


**Vol-3250**

**urn:nbn:de:0074-3250-0**



# STAF-WS 2022

# Software Technologies: Applications and Foundations Workshops 2022

**STAF 2022 Workshop Proceedings: 10th International Workshop on Bidirectional Transformations (BX 2022), 2nd International Workshop on Foundations and Practice of Visual Modeling (FPVM 2022) and 2nd International Workshop on MDE for Smart IoT Systems (MeSS 2022) (co-located with Software Technologies: Applications and Foundations federation of conferences (STAF 2022))**

**Nantes, France, July 5--8, 2022.**

**Edited by**

**Catherine Dubois** *
**Julien Cohen** **

* ENSIIE, Evry-Courcouronnes, France
** Nantes Université, Nantes, France

# Table of Contents

- Model-Driven Development of Digital Twins for Supervision and Simulation of Sensor-and-Actuator Networks (Extended Abstract)
  *Gaël Pichot*, *Jérôme Rocheteau*, *Christian Attiogbé*
- SpeakWell or Be Still: Solving Conversational AI with Weighted Attribute Grammars (Poster)
  *Vadim Zaytsev*
- Smart Home Model Verification with AnimUML (Poster)
  *Frédéric Jouault*, *Ciprian Teodorov*, *Matthias Brun*
- SimulateIoT-Federations: Domain Specific Language for designing and executing IoT simulation environments with Fog and Fog-Cloud federations (Poster)
  *José A. Barriga*, *Pedro J. Clemente*

---

2022-08-29: submitted by Catherine Dubois, metadata incl. bibliographic data published under Creative Commons CC0

2022-10-26: published on CEUR Workshop Proceedings (CEUR-WS.org, ISSN 1613-0073) |valid HTML5|

# Message from the STAF 2022 Workshop Chairs

This volume contains the technical papers presented at three workshops co-located with the 2022 edition of the *Software Technologies: Applications and Foundations (STAF)* federation of conferences on software technologies, namely the tenth International Workshop on Bidirectional Transformations (BX 2022), the second International Workshop on Foundations and Practice of Visual Modeling (FPVM 2022) and the second International Workshop on MDE for Smart IoT Systems (MeSS 2022).

The STAF conferences, satellite events and workshops bring together leading researchers and practitioners from academia and industry to advance the state of the art in practical and foundational advances in software technology. They address all aspects of software technology, from object-oriented design, testing, mathematical approaches to modeling and verification, transformation, model-driven engineering, aspect-oriented techniques, and tools.

STAF 2022 was organized as an hybrid event by IMT-Atlantique and Nantes Université, from July 5 to July 8, 2022 in Nantes (France). STAF 2022 hosted the following workshops:

- 10th International Workshop on Bidirectional Transformations (Bx 2022), organizers: Li-yao Xia (University of Pennsylvania, US), He Xiao (SCCE USTB, Beijing, China) and Vadim Zaytsev (University of Twente, Netherlands),
- 2nd International Workshop on Foundations and Practice of Visual Modeling (FPVM 2022), organizers: Amleto Di Salle (University of L'Aquila, L'Aquila, Italy), Ludovico Iovino (Gran Sasso Science Institute, L'Aquila, Italy), Alfonso Pierantonio (University of L'Aquila, L'Aquila, Italy) and Juha-Pekka Tolvanen (MetaCase, Finland),
- 13th International Workshop on Graph Computation Models (GCM 2022), organizers: Reiko Heckel (University of Leicaster, UK) and Chris Poskitt (Singapore Management University, Singapore),
- 2nd International Workshop on MDE for Smart IoT Systems (MeSS 2022), organizers: Federico Ciccozzi (Malardalen University, Sweden), Nicolas Ferry (University of Nice Côte D'Azur, France), Ludovico Iovino (Computer Science Department – Gran Sasso Science Institute, Italy), Sebastien Mosser (McMaster University, Canada), Arnor Solberg (Tellu AS, Norway) and Manuel Wimmer (KU Linz, Austria).

We would like to thank the workshop organizers who made their events successful. We are grateful to the local organizers of STAF 2022 for their trust and support. After the pandemic, it was a real pleasure to discuss research topics in person.

*STAF 2022 Workshops, 05–08 July, 2022, Nantes, France*



Finally, we would also like to thank all the participants and people who have contributed to the events as well as the respective Program Committee members and external reviewers who carried out thorough and careful reviews.

August, 2022

Catherine Dubois (Ecole Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise, France)
Julien Cohen (Nantes Université, France)

# Report on the Tenth International Workshop on Bidirectional Transformations (BX 2022)

Xiao He[1], Li-yao Xia[2] and Vadim Zaytsev[3]

[1]University of Science and Technology Beijing, China
[2]University of Edinburgh, United Kingdom
[3]University of Twente, The Netherlands

**Abstract**

Bidirectional transformations (BX) are a mechanism for maintaining the consistency between two or more related and heterogeneous sources of information (e.g., relational databases, software models and code, or any other artefacts following standard or domain-specific formats). The strongest argument in favour of BX is its ability to provide a synchronisation mechanism that is guaranteed to be correct by construction. BX has been attracting a wide range of research areas and communities, with prominent presence at top conferences in several different fields (namely databases, programming languages, software engineering, graph transformation). The fast-growing complexity of software- or data-intensive systems forced industry and academia to use and investigate different development techniques to manage many different aspects of the systems. Researchers are actively investigating the use of bidirectional approaches to tackle a diverse set of challenges with various applications including model-driven software development, visualisation with direct manipulation, big data, databases, domain-specific languages, serialisers, data transformation, integration and exchange. BX 2022 is a dedicated venue for BX in all relevant fields and is part of a workshop series that was created in order to promote cross-disciplinary research and awareness in the area.

## 1. Organisation

The BX workshop series has been running steadily since its conception in 2012 (rotating among ETAPS, STAF, IDBT/ICDT, ⟨Programming⟩, PLW) until 2020 when it was postponed/cancelled together with the rest of STAF 2020 programme due to the pandemic. It has resumed its course in 2021 in a virtual format, and this 2022 edition ran smoothly in a hybrid format, accommodating physical participants in Nantes in France, as well as remote participants via Zoom and Webex. Many names like Frédéric Jouault or Soichiro Hidaka, familiar to the regular BX community but not belonging to on-site attendees, have lighted up on the screen throughout the day and channelled their questions and comments to presenters.

We have received five submissions, and after collecting 2–3 reviews per paper, we ended up with two accepted submissions. Then, we have solicited invited presentations from authors of recently published BX-related papers at ICSE 2022, JSS 189 (2022) and SoSyM 20:5 (2021). The last session of the day was decided to be spent on a plenary discussion.

*BX'22: Tenth International Workshop on Bidirectional Transformations, 5 July 2022, Nantes, France*
hexiao@ustb.edu.cn (X. He); xialiyao@seas.upenn.edu (L. Xia); vadim@grammarware.net (V. Zaytsev)
https://ustbmde.bitbucket.io/hexiao (X. He); https://poisson.chat (L. Xia); https://grammarware.net (V. Zaytsev)
0000-0003-2673-4400 (L. Xia); 0000-0001-7764-4224 (V. Zaytsev)

We would like to thank the programme committee members, dedicating their time and expertise to reviewing the submissions:

- **Jens Weber**, University of Victoria
- **Michael Johnson**, Macquarie University
- **Ravi Chugh**, University of Chicago
- **Perdita Stevens**, The University of Edinburgh
- **Fernando Orejas**, Universitat Politècnica de Catalunya
- **Leen Lambers**, Brandenburgische Technische Universität Cottbus-Senftenberg
- **Hsiang-Shang Ko**, Institute of Information Science, Academia Sinica, Taiwan
- **Kazutaka Matsuda**, Tohoku University

Our appreciation also goes to all the remote participants who had to bear with the occasional technical imperfections, and especially to physical participants who were literally there for us, and made BX into a successful event with lively discussions and useful insights.

## 2. Keynote

**Zhenjiang Hu** is a chair professor in School of Computer Science of Peking University, and a professor of NII by special appointment. He received his B.S. and M.S. degrees from Shanghai Jiao Tong University in 1988 and 1991, respectively, and Ph.D. degree from University of Tokyo in 1996. He was a lecturer (1997–2000) and an associate professor (2000–2008) at University of Tokyo, a full professor at NII (2008-2019), and a full professor at University of Tokyo (2018-2019), before joining Peking University in 2019. His main research interest is in programming languages and software engineering in general, and functional programming and bidirectional programming in particular. He is Fellow of IEEE, Fellow of JFES (Japan Federation of Engineering Society), Member of Engineering Academy of Japan, and Member of Academy of Europe.

**Dejima: A Bidirectional Collaborative Framework for Decentralized Data Management**

*Data management systems are now moving from "centralised" towards "decentralised", where data are maintained in different sites with autonomous storage and computation capabilities. There are two fundamental issues with such decentralized systems: local privacy and global consistency. By local privacy, the owner of the data wish to control what information should be exposed and how it should be used or updated by other peers. By global consistency, the systems wish to have a globally consistent and integrated view of all data. In this talk, we shall report the progress of our BISCUITS project [1] that attempts to systematically solve these two issues in decentralized systems. In particular, we present a new bidirectional transformation-based approach to controlling and sharing distributed data based on the view, describe Dejima [2], a new architectures for data integration via bidirectional updatable views, and discuss various applications.*

The keynote presentation itself covered the history of bidirectional transformations, tracing back to its origins all the way back to VLDB 1978 [3] and POPL 2005 [4] and the seminal Shonan report of 2008 [5]. A fundamental distinction was made between, on one side, forward-based or get-based BX for trees [4], tables [6], strings [7], XML [8] and graphs [9], and, on the other side,

backward-based of putback-based BX [10] for trees [11], tables [12] and graphs [13]. Workshop participants could hear some advise on how to develop dependable, correct by construction, BX, with examples from the presenter's own projects like BiGUL [14] and BIRDS [15], as well as the newest of them: Dejima [2].

## 3. Papers

**Weixin Zhang** presented *Decomposition Without Regret*, joint work with Cristina Daivid and Meng Wang, and used the well-known Expression Problem [16] to motivate the need to support both object-oriented and functional decomposition. Their tool called Cook, in memory of late William Cook, who contributed heavily to the understanding of abstraction and decomposition [17]. The authors have also extended their talk proposal into a two-page summary which can be found in this STAF 2022 post-proceedings.

**Nils Weidmann** presented *Bidirectional Transformations in Practice: An Automotive Perspective on Traceability Maintenance*, joint work with Anthony Anjorin and Katharina Artic. Within the context of the automotive industry, they discuss current and potential applications of BX to maintaining models in tandem between requirement management and architectural modelling tools, ensuring their consistency throughout the evolution of a project. Key constraints are identified, stemming from the high-level of abstraction of the models under study and the regulatory nature of the industry. Three main solution strategies are investigated, as a way to classify existing tools and to highlight possible avenues for BX applications. Their nine page submission can also be found in this STAF 2022 post-proceedings.

## 4. Invited talks

**Bernhard Westfechtel** presented *BXtendDSL: A layered framework for bidirectional model transformations combining a declarative and an imperative language*, joint work with Thomas Buchmann and Matthias Bank. A common approach in BX is to design domain-specific languages with a strong consistency guarantee between the two directions of a program, but such languages generally sacrifice expressiveness, reducing their practicality. BXtendDSL follows a more pragmatic approach. From a high-level declarative DSL, code is generated against a low-level API, viewed as an imperative DSL. The generated code offers extension points, enabling users to handle operational details that go beyond the scope of the high-level DSL. The resulting language is not only expressive, but also concise and scalable. The six page submission titled *Engineering Bidirectional Model Transformations* that the authors submitted for the STAF 2022 post-proceedings, can be seen as a follow-up to their JSS paper [18]

**Gábor Bergmann** presented *Controllable and decomposable multidirectional synchronizations*. Whereas BX has traditionally focused on transformations between a source and a single view, this work studies systems with many concurrently synchronised views, called multidirectional synchronisations (MX). This work identifies fundamental challenges in this setting. Notably, a "whack-a-mole" behaviour may arise, where even with individually well-behaved views, local updates can never reach a globally consistent state. Generalisations of standard BX properties as well as novel *axioms of regularity* are investigated. Remarkably, history-ignorance and very-

well-behaved-ness, which coincide for BX, generalise differently for MX. The author decided that his original SoSyM paper [19] does not require a follow-up yet.

**Xing Zhang** presented *Towards Bidirectional Live Programming for Incomplete Programs*, joint work with Zhenjiang Hu. Bidirectional live programming allows programmers to edit a program by modifying its output. This work extends that idea to also allow editing of incomplete programs. They study a core language for bidirectional live programming featuring explicit holes. During execution, holes in the program become holes in the output. A key mechanism is that output holes also carry the environment in which they were created, allowing one to distinguish different instances of the same hole in the source program. This language is implemented in a tool called Bidirectional Preview, providing a support to perform benchmarks and to demonstrate compelling examples. The authors decided that their original ICSE paper [20] does not require a follow-up yet.

## 5. Discussion panel

We would like to thank Nils Weidmann, Robbert Jongeling, Massimo Tisi, Soichiro Hidaka, James William Pontes Miranda and Simon Dierl for their contributions to this discussion. All three workshop chairs also participated.

Some of the major discussion points were:

- The Bx Examples Repository [21] at http://bx-community.wikidot.com/examples:home
- Forward-facing and backward-facing BX
- Challenges and competitions organised by other communities [22, 23, 24]
- Possible competition/comparison of approaches or even DSLs
- Impact of shared cases on replicability if the core of the problem for each case is known and stateed
- Using JANI [25] https://jani-spec.org as an interchange format helped the community of probabilistic model checking to foster tool interoperation and comparison, could we at some point converge to such a format for BX?
- “Editing generated code” as one of the most common BX cases in practice
- Many alternative possible explorable models of synchronisation with negotiated updates
- Smell detection in synchronisation problems
- More demanding properties and stronger guarantees vs realistic properties and weaker guarantees
- BX in an advisory role to suggest manual changes
- Maintenance of BX in an industrial context
- Restoring consistency with machine learning
- (Partial) bidirectionalisation and defining a bidirectional semantics for a language
- Impossibility to prove nice theoretical properties in a practical context
- Using known theory to study non-well-behaved systems
- Tolerance to imperfect/incomplete practical solutions

## References


[1] Z. Hu, et al., BISCUITS: Bidirectional Information Systems for Collaborative, Updatable, Interoperable, and Trusted Sharing, http://www.biscuits.work, 2017.

[2] K. Miyake, Dejima Prototype, https://github.com/ekayim/dejima-prototype, 2020.

[3] U. Dayal, P. A. Bernstein, On the Updatability of Relational Views, in: Proceedings of the Fourth International Conference on Very Large Data Bases (VLDB), IEEE Computer Society, 1978, pp. 368–377.

[4] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, A. Schmitt, Combinators for Bi-directional Tree Transformations: A Linguistic Approach to the View Update Problem, in: Proceedings of the 32nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (POPL), ACM, 2005, pp. 233–246. doi:10.1145/1040305.1040325.

[5] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, J. F. Terwilliger, Bidirectional Transformations: A Cross-Discipline Perspective, in: Proceedings of the Second International Conference on Theory and Practice of Model Transformations (ICMT), volume 5563 of *LNCS*, Springer, 2009, pp. 260–283. doi:10.1007/978-3-642-02408-5_19.

[6] J. N. Foster, T. J. Green, V. Tannen, Annotated XML: Queries and Provenance, in: Proceedings of the 27th Symposium on Principles of Database Systems (PODS), ACM, 2008, pp. 271–280. doi:10.1145/1376916.1376954.

[7] A. Bohannon, J. N. Foster, B. C. Pierce, A. Pilkiewicz, A. Schmitt, Boomerang: Resourceful Lenses for String Data, in: Proceedings of the 35th Symposium on Principles of Programming Languages (POPL), ACM, 2008, pp. 407–419. doi:10.1145/1328438.1328487.

[8] D. Liu, Z. Hu, M. Takeichi, An Environment for Maintaining Computation Dependency in XML Documents, in: Proceedings of the Fifth Symposium on Document Engineering (DocEng), ACM, 2005, pp. 42–51. doi:10.1145/1096601.1096616.

[9] S. Hidaka, Z. Hu, K. Inaba, H. Kato, K. Matsuda, K. Nakano, Bidirectionalizing Graph Transformations, in: Proceeding of the 15th International Conference on Functional Programming (ICFP), ACM, 2010, pp. 205–216. doi:10.1145/1863543.1863573.

[10] Z. Hu, H. Pacheco, S. Fischer, Validity Checking of Putback Transformations in Bidirectional Programming, in: Proceedings of the 19th International Symposium on Formal Methods (FM), volume 8442 of *LNCS*, Springer, 2014, pp. 1–15. doi:10.1007/978-3-319-06410-9_1.

[11] H. Ko, Z. Hu, An axiomatic basis for bidirectional programming, Proceedings of the ACM on Programming Languages 2 (2018) 41:1–41:29. doi:10.1145/3158129.

[12] V. Tran, H. Kato, Z. Hu, Programmable View Update Strategies on Relations, Proceedings of the VLDB Endowment 13 (2020) 726–739. URL: http://www.vldb.org/pvldb/vol13/p726-tran.pdf. doi:10.14778/3377369.3377380.

[13] X. He, Z. Hu, Putback-based bidirectional model transformations, in: Proceedings of the Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE), ACM, 2018, pp. 434–444. doi:10.1145/3236024.3236070.

[14] H. Ko, T. Zan, Z. Hu, BiGUL: A Formally Verified Core Language for Putback-based Bidirectional Programming, in: Proceedings of the 21st Workshop on Partial Evaluation and Program Manipulation (PEPM), ACM, 2016, pp. 61–72. doi:10.1145/2847538.2847544.

[15] V. Tran, H. Kato, Z. Hu, BIRDS: Programming view update strategies in Datalog, Proceedings of the VLDB Endowment 13 (2020) 2897–2900. URL: http://www.vldb.org/pvldb/vol13/p2897-tran.pdf. doi:10.14778/3415478.3415503.

[16] P. Wadler, The expression problem, Posted on the Java Genericity mailing list, http://homepages.inf.ed.ac.uk/wadler/papers/expression/expression.txt, 1998.

[17] W. R. Cook, On understanding data abstraction, revisited, in: Proceedings of the 24th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages, and Applications, OOPSLA 2009, October 25-29, 2009, Orlando, Florida, USA, ACM, 2009, pp. 557–572. doi:10.1145/1640089.1640133.

[18] T. Buchmann, M. Bank, B. Westfechtel, Bxtenddsl: A layered framework for bidirectional model transformations combining a declarative and an imperative language, Journal of Systems and Software 189 (2022) 111288. doi:10.1016/j.jss.2022.111288.

[19] G. Bergmann, Controllable and Decomposable Multidirectional Synchronizations, Software and Systems Modeling 20 (2021) 1735–1774. doi:10.1007/s10270-021-00879-w.

[20] X. Zhang, Z. Hu, Towards bidirectional live programming for incomplete programs, in: Proceedings of the 44th IEEE/ACM 44th International Conference on Software Engineering (ICSE), ACM, 2022, pp. 2154–2164. doi:10.1145/3510003.3510195.

[21] J. Cheney, J. McKinna, P. Stevens, J. Gibbons, Towards a repository of bx examples, in: Proceedings of the Workshops of the EDBT/ICDT 2014 Joint Conference, volume 1133 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014, pp. 87–91. URL: http://ceur-ws.org/Vol-1133/paper-14.pdf.

[22] B. E. Aydemir, A. Bohannon, M. Fairbairn, J. N. Foster, B. C. Pierce, P. Sewell, D. Vytiniotis, G. Washburn, S. Weirich, S. Zdancewic, Mechanized metatheory for the masses: The poplmark challenge, in: Proceedings of the 18th International Conference on Theorem Proving in Higher Order Logics (TPHOLs), volume 3603 of *LNCS*, Springer, 2005, pp. 50–65. doi:10.1007/11541868_4.

[23] D. Beyer, Progress on software verification: Sv-comp 2022, in: D. Fisman, G. Rosu (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, Springer, Cham, 2022, pp. 375–402.

[24] A. Hartmanns, M. Klauck, D. Parker, T. Quatmann, E. Ruijters, The quantitative verification benchmark set, in: Proceedings of the 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 11427 of *LNCS*, Springer, 2019, pp. 344–350. doi:10.1007/978-3-030-17462-0_20.

[25] C. E. Budde, C. Dehnert, E. M. Hahn, A. Hartmanns, S. Junges, A. Turrini, JANI: quantitative model and tool interaction, in: Proceedings of the 23rd International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS), volume 10206 of *LNCS*, 2017, pp. 151–168. doi:10.1007/978-3-662-54580-5_9.

# Decomposition Without Regret (Poster)

Weixin Zhang[1], Cristina David[1] and Meng Wang[1]

[1]*University of Bristol, United Kingdom*

Abstract

Programming languages are embracing both functional and object-oriented paradigms. A key difference between the two paradigms is the way of achieving data abstraction. That is, how to organize data with associated operations. There are essential tradeoffs between functional and object-oriented decomposition regarding extensibility and expressiveness. Unfortunately, programmers are usually forced to select a particular decomposition style in the early stage of programming. Once the wrong design decision has been made, the price for switching to the other decomposition style could be rather high since pervasive manual refactoring is often needed.

In this talk, we show a bidirectional transformation system between functional and object-oriented decomposition. We formalize the core of the system in the **FOOD** calculus, which captures the essence of functional and object-oriented decomposition. We prove that the transformation preserves the type and semantics of the original program. We further implement **FOOD** in Scala as a translation tool called **Cook** and conduct several case studies to demonstrate the applicability and effectiveness of **Cook**.

**Keywords**

Bidirectional program transformation, Functional decomposition, Object-oriented decomposition

Programming languages are embracing multiple paradigms, in particular functional and object-oriented paradigms. Modern languages are designed to support multi-paradigms. Well-known examples include OCaml, Swift, Rust, TypeScript, Scala, F#, and Kotlin. Meanwhile, mainstream object-oriented languages such as C++ and Java are gradually extended to support functional paradigms. When multiple paradigms are available within one programming language, a natural question arises: *which paradigm to choose when designing programs*?

A fundamental difference between functional and object-oriented paradigms is the way of achieving *data abstraction* [1, 2]. That is, how to organize data with associated operations. If we view a program as a matrix, data variants and operations are then the rows and columns of that matrix respectively. Object-oriented programming decomposes the program *by row* and is *operation first*: we first declare an interface that describes the operations supported by the data and then implement that interface with some classes. Conversely, functional programming decomposes the program *by column* and is *data first*: we first represent the data using an algebraic datatype and then define operations by pattern matching on that algebraic datatype.

There are important tradeoffs between functional and object-oriented decompositions in terms of extensibility and expressiveness. As acknowledged by the notorious Expression Problem [1, 3, 4], these two decomposition styles are complementary in terms of *extensibility*. Object-oriented decomposition makes it easy to extend data variants through defining new classes. On the other hand, functional decomposition makes it easy to add new operations on

*STAF 2022 Workshop: Tenth International Workshop on Bidirectional Transformations (BX 2022)*

weixin.zhang@bristol.ac.uk (W. Zhang); cristina.david@bristol.ac.uk (C. David); meng.wang@bristol.ac.uk (M. Wang)

expressions. Besides extensibility, object-oriented and functional decomposition have different expressive power. Object-oriented decomposition facilitates code reuse through inheritance and enables *interoperability* between different implementations of the same interface whereas functional decomposition allows inspection on the internal representation of data through (nested) pattern matching, simplifying abstract syntax tree transformations.

Unfortunately, programmers are forced to decide a decomposition style in the early stage of programming. A proper choice, however, requires predictions on the extensibility dimension and kinds of operations to model, which may not be feasible in practice. Once the wrong design decision was made, the price for switching to the other decomposition style could be rather high since pervasive manual refactoring is often needed.

A better way, however, allows programmers to choose a decomposition style for prototyping without regret. When the design choice becomes inappropriate, a tool automatically transforms their code into another style without affecting the semantics. Even at later stages, such a automatic translation tool could be used to make extensions of data variants or operations easier by momentarily switching the decomposition, adding the extension, and then transforming the program back to the original decomposition. Furthermore, studying the transformation between the two styles can provide a theoretical foundation for compiling multi-paradigm languages into single-paradigm ones. From an educational perspective, the tool can help novice programmers to understand both decomposition styles better.

To address this issue, we propose a bidirectional transformation between functional and object-oriented decomposition based on the observation that restricted forms of functional and object-oriented decomposition are *symmetric*. We formalize an automatic, type-directed transformation in the core calculus **FOOD**, which captures the essence of Functional and Object-Oriented Decomposition. We prove that the transformation preserves the type and semantics of the original program. We further implement **FOOD** in Scala as a translation tool called **Cook** and conduct several case studies to demonstrate the applicability of **Cook**. Interested readers may want to consult the full paper for more details [5].

## References


[1] J. C. Reynolds, User defined types and procedural data structures as complementary approaches to data abstraction, in: D. Gries (Ed.), Programming Methodology, A Collection of Articles by IFIP WG2.3, Springer-Verlag, New York, 1978, pp. 309–317. Reprinted from S. A. Schuman (ed.), *New Advances in Algorithmic Languages 1975*, Inst. de Recherche d'Informatique et d'Automatique, Rocquencourt, 1975, pages 157-168. Also in taoop.

[2] W. R. Cook, On Understanding Data Abstraction, Revisited, in: Proceedings of the 24th ACM SIGPLAN Conference on Object Oriented Programming Systems Languages and Applications, OOPSLA '09, 2009, pp. 557–572. doi:10.1145/1639949.1640133.

[3] W. R. Cook, Object-oriented programming versus abstract data types, in: Foundations of Object-Oriented Languages, Springer, 1991, pp. 151–178. doi:10.1007/BFb0019443.

[4] P. Wadler, The Expression Problem, 1998. Note to Java Genericity mailing list.

[5] W. Zhang, C. David, M. Wang, Decomposition without regret, 2022. URL: https://arxiv.org/abs/2204.10411. doi:10.48550/ARXIV.2204.10411.

# Bidirectional Transformations in Practice: An Automotive Perspective on Traceability Maintenance (Short Paper)

Anthony Anjorin[1], Nils Weidmann[2] and Katharina Artic[1]

[1]IAV GmbH, Germany
[2]Paderborn University, Germany

**Abstract**

Bidirectional transformations (bx) are used to maintain the consistency of two or more artefacts as they are concurrently updated, typically by different people, often using different tools. While there has been active research on bx for some time with numerous examples and industrial case studies from research projects, it is still a valid question if bx is absolutely necessary in practice. Indeed, if productivity and simplicity are most important, perhaps processes and tool chains can be chosen to avoid bx as much as possible. In this experience report, we provide an automotive perspective on the need for and application of bx to traceability maintenance. We share our experiences from relevant projects, focusing on challenges and constraints in the problem domain, and discussing solution strategies we have applied and evaluated. Our aim is to provide a concrete characterisation of bx-related solution strategies to traceability maintenance in practice, which we hope serves as motivation and input for bx researchers.

**Keywords**

Bidirectional Transformations, Automotive Engineering, Traceability Maintenance

## 1. Introduction

Automotive projects require the concurrent engineering of multiple artefacts and involve groups of experts working on multiple artefacts with different tools. The development of these artefacts or *models* must often comply with process assessment and reference standards such as Automotive SPICE (ASPICE) [1], with relevant emission laws, and with numerous regulations concerning, e.g., functional safety and security. These standards typically demand a series of systematic and documented reviews as a means of ensuring that all models are consistent and that quality goals are met. In this context, mappings between semantically related elements of different domain models are described by *traceability links*, which can be technically represented in various forms [2] (e.g., in form of correspondences when describing a inter-model consistency by means of triple graph grammars [3]). Traceability links between model elements play an important role in supporting reviews and (manual) consistency checks, and are often stipulated or at least suggested as a best practice by many standards. Maintaining traceability links productively, i.e., letting users directly operate on them instead of just managing them in the

*Tenth International Workshop on Bidirectional Transformations (BX 2022)*
tony@anjorin.de (A. Anjorin); nils.weidmann@upb.de (N. Weidmann); katharina.artic@iav.de (K. Artic)

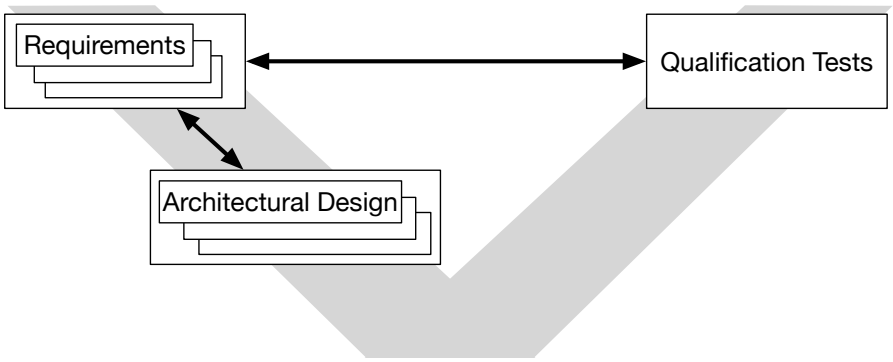


**Figure 1:** Our focus in the automotive domain

background, is thus a relevant challenge in this context, especially as the connected models co-evolve.

Bidirectional transformations (bx) have been actively researched for some time as a means of maintaining the consistency of two or more artefacts [4]. While there is now a curated collection of bx examples [5] as well as numerous reports on industrial case studies from research projects [6, 7, 8, 9, 10], we claim that input from industrial practitioners on applying bx in practice is still largely missing. As the research questions identified for research projects aim to cover novel aspects and tend to be increasingly visionary, it is useful to have a reality check with practical challenges in real-world projects as a means of possibly steering future research questions and work. In this paper, therefore, we report on applying bx to traceability maintenance [11] in the automotive domain. We share our experiences from relevant projects, focusing on typical limitations and constraints in the problem domain, and discussing bx-related solution strategies we have applied and evaluated, as well as open challenges.

As depicted in Figure 1, our primary focus in most projects has been in the upper-left corner of the V-model often stipulated by automotive process reference standards for (software) systems engineering. In the diagram, rectangles represent models, while the bidirectional black arrows represent traceability links between model elements. The stacked rectangles for requirements and architectural design indicate that the models we are interested in often comprise multiple layers. E.g., stakeholder, system, and component for requirements, and functional, logical, and physical for architectural design models. Qualification tests are expected to be on the same abstraction level as requirements. Relevant traceability links are mostly between requirements and architectural design, as well as between requirements and qualification tests, and are typically required to be bidirectionally navigable. The goal is often to support a holistic system view without committing prematurely to a purely software-based or hardware-based view of the system. In a few projects, however, we have also addressed the problem of connecting the conceptual, modelling world to the final executable code, especially when the latter is not automatically generated. This focus of projects in the last 3-5 years shapes and limits the automotive perspective we can report on, and does not necessarily reflect other parts of the automotive domain.

## 2. A characterisation of the problem domain

To simplify the discussion, we focus in the rest of the paper on the traceability links between requirements and architectural design models. Qualification tests can be handled similarly.

Although some requirements can be generated automatically from parts of the architecture in a few cases, the norm in our projects is that both requirements and architectural design models are created and maintained manually. Similarly, most traceability links between requirements and architectural elements can only be created and checked manually, and represent an explicit documentation of a relationship between the elements, e.g., "realised by". There is typically no chance to define a formal consistency relation from which, e.g., traceability links can be created or checked automatically as correspondences. The need for some form of automation comes more from the task of *maintaining* traceability links possibly across tool boundaries while the involved models constantly change. According to our experience, the following requirements for productive traceability maintenance are to be adequately addressed by an acceptable solution, irrespective of the applied solution strategy:

- **R1** Traceability links between requirements and architectural elements can be created, navigated, versioned, and generally used productively in either the requirements management tool or the architectural modelling tool (or sometimes in both). Convincing end users to use an additional tool solely for traceability maintenance is – in our experience – difficult.
- **R2** Traceability links that are clearly broken by changes that were made to one or both of the models are marked but preserved in a way that they can be reviewed and handled manually.
- **R3** A set of "suspect" traceability links can be determined for a manual review in some configurable manner (e.g., links connected to elements that have changed). This point is often optional as (R2) already requires a considerable amount of manual work.

The problem domain can be further characterised by the following constraints and limitations:

**The underlying development process must first be clarified:** In our experience, developing a feasible solution for traceability maintenance always requires a clear development process defining who changes what when and with which tool. In most projects, however, the development process is usually implicit, unclear, and not yet communicated to and accepted by all relevant stakeholders. Some questions that we investigate include: Why are traceability links required and by whom? Who is going to create and maintain these traceability links when using which tool? What parts of which models can be changed by whom in which tool?

**A complex combination of different concerns:** A further challenge is that traceability maintenance must be usually combined with the following intertwined concerns: (i) A flexible versioning of all models is required as different engineers work concurrently on different releases. This affects traceability as links might have to be created to elements from a particular baseline. The problem here is that many modelling tools support versioning in their own unique way, if they do at all. (ii) Variant management is also a concern as multiple, similar systems are often planned and designed together to promote reuse of certain parts of models. This also

affects traceability, however, as a compatible strategy for variant management of traceability links must also be established.

**A restricted choice of available technology:** Working for different clients on different projects, the available technology in the solution domain tends to be fixed or at least severely limited by numerous factors including the set of tools already in use at a company, the tools for which the client's IT is prepared to obtain and support licenses, and of course budgetary constraints. While we sometimes provide consulting regarding the choice of tools, we also often have to cope with the current tools in use.

**No compiler or test suite to validate automated decisions:** In contrast to working with executable code or simulations, working with models on a high, conceptual level typically means that one cannot rely on a powerful compiler or an extensive test suite to catch most mistakes and inconsistencies. Indeed, while we strive to implement as many basic validation rules as possible, automated decisions made by a tool, e.g., whether a link between a certain requirement and an architectural model element is still valid after applying a change, must be manually reviewed.

**There are indeed factors that simplify the task of traceability maintenance:** Compared to working with code and parser-based systems, most requirements management tools and architectural modelling tools provide unique identifiers for all models elements. While this comes with its own set of challenges, unique identifiers generally simplify numerous tasks including determining changes (deltas) and correspondences (corrs) by simply comparing two models.

As the requirement and architectural models are on a relatively high-level of abstraction, scalability is usually not an issue in this context. Indeed, as numerous manual reviews and expert discussions must still be possible, clarity is more important than completeness.

## 3. Current solution strategies

We now discuss the solution domain describing the most common solution strategies we have seen and applied ourselves in projects. We again restrict the following discussion to requirements management (RM) and architectural modelling (AM).

### 3.1. A single tool for all models

A common solution strategy is to provide an all-in-one tool that supports both RM and AM (and all else that is required) in an integrated manner. Such a tool has a better chance of supporting versioning, variant management, and traceability maintenance in a compatible manner. Traceability maintenance is typically supported by immediately checking all consequences of a change, and if necessary rejecting it or at least prompting for a confirmation from the user. Although this is attractive, it is also problematic. We have seen this one-size-fits-all strategy fail in practice due to missing acceptance from a group of users. Such a tool tends to become quite

complex and unwieldy, e.g., for users only or primarily concerned with RM. Such tools often have a primary focus and end up being, e.g., much better for AM than for RM even though both are supported.

In general, this strategy eventually breaks down as new models are added and have to be maintained using separate tools, resulting in an awkward mix of almost everything in one complex tool and still a number of separate tools added ad-hoc to the mix. Nonetheless, many tools on the market currently take this approach such as Enterprise Architect,[1] Capella,[2] PREEvision,[3] and Cameo systems modeller.[4] As points R1 – R3 can be fully addressed by such an integrated solution, we believe this can be a viable solution strategy if the tool is crafted or at least substantially tailored for a specific client and project with a stable set of models.

### 3.2. Separate tools with a bx to synchronise a common overlap

A flexible solution that still addresses R1 adequately, i.e., allows users to perform traceability maintenance in their own tool, is to identify the parts of different models that are relevant for traceability, and to keep this "overlap" synchronised using bx as all models co-evolve.

This strategy is depicted in Figure 2 using notation and terminology from the lens framework for bx (we refer readers new to lenses to, e.g., Johnson et al. [12] for a unified overview and comparison of the different lens formal frameworks for bx). Arrows with a white fill represent applications of the functions indicated via the label of the arrow, unidirectional arrows with a black fill represent changes (deltas) applied to the models, while bidirectional arrows with a black fill represent traceability links. Grey circles/rectangles indicate parts of the models representing requirements/architectural design. Dashed rectangles indicate the boundaries of individual models in either tool.

An RM tool containing requirement models is to be used together with an AM tool containing architectural design models. The decision has been made in this case to enable traceability maintenance only in the AM tool. To support this, a relevant part of the requirement models must be identified, and then propagated to the AM tool where it has to be represented in some manner. Now users of the AM tool have representatives of relevant parts of the requirements models in their own tool and can create and use traceability links. When the requirement models are changed in the RM tool, however, consistency must be reestablished as follows (the steps are indicated in Figure 2 in small black circles)

1. A user of the RM tool makes changes ($\Delta_{RM}$) to the requirement model.
2. The synchronisation starts by extracting the relevant parts of the changed requirements model using the function $get_{RM}$.
3. To determine what was changed, the same overlap is extracted from the current architectural design model in the AM tool using the function $get_{AM}$. Assuming a consistent starting point, this is expected to be identical to what $get_{RM}$ would produce when applied to the old requirements model (depicted as a greyed out white arrow in the figure).

[1] https://sparxsystems.com/products/ea/
[2] https://www.eclipse.org/capella/
[3] https://www.vector.com/de/de/produkte/produkte-a-z/software/preevision/
[4] https://www.3ds.com/products-services/catia/products/no-magic/cameo-systems-modeler/

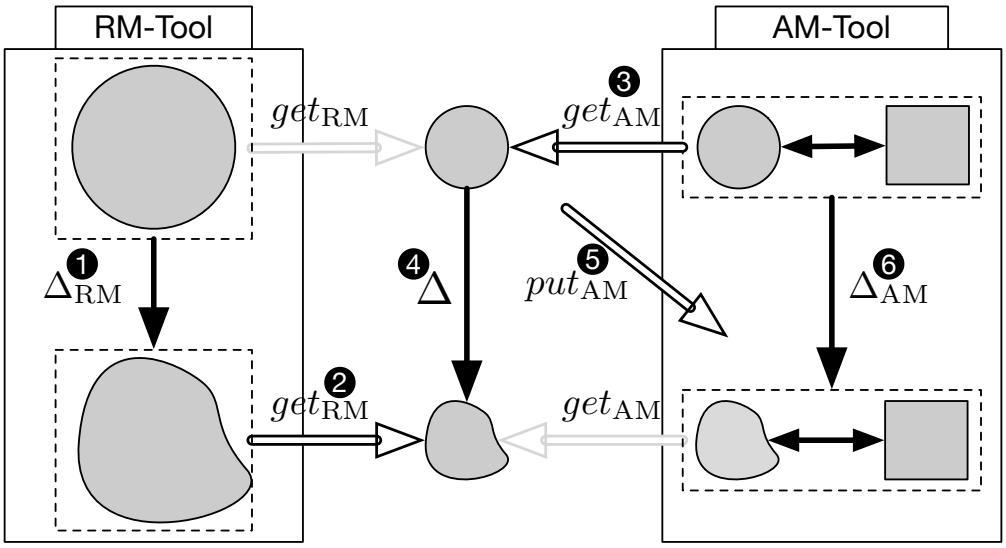


**Figure 2:** Using bx to synchronise a common overlap for traceability maintenance

4. The set of relevant changes $\Delta$ can now be computed by comparing the unique identifiers of the elements in the extracted overlaps (the source and target of the arrow $\Delta$). For this to be possible, note that the identifiers used in the RM tool must be saved as part of the representation in the AM tool.
5. The set of relevant changes $\Delta$ can now be propagated to the AM tool using $put_{AM}$ in such a way that R2 and R3 are addressed adequately. This includes, e.g., conservative strategies of handling deletions, so that the AM user can review and ultimately decide how to handle broken traceability links.
6. The result of applying $put_{AM}$ is represented as the induced change $\Delta_{AM}$ of the model in the AM tool, typically only affecting the overlap from the requirements model and of course traceability links to architectural model elements. If $get_{AM}$ is now applied to the resulting architectural model (depicted as a greyed out white arrow in the figure), the result is expected to be identical to the overlap produced by $get_{RM}$ in Step 2.

To simplify the required bx, process-based conventions can be established to limit which parts of the models can be changed in which tools. To apply Figure 2, for instance, all users must agree that (i) traceability links can only be created and maintained in the AM tool, (ii) the representation of the relevant parts of the requirements models in the AM tool should only be directly manipulated by $put_{AM}$ and not by users of the AM tool.

To implement this solution strategy in practice, we therefore spend a substantial part of our time performing a process analysis, i.e., suggesting and evaluating different processes, discussing cost/effort vs. benefit in each case, and interviewing relevant stakeholders to determine what is feasible/acceptable and what not.

In our experience, modern RM tools such as Codebeamer[5] and Jama[6] are well-suited for applying this strategy as they (i) provide a relatively complete (REST-)API for flexible data manipulation, and (ii) are flexible enough to allow new types of model elements to be introduced to represent, e.g., architectural elements or test cases.

### 3.3. An extra (backend) tool for traceability maintenance

A final strategy involves using separate tools that support creating traceability links to proxies in other tools. A proxy in this context is a model element that might or might not exist in another tool. To ensure that R1 is adequately addressed the tool must support creating and navigating such external links to proxies in a native manner.

A separate tool is responsible for resolving these proxies in the background, reporting links that have become broken, and providing candidate elements of a certain type to support/simplify link creation. OSLC[7] can be used as a standard for connecting the tools as services via REST APIs. Graph databases such as Neo4j[8] can be used to check all proxies and resolve traceability links. All modelling tools push/update periodically a part (basically the overlapping in Figure 2) of their models to the graph database, which can then attempt to resolve proxies by connecting these interface elements and reporting on success/failure. We have seen this strategy successfully implemented for modern, domain-specific tools, which could be implemented from scratch. Reusing current tools on the market, however, makes it difficult to address R1, i.e., to ensure productive traceability maintenance for end users in their favourite tool.

## 4. How can future bx research help?

In our experience, it is currently difficult to directly (re)use a bx tool in practice. This is mainly due to the effort involved in connecting the bx tool as required to the actual models and concrete tools, especially when a project poses constraints on the choice of software technology. Usually, bx tools that are developed in academia enforce a specific format for persisting models (e.g., Ecore-compliant XML Metadata Interchange (XMI) files), and do not allow for being used as an all-in-one tool (as other tools are much more suitable for modelling purposes) or as a backend for traceability maintenance (due to inappropriate interfaces). Using an existing bx tool for synchronising a common overlap involves substantial additional implementation efforts, as the bx must react to events that are triggered in one of the modelling tools.

It turns out, therefore, that bx foundations are what can be really transferred to industry and used to investigate new application scenarios as well as organise and communicate implemented solutions. As a consequence, bx could be further promoted by presenting established formal foundations in an accessible manner for practitioners in the application area of (model-based) systems engineering.

Concerning formal bx foundations, it would be helpful to clarify the relationship between bx, version management, and variant management. To the best of our knowledge, there exists no

[5] https://codebeamer.com/cb
[6] https://www.jamasoftware.com/solutions/requirements-management/
[7] https://open-services.net
[8] https://neo4j.com

unified formal treatment of these three concerns together, even though they often have be to handled in combination in practice. There are different ways of handling bx (state-based, delta-based, different laws), different ways to support versioning (branch-based with a diff+merge, locks, online vs. offline), and different approaches to variant management (composition-based vs. annotation-based) – having a formal framework with unifying definitions and laws covering all these concerns would help master the complexity and confusion in practice.

Concerning bx tooling, we suggest inspecting existing modelling tools that are currently being used in practice by a community of practitioners in a respective ecosystem. To increase the chance of actually using bx-related implementations, the bx community has to provide tool-specific connectors that can be directly used and configured as required.

Finally, we believe it is particularly unrealistic to make too many assumptions about the data stored in tools. Expecting the complete data to be exported to a certain format, manipulated by a bx, and then imported back to the modelling tool is typically unwieldy if not infeasible in practice. We suggest instead developing bx tooling that leaves all data in the respective modelling tools, communicating with the tools via their respective APIs as required.

## 5. Conclusion

In this paper, we shared an automotive perspective on traceability maintenance with a primary focus on the upper-left corner of the V-model.

We reported on the main requirements, challenges and constraints of this problem domain.

In our experience, the pivotal requirement is typically that end users want to create and make use of traceability links directly in their modelling tool of choice without too much of a distinction between these links and other "normal" links in the modelling tool.

This perhaps explains some of the solution strategies we discussed, especially the overly ambitious attempt to build an all-in-one tool despite repeated failure in the past.

A viable, pragmatic solution strategy employs bx to propagate parts of models in one tool to other tools, enabling traceability maintenance in the tools at the price of having to keep these parts synchronised.

Finally, we suggested future directions of bx research which would help promote a transfer of bx to industry, especially for model-based systems engineering.

## References


[1] Automotive SPICE® Process Reference and Assessment Model - RELEASE 3.1 - 01 November 2017, https://www.automotivespice.com/fileadmin/software-download/AutomotiveSPICE_PAM_31.pdf, 2017. Accessed: 2022-05-13.

[2] E. R. Batot, S. Gérard, J. Cabot, A survey-driven feature model for software traceability approaches, in: E. B. Johnsen, M. Wimmer (Eds.), Fundamental Approaches to Software Engineering - 25th International Conference, FASE 2022, Munich, Germany, April 2-7, 2022, Proceedings, volume 13241 of *LNCS*, Springer, 2022, pp. 23–48.

[3] A. Anjorin, E. Leblebici, A. Schürr, 20 years of triple graph grammars: A roadmap for future research, Electron. Commun. Eur. Assoc. Softw. Sci. Technol. 73 (2015).

[4] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, J. Terwilliger, Bidirectional Transformations: A Cross-Discipline Perspective, in: R. F. Paige (Ed.), ICMT 2009, volume 5563 of *LNCS*, Springer, 2009, pp. 260–283.

[5] J. Cheney, J. McKinna, P. Stevens, J. Gibbons, Towards a Repository of Bx Examples, in: K. S. Candan, S. Amer-Yahia, N. Schweikardt, V. Christophides, V. Leroy (Eds.), Proceedings of the Workshops of EDBT/ICDT 2014, volume 1133 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2014, pp. 87–91.

[6] F. Hermann, S. Gottmann, N. Nachtigall, H. Ehrig, B. Braatz, G. Morelli, A. Pierre, T. Engel, C. Ermel, Triple Graph Grammars in the Large for Translating Satellite Procedures, in: D. D. Ruscio, D. Varró (Eds.), ICMT 2014, volume 8568 of *LNCS*, Springer, 2014, pp. 122–137.

[7] D. Blouin, A. Plantec, P. Dissaux, F. Singhoff, J. Diguet, Synchronization of Models of Rich Languages with Triple Graph Grammars: An Experience Report, in: D. D. Ruscio, D. Varró (Eds.), ICMT 2014, volume 8568 of *LNCS*, Springer, 2014, pp. 106–121.

[8] H. Giese, S. Hildebrandt, S. Neumann, Model Synchronization at Work : Keeping SysML and AUTOSAR Models Consistent, Festschrift Nagl 5765 (2010) 555–579.

[9] J. Greenyer, J. Rieke, Applying Advanced TGG Concepts for a Complex Transformation of Sequence Diagram Specifications to Timed Game Automata, in: A. Schürr, D. Varró, G. Varró (Eds.), AGTIVE 2011, volume 7233 of *LNCS*, Springer, 2012, pp. 222–237.

[10] N. Weidmann, S. Salunkhe, A. Anjorin, E. Yigitbas, G. Engels, Automating Model Transformations for Railway Systems Engineering, J. Object Technol. 20 (2021) 10:1–14.

[11] S. Maro, A. Anjorin, R. Wohlrab, J. Steghöfer, Traceability Maintenance: Factors and Guidelines, in: D. Lo, S. Apel, S. Khurshid (Eds.), ASE 2016, ACM, 2016, pp. 414–425.

[12] M. Johnson, R. D. Rosebrugh, Unifying Set-Based, Delta-Based and Edit-Based Lenses, in: A. Anjorin, J. Gibbons (Eds.), Bx 2016, volume 1571 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2016, pp. 1–13.

# Engineering Bidirectional Model Transformations (Short Paper)

Thomas Buchmann[1], Bernhard Westfechtel[1]

[1]Applied Computer Science I, University of Bayreuth, D-95440 Bayreuth, Germany

**Abstract**

Bidirectional transformations have been studied in a wide range of application domains. In model-driven software engineering, they are required for roundtrip engineering processes. We present a pragmatic approach to engineering bidirectional model transformations that assists transformation developers by domain-specific languages, frameworks, and code generators and provides for conciseness, expressiveness, and scalability. We also discuss different variants of transformation development processes as well as their advantages and drawbacks.

**Keywords**

Model-driven software engineering, roundtrip engineering, bidirectional transformation

## 1. Background

*Bidirectional transformations (bx)* occur in different application domains, including e.g. databases, programming languages, and software engineering [1]. Programming bidirectional transformations in a conventional programming language is both laborious and error-prone: Both transformation directions have to be programmed separately, and consistency of forward and backward transformations has to be checked by testing.

In response to these problems, a wide variety of bx approaches have been developed in research [2]. In *functional approaches*, a bidirectional transformation is defined by a function operating in one direction; the opposite direction is derived automatically [3, 4]. In *relational approaches*, a bidirectional transformation is defined by a set of relations between source and target elements [5, 6]. In *grammar-based approaches*, a set of grammar rules defines consistent pairs of source and target models [7, 8].

A recurring theme driving bx research are *roundtrip properties*, also referred to as *bx laws* [9]. Roundtrip properties are constraints on the interplay of forward and backward transformations. The goal of many bx approaches consists in the construction of bidirectional transformations that are *provably correct* with respect to roundtrip properties.

However, our *empirical evaluations* [8, 10, 11] demonstrate limitations of bx approaches with respect to *expressiveness*, i.e., the capability to solve a given bx problem. These limitations follow from the conditions that transformations have to satisfy in order to guarantee roundtrip properties. Additional shortcomings were observed with respect to *conciseness* — the ability

*Tenth International Workshop on Bidirectional Transformations (BX 2022), July 08, 2022, Nantes, France*

Thomas.Buchmann@uni-bayreuth.de (T. Buchmann); Bernhard.Westfechtel@uni-bayreuth.de (B. Westfechtel)



CEUR Workshop Proceedings (CEUR-WS.org)

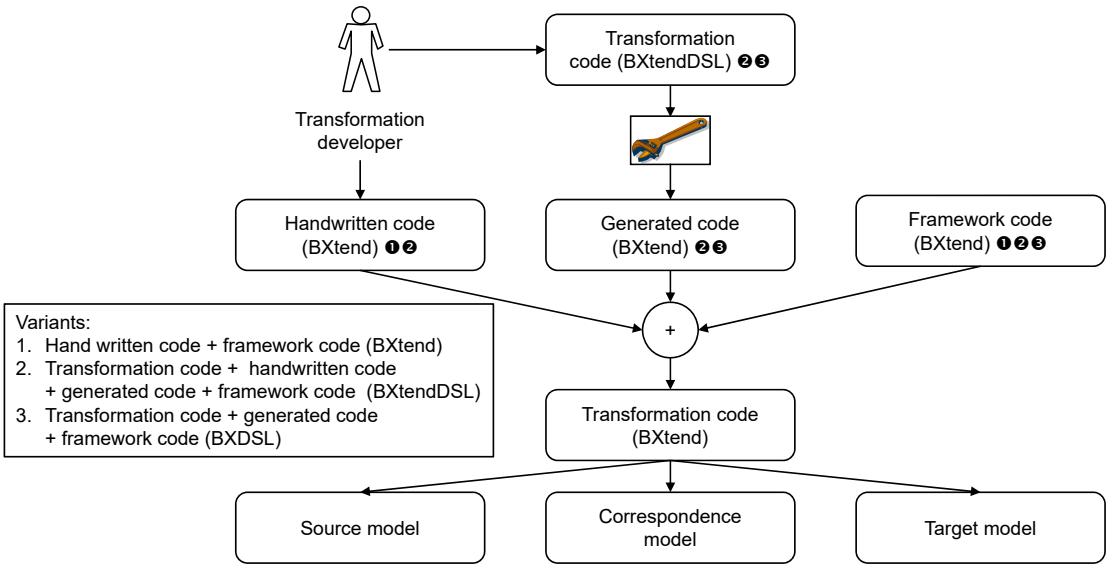


**Figure 1:** Layered approach to bidirectional transformations

to provide for short solutions with respect to size metrics — and *scalability* — the ability to perform transformations efficiently on large data sets.

## 2. Contribution

Based on our experience gained from empirical evaluations, we have been developing *frameworks* for *engineering bidirectional model transformations* [12]. Our goal is to support transformation developers in the specification and implementation of bidirectional transformations in the context of *model-driven software engineering* [13]. Thus, the artifacts that are subject to transformations are *models* of software systems; the Eclipse Modeling Framework [14] serves as the underlying technological ecosystem. In particular, we focus on *roundtrip engineering* (e.g., between model and code), where the source model and the target model are considered as peers and updates may be propagated in both directions. While roundtrip properties are guaranteed to a certain extent, we have addressed primarily conciseness, expressiveness, and scalability.

In our work, bidirectional transformations share the following features: (1) Bidirectional transformations are *symmetric* inasmuch as source and target model are considered as peers (rather than asymmetric transformations, working on sources and views). (2) Each transformation execution is *directed*, i.e., it reads the source model and updates the target model (or vice versa). (3) Each transformation is *executed on demand* only (no live propagation of changes). (4) Transformations are *correspondence-based*, a correspondence model is stored persistently to allow for precise change propagations. (5) Finally, transformations are *state-based*, i.e., they rely on model states only rather than on operational deltas.

As illustrated in Figure 1, we provide a *layered approach* to developing bidirectional model transformations. This approach, which is called *BXTendDSL*, combines declarative with im-

perative programming and is labeled as number ❷ in Figure 1. Before, we briefly present its precursor *BXtend* [15], which entirely relies on *imperative programming* (number ❶). Finally, we discuss a potential *purely declarative approach* (*BXDSL*), which is subject to current and future work (number ❸).

## 2.1. BXtend

The acronym BXtend is composed from BX (bidirectional transformations) and Xtend, an object-oriented programming language that is based on Java [16]. The BXtend framework [15] provides an *internal domain-specific language (DSL)* for bidirectional transformations. An internal DSL offers an application programming interface in a host language; it is easier to implement than an external DSL and does not require the transformation developer to learn a new language. Xtend was selected as a host language because it offers high-level support for object-oriented, procedural, and functional programming.

Programming bidirectional transformations from scratch is laborious. BXtend reduces this effort considerably by providing a generic framework on top of which specific code for implementing transformation rules has to be written. The framework includes an implementation of a correspondence model as well as algorithms for incremental model transformations in both directions. The evaluations which we conducted so far confirm conciseness, expressiveness, and scalability. In the Families to Persons benchmark published in [8], the BXtend solution is the only one that passes all test cases. Furthermore, BXtend proves to be scalable. Even conciseness is reasonable although both transformation directions have to be programmed manually.

## 2.2. BXtendDSL

The BXtend framework shows two shortcomings: First, a transformation definition written in the BXtend internal DSL contains redundant code, i.e., similar code fragments in forward and backward direction. Second, roundtrip properties may be ensured by testing only. In response to these problems, we developed BXtendDSL [12, 17] that adds a declarative layer on top of the imperative layer. At the declarative layer, the transformation developer specifies the transformation in a purely declarative language (reflected in the suffix of the acronym); here, we decided to provide an *external DSL* with a short and intuitive syntax. After code generation, the transformation developer employs the internal DSL to complete the transformation definition.

The external DSL is a small and light-weight *relational language* that is based on rules describing relations between corresponding source and target patterns. Intentionally, the DSL is *computationally incomplete*: In all of the transformation cases that we have studied so far, the declarative code needs to be supplemented with imperative code that offers the required flexibility to solve the transformation case at hand. In this way, the external DSL can be kept small, avoiding the reimplementation of functionality that is available in the BXtend internal DSL anyway. Accordingly, *code generation* is *partial*, as e.g. in the EMF code generator [14], which generates incomplete code from a structural metamodel.

In the external DSL, a transformation is defined by a sequence of *rules* that may *depend* on each other. The DSL allows to declare $m : n$ *dependencies* between *source and target objects*, as well as $m : n$ dependencies at the level of *structural features* (attributes and references).

Furthermore, the transformation developer may define *extension points* that have to be filled with imperative code if the mapping cannot be handled by generated code alone.

Furthermore, the external DSL guarantees *roundtrip properties* under certain restrictions. Roughly following [18], *correctness* means that a transformation restores consistency among the participating models, and *hippocraticness* implies that models that are already mutually consistent are not updated. Correctness and hippocraticness are guaranteed under *well-behavedness conditions* that are defined by means of OCL [19] constraints on transformation definitions in the declarative DSL.

Our benchmark evaluations [12] reconfirm conciseness, expressiveness, and scalability. Compared to BXtend, both expressiveness and scalability are not affected adversely. Furthermore, solutions in the layered framework (comprising the manual code written on both layers) are considerably more concise than the corresponding BXtend solutions (and solutions in other tools/languages).

### 2.3. BXDSL

BXtendDSL offers two main advantages over BXtend: conciseness and guarantee of roundtrip properties (under well-behavedness conditions). However, it exhibits two shortcomings: First, the transformation developer has to switch between different levels of abstraction. Second, the well-behavedness conditions are restrictive; each of the transformation cases we have studied so far violates at least one of these conditions, implying the need for imperative programming. Thus, the question arises whether it is possible to design a declarative language (with the fictitious name *BXDSL*) that is computationally complete (and thus does not require imperative code as a supplement) and relaxes the well-behavedness conditions such that a wider range of transformations can be specified that are provably correct.

So far, we have not achieved these goals. Currently, we are working on an extension of BXtendDSL so that more work can get done at the declarative layer [20]. However, the new language version is not powerful enough to solve bx transformations completely, and still requires complementary imperative code, thus retaining the layered approach described above. Based on the experiences we have gained so far in bidirectional transformations, we consider it unlikely that a single bx language may be designed that guarantees round-trip properties without restrictions on the use of the language. Furthermore, the fictitious language would have to include unidirectional language constructs, as they are already present in BXtendDSL and were proposed e.g. in [21] as extensions to the relational bx language QVT-R [5].

## 3. Conclusion

In this paper, we summarized our research in the bx domain. Our approach aims at engineering bidirectional model transformations by offering domain-specific languages, frameworks, and code generators. In this way, we intend to reduce the amount of work that transformation developers have to invest. Our main focus lies on the quality attributes conciseness, expressiveness, and scalability. We also discussed different variants of transformation development processes. So far, we consider the layered approach of BXtendDSL the best choice in terms of concisensess, expressiveness, and scalability. Altogether, our research complements other bx

research that primarily focuses on roundtrip properties and provably correct transformations. These issues have been addressed to a limited extent in BXtendDSL, as well, but have not been the major driving force of our research, which rather follows the modest goal of making the life of bx transformation developers easier.

## References


[1] K. Czarnecki, J. N. Foster, Z. Hu, R. Lämmel, A. Schürr, J. F. Terwilliger, Bidirectional transformations: A cross-discipline perspective, in: R. F. Paige (Ed.), Proceedings of the Second International Conference on Theory and Practice of Model Transformations (ICMT 2009), volume 5563 of *Lecture Notes in Computer Science*, Springer-Verlag, Zurich, Switzerland, 2009, pp. 260–283.

[2] S. Hidaka, M. Tisi, J. Cabot, Z. Hu, Feature-based classification of bidirectional transformation approaches, Software and Systems Modeling 15 (2016) 907–928.

[3] J. N. Foster, M. B. Greenwald, J. T. Moore, B. C. Pierce, A. Schmitt, Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem, ACM Transactions on Programming Languages and Systems 29 (2007) 17:1–17:65.

[4] H. Ko, T. Zan, Z. Hu, BiGUL: a formally verified core language for putback-based bidirectional programming, in: M. Erwig, T. Rompf (Eds.), Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 - 22, 2016, ACM, 2016, pp. 61–72. URL: https://doi.org/10.1145/2847538.2847544. doi:10.1145/2847538.2847544.

[5] OMG, Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification, formal/2015-02-01 ed., Needham, MA, 2015.

[6] A. Cicchetti, D. Di Ruscio, R. Eramo, A. Pierantonio, JTL: A bidirectional and change propagating transformation language, in: B. Malloy, S. Staab, M. van den Brand (Eds.), Proceedings of the Third International Conference on Software Language Engineering (SLE 2010), volume 6563 of *Lecture Notes in Computer Science*, Springer-Verlag, Eindhoven, The Netherlands, 2010, pp. 183–202.

[7] A. Schürr, Specification of Graph Translators with Triple Graph Grammars, in: G. Tinhofer (Ed.), Proceedings of the 20th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 1994), volume 903 of *LNCS*, Springer-Verlag, Herrsching, Germany, 1994, pp. 151–163.

[8] A. Anjorin, T. Buchmann, B. Westfechtel, Z. Diskin, H. Ko, R. Eramo, G. Hinkel, L. Samimi-Dehkordi, A. Zündorf, Benchmarking bidirectional transformations: theory, implementation, application, and assessment, Software and Systems Modeling 19 (2020) 647–691. URL: https://doi.org/10.1007/s10270-019-00752-x. doi:10.1007/s10270-019-00752-x.

[9] F. Abou-Saleh, J. Cheney, J. Gibbons, J. McKinna, P. Stevens, Introduction to bidirectional transformations, in: J. Gibbons, P. Stevens (Eds.), Bidirectional Transformations - International Summer School, Oxford, UK, July 25-29, 2016, Tutorial Lectures, volume 9715 of *Lecture Notes in Computer Science*, Springer, 2016, pp. 1–28. URL: https://doi.org/10.1007/978-3-319-79108-1_1. doi:10.1007/978-3-319-79108-1_1.

[10] T. Buchmann, B. Westfechtel, Using triple graph grammars to realize incremental round-

trip engineering, IET Software 10 (2016) 173–181. URL: http://digital-library.theiet.org/content/journals/10.1049/iet-sen.2015.0125.

[11] B. Westfechtel, Case-based exploration of bidirectional transformations in QVT relations, Software and Systems Modeling 17 (2018) 989–1029. doi:10.1007/s10270-016-0527-z.

[12] T. Buchmann, M. Bank, B. Westfechtel, BXtendDSL: A layered framework for bidirectional model transformations combining a declarative and an imperative language, J. Syst. Softw. 189 (2022) 111288. URL: https://doi.org/10.1016/j.jss.2022.111288. doi:10.1016/j.jss.2022.111288.

[13] M. Völter, T. Stahl, J. Bettin, A. Haase, S. Helsen, Model-Driven Software Development: Technology, Engineering, Management, John Wiley & Sons, Chichester, UK, 2006.

[14] D. Steinberg, F. Budinsky, M. Paternostro, E. Merks, EMF Eclipse Modeling Framework, The Eclipse Series, 2nd ed., Addison-Wesley, Boston, MA, 2009.

[15] T. Buchmann, Bxtend - A framework for (bidirectional) incremental model transformations, in: S. Hammoudi, L. F. Pires, B. Selic (Eds.), Proceedings of the 6th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2018, Funchal, Madeira - Portugal, January 22-24, 2018., SciTePress, 2018, pp. 336–345. URL: https://doi.org/10.5220/0006563503360345. doi:10.5220/0006563503360345.

[16] L. Bettini, Implementing Domain-Specific Languages with Xtext and Xtend, Packt Publishing, Birmingham, UK, 2016.

[17] M. Bank, T. Buchmann, B. Westfechtel, Combining a declarative language and an imperative language for bidirectional incremental model transformations, in: S. Hammoudi, L. F. Pires, E. Seidewitz, R. Soley (Eds.), Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development, MODELSWARD 2021, Online Streaming, February 8-10, 2021, SCITEPRESS, 2021, pp. 15–27. URL: https://doi.org/10.5220/0010188200150027. doi:10.5220/0010188200150027.

[18] P. Stevens, Bidirectional model transformations in QVT: Semantic issues and open questions, Software and Systems Modeling 9 (2010) 7–20.

[19] OMG, Object Constraint Language, formal/2014-02-03 ed., OMG, Needham, MA, 2014.

[20] O. Hacker, BXtendDSL 2: Weiterentwicklung einer hybriden Sprache für bidirektionale Modell-zu-Modell Transformationen, 2022. Master thesis (in German), University of Bayreuth, Germany.

[21] B. Westfechtel, A case study for evaluating bidirectional transformations in QVT Relations, in: J. Filipe, L. Maciaszek (Eds.), Proceedings of the 10th International Conference on the Evaluation of Novel Approaches to Software Engineering (ENASE 2015), SCITEPRESS, Barcelona, Spain, 2015, pp. 141–155.

# Workshop on Foundations and Practice of Visual Modeling (FPVM)

Amleto Di Salle[1], Ludovico Iovino[2], Alfonso Pierantonio[1] and Juha–Pekka Tolvanen[3]

[1]*Gran Sasso Science Institute, L'Aquila, Italy*
[1]*University of L'Aquila, L'Aquila, Italy*
[3]*Metacase, Finland*

The sheer complexity of software systems nowadays makes modeling artifacts pervasive throughout the development process, be it use requirements, analysis, design, or development. Whether models are used for communication or prescriptive purposes, their syntax and pragmatics affect usability and represent contributory factors concerning the accidental complexity. The diversity of modeling notations and approaches permits classifying them according to different taxonomies. General-purpose and domain-specific modeling languages can be created with different intended scopes, although all of them can make use of graphical, textual, maps, matrices, tables, and combinations regarding its concrete syntax. These representations have the undoubted advantage of capturing and increasing understanding of complex software systems and better grasping the rationale behind them. In essence, a visual modeling language creates a joint base for the modeler by improving their communication and lays a solid foundation for the implementation.

**FPVM 2022** aims to promote and foster discussions on many aspects of visual modeling languages, including novel and visionary ideas and techniques, notations for the generations of support tools for visual languages, the usability of tools and meta-tools.

The second edition of FPVM was held at Nantes (France) on July 05, 2022, and co-located with the Software Technologies: Applications and Foundations (STAF).

FPVM has received four submissions. After a thorough peer-review process involving three members from the program committee per each submission, three submissions have been accepted for publication (acceptance rate was 75%).

We would like to thank the FPVM program committee for making the workshop possible. Additionally, we would like to thank the STAF workshop chairs, Catherine Dubois and Julien Cohen, for their help and support and STAF for hosting the workshop.

## FPVM Organization

### Workshop Chairs

*STAF '22: Workshop Foundations and Practice of Visual Modeling (FPVM), 05–08 July, 2022, Nantes, FR*
amleto.disalle@univaq.it (A. Di Salle); ludovico.iovino@gssi.it (L. Iovino); alfonso.pierantonio@univaq.it (A. Pierantonio); jpt@metacase.com (J. Tolvanen)
0000-0002-0163-9784 (A. Di Salle); 0000-0001-6552-2609 (L. Iovino); 0000-0002-5231-3952 (A. Pierantonio)

| | |
|---|---|
| Amleto Di Salle | University of L'Aquila, Italy |
| Ludovico Iovino | Gran Sasso Science Institute, Italy |
| Alfonso Pierantonio | University of L'Aquila, Italy |
| Juha–Pekka Tolvanen | Metacase, Finland |

**Program Committee**

| | |
|---|---|
| Luca Berardinelli | Johannes Kepler University Linz, Austria |
| Paolo Bottoni | Sapienza University of Roma, Italy |
| Alessio Bucaioni | Mälardalen University, Sweden |
| Federico Ciccozzi | Mälardalen University, Sweden |
| Benoit Combemale | University of Rennes 1, France |
| Juan De Lara | Autonomous University of Madrid, Spain |
| Davide Di Ruscio | University of L'Aquila, Italy |
| Steven Kelly | Metacase, Finland |
| Dimitris Kovolos | University of York, United Kindgom |
| Sebastien Mosser | Université du Québec à Montréal, Canada |
| Nikolaus Regnat | Siemens, Germany |
| Maria Teresa Rossi | Gran Sasso Science Institute, Italy |
| Matthias Tichy | University of Ulm, Germany |
| Hans Vangheluwe | University of Antwerp, Belgium and McGill University, Canada |
| Manuel Wimmer | Johannes Kepler University Linz, Austria |

# Optimistic Versioning for Conflict-tolerant Collaborative Blended Modeling

Joeri Exelmans[1], Jakob Pietron[2], Alexander Raschke[2], Hans Vangheluwe[1] and Matthias Tichy[2]

[1]Department of Computer Science, University of Antwerp – Flanders Make, Antwerp, Belgium
[2]Institute of Software Engineering and Programming Languages, Ulm University, Ulm, Germany

**Abstract**

Optimistic versioning is a key component in supporting collaborative workflows. Text-based versioning has been widely adopted for versioning code, but in model-driven engineering, dealing with visual concrete syntaxes, new methods are required. In the case of blended modeling, a mixture of both textual and visual syntaxes, concurrently editable and synchronizable, introduces additional challenges.

We propose a type of operation-based versioning to record not only user edits, but also bi-directional change propagations between concrete and abstract syntax. This way we can support blended modeling with layout continuity, and flexible handling of missing information (e. g., layout information) when rendering changes from abstract to concrete syntax. In addition, the proposed versioning approach enables collaborative conflict resolution by allowing partial conflict resolution, thus deferring a final resolution.

**Keywords**

versioning, blended modeling, conflict-tolerant, operation-based

## 1. Introduction

Model-driven engineering (MDE) has become widely accepted as prime enabler for the creation of increasingly complex software-intensive systems. In addition to graphical models, various model representations such as tabular or textual ones are typically used. The flexible use of different representations (concrete syntax (CS)) for one and the same model (abstract syntax (AS)) is also called *blended modeling*. The ability to switch between different representations allows the user to choose the one that is most useful and efficient for the current task [1].

Complex systems are developed in teams in both asynchronous and synchronous collaborative environments [2]. Synchronous collaboration has gained importance since the Corona pandemic, where developers who normally develop models in the same room, e. g. on whiteboards, were forced to use (online) tools for collaboration [3].

In addition to the complexity of concurrency in (a)synchronous collaboration (i.e. branching, merging, and dealing with conflicts), an orthogonal problem of blended modeling is the challenge of (multi-)CS and AS synchronization. For instance, there may be concurrency on the same

*FPVM 2022: 2nd International Workshop on Foundations and Practice of Visual Modeling, July 4–8, 2022, Nantes, France*
joeri.exelmans@uantwerpen.be (J. Exelmans); jakob.pietron@uni-ulm.de (J. Pietron); alexander.raschke@uni-ulm.de (A. Raschke); hans.vangheluwe@uantwerpen.be (H. Vangheluwe); matthias.tichy@uni-ulm.de (M. Tichy)
0000-0002-6916-5140 (J. Exelmans); 0000-0001-8308-6636 (J. Pietron); 0000-0002-6088-8393 (A. Raschke); 0000-0003-2079-6643 (H. Vangheluwe); 0000-0002-9067-3748 (M. Tichy)

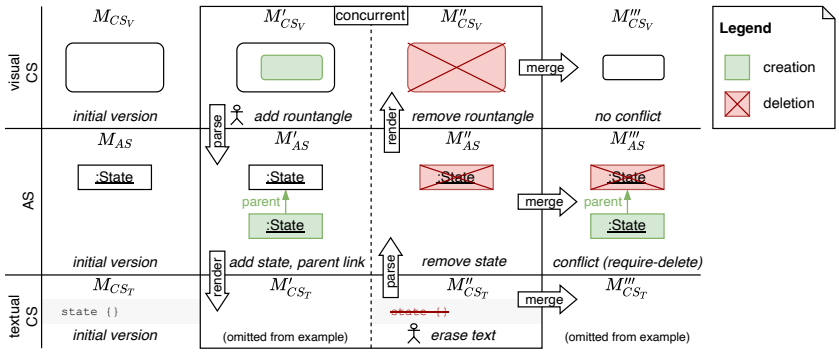


**Figure 1:** Running example: Blended modeling scenario with concurrent user edits

or on different CSs that are being synchronized (see Figure 1 and Section 2). Another issue of blended modeling is that of missing information when switching between CS representations. For example, if an element is added in textual syntax, the position of the new corresponding element in a graphical syntax remains unknown.

Versioning systems for "code" (e.g. git [4], SVN [5]) are only suited for recording textual CS. Model versioning systems attempt to overcome this limitation by recording, comparing and merging instead at the level of the AS. Only a single (visual) CS is assumed to exist, with a 1:1 mapping between CS and AS elements. This hinders support for blended modeling.

To overcome this limitation, we propose to combine operation-based versioning with incremental bi-directional change propagation to enable (a) arbitrary mappings between CS and AS, with traceability between CS and AS, (b) blended modeling with layout continuity and flexible handling of missing (layout) information, and (c) reuse of CS editing environments for different languages. Additionally, as part of our operation-based versioning approach, we propose a new way to persist merge conflicts, to allow recording of the steps taken in conflict resolution.

Our approach should not be confused with projectional editing [6], where CS operations directly impact the AS without using a parser. On the contrary, we allow arbitrary CS/AS mappings, increasing the flexibility for the modeler and ultimately the usability of the modeling environment [7, 8].

The remainder of the paper is structured as follows. In Section 2, we introduce a running example that is used to illustrate our proposed solution presented in Section 3. Section 4 discusses related work and Section 5 concludes with an outlook on future work.

## 2. Running Example

To illustrate our approach, we introduce a (visual) CS and AS for a very limited subset of the Statecharts formalism. On the CS side, we have 2D drawings of rountangles (rounded rectangles) with geometries. On the AS side, we can have State objects, between which a "parent" association exists (every State can have 1 parent). The correspondence relation between CS and AS is as follows: There exists a one-to-one mapping between rountangles and State objects, and whenever a rountangle is geometrically inside another, a parent link between their

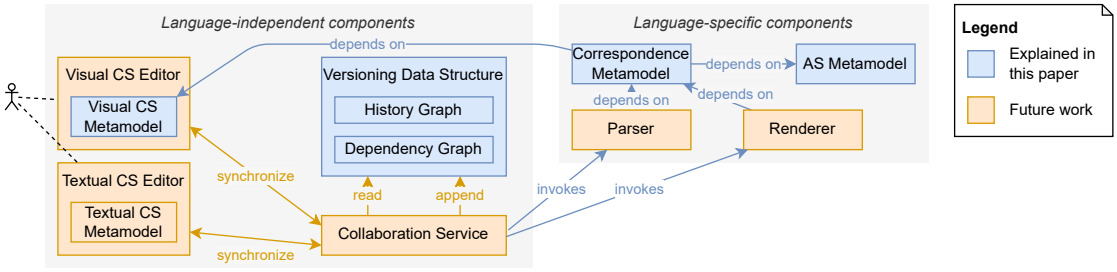


**Figure 2:** "Big picture": A possible architecture

corresponding State objects must exist. Additionally, we assume that there is a textual CS, that we do not describe here as it does not add additional insight.

Figure 1 shows evolving CS and AS models. Initially, we have a version $M_{CS_V}$ of a visual CS model with a single rountangle, $M_{AS}$ of the corresponding AS model with a single State, and $M_{CS_T}$ the corresponding textual CS model. Then, concurrently, a change happens to the visual and textual CS models. In the visual model, an inner rountangle is added (producing $M'_{CS_V}$) and, concurrently, in the textual model, some text is erased (producing $M''_{CS_T}$). Both changes could be propagated (parsed) to the AS, and subsequently rendered to the other CS. How do we represent these changes, and how do we merge them? We intuitively understand that there will be a merge conflict, at least at the level of the AS: a State object is being deleted, while at the same time, it is the target of a newly created parent link.

There may be many meaningful ways to resolve such a conflict, but this is not our focus here. In this paper the main focus is on recording (concurrent) changes and (concurrent) change propagations, on and between CS and AS.

Note that in the remainder of this paper, we assume that text removal in $M''_{CS_T}$ has already been parsed to produce $M''_{AS}$. This way, we can focus on the visual CS and the AS, which are sufficient to explain bi-directional change propagation (the main building block of blended modeling), combined with concurrency.

## 3. Solution

We introduce a set of components that can become part of a collaboration architecture. Their role in a possible architecture is shown in Figure 2.

We explicitly distinguish between language-specific and language-independent components. In our approach, CS and AS are separate, evolving models, each conforming to their own metamodel, as proposed by Van Tendeloo [9]. AS metamodels will always be language-specific, e.g. specific to Statecharts. CS metamodels (and their editors) will often be language-independent. For instance, a metamodel for vector graphics drawings may serve as a CS meta model for both Statecharts and Petri Nets. Users only interact directly with a model through a CS. Obviously, the definition of a mapping between CS and AS will be language-specific. In our approach, this mapping consists of a *correspondence metamodel*, and a *parser* and *renderer* function. We will explain these concepts, and specify an interface for parser and renderer functions.

The language-independent component *collaboration service* is left underspecified. Among its

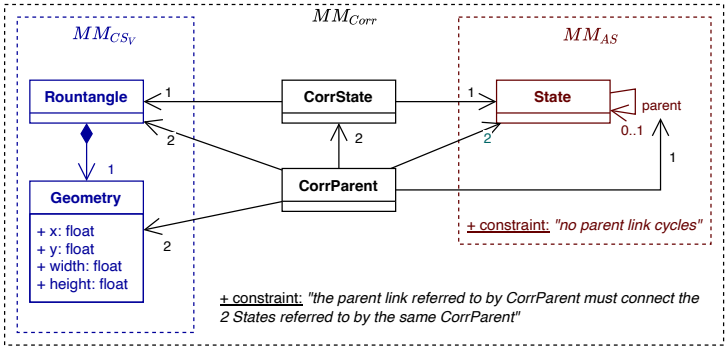


**Figure 3:** Running example: Concrete syntax, abstract syntax and correspondence metamodels

many tasks is synchronization of CS editors with evolving CS models, (networked) synchronization between collaborators, and synchronization between CS and AS models. For the latter, it invokes the parser and renderer functions. This service uses the *versioning data structure* to record the edit history of models. We will explain this component, and why it is especially well suited for blended modeling.

The components presented in this paper are deployment-independent. For instance, the versioning data structure could be deployed centrally or decentrally.

## 3.1. Incremental parsing and rendering

A change to a CS may cause corresponding changes to the AS. Change propagation from CS to AS is called *parsing*. Subsequently, this change to the AS may cause changes to other CSs. Change propagation from AS to CS is called *rendering*[1].

Parsing and rendering must happen incrementally for several reasons: The first reason is performance (not having to parse/render from scratch after every change). The second reason is *layout continuity* when rendering a visual update (not regenerating a new layout from scratch, which would be confusing to users familiar with an existing layout). The third reason is that in a multi-user collaboration scenario, propagating concurrent changes must cause new concurrent changes (instead of new concurrent models), that can be merged accurately and efficiently.

## 3.2. Correspondence model

The user only interacts with a model through a CS, so any information (e. g., inconsistencies) from the AS or semantic level must also be visualized in the CS. CS and AS can relate to each other in nontrivial ways, and inferring this information a posteriori would be complex, and nondeterministic (more than one solution). We therefore persist traceability information between CS and AS elements, and keep it up-to-date, every time changes are propagated.

We persist traceability information between one CS and one AS model in a *correspondence model*, an idea taken from triple graph grammars (TGGs) [10]. For any pair of CS and AS metamodels that can be synchronized, a correspondence metamodel must be defined. This

[1]Van Tendeloo uses the terms “comprehension” and “perceptualization” instead of “parsing” and “rendering”, resp. We find “parsing” and “rendering” more intuitive.

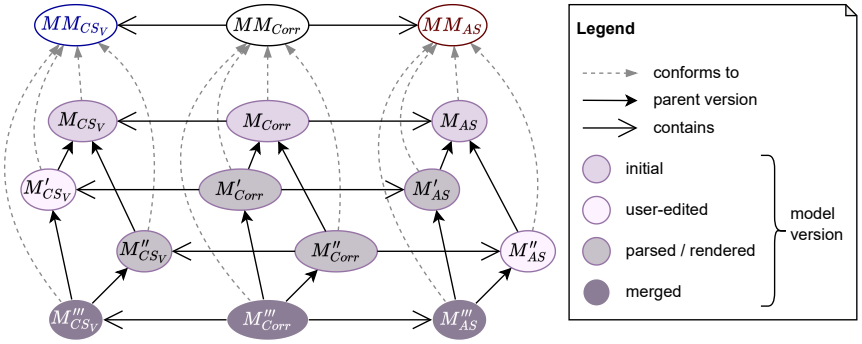


**Figure 4:** Running example: Edit history of CS, AS and Correspondence models

metamodel contains all types of the CS and AS metamodels that it relates to, and also the different correspondence object types, that associate CS to AS types. At the instance level, a correspondence model includes one CS and one AS model (it acts as an "overlay" on a $(CS, AS)$ model pair), and a set of correspondence objects between their elements.

**Running example:** Figure 3 shows the metamodels (MM) for "the AS of the CS" ($MM_{CS_V}$, in blue), the AS ($MM_{AS}$, in red) and the correspondence ($MM_{Corr}$). The correspondence MM includes the elements of the CS and AS MMs, and defines two correspondence object types (*CorrState* and *CorrParent*). *CorrState* relates one CS rountangle to one AS State, and *CorrParent* relates two CS rountangles with geometrical enclosure to a parent link between their corresponding AS States.

Note that although we adopt some ideas from TGGs, we do not require change propagation rules to be specified as TGGs - our approach is neutral with respect to rule specification, and rules may even be specified imperatively.

### 3.3. Persistence of (propagated) changes

Instances of CS, AS, and correspondence models evolve. We rely on an underlying versioning system (explained in Section 3.4) to persist every change in each of these models (operation-based versioning [11]). A change may be the effect of an edit operation, or a propagated change (ultimately traceable to an edit operation). Changes can be replayed, so in essence, after every change, a permanent and immutable version is created.

**Running example**: Figure 4 shows the three metamodels from Figure 3 at the top. Below, we see a history graph of the model versions from Figure 1, and their conformance to the meta-level. We saw that $M'_{CS_V}$ and $M''_{AS}$ were produced by concurrent user edits. In our history graph, we simply record these model versions, and the relation to their parent version ($M_{CS_V}$ and $M_{AS}$, respectively). Parsing and rendering only produces new model versions. Parsing $M'_{CS_V}$ produces $M'_{AS}$ and $M'_{Corr}$, and rendering $M''_{AS}$ produces $M''_{CS_V}$ and $M''_{Corr}$. All pairs $(M'_x, M''_x)$ are concurrent, and when merged, produce model $M'''_x$ (with $x \in \{CS_V, AS, Corr\}$).

We will now explain our versioning approach in more detail. We will see what precisely makes up a model version, how we persist changes, how we relate changes to CS and AS to each other, and how we detect conflicts.

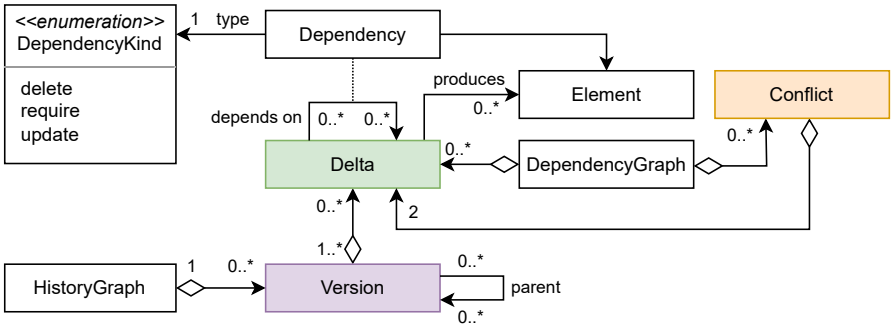


**Figure 5:** Metamodel of the versioning data structure

## 3.4. Versioning Data Structure

In this section, we present a new and application-agnostic versioning approach. Our approach can be classified as *operation-based* according to Brosch et al. [11], meaning that we persist changes between versions, instead of persisting snapshots of versions. Snapshots can be reconstructed by *replaying* changes. By persisting changes, we do not have to perform *diffing*, a complex and error-prone process. Persisting changes also enables persisting traceability information between propagated changes and user edits, which is crucial in order to support incremental parsing and rendering.

Our versioning approach consists of two data structures, called *History Graph* and *Delta Graph*. We will first explain the *Delta Graph*, which consists of deltas and dependency links between them.

**Deltas.** A delta (colored in green) records a change to a model. A delta may be caused by a user edit, or a propagated change (model synchronization). We only record the *effect* of the change (i. e., atomic CRUD operations on model elements), as opposed to the *cause* (e.g. the edit command). Rcording the cause as well may have advantages for edit history comprehensibility and analysis [12, 13], but this is not our focus. Deltas are transactional, meaning that they are either fully applied to the model, or not at all.

**Dependencies.** Deltas can depend on the effect of other deltas. We persist these *dependencies* between deltas in a directed, acyclic and append-only *Dependency Graph* in order to efficiently detect conflicts between deltas, as we will discuss later. Dependencies can be of three different types: *update*, *require*, and *delete*.

Figure 6 shows the *Dependency Graphs* for CS, AS, and correspondence of our running example. Delta $cs_1$ results in the creation of the outer rountangle. Further, there is delta $cs_2$, that creates the inner rountangle. The two deltas do not depend on each other and can, therefore, be applied to the model in any order, producing $M'_{CS_V}$. However, delta $cs_3$ deletes the outer rountangle, and in consequence, $cs_3$ has a *delete* dependency on $cs_1$. If there were a delta $cs_x$ (not part of our running example - only for illustrative purposes) that resizes the outer rountangle, it would have an *update* dependency on $cs_1$. The third type, *require*, occurs when a delta requires the existence of an element created by an other delta without updating the required element itself, e.g., when connecting an edge to an element, as in $as_2$.

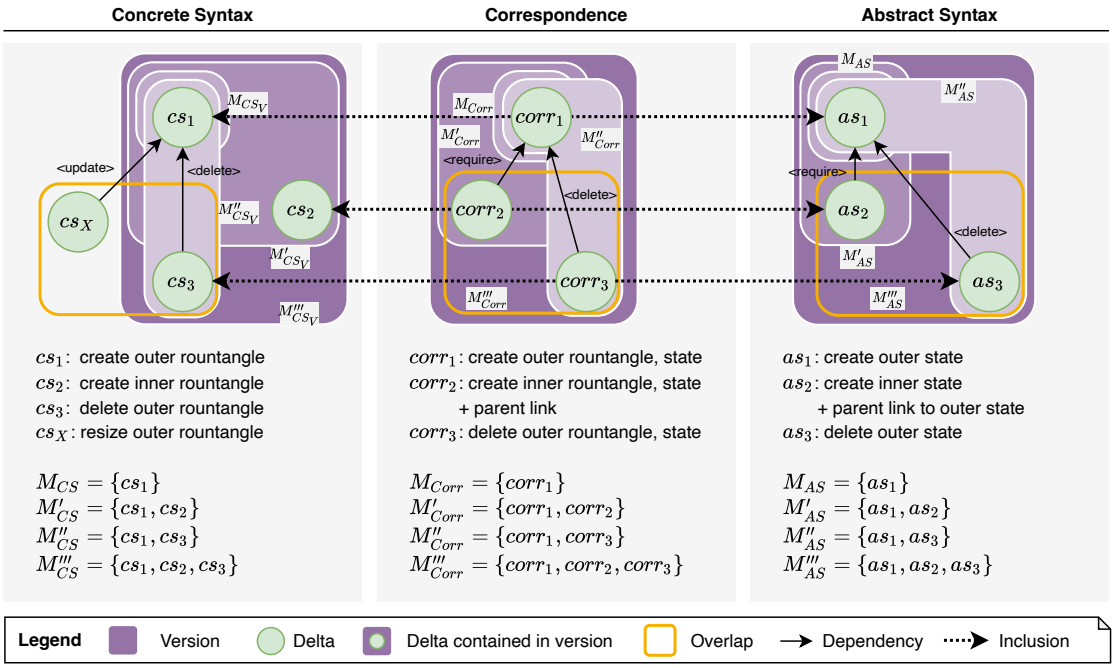


**Figure 6:** Running example: The three dependency graphs of $CS$, $Corr$, and $AS$ consisting of deltas. Their effect on the model is listed below. Additionally, the purple layers represent the different versions.

**Conflicts.** Two deltas that share a dependency on the same model element are *conflicting* if at least one of the deltas alters that model element. *Conflict* types are *update-update*, *update-delete*, and *require-delete* [11].

Continuing our running example, deltas $cs_1$ and $cs_2$ are independent and not conflicting, because they do not depend on a common delta. In consequence, they can both be part of the model state without any *Conflict* as it is the case in $M'_{CS_V}$. However, $cs_x$ (update geometry of outer rountangle) and $cs_3$ (delete outer rountangle) depend concurrently on the same model element $cs_1$ and are, consequently, in an *update-delete* conflict as indicated in orange color in Figure 6 on the left.

While the *Dependency Graph* imposes a partial order on the execution of deltas to obtain a valid state, it does not contain any information about the actual order in which deltas were added (or omitted, explained later) by different collaborators, altering the model state. Moreover, we want to be able to point to specific model *versions*. We therefore introduce the *History Graph*, which contains versions and their (partial) order.

**Versions.** A version (colored in various shades of purple), is a (possibly empty) set of deltas, that when replayed, produces a model state. In a version, all dependencies of all contained deltas must also be contained, a property called *left-closedness* in the theory of Event Structures [14]. The order of versions is persisted in a *History Graph*, which is also directed, acyclic and append-only. Versions refer to their predecessor(s) by *parent* links, whose semantics are identical to parent links in git.

Figure 4 of our running example already showed a *History Graph*, with branching into

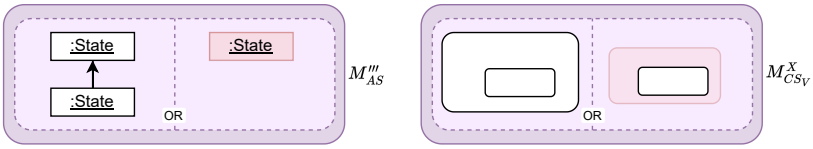


**Figure 7:** Running example: Instances of versions and $M'''_{AS}$ and $M^{X}_{CS_V}$ in a superposition each.

concurrent versions ($M'_x$ and $M''_x$) and merging ($M'''_x$). Figure 6 visualizes the different versions as sets of deltas.

The *difference* between any two versions $A$ and $B$ is simply a set of *added* deltas ($B \setminus A$) and a set of *removed* deltas ($A \setminus B$). Versions can be trivially merged by taking the union of their deltas, which preserves left-closedness.

**Superpositions.** Versions are allowed to contain conflicting deltas. This way, conflicting states are not just temporary, in-memory phenomena during the merge process, but persistent, which has two advantages: (1) We can record not just the occurence, but also steps taken in the resolution of the conflict (as a sequence of versions). This may be valueable information. (2) Conflicts are non-blocking, and the modeler can continue working on non-conflicting parts of the model.

We use the term *superposition* for versions containing conflict(s) (reference to quantum physics), because the version can be interpreted as containing all possible conflict resolutions. A conflict (= a pair of deltas) is resolved by excluding from the next version at least one of the conflicting deltas (and its dependants). Possibly a new delta is introduced, replacing both of the conflicting deltas.

In Figure 6, the version $M'''_{AS}$ is in a superposition, because it contains deltas $as_2$ and $as_3$ in a *require-delete* conflict. The possible (non-conflicting) versions contained in the superposition are $\{as_1\} \times \{as_2, as_3, as_{n_1}, as_{n_2}, ...\}$, where $as_{n_i}$ is a new (e.g. manually added) delta that replaces both conflicting deltas.

Conflicts in versions can be visualized by presenting the effects of non-conflicting deltas in a side-by-side view. Figure 7 shows $M''_{AS}$ and $M^{X}_{CS_V}$ in their the superposition.

## 3.5. Parsing and rendering interface

We now define an interface for the parsing and rendering behavior. Parsing and rendering does not alter models in-place, and their only effect is that they produce new model versions. We specify their interfaces as *pure functions* (i.e. functions without side-effects, that only have read access to their inputs) that return the newly produced model versions. Purity keeps us honest, forcing us to be explicit about all inputs, and guaranteeing that results are repeatable, which has benefits in distributed environments: For instance, in live collaboration on the same CS model, all users could parse the model locally (to reduce latency), while guaranteeing that their results are identical.

In order to parse incrementally, the parsing and rendering functions need access to the most recent correspondence model, and to the respective CS or AS changes. As output, a new correspondence model is produced, which includes a new AS or CS model, respectively.

**Running example:** In our example, the inputs and outputs are as follows:

$$parse(M_{Corr}, d(M_{CS_V}, M'_{CS_V})) = (M'_{Corr}, M'_{AS})$$
$$render(M_{Corr}, d(M_{AS}, M''_{AS}), *) = (M''_{Corr}, M''_{CS_V})$$

where $d$ is the difference (i.e. added, removed elements) between two model versions, which we can easily compute from our deltas. In the rendering function, the asterisk $*$ denotes the possibility of having additional parameters, which we will motivate in the next section.

The types of parsing errors that can occur are specific to both CS and AS, so we feel that they should be defined on the level of the correspondence metamodel. When a parsing error occurs, an object describing the error should be created in the returned correspondence model.

### 3.6. Dealing with missing information

Rendering is usually non-deterministic in the sense that an AS model can be mapped correctly onto many CS models, due to missing information (e.g. about layout). An automated rendering *function* can however only produce a single result, which may not always match the user's intention (which ultimately remains unknown to the computer). Therefore, we believe we must support human interaction in rendering. We see several complementary ways to support this.

First, the rendering function could have additional input parameters, such as a random seed or a layout heuristic to optimize, that tweak the result. We could further present a number of pre-rendered solutions to choose from, based on frequently chosen parameter values. Furthermore, after rendering, the user can make manual improvements in the CS model. Perhaps a feature in the CS editor to "freeze" the AS (guaranteeing that the user cannot accidentally alter the AS while altering the CS) could be beneficial here. This would be easy to implement since we know when a CS change causes a AS change. Finally, by relying on versioning to record *everything*, changes do not have to be rendered (or parsed) immediately. For instance, in a scenario where a Statechart is being edited through a textual syntax, (interactively) rendering the visual syntax may be postponed until the visual syntax is actually opened. In the end, empirical study should point out preferred workflows. Our work can become a basis for such study.

## 4. Related Work

For space reasons, we limit ourselves in the following to model versioning approaches. A comprehensive overview of this topic by Brosch et al. can be found in [11]. Various (overlapping) definitions of conflicts and/or inconsistencies are given in the literature. Mens [15] distinguishes syntactic, structural, and semantic conflicts, but also calls all of them inconsistencies. Taentzer et al. provide in [16] a precise formal definition of conflicts based on graph theory. They introduce the terms state-based and operation-based conflict (not to be mixed up with state-based vs. operation-based versioning approaches). Our definition of conflict matches the latter.

Brosch et al. present in [17, 18] a taxonomy of conflicts together with a visualization of the different conflicts (conflict diagram). In this taxonomy, conflicts are either "overlapping changes" (competing changes, similar to our "conflicts") or (constraint) "violations". The overall approach is based on a tight coupling between AS and CS, but is extensible with language-specific features.

In the graph transformation rules applied for conflict detection and resolution, only the elements of the AS are considered and also only (abstract) elements of the conflict diagram are generated. Layout problems arising during the rendering process of the conflict diagram are partially solved using simple heuristics preserving layout continuity. Layout conflicts in the original model are only handled in the simplest way by just preserving the layout of the merging user. More complex layout problems are not considered (and even cannot due to the restriction to the level of AS).

In contrast to this approach, we explicitly distinguish between CS and AS and thus, are able to detect and handle conflicts more precisely. In addition, due to our loose coupling of CS and AS via a correspondence model, we directly support blended modeling environments which is not the case in the AMOR project [19]. The proposed "conflict-tolerant merging of models" [13] by Wieland et al. is enabled by design in our approach including the information about "how this conflict was resolved and who was responsible for the resolution decision" [13]. The persistence of all (conflicting) deltas together with meta-information of the delta author fulfills this requirement.

In [9], van Tendeloo et al. present a more flexible framework for collaborative model development with a clear separation of CS and AS, each corresponding to their own metamodel. Our underlying idea of a flexible, language-independent modeling environment that supports the indeterminacy needed for blended modeling is based on this work [20]. The problem with van Tendeloo's framework is that it leaves open when and how the synchronization of CS and AS can and should be done. Our approach attempts to bridge this gap.

The work of Pietron et al. [12] presents an operation-based versioning system propagating user-performed edit operations. Compared to the approach in the course of this paper, their work focuses mainly on the AS and does not support multiple CS and their synchronization. Some CS-related operations, such as updating the layout of an element, are supported but lack a clear distinction from AS-related operations.

## 5. Conclusion

We presented a set of components and interfaces for collaborative modeling environments supporting CS reuse and blended modeling through loose CS/AS coupling, and bi-directional synchronization. By explicitly versioning CS, AS, and their correspondence, we can distinguish between conflicts on each of these levels, and synchronizations can happen asynchronously or even be postponed, which is especially useful when dealing with missing information while rendering. By allowing versions with unresolved conflicts to be persisted, we support deferred resolution of merge conflicts in a collaborative way, as suggested by Wieland et al. [13].

Currently, we are working on a formal, yet abstract description of a conflict and inconsistency detection algorithm and its demonstration in a prototypical web-based implementation of such a modeling environment. Another future research direction is the consideration of multiple layers between CS and AS with increasing abstraction levels (insideness, connectedness relation, etc.), to allow even more CS reuse between languages.

## Acknowledgments

Author J. Exelmans is an SB PhD fellow at FWO (1S70622N). Author J. Pietron is partly funded by the project *GENIAL!*, which is partly funded by the German Federal Ministry of Education and Research (BMBF) within the research programme ICT 2020 (reference number: 16ES0875).

## References

[1] F. Ciccozzi, M. Tichy, H. Vangheluwe, D. Weyns, Blended Modelling - What, Why and How, in: 2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2019, pp. 425–430. doi:10.1109/MODELS-C.2019.00068.

[2] S. Abrahão, F. Bourdeleau, B. H. C. Cheng, S. Kokaly, R. F. Paige, H. Störrle, J. Whittle, User experience for model-driven engineering: Challenges and future directions, in: 20th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2017, IEEE, 2017, pp. 229–236. doi:10.1109/MODELS.2017.5.

[3] I. David, K. Aslam, S. Faridmoayer, I. Malavolta, E. Syriani, P. Lago, Collaborative model-driven software engineering: A systematic update, in: 24th International Conference on Model Driven Engineering Languages and Systems, MODELS 2021, IEEE, 2021, pp. 273–284. doi:10.1109/MODELS50736.2021.00035.

[4] git Version Control System, 2022. URL: https://git-scm.com/, last visited: 10/05/2022.

[5] Apache Subversion, 2022. URL: https://subversion.apache.org/, last visited: 10/05/2022.

[6] T. Berger, M. Völter, H. P. Jensen, T. Dangprasert, J. Siegmund, Efficiency of projectional editing: A controlled experiment, in: Proceedings of the 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), ACM, New York, NY, USA, 2016, p. 763–774. doi:10.1145/2950290.2950315.

[7] B. Nuseibeh, S. Easterbrook, A. Russo, Making inconsistency respectable in software development, Journal of Systems and Software 58 (2001) 171 – 180. doi:10.1016/S0164-1212(01)00036-X.

[8] E. Guerra, J. de Lara, On the Quest for Flexible Modelling, in: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems (MODELS), MODELS '18, ACM, New York, NY, USA, 2018, pp. 23–33. doi:10.1145/3239372.3239376.

[9] Y. V. Tendeloo, H. Vangheluwe, Unifying model- and screen sharing, IEEE 27th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE) (2018) 127–132. doi:10.1109/WETICE.2018.00031.

[10] A. Schürr, Specification of graph translators with triple graph grammars, in: E. W. Mayr, G. Schmidt, G. Tinhofer (Eds.), Graph-Theoretic Concepts in Computer Science, Springer Berlin Heidelberg, Berlin, Heidelberg, 1995, pp. 151–163. doi:10.1007/3-540-59071-4_45.

[11] P. Brosch, G. Kappel, P. Langer, M. Seidl, K. Wieland, M. Wimmer, An Introduction to Model Versioning, in: M. Bernardo, V. Cortellessa, A. Pierantonio (Eds.), Formal Methods

for Model-Driven Engineering, volume LNCS 7320, Springer, Berlin, Heidelberg, 2012, pp. 336–398. doi:10.1007/978-3-642-30982-3_10.

[12] J. Pietron, F. Füg, M. Tichy, An operation-based versioning approach for synchronous and asynchronous collaboration in graphical modeling tools, in: L. Iovino, L. M. Kristensen (Eds.), STAF 2021 Workshop Proceedings, volume 2999 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2021, pp. 88–89. URL: http://ceur-ws.org/Vol-2999/fpvmdata4mdepaper3.pdf.

[13] K. Wieland, P. Langer, M. Seidl, M. Wimmer, G. Kappel, Turning Conflicts into Collaboration, Computer Supported Cooperative Work (CSCW) 22 (2013) 181–240. doi:10.1007/s10606-012-9172-4.

[14] G. Winskel, An introduction to event structures, in: J. W. de Bakker, W. P. de Roever, G. Rozenberg (Eds.), Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency, School/Workshop, Noordwijkerhout, The Netherlands, May 30 - June 3, 1988, Proceedings, volume 354 of *Lecture Notes in Computer Science*, Springer, 1988, pp. 364–397. doi:10.1007/BFb0013026.

[15] T. Mens, A state-of-the-art survey on software merging, IEEE Transactions on Software Engineering 28 (2002) 449–462. doi:10.1109/TSE.2002.1000449.

[16] G. Taentzer, C. Ermel, P. Langer, M. Wimmer, Conflict Detection for Model Versioning Based on Graph Modifications, in: H. Ehrig, A. Rensink, G. Rozenberg, A. Schürr (Eds.), Graph Transformations, Lecture Notes in Computer Science, Springer, Berlin, Heidelberg, 2010, pp. 171–186. doi:10/dcjxkr.

[17] P. Brosch, M. Seidl, M. Wimmer, G. Kappel, Conflict visualization for evolving UML models, J. Object Technol. 11 (2012) 2: 1–30. doi:10.5381/jot.2012.11.3.a2.

[18] P. Brosch, Conflict Resolution in Model Versioning, Ph.D. thesis, Vienna University of Technology, Vienna, 2012. URL: https://publik.tuwien.ac.at/files/PubDat_208975.pdf.

[19] AMOR — Adaptable Model Versioning, 2009. URL: http://modelversioning.org/, last visited: 10/05/2022.

[20] L. Nachreiner, A. Raschke, M. Stegmaier, M. Tichy, CouchEdit: A Relaxed Conformance Editing Approach, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings (MODELS-C), ACM, 2020, pp. 1–5. doi:10.1145/3417990.3421401.

# From Object to Class Models: More Steps towards Flexible Modeling (Short Paper)

Martin Gogolla[1], Bran Selic[2], Andreas Kästner[1], Larousse Degrandow[1] and Cyrille Namegni[1]

[1]University of Bremen, Computer Science, 28334 Bremen, Germany
[2]Malina Software Corp., Ottawa K2J 2J3, Canada

Abstract

This contribution discusses a flexible modeling approach that proposes to develop class diagrams starting from object diagrams. On the one hand, the contribution explains in a general way the benefits of flexible visual modeling by allowing a lively development process through relaxing the formal requirements for artefacts in the work process. On the other hand, the contribution shows a concrete example and explains an implementation in a tool, in particular how to cover whole-part relationships, generalization and an improved handling for association multiplicities. The aim is to give developers the option to let their ideas flow in a free way with few creativity restrictions by a tool. Flexibility may be gained by transitioning in the work process between specific, instance-based visual models and generic, type-based visual models where both kinds of models allow for incompleteness or inconsistency.

**Keywords**

Object model, Class model, Flexible modeling, Incomplete model, Inconsistent model

## 1. Motivation

There is no doubt that precision plays a fundamental role in all good engineering. It is particularly significant in software engineering, which is founded to a great extent on applied mathematical logic.

In essence, precision implies the elimination of ambiguity, which is typically a source of uncertainty and can ultimately lead to invalid or inappropriate design decisions. In engineering, precision is typically achieved by the application of some type of formal mathematical methods. By means of formal mathematical constraints and validity rules, it is possible to define elements of a design in a way that eliminates subjectivity or the possibility of misinterpretation.

However, this level of precision does not come easily. Often it is only possible if we have sufficient understanding of the topic. And therein "lies the rub"; reaching an understanding of some complex aspect takes time. One of the most effective means for reaching understanding is direct experience with the subject matter. This typically involves trial and error, so that we can appreciate not only what works and why, but equally important, what does not work and why. For this reason, prototyping is essential to most complex engineering projects.

*2nd Int. Workshop on Foundations and Practice of Visual Modeling, July 4–8, 2022, Nantes, France, Co-located with STAF 2022*

gogolla@uni-bremen.de (M. Gogolla); selic@acm.org (B. Selic); andreask@uni-bremen.de (A. Kästner); degrandow@uni-bremen.de (L. Degrandow); jikename@uni-bremen.de (C. Namegni)

In this process of forming an understanding through trial and error, introducing formal constraints too early in this process can create a kind of "bureaucratic" hurdle that can stand in the way of not only understanding but also creativity. Innovative ideas often start off in vague and imprecise form, and need to be refined gradually before an informed decision can be made whether to adopt them or discard them. Consequently, formal methods should only be applied when sufficient information (i.e., understanding) has been attained.

Based on the above, at the core of the work described here is the idea of a flexible design approach, which allows formality and, hence, precision, to be introduced gradually and selectively, as design and understanding progress. Thus, we may start off with an early informal model of a proposed design. This initial model may suffer from incompleteness and even inconsistency, but it may still be useful in helping designers gain an understanding of its properties. If this ambiguos model appears promising, we may then decide to apply formal approaches selectively, to help us gain confidence. This "mild" level of formality might reveal fundamental flaws in the design, at which point it may be amended or even discarded due to serious flaws. Over time and as the design is refined, the degree of formal checking can be gradually increased, ultimately reaching the fullest extent possible.

One of the advantages of such an approach is early detection of design flaws in what may have seemed as a promising approach. This is because the overhead involved in "full" formal validation is avoided. This overhead involves specifying the full set of details required to avoid incompleteness and inconsistency errors even of an early putative model is eliminated. On the other hand, by allowing selective application of formal checking, it allows designers to detect key flaws in those areas where they may be most uncertain. The end result is likely to be a faster path to the ultimate solution.

The paper is structured as follows. Section 2 discusses the technical context of our contribution. In Section 3 we present our view and positions on flexible visual modeling. In Section 4 we put forward the technical content of a recent tool extension. The paper closes with a short summary, some conclusions and future work.

## 2. Context

A key ingredient to a high level of productivity and product quality as promised by Model-Based Engineering (MBE), e.g., with UML [1, 2, 3], is computer-supported automation. But practical experience with current MBE tools indicates that we are still far from this ideal. Typically, tools are difficult to learn and use and are complex. Frustrating situations where the tools are forcing users into workarounds and constrained operating modes are frequent. This is contrary to free expression of ideas. For achieving effective tool support, the transition between an informal, provisional mode and a formal, precise mode is crucial. This contribution is a further step into that way of working with tools.

To enable a development process that includes an ability that is similar to informal diagrams sketched "on a napkin", we are working on a flexible modeling approach [4, 5], which focuses on objects [6]. Starting with incomplete or even inconsistent UML object diagrams, we have developed an automated transformation of these into class diagrams, as a plugin for the USE tool [7, 8]. Both the object diagrams as well as the class diagrams use a flexible syntax which

comes close to an informal drawing tool while still following an internal meta-model. In the work process, developers have the option to incrementally create the object diagram while receiving feedback from the resulting class diagram. This paper extends on the technical side the previous work [4, 5] by handling whole-part relationships and inheritance as well as an improved handling for association multiplicities. The current extension of the USE tool has been applied for smaller teaching projects, in particular by students developing course projects; a systematic study is planned for future work. The paper also presents an extension on the conceptual side by summarizing the benefits of our approach and its potential to the development process.

Related approaches for flexible development of systems and transformations have been proposed: Related works on example based modeling include [9, 10, 11, 12]; flexible transformations, partly on an example focused basis, are [13, 14, 15]; uncertainty and partiality in modeling has been studied in [16, 17, 18, 19]. General dimensions of flexible modeling together with concrete application options are presented in [20]. The work in [21] discusses flexible typing. [22] concentrates on flexibility in domain-specific modeling. The Typing Requirements Models (TRM) in [23] permit a high degree of variability and flexibilty for typing model transformations and lead to improved reuse options.

## 3. Positions on and Benefits of Flexible Modeling

**Visualization techniques and methodologies:** Our approach utilizes mainstream modeling visualization techniques with slight modifications and proposes a particular methodology for their application. Basically, we start from conventional UML class and object diagrams, and we extend them to what we call "imperfect" class and object diagrams. That means our class and object diagram do not follow strictly the conventional UML metamodel, but an extended one that allows incomplete and inconsistent diagrams. This opens in the work process to developers the option to let their ideas flow in a free way without having to obey conventional metamodel restrictions (e.g., "Attributes must have a type.") and typical tool requirements (e.g., "Only class diagrams valid w.r.t. the conventional UML metamodel can be stored"). The principle of modifying and relaxing a language metamodel to allow for incompleteness and inconsistency can be applied to other UML sub-languages as well. For example, allowed UML operation call sequences that are abstracted to UML protocol state machines could be relaxed to "imperfect" UML operation call sequences (where, e.g., not all all calls of a fixed operation have the same number of parameters) and "imperfect" UML protocol state machines (where, e.g., not all operation calls have a corresponding operation in the class diagram).

**Visualizing errors in models:** In our approach we have implemented a particular way of handling incompleteness, inconsistency and incorrectness (we call them the three "incos"), as displayed in Fig. 1. Currently, the focus is on class and object diagrams, but the principles can be extended to other UML diagrams, or more generally to other kinds of models. *Incompleteness* is indicated by elements with a plus mark or by dashed elements in our object diagrams. In our class diagrams a question mark indicates an incomplete specification. *Inconsistency* is put forward by elements marked with an exclamation in our class diagrams. *Incorrectness* is presented with dashed elements in our class diagrams. The technical realization and the justification for the different kinds of representation will be discussed and become clear in the

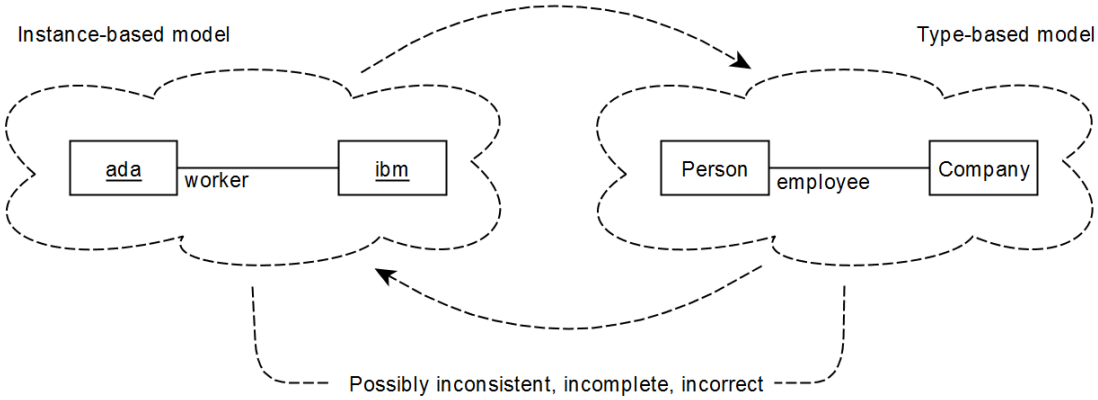


**Figure 1:** Idealized work process with imperfect instance- and type-based models.

following section. We emphasize that these additional language features (for the three incos) in our diagrams extend the conventional UML notation.

To our knowledge, the three "incos" have not been formally defined, and it is impossible to do so. Nevertheless, we want to state an informal explanation on how we view these three, overlapping notions. *Incompleteness* refers to the observation that an important aspect is yet missing in the model or description. *Inconsistency* expresses that there are at least two details in the model that contradict each other. *Incorrectness* comes in our view in two shades, namely syntactic and semantic incorrectness: Syntactic incorrectness means that the model does not meet its metamodel, and semantic incorrectness means that the model does not completely meet the real-world excerpt that it is intended to describe.

**Collaborative development with human-in-the-loop:** Our approach relies on model improvement through iteration: we start with an imperfect object diagram and from this we derive a first imperfect class diagram; by adding more possibly improved object diagrams or object diagram for further scenarios and through repeating the (object,class) transitions, ultimately a settled class diagram describing correctly all developed scenarios is achieved. As mentioned already, the (object,class) transitions could be generalized to instance-based artefacts alternated by type-based artefacts, e.g., by transitioning between example command sequences and protocol state machines.

**Imperfect artefacts:** In any case, our aim is to give developers the option to let their ideas flow in a free way with few creativity restrictions by a tool and the implicit steps in the work process. Fexibility may be gained by transitioning in the work process between specific, instance-based visual models and generic, type-based visual models where both kinds of imperfect models allow for the three incos: incompleteness, inconsistency, and incorrectness.

## 4. Whole-Part Relationships, Generalization, Multiplicities

This section discusses the newly designed and implemented tool functionality by means of an example. Figure 2 shows the formation of the (incomplete) output class diagram in the right side

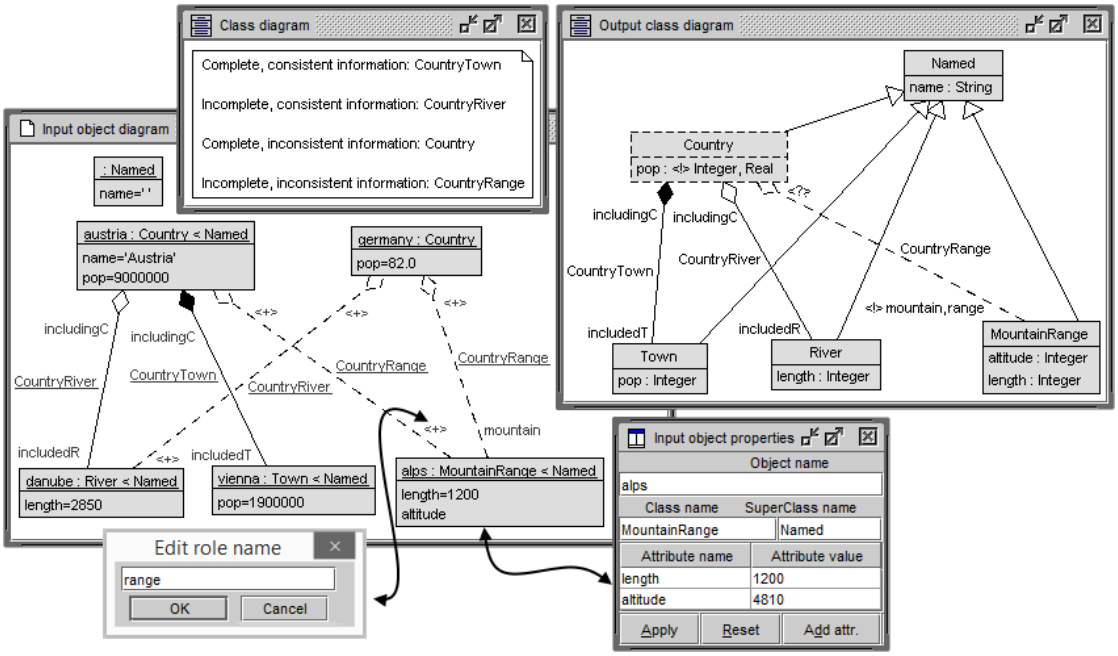


**Figure 2:** Utilizing Features for Whole-Part Relationships and Generalization

on the basis of the (incomplete) input object diagram in the left side. The input object diagram specifies incomplete objects (e.g., the attribute 'name' is present in the object 'austria' but missing in the object 'germany') and incomplete aggregation and composition links (e.g., role names are present for the left 'CountryRiver' link but are partly missing for 'CountryRange'). In the input and in the output, graphical elements drawn using *continuous, solid* lines and contours represent fully specified entities, whereas those drawn using *dashed* lines and contours stand for entities with somewhat incorrect specification, i.e., incomplete or inconsistent information. The intention of our approach is that such incomplete, even inconsistent object diagrams may be used when new ideas and concepts are introduced into the models (e.g., typically in the early phases of the software development process). Developers should have the freedom to let their ideas flow in a *natural* way, even if their diagrams do not (yet) meet all formal requirements of the underlying modeling language or tool.

There are four cases w.r.t. available information in the object diagram for mapping objects and links to classes and associations (more specifically to aggregations and compositions).

**Complete, consistent information:** The composition 'CountryTown' can be completely derived (obtaining composition name and role names), given the complete and consistent object diagram information.

**Incomplete, consistent information:** The aggregation 'CountryRiver' can also be completely derived, although some links in the object diagram are only partially specified. The incomplete object diagram information can be matched against the more complete class diagram information.

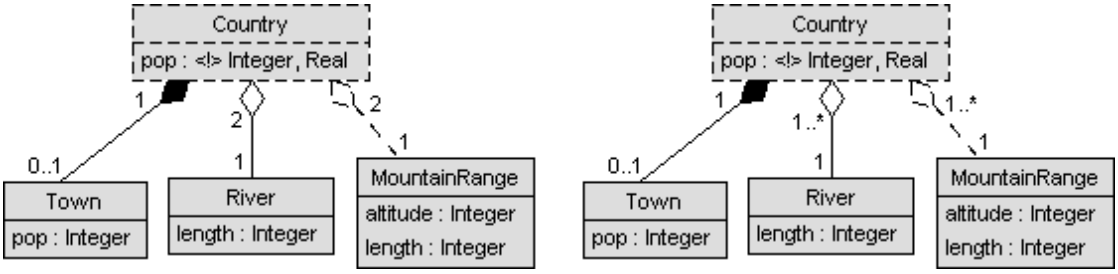


**Figure 3:** Exact and Standard Multiplicities

**Complete, inconsistent information:** The object diagram information for class 'Country' is complete, but inconsistently specified (contradicting attribute datatypes). This contradiction is highlighted in the resulting class diagram; i.e., the dashed rectangle of the class 'Country' flags it as a 'to-be-improved' element.

**Incomplete, inconsistent information:** The aggregation 'CountryRange' is incomplete, as role names on the 'Country' side are missing in the object diagram. The object diagram information is inconsistent because of mutually contradictory role names ('mountain' vs. 'range') on the other aggregation side. The aggregation 'CountryRange' is also flagged as 'to-be-improved', using the dashed aggregation link.

We have decided to use the same visual elements (i.e., for the 'incos' incompleteness, inconsistency, incorrectness) in the object and class diagram (or to say it more generally for the instance-based description and the type-based description). In the specific situation with deriving a class diagram from object diagrams, basically only the object diagram can be manipulated by the developer and the class diagram is automatically derived from the object diagram. In an even more flexible (but also more complicated) approach, both descriptions could be edited. To sum up in simple words, the plus (in the object diagram) and question mark (in the class diagram) stands for incompleteness, the exclamation mark for inconsistency, and the dashed elements for incorrectness resp. for items that still need more work.

The specification of *inheritance* in the object model is indicated by allowing the name of a superclass (as a type) in the name field of the object's rectangle. The determination of attributes of the superclass is done by prototypical objects. For example, in Figure 2, there is one such unnamed prototypical object whose type is the superclass 'Named'. This object has an attribute 'name', whose value is specified as an empty String value. Superclasses could also be identified (maybe in an even smoother way) by a refactoring process after a number of objects have been constructed. This is currently not possible in the tool.

The handling of *multiplicities* (as in Fig.3) has been improved compared to earlier versions of the tool. In the output class model there is now a new option to either use multiplicities as exactly stated in the object model (e.g. '0..1' or '2') or to use only multiplicities from a fixed collection of frequently applied standard multiplicities ('0..1', '1', '0..*', '1..*').

## 5. Conclusion and Future Work

The contribution has shown how principles of flexible modeling can be applied and how an existing approach for flexible visual modeling can be extended. A central goal was the early detection of design flaws avoiding the overhead of "full" formal validation. Incompleteness and inconsistency are temporarily accepted through selective application of formal checking giving designers the option to detect key flaws in those areas where they may be most uncertain.

Much more work on other modeling aspects (e.g., transitioning from prototypical behavior models in the form of example command sequences to complete protocol state machines) remains to be done. The principle of transitioning between instance-based and type-based imperfect descriptions and models can probably be carried over to more modeling areas. In a collaborative modeling context, different objects and links from different developers could be presented and handled differently. Last but definitively very important, user studies must give more feedback about the applicability of the approach.

## References


[1] G. Booch, J. Rumbaugh, I. Jacobson, The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[2] J. Rumbaugh, I. Jacobson, G. Booch, The Unified Modeling Language Reference Manual, 2nd ed., Addison-Wesley, 2005.

[3] B. Rumpe, Modellierung mit UML, Springer, 2004.

[4] A. Kästner, M. Gogolla, B. Selic, From (Imperfect) Object Diagrams to (Imperfect) Class Diagrams, in: O. Haugen, R. Paige (Eds.), Proc. 21th Int. Conf. Model Driven Engineering Languages and Systems (MoDELS'2018), ACM/IEEE, 2018, pp. 13–22.

[5] A. Kästner, M. Gogolla, B. Selic, Towards Flexible Object and Class Modeling Tools: An Experience Report, in: D. di Ruscio, J. de Lara, A. Pierantonio (Eds.), Proc. 4th Flexible MDE Workshop (FlexMDE 2018), CEUR Proceedings 2245, 2018, pp. 233–242.

[6] B. Selic, Career Award Talk, YouTube https://youtu.be/9qPbGksB3d4?t=20m32s, 2016.

[7] M. Gogolla, F. Büttner, M. Richters, USE: A UML-Based Specification Environment for Validating UML and OCL, Science of Computer Programming 69 (2007) 27–34.

[8] M. Gogolla, F. Hilken, K.-H. Doan, Achieving Model Quality through Model Validation, Verification and Exploration, Journal on Computer Languages, Systems and Structures, Elsevier, NL 54 (2018) 474–511.

[9] J. J. López-Fernández, J. S. Cuadrado, E. Guerra, J. de Lara, Example-driven meta-model development, Software & Systems Modeling 14 (2015) 1323–1347.

[10] S. Maoz, J. O. Ringert, B. Rumpe, Modal Object Diagrams, in: M. Mezini (Ed.), ECOOP 2011 – Object-Oriented Programming, Springer Berlin Heidelberg, 2011, pp. 281–305.

[11] D. Zayan, A. Sarkar, M. Antkiewicz, R. S. P. Maciel, K. Czarnecki, Example-driven modeling: on effects of using examples on structural model comprehension, what makes them useful, and how to create them, Software & Systems Modeling (2018).

[12] D. Wüest, N. Seyff, M. Glinz, Flexisketch: a lightweight sketching and metamodeling

approach for end-users, Softw. Syst. Model. 18 (2019) 1513–1541. URL: https://doi.org/10.1007/s10270-017-0623-8. doi:10.1007/s10270-017-0623-8.

[13] G. Kappel, P. Langer, W. Retschitzegger, W. Schwinger, M. Wimmer, Model Transformation By-Example: A Survey of the First Wave, in: A. Düsterhöft, M. Klettke, K.-D. Schewe (Eds.), Conceptual Modelling and Its Theoretical Foundations, Springer Berlin Heidelberg, 2012, pp. 197–215.

[14] T. Mens, P. Van Gorp, A Taxonomy of Model Transformation, Electronic Notes in Theoretical Computer Science 152 (2006) 125–142.

[15] W. Smid, A. Rensink, Class Diagram Restructuring with GROOVE, in: P. Van Gorp, L. Rose, C. Krause (Eds.), Proceedings Sixth Transformation Tool Contest, Electronic Proceedings in Theoretical Computer Science, 2013, pp. 83–87.

[16] R. Salay, M. Chechik, M. Famelis, J. Gorzny, A Methodology for Verifying Refinements of Partial Models, Journal of Object Technology 14 (2015). URL: https://doi.org/10.5381/jot.2015.14.3.a3. doi:10.5381/jot.2015.14.3.a3.

[17] O. Semeráth, D. Varró, Graph Constraint Evaluation over Partial Models by Constraint Rewriting, in: E. Guerra, M. van den Brand (Eds.), Theory and Practice of Model Transformation, Springer International Publishing, Cham, 2017, pp. 138–154.

[18] R. Salay, M. Famelis, M. Chechik, Language independent refinement using partial modeling, in: J. de Lara, A. Zisman (Eds.), Fundamental Approaches to Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2012, pp. 224–239.

[19] M. Famelis, S. Santosa, Mav-vis: A notation for model uncertainty, in: ICSE Workshop on Software Engineering for Adaptive and Self-Managing Systems, 2013, pp. 7–12.

[20] E. Guerra, J. de Lara, On the quest for flexible modelling, in: A. Wasowski, R. F. Paige, Ø. Haugen (Eds.), Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018, ACM, 2018, pp. 23–33.

[21] A. Zolotas, N. Matragkas, S. Devlin, D. S. Kolovos, R. F. Paige, Type inference in flexible model-driven engineering using classification algorithms, Softw. Syst. Model. 18 (2019) 345–366.

[22] F. R. Golra, A. Beugnard, F. Dagnat, S. Guérin, C. Guychard, Using free modeling as an agile method for developing domain specific modeling languages, in: B. Baudry, B. Combemale (Eds.), Proceedings of the ACM/IEEE 19th International Conference on Model Driven Engineering Languages and Systems, Saint-Malo, France, October 2-7, 2016, ACM, 2016, pp. 24–34.

[23] J. de Lara, J. D. Rocco, D. D. Ruscio, E. Guerra, L. Iovino, A. Pierantonio, J. S. Cuadrado, Reusing model transformations through typing requirements models, in: M. Huisman, J. Rubin (Eds.), FASE 2017, Part of ETAPS, volume 10202 of *LNCS*, Springer, 2017, pp. 264–282.

# Model Slicing on Low-code Platforms

Ilirian Ibrahimi[1,2,*], Dimitris Moudilos[2]

[1]Johannes Kepler University, Institute of Software Engineering, Altenberger Straße 69, Linz, Austria
[2]CLMS UK. Battle House,1 East Barnet Road,New Barnet, Herts EN4 8RR, UK, and Andrea Papandreo 19, Athens, Greece

**Abstract**

Low-code platforms (LCP) use models as the main artifact during the software development process. Typically, the modeling activity concerns both structural and behavioral aspects of the generated application, like the underlying data model (DM), User Interface (UI), and business logic (BL), resulting in a collection of interconnected models. Thus, reusing model fragments across different projects would be a highly beneficial feature for LCPs and their users.

This paper presents a model slicing approach for LCP models that combines DM, UI, and BL modeling concerns. A model slice consists of a DM class which serves as an input for the approach, its DM constraint-related classes e.g., base classes, and its related UI entities as well as BL functions.

The model slicer operates on separated model repositories which will be queried to find related entities to the DM input class and integrate them automatically into the LCP. We conducted an experimental evaluation with zAppDev models and concluded that 77.78% of the DM classes are cross-connected to any entity among the zAppDev models. Hence all these connected entities can be extracted as model slices and reused automatically.

**Keywords**

MDE, Low-code platfroms, Model slicing, Model reuse, Knowledge graphs

## 1. Introduction

Low-code platforms (LCP) are cloud-based applications that serve for building full-stack software applications without necessarily requesting coding knowledge. One of the main capabilities of an LCP is modeling the software application by designing its data models (DM), the user interface - Form models, and business logic model(s) (BL), writing as less as possible domain-specific code for implementing and deploying a complete software application.

Typical software engineering activities like coding in a general-purpose language, code formatting, modularization, database configuration, deployment, etc, are automated [1, 2]. By giving priority to modeling rather than coding, LCP enables so-called *citizen developers*, i.e., stakeholders with very limited or even no coding experience, the opportunity to create full-stack software applications which makes the LCPs more popular and useful in the software development industry [3].

LCPs leverage model-driven engineering techniques (MDE) so that models are the cornerstone artifacts that drive the overall engineering process [4, 5]. Some of these models as a whole or

*Staf 2022 Workshop - 2nd International Workshop on Foundations and Practice of Visual Modeling (FVPM)*
*Corresponding author.
ibrahimi.ilirian@gmail.com (I. Ibrahimi); d.moudilos@clmsuk.com (D. Moudilos)
https://github.com/iliriani (I. Ibrahimi); https://clmsuk.com/ (D. Moudilos)

part of it (i.e., their elements) might be shared among different LCP systems. Thus, finding any solution on how to reuse these models, which may be of different languages like XML, JSON, etc., and different levels like DM, UI models, etc., and providing all this information automatically to the user in the domain modeling stage would be a novel and highly on-demand task.

Hence, this paper presents an approach for model reuse through model slicing on LCP. To get the information on heterogeneous models, our approach converts all the heterogeneous models to a homogeneous graph which will serve as a knowledge graph (KG). And to cope with the different levels of models, we created two different repositories, one for the DM, and another for the Form models. The two repositories persist the KG for the DM and the Forms respectively. The model slicing approach gets as input a DM class and queries both repositories in order to get related entities to it. The required entities within the DM repository like base class, composition, etc. will constitute the horizontal slice since they belong to the same model level (i.e., DM) as the input class. And the related entities from the Form repository will constitute the vertical slice since they belong to a different level than the input class. Both slices will be merged as a single model slice and provided to the developer[1] in a JSON format.

As a proof of concept, we have tested our approach on the zAppDev[2] LCP by using 5 different zAppDev DM ( in XML) with a total of 27 different domain classes, and 47 different Form models related to these domain models. The evaluation revealed that 77.78% of the given DM classes had any kind of cross-model relation i.e. we could extract successfully 21 distinct cross-language and cross-level model slices from these zAppDev models. The approach has been developed on Spring boot and is provided as a REST API.

In the rest of this paper, we present in Section 2 a running example in order to better understand the aim of the model slicing approach. In Section 3 we outline and explain how the model slicing approach works. In Section 4 we present an experimental evaluation of our approach. Afterward, in Section 5 we present some related work to model slicing, and finally, in Section 6 we provide the conclusion and the tentative future work.

## 2. Running Example

To clarify the concepts used throughout this paper, we will initially provide some background information.

### 2.1. Background Information about the zAppDev LCP

zAppDev is a web-based, model-driven development environment, allowing developers of any technology and proficiency level to easily create, edit and reuse models of software artifacts (e.g. database models, business logic models, user interface models, and more), covering the complete application development lifecycle while having total control of the process. As explained in [1] an LCP typically consists of 4 different layers which are included as well on zAppDev and are comprised of 1. The application layer is represented by the Form models, 2. The service

[1]For the sake of brevity, in this paper we will use the terms developer interchangeably for citizen developer
[2]https://zappdev.io/

integration layer by API adapters, 3. The data integration layer is represented by data models, the service models, API Adapters, and finally 4. The deployment layer is represented by the Cloud.

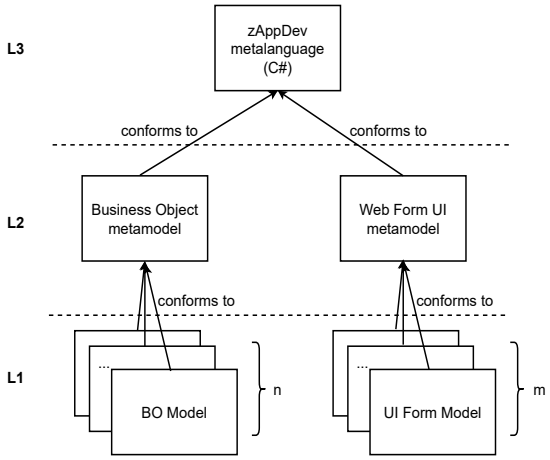


Figure 1: Metamodeling layers on the zAppDev LCP

The layers of interest for us are the application and data integration layers and their corresponding models since these are the only models the developers have direct contact with. Concretely, in this work, we will work with zAppDev data models a.k.a **business object models (BOs)**, and Form models a.k.a **UI-Form models**. The BO and UI-Form models belong to the zAppDev L1 level of the 3-level meta-modeling architecture as depicted in Fig. 1. Both, the BOs and the UI-Form models are instances of the Business Object metamodel and Web Form UI metamodel respectively (L2 level). Whereas both metamodels conform to the zAppDev metalanguage written in the c# programming language (L3 level). The metamodels and the metalanguage are embedded on zAppDev and the developers have no access to them, they start the work directly by creating BO models as instances of the BO metamodel and auto-generate the UI-Form models from the BO models or they can initially design the UI-Form model and connect it afterwards to the relevant BO model. Lastly, the developers define the business logic of any UI component within the UI-Form model by using the Mamba language.

Now, by explaining a running example we will be trying to clarify how this approach will extract model slices from LCP models.

## 2.2. Running example

Assume that in an LCP there is an `Invoice` software containing a BO, a UI-Form model named "Invoice Form" auto-generated from the BO classes or manually constructed by the developer, and the business logic functions related to the BO classes. The architecture of the `Invoice` software is depicted in Fig. 2. In the zAppDev LCP, the BOs are presented in XML format, the UI-Forms as JSON files, and the DSL functions are written in the Mamba[3] language. As we can see in Fig. 2 on the left part, the BO is constructed from the classes `Invoice`, `Client`, and `Company`. We aim to get only the related cross-level and cross-language artifacts to the `Client` BO class. As depicted in Fig. 2, the `Client` BO class is related to the Invoice Form with a label with the text `Client` on it and a combo box. Further, we can see that the `Client` class has an Edit function related to it. To emphasize the connection of all the `Client` related cross-level and cross-language entities we rounded and connected them with a blue cycle and blue lines.

[3]https://docs.zappdev.com/MambaLanguage/About/

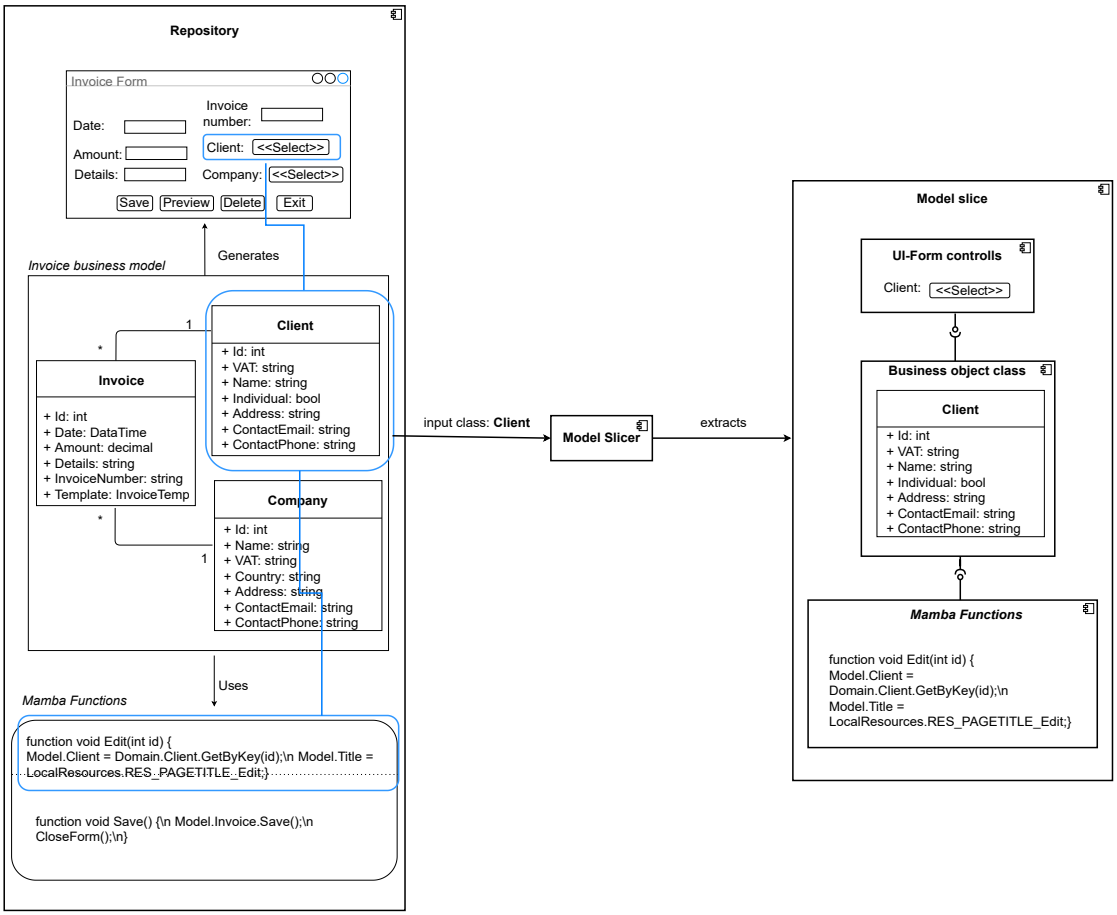


**Figure 2:** Model reuse through model slicing - running example

Now the idea of a model slicer approach on an LCP as shown in Fig. 2 would be to provide to the approach only the `Client` BO class as input and it would be capable to compute and extract all the cross-related entities to the `Client` BO class and integrate those on an LCP. As shown on the right side of Fig. 2 the extracted model slice from the approach is itself a model which contains only the `Client` relevant entities across the LCP models.

Concluding, the model slicer tends to find connected entities to the BO input classes on cross-level and cross-language models and presents these as a model slice to the developers so everything that is connected within the entire production line on any LCP can be reused and integrated automatically on an LCP. Inspired by this running example, we have created a model slicing approach that will be explained in more detail in Section 3.

## 3. Approach

This section will be presenting in more detail how the model slicer extracts model slices from cross-level models. The overview of the model slicer is presented in Fig. 3.

### 3.1. Repositories

The first step toward model slicing is persisting the models in a repository so they can be reused for different business needs afterwards. Since all models in zAppDev are graph-based, the repository of our approach is also graph-based. We selected the Resource Description Framework (RDF) [6] as our model format since it is a graph-based model and the standard format of W3C[4], this is relevant to LCPs which are cloud-based. Thus, various models will be converted and merged into a single RDF graph. We will use and refer to this RDF graph as the knowledge graph (KG) of our approach.

Since developing any software on an LCP ones needs to create its' DM, its' UI in a Form model, and the business logic which is persisted in any of these two (especially in zAppDev), we have created two different repositories for the model slicer, one which persists the knowledge graph for the DM, and another for the Form models. Thus, as explained in Fig. 3, *step 1* of our approach is creating the repositories which persist the knowledge graph for the DM and Form models by converting them to RDF and merging them to their respective knowledge graphs.

### 3.2. Input Class

*Step 2* of our approach is getting the input class. The input class is the core entity of any model slice because any other entity of the model slice has to be related in a specific form to it. Hence, the input class defines the slicing criterion for our approach. After having the input class, the slice service - which is responsible for the business logic part of the model slicer - will query the repositories to find relevant connections between the existing entities within the repositories and the input class.

### 3.3. Horizontal Slice

The first check the input class will go through is if it has any **constraints class** that has to be integrated with the input class. For instance, if the input class inherits a class within the BO classes, or has a composition class, then the base/composition class has to be integrated as well in the LCP in order to avoid model validation errors. Thus in *step 3*, our approach checks in the DT (BO) repository if there is any such constraints class related to the input class, and if any such class can be found it will be provided to the developers together with the input class. Since the input class and the constraint classes are on the same level (within the business object) we call this kind of relation as horizontal slicing. Since we are slicing only zAppDev models so far, the horizontal slice is defined only by checking if the input class has any base class within the BO repository.

### 3.4. Vertical Slice

Next, the model slicer service will check if there is any related entity within the Form KG to the input class. In step 4 the model slicer will return all the relevant information about the related entities to the input class. In our case, we query the information about UI components, i.e., the UI component name, the UI component data source - which shows to which specific

[4]https://www.w3.org/TR/2004/REC-rdf-concepts-20040210

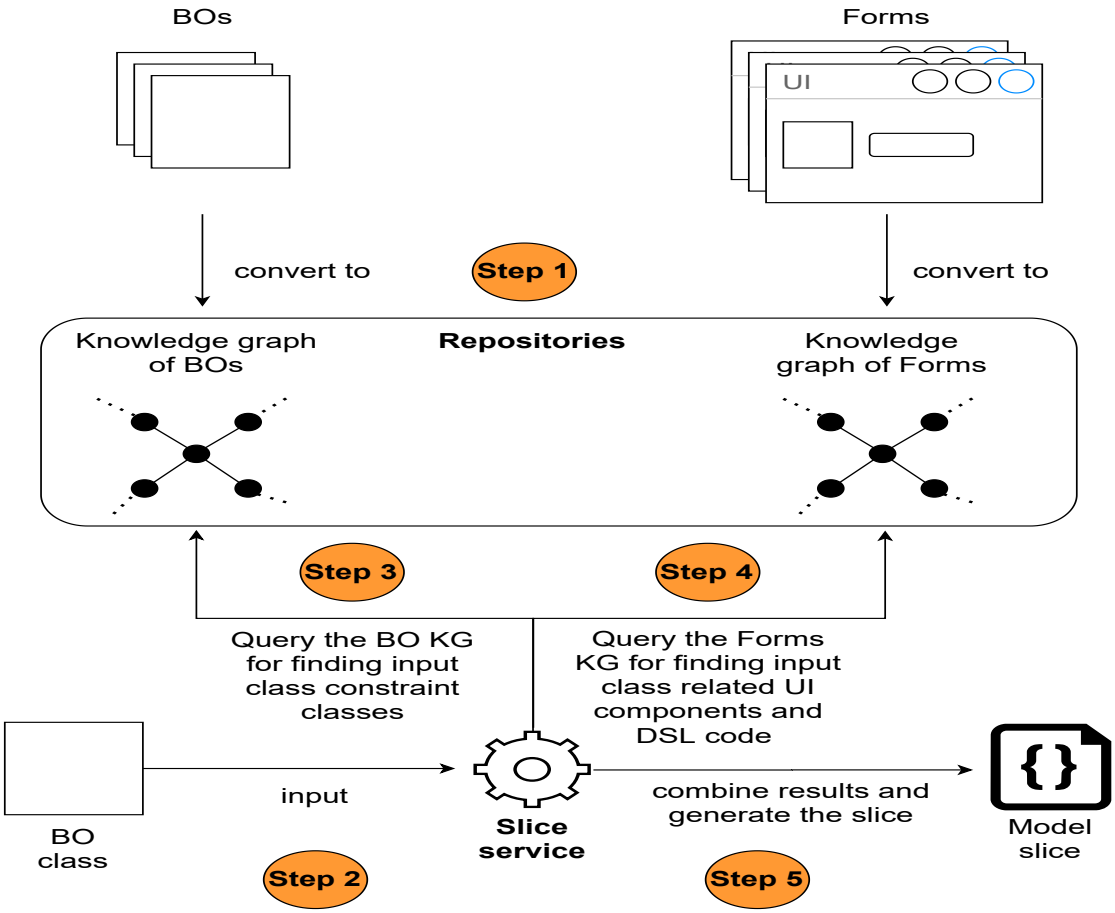


**Figure 3:** Model slicing approach overview

attribute of the input class the UI component is related, and the type of the UI component, e.g. TextBoxControll, CheckBox, etc. Although a lot of other relevant information can be retrieved e.g. the cascade style sheet (CSS) information about the UI components, UI components layout information, etc. Since in zAppDev the DSL functions a.k.a. Mamba functions are persisted within the Form models, in step 4 the model slicer also queries for related Mamba functions to the input class. Hence the UI components and DSL functions are not in the same model level as the input class, we call this relation of connected cross-level models a vertical slice.

Finally, the extracted horizontal and vertical slices will be merged as a single model slice and integrated into the LCP.

An example of model slicing is presented in Fig. 4. We see that the meta-class `Class 1` is connected to the UI components `Comp. 1` and `Comp. 2` and also to the DSL functions *Func. 1* and *Func. 2*, thus the connection of all these entities would give a single slice (Slice 1). Further in Fig. 4, we can see that the meta-class `Class 3` has a constraint class `Class 2` within the BO, it is also related to the component `Comp 3` on the UI - Form model, and it has also a related DSL function *Func. 3*. The connection of these related entities to `Class 2` would produce another

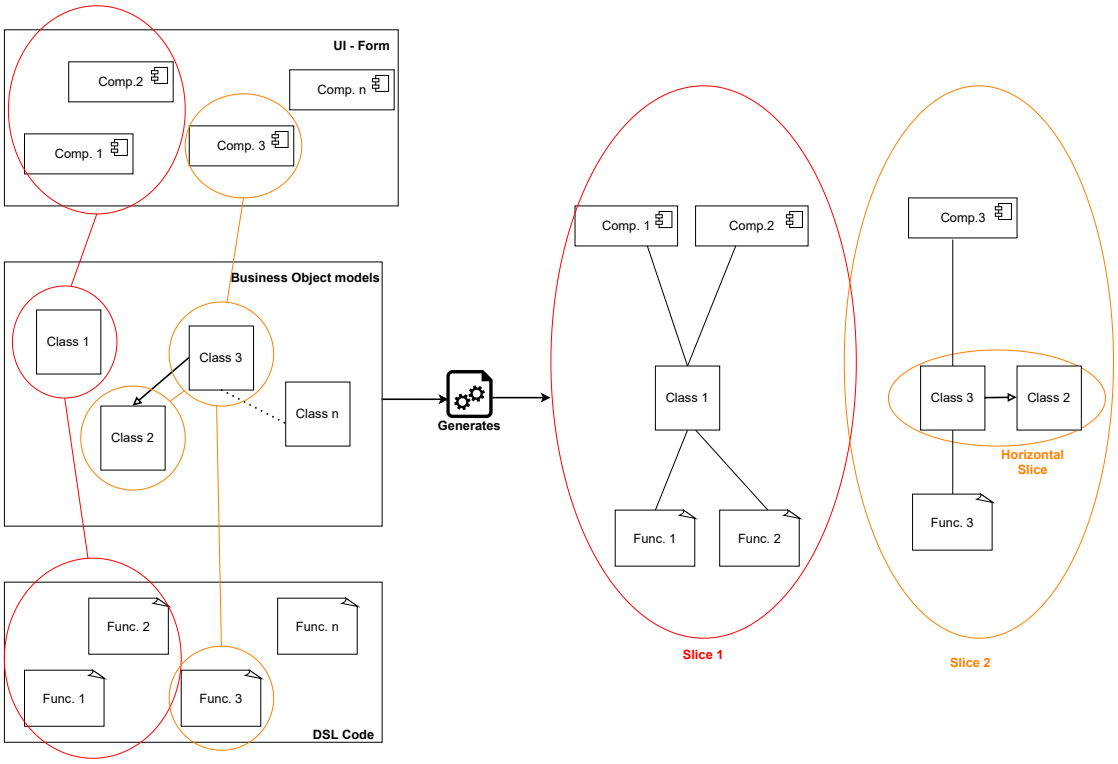


**Figure 4:** Model reuse through model slicing

| Model slices on zAppDev models | | | | |
|---|---|---|---|---|
| BO models | BO classes | UI-Form models | Horizontal Slices | Vertical Slices |
| 1 CoreBO | 10 | | 0 | 9 |
| 2 DTOs | 2 | | 0 | 2 |
| 3 Expenses | 5 | 47 | 0 | 4 |
| 4 ProjectBO | 3 | | 0 | 3 |
| 5 TaskBO | 17 | | 1 | 8 |
| Total | 48 (27 Distinct) | 47 | 1 | 26 (21 Distinct) |

**Table 1**
Model slices on zAppDev models

modeling slice (Slice 2).

## 4. Experimental evaluation

This section demonstrates how the model slicer extracts model slices on real LCP models.

As a proof of concept, we got 5 different BOs and 47 different UI-Form models generated by these business objects. We have created two different knowledge graphs which contain all

the information about the BOs and the Form models respectively. Then we selected each class iteratively from the BOs and gave this as an input class to the model slicer and checked the extracted model slices. In Fig. 5 we have presented the information that will be extracted by the model slicer. In this demo, the model slicer will try to extract a model slice related to the `Client` class.

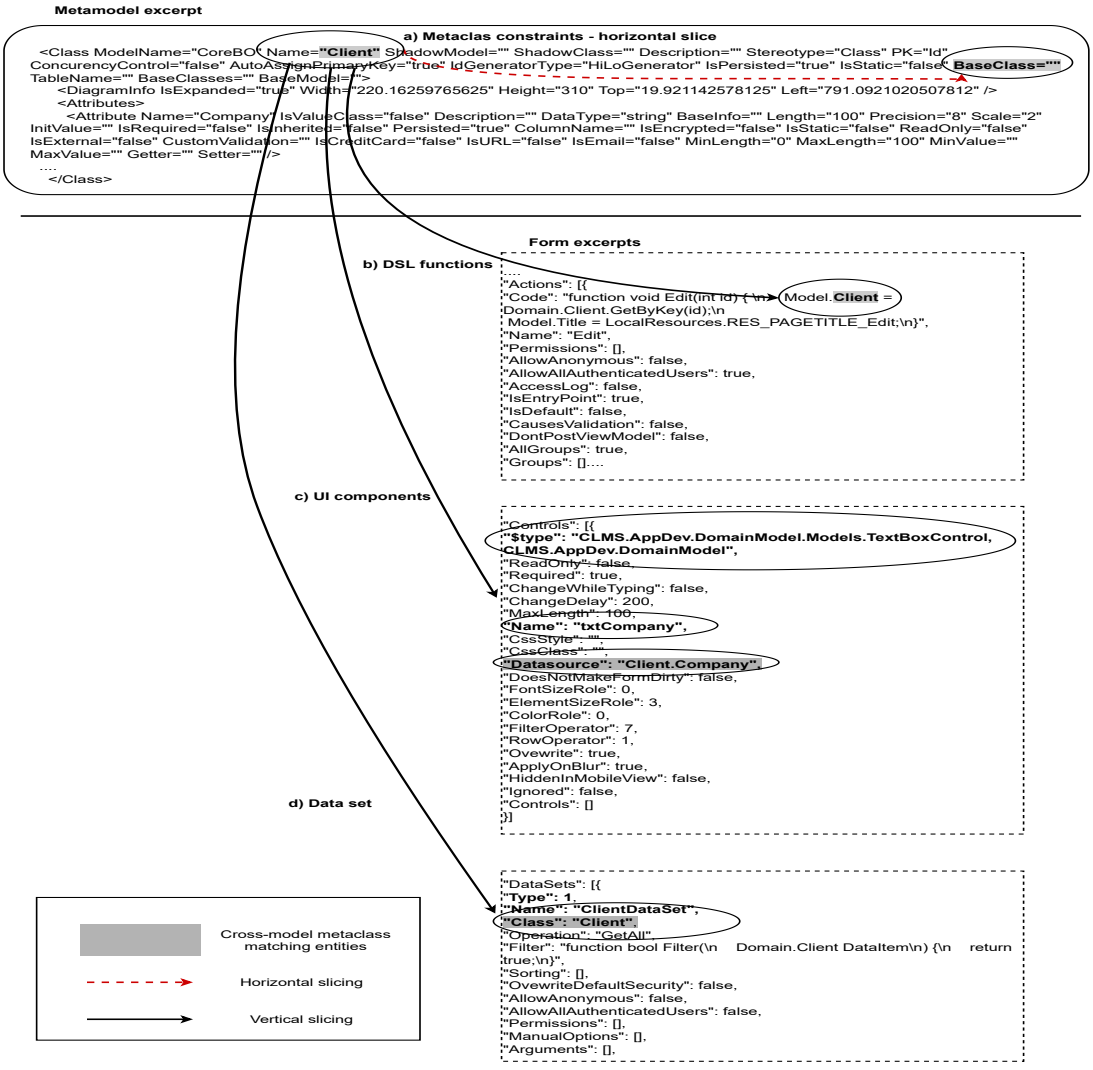


**Figure 5:** Model slicing in zAppDev

Initially, the model slicer will check within the BOs at the *baseClass* element to find any base class so it can create the horizontal slice ( a) ). This check is presented with the red dashed arrow. In this demo, the `Client` class hasn't any base class, and since at the time of writing this paper this is the only checked constraint for BO classes, will the model slicer not define any horizontal slice. Next, the model slicer will check for extracting the relevant Mamba functions from the zAppDev Form models ( b) ). The model slicer will check within the *Code* notation to find any related

function to the `Client` class. The found functions will be returned as part of the vertical slice.

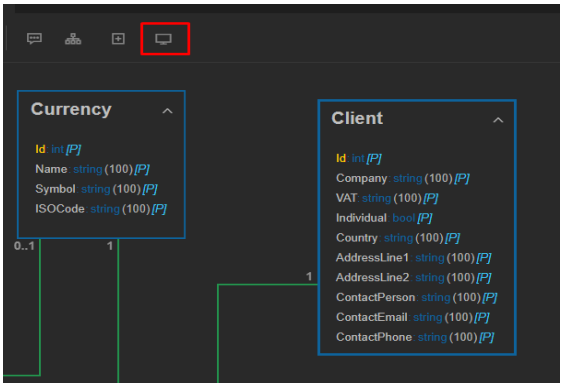


Figure 6: Triggering the model slicer service for the Client class

Next in c), the model slicer will extract the required information about the relevant UI components to the `Client` class. First, the model slicer will check if there is any *Datasource* notation that is related to the `Client` class, if there is any, then it gets the information about that Datasource related *Name* and *$type* notation. These three notations: *Name, Datasource,* and *$type* will be returned also as part of the vertical slice.

Finally, the model slicer checks for any related *dataset* to the `Client` class ( d) ). It checks the notation *Class* within the *Dataset* notation if it is Client. If there is a match, then will the model slicer get the respective *Name, Operation,* and *Filter* information. All this information will also be returned as part of the vertical slice.

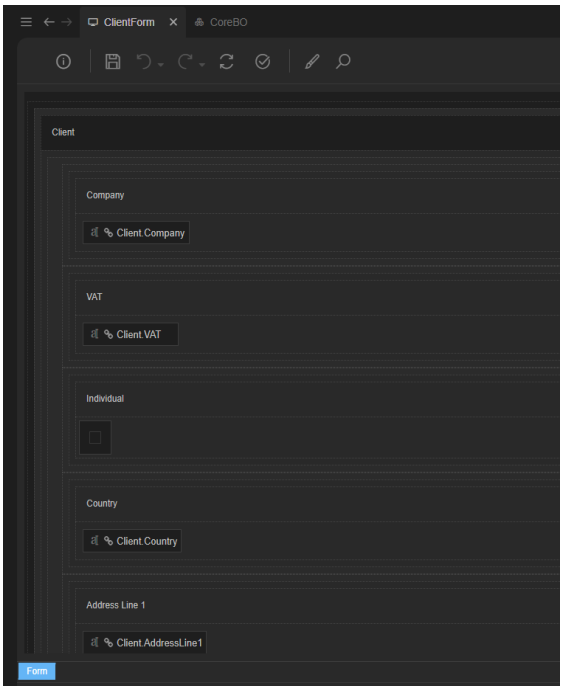


Figure 7: Integration of the Client model slice in zAppDev

In the end, all the information about the horizontal and vertical slices will be merged in a single JSON file and integrated into the zAppDev LCP. This model slice is extracted after selecting the `Client` BO class and clicking the "Create forms from Slice" button located in the menu bar as shown in Fig. 6. The model slice is integrated on the zAppDev LCP as a UI-Form model including the extracted information for the `Client` BO class. A snapshot of the integrated model slice on zAppDev is depicted in Fig. 7.

In Table 1 we have outlined the results of how many model slices could be extracted from the zAppDev models. For the study, we have used 5 different business object models and 47 different UI-Form models. We listed all the BO classes from all the 5 BO models and counted 27 distinct classes. We iterated through each of these BO classes and set each of them successively as an input class to the model slicer. Of all these BO classes only one class had a base class (PMOUser had as base class ApplicationUser) i.e., a horizontal slice. Also from the set of 27 distinct BO classes, the model slicer could extract 21 vertical slices which

means that 21 BO classes have at least one related entity on the UI-Form models. From this amount of data we got from zAppDev, we could conclude that **77.78%** of the BO classes are related at least to a base class or at least to one UI-Form model entity. This fact reveals the emerging need for a cross-language and cross-level model reuse approach on LCP that can be facilitated through the model slicer provided in this work.

The model slicer has been developed using Spring boot and is provided as a REST API for use in the zAppDev LCP. The source and the repositories containing the KG used for the evaluation are available on GitHub[5]

## 5. Related Work

Although to the best of our knowledge this is the first approach towards model ruse through slicing on cross-level and cross-language LCP models, inspired by program slicing approaches [7, 8], we will show some related works to model slicing.

Salay et al. [9] present an algorithm for megamodels slicing. The algorithm gets as input the megamodel and by using the traceability relation among the entities of the models that construct the megamodel it extracts the model slice. Our approach is search-based and not static based, i.e., it doesn't iterate through the cross-level model entities of a megamodel, it queries different repositories to find relevant matches to the input class.

Taenzer et al. [10] present a formal framework for creating model slicers that are capable to change incrementally a model slice after performing any change on it. Our approach is search-based and generates model slices from a single input class to support the LCP users during the modeling process. It is not required that we implement the update of model slicing since it can be updated directly by the LCP users based on their needs after being integrated.

Compare to the approaches that enable model slicing for a specific model type [11, 12, 13, 14, 15, 16] our approach checks for related entities among different models of different types and extract them as a model slice.

## 6. Conclusion and Future work

In this work, we have presented an approach that enables the reuse of cross-related models on an LCP through model slicing. The model slicing approach gets as input a data model class and queries the data model for any constraint-related class, and also the Form model repositories to get UI components and DSL functions. The current approach enables model slicing of zAppDev models but conceptually it can be used for any LCP.

In future work, we plan to fine-grain the model slicer by proving UI component layout information, slicing the class attributes, etc. We also aim to integrate the model slicer on a model recommendation approach so that after selecting a suggested data model class, all its cross-related model entities will be integrated automatically.

[5]https://github.com/iliriani/Model-slicer

## Acknowledgments

This project has received funding from the EU Horizon 2020 research and innovation programme under the Marie Skłodowska Curie grant agreement No 813884.

## References

[1] A. Sahay, A. Indamutsa, D. Di Ruscio, A. Pierantonio, Supporting the understanding and comparison of low-code development platforms, Proceedings - 46th Euromicro Conference on Software Engineering and Advanced Applications, SEAA 2020 (2020) 171–178. doi:10.1109/SEAA51224.2020.00036.

[2] R. Waszkowski, Low-code platform for automating business processes in manufacturing, IFAC-PapersOnLine 52 (2019) 376–381. URL: https://doi.org/10.1016/j.ifacol.2019.10.060. doi:10.1016/j.ifacol.2019.10.060.

[3] P. Vincent, K. Iijima, M. Driver, J. Wong, Y. Natis, Licensed for Distribution Magic Quadrant for Enterprise Low-Code Application Platforms (2019) 1–34. URL: https://www.gartner.com/doc/reprints?id=1-1ODOM46A{&}ct=190812{&}st=sb.

[4] D. Di Ruscio, D. Kolovos, J. de Lara, A. Pierantonio, M. Tisi, M. Wimmer, Low-code development and model-driven engineering: Two sides of the same coin?, Software and Systems Modeling (2022). URL: https://doi.org/10.1007/s10270-021-00970-2. doi:10.1007/s10270-021-00970-2.

[5] A. Bucaioni, A. Cicchetti, F. Ciccozzi, Modelling in low-code development: a multi-vocal systematic review, Software and Systems Modeling (2022). URL: https://doi.org/10.1007/s10270-021-00964-0. doi:10.1007/s10270-021-00964-0.

[6] O. Lassila, R. R. Swick, Resource description framework (RDF) model and syntax specification. World Wide Web Consortium Recommendation (1999). URL: http://www.w3.org/TR/REC-rdf-syntax.

[7] H. V. Nguyen, C. Kästner, T. N. Nguyen, Cross-language program slicing for dynamic web applications, in: Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2015, Association for Computing Machinery, New York, NY, USA, 2015, p. 369–380. URL: https://doi.org/10.1145/2786805.2786872. doi:10.1145/2786805.2786872.

[8] M. Weiser, Program slicing, in: Proceedings of the 5th International Conference on Software Engineering, ICSE '81, IEEE Press, 1981, p. 439–449.

[9] R. Salay, S. Kokaly, M. Chechik, T. S. E. Maibaum, Heterogeneous megamodel slicing for model evolution, in: ME@MoDELS, 2016.

[10] G. Taentzer, T. Kehrer, C. Pietsch, U. Kelter, A formal framework for incremental model slicing, in: FASE, 2018.

[11] R. Ahmadi, J. Dingel, E. Posse, Slicing uml-based models of real-time embedded systems, 2018. doi:10.1145/3239372.3239407.

[12] S. Sen, N. Moha, B. Baudry, J.-M. Jézéquel, Meta-model Pruning, in: ACM/IEEE 12th International Conference on Model Driven Engineering Languages and Systems (MODELS'09), Denver, Colorado, USA, United States, 2009. URL: https://hal.inria.fr/inria-00468514.

[13] A. Bergmayr, M. Wimmer, W. Retschitzegger, U. Zdun, Taking the pick out of the bunch - type-safe shrinking of metamodels, in: S. Kowalewski, B. Rumpe (Eds.), Software Engineering 2013, Gesellschaft für Informatik e.V., Bonn, 2013, pp. 85–98.

[14] H. Kagdi, J. I. Maletic, A. Sutton, Context-free slicing of uml class models, in: Proceedings of the 21st IEEE International Conference on Software Maintenance, ICSM '05, IEEE Computer Society, USA, 2005, p. 635–638. URL: https://doi.org/10.1109/ICSM.2005.34. doi:10.1109/ICSM.2005.34.

[15] P. Kelsen, Q. Ma, C. Glodt, Models within models: Taming model complexity using the sub-model lattice, in: D. Giannakopoulou, F. Orejas (Eds.), Fundamental Approaches to Software Engineering, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 171–185.

[16] A. Blouin, B. Combemale, B. Baudry, O. Beaudoux, Modeling model slicers, in: J. Whittle, T. Clark, T. Kühne (Eds.), Model Driven Engineering Languages and Systems, Springer Berlin Heidelberg, Berlin, Heidelberg, 2011, pp. 62–76.

# Preface of MeSS 2022

Federico Ciccozzi[1], Nicolas Ferry[2], Ludovico Iovino[3], Sébastien Mosser[4], Arnor Solberg[5] and Manuel Wimmer[6]

[1]*Malardalen University, Sweden*
[2]*University of Nice Côte d'Azur, France*
[3]*Gran Sasso Science Institute*
[4]*McMaster University, Canada*
[5]*Tellu AS, Norway*
[6]*JKU Linz, Austria*

The next generation IoT systems needs to perform distributed processing and coordinated behavior across IoT, edge and cloud infrastructures. Smart IoT Systems have the potential to flourish innovations in many application domains. For instance, the typical components of a smart city include infrastructure, transportation, intelligent energy consumption, health-care, and technology. These ingredients are what make the cities smart, efficient and optimized respect to the citizen and administration needs. The Internet of Things is an emerging paradigms that can contribute to make smart cities efficient and responsive.

On the one hand, Model-driven engineering (MDE) techniques can support the design, deployment, and operation of smart IoT systems. For instance, to manage abstractions in IoT systems definition and to provide means to automate some of the development and operation activities of IoT systems, e.g., domain specific modeling languages can provide a way to represent different aspects of systems leveraging a heterogeneous software and hardware IoT infrastructure and to generate part of the software to be deployed on it. On the other hand, the application of modeling techniques in the IoT poses new challenges for the MDE community.

Due to its cross-domain nature, this topic has a high potential for synergies (i) within the MDE community – model evolution, models@run.time, model transformations, multi-paradigm modeling and model validation for examples, and (ii) across the MDE and IoT communities. The International Workshop on MDE for Smart IoT Systems (MeSS) is one of the most accurate venues to offer researchers a dedicated forum to discuss fundamental as well as applied research that attempts to exploit model-driven techniques in the IoT domain. The program of this sixth edition (counting also the precursor workshop MDE4IoT) consisted of the 7 accepted extended abstract presentation. All the abstracts submitted to the workshop underwent through a peer-reviewing process and all the accepted abstracts will be invited for a special issue on JOT - The Journal of Object Technology. The workshop has been held has full day event of the Software Technologies: Applications and Foundations (STAF) conference on the July 5$^{th}$, 2022.

We would like to thank the STAF 2022 organization for giving us the opportunity to organize this workshop, especially to the workshop chairs Catherine Dubois (Ecole Nationale Supérieure d'Informatique pour l'Industrie et l'Entreprise, France) and Julien Cohen (Université de Nantes,

*STAF'22 Workshop Proceedings*

France), who were always very helpful and supportive. Many thanks to all those that submitted papers, and particularly to the presenters of the accepted papers. Last but not least, our thanks go to the reviewers and the members of the Program Committee, for their timely and accurate reviews and for their help in choosing and suggestions for improving the selected papers.

July 2022

Federico Ciccozzi
Nicolas Ferry
Ludovico Iovino
Sébastien Mosser
Arnor Solberg
Manuel Wimmer

**Program Committee**

| | |
|---|---|
| Ankica Barisic | I3S Laboraroty, France |
| Nicolas Belloir | IRISA, France |
| Martina De Sanctis | Gran Sasso Science Institute, Italy |
| Stefan Klikovits | National Institute of Informatics, Japan |
| Judith Michael | RWTH Aachen, Germany |
| Davide Di Ruscio | Università degli Studi dell'Aquila, Italy |
| Hui Song | SINTEF, Norway |
| Romina Spalazzese | Malmö University, Sweden |
| Matthias Tichy | Ulm University, Germany |
| Andreas Wortmann | University of Stuttgart, Germany |
| Wolfgang Kastner | TU Wien, Austria |

# Efficiently Engineering IoT Architecture Languages—An Experience Report (Poster)

Jörg Christian Kirchhof[1], Anno Kleiss[1], Judith Michael[1], Bernhard Rumpe[1] and Andreas Wortmann[2]

[1]*Software Engineering, RWTH Aachen University, Germany, https://se-rwth.de/*
[2]*Institute for Control Engineering of Machine Tools and Manufacturing Units (ISW), University of Stuttgart, Germany, https://www.isw.uni-stuttgart.de/*

**Keywords**
Internet of Things, Model-Driven Engineering, Architecture Description Language

## 1. Introduction

Engineering architecture description languages (ADLs) is complex. Yet, research and industry have developed over 120 ADLs for various purposes. Many of these languages share similar concepts and elements. Instead of creating a novel ADL for the Internet of Things (IoT) from scratch, we created the MontiThings IoT ADL through systematically reusing (parts of) various stand-alone languages. MontiThings is an ecosystem for the model-driven development, deployment, and analysis of IoT applications [1, 2, 3]. MontiThings can generate C++ code from its models and also provides the necessary scripts to containerize the code using Docker. In this paper, we detail the MontiThings ADL, its constituents, and language reuse mechanisms. Researchers and practitioners in the engineering of (IoT) ADLs can benefit from these insights to prevent creating another 120 ADLs from scratch.

## 2. Language Features

MontiThings is developed using the MontiCore Language Workbench [4], leveraging the library of composable modeling languages [5] offered by MontiCore (*cf.* Fig. 1).

**Type System** MontiThings offers primitive types similar to Java or C++ (`byte`, `int`, `long`, `float`, `boolean`, `char`, `String`). Additionally, collections types can group objects (Set, List, Map). To facilitate working with sensor data, MontiThings enables using SI units as primitive types (*e.g.,* `kg`, `dB`, `km`, `s`, `°C`, ...) by extending MontiCore's SI Unit language. MontiThings automatically converts the SI unit values if necessary (*e.g.,* `m/s` to `km/h`). Developers can specify their own types using class diagrams from MontiCore's CD4Analysis language.

*MESS'22: International workshop on MDE for Smart IoT Systems, July 04–08, 2022, Nantes, France*
0000-0002-8188-3647 (J. C. Kirchhof); 0000-0002-1378-3097 (A. Kleiss); 0000-0002-4999-2544 (J. Michael); 0000-0002-2147-1966 (B. Rumpe); 0000-0003-3534-253X (A. Wortmann)

**Figure 1:** Overview of MontiThings' language inheritance hierarchy.

**Behavior** MontiThings components offer three modes of computation: initialization, *i.e.,* behavior executed when starting the component, cyclic behavior, *i.e.,* behavior executed in time intervals, and event-based behavior, *i.e.,* behavior executed in response to receiving a message. The three modes can be combined within the same component but only one behavior can be executed simultaneously per component to prevent race conditions. The behavior of MontiThings components can be specified in four ways: Composed components instantiate and connect subcomponents to specify their own behavior. Atomic components do not have subcomponents, but specify their behavior through a programming language embedded in the model (using MontiCore's `MCCommonStatements` language), statecharts (from MontiCore's statechart language) or handwritten C++ code.

**Expressions and Literals** MontiThings' expressions are mainly built on top of the expressions provided by MontiCore out of the box, *i.e.,* assignments, mathematical, and boolean operators (*e.g.,* +, −, <=, or `||`). Additionally, MontiThings' reuses expressions from object constraint language (OCL) such as `@pre`, set expressions such as union or intersection of sets or checking if a set contains a given element. MontiCore's literal grammars offer MontiThings the capability to create numbers, strings, or boolean values. The `SIUnitLiterals` enable creating numbers with international system of units (SI) types such as `17 km/h`. To instantiate classes from class diagrams, MontiThings uses object diagrams using a JSON-like syntax.

## 3. Discussion, Recommendations, and Conclusion

MontiThings defines about 710 lines of grammar in 14 grammars and reuses 4371 lines of grammar from 46 grammars from the MontiCore project. We therefore assumed that language libraries can substantially reduce the effort required to implement new (IoT) ADLs. To test this assumption, we tried to re-implemented Ericsson's IoT ADL *Calvin* [6, 7, 8]. We were able to parse (slightly adapted) Calvin models using only existing language components adapted with about 50 lines of grammar to match Calvin's syntax. Another 15 lines of grammar were needed to add CalvinScript, which is not covered by MontiCore's language library. The main limitation to this approach is that deviating from the language library's infrastructure (*e.g.,* type system) may require a non-negligible amount of work. As re-implementing the Calvin's Python-like type system would have implied significant changes, we kept MontiThings' type system. Thus, the Calvin models needed to be adapted to contain explicit type names. Nevertheless, this experiment suggests that future IoT ADLs could reuse models of existing IoT ADLs with only limited effort. Providing parsers for other IoT ADLs can lower the entry barrier for modelers who are already invested in another IoT ADL.

The intensive reuse of existing language components enables us to reduce the effort for creating a new language. Language engineers can extend already mature and tested languages but the approach still allows to add needed domain-specific extensions. Thus, we recommend

language engineers to not build new IoT languages from scratch but utilize language libraries and consider the option of providing parsers for existing IoT ADLs.

## Source Code

MontiThings is available on GitHub: https://github.com/MontiCore/montithings

## Acknowledgments

Funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy—EXC 2023 Internet of Production—390621612. Website: https://www.iop.rwth-aachen.de

## References

[1] J. C. Kirchhof, B. Rumpe, D. Schmalzing, A. Wortmann, MontiThings: Model-driven Development and Deployment of Reliable IoT Applications, Journal of Systems and Software 183 (2022) 111087.

[2] J. C. Kirchhof, A. Kleiss, B. Rumpe, D. Schmalzing, P. Schneider, A. Wortmann, Model-Driven Self-Adaptive Deployment of Internet of Things Applications with Automated Modification Proposals, ACM Transactions on Internet of Things (In Press, 2022, DOI: https://doi.org/10.1145/3549553).

[3] J. C. Kirchhof, L. Malcher, B. Rumpe, Understanding and Improving Model-Driven IoT Systems through Accompanying Digital Twins, in: E. Tilevich, C. De Roover (Eds.), Proceedings of the 20th ACM SIGPLAN International Conference on Generative Programming: Concepts and Experiences (GPCE '21), ACM SIGPLAN, 2021, pp. 197–209.

[4] K. Hölldobler, O. Kautz, B. Rumpe, MontiCore Language Workbench and Library Handbook: Edition 2021, Aachener Informatik-Berichte, Software Engineering, Band 48, Shaker Verlag, 2021. URL: https://www.monticore.de/handbook.pdf.

[5] A. Butting, R. Eikermann, K. Hölldobler, N. Jansen, B. Rumpe, A. Wortmann, A Library of Literals, Expressions, Types, and Statements for Compositional Language Design, Special Issue dedicated to Martin Gogolla on his 65th Birthday, Journal of Object Technology 19 (2020) 3:1–16.

[6] O. Angelsmark, P. Persson, Requirement-Based Deployment of Applications in Calvin, in: I. Podnar Žarko, A. Broering, S. Soursos, M. Serrano (Eds.), Interoperability and Open-Source Solutions for the Internet of Things, Springer International Publishing, Cham, 2017, pp. 72–87.

[7] P. Persson, O. Angelsmark, Calvin – Merging Cloud and IoT, Procedia Computer Science 52 (2015) 210 – 217. 6th Int. Conf. on Ambient Systems, Networks and Technologies (ANT).

[8] P. Persson, O. Angelsmark, Kappa: Serverless iot deployment, in: Proceedings of the 2nd International Workshop on Serverless Computing, WoSC '17, Association for Computing Machinery, New York, NY, USA, 2017, pp. 16–21.

# Key-Value vs Graph-based data lakes for realizing Digital Twin systems (Poster)

Daniel Pérez-Porras, Paula Muñoz, Javier Troya and Antonio Vallecillo

*ITIS Software. University of Málaga, Spain*

**Keywords**
Digital twins, Data Lake, NoSQL databases, Graph databases

## 1. Introduction

A Digital Twin (DT) is a comprehensive digital representation of an actual system, service or product (the Physical Twin, PT), synchronized at a specified frequency and fidelity [1]. The digital twin includes the properties, condition and behavior of the physical entity through models and data, and is continuously updated with real-time data about the PT performance, maintenance, and health status throughout its entire lifetime [2]. The exchange of data between the digital and the physical twins takes place through bi-directional data connections. Additionally, a DT system can also comprise a set of services that permit exploiting the data exchanged by the two twins [3].

Engineering DT systems is challenging for many reasons, one of them being their complexity [4]. The problem we would like to address in this paper is how to implement the connections between the twins in an effective and efficient way. Usually, these connections are achieved through a Data Lake. As defined in [5], a data lake is "a flexible, scalable data storage and management system, which ingests and stores raw data from heterogeneous sources in their original format, and provides query processing and data analytics in an on-the-fly manner."

In a previous work [6] we defined a framework for the specification and deployment of DT systems. It uses UML models to specify the digital twins, and connects them through a Data Lake repository implemented in Redis (https://redis.io/), which provides the bi-directional communication infrastructure. This open-source lightweight in-memory data structure is optimized to deliver fast responses to a massive amount of petitions. Redis is a key-value database and supports various abstract data structures such as strings, lists, sets or maps (called 'hashes'). However, Redis does not easily allow complex queries: to retrieve hashes by the values of their fields, it is necessary to store additional records that include the field value and a reference to the hash key. This makes the database structure and contents dependent on the queries that need to be performed.

MESS@STAF 2022: International workshop on MDE for Smart IoT Systems, July 04–08, 2022, Nantes, France

daniperezporras@uma.es (D. Pérez-Porras); paulam@uma.es (P. Muñoz); jtroya@uma.es (J. Troya); av@uma.es (A. Vallecillo)
0000-0003-2939-5803 (P. Muñoz); 0000-0002-1314-9694 (J. Troya); 0000-0002-8139-9986 (A. Vallecillo)

Alternative solutions use time series databases, such as InfluxDB or TimescaleDB, and even define temporal models for handing them at a higher level of abstraction [7]. While very efficient for querying time-sensitive information, these solutions are not optimal for implementing more general queries, needed when efficient data analysis is required.

In this work we explore the use of Graph databases [8] to store and query the information handled in data lakes. Graph databases use graph structures to perform semantic queries; they store the information as nodes, edges, and properties. Examples of Graph databases include Neo4j, ArangoDB or OrientDB, to name a few. They all count with specialized query languages such as Gremlin, Cypher, SPARQL, or GraphQL.

We have implemented a data lake using Neo4j, and evaluated its performance and expressiveness against our previous implementation with Redis. As expected, Neo4j allows easy specification of more complex queries using its Cypher query language. The same queries in Redis require the addition of ad-hoc records with the corresponding key-value mappings if the queries were not contemplated beforehand in the Redis record structure. Interestingly, the additional queries forced by these new records introduce a performance penalty that makes Neo4j's response times better than those of Redis. Furthermore, the response times obtained for simple queries in Redis are not very different from those of Neo4j. All this makes Neo4j appear to be a better solution than Redis for realizing data lakes. The description of the tests carried out and their results are availble from https://github.com/atenearesearchgroup/dt-graph-database. As future work, we are defining a benchmark with different types of queries that will be used to compare implementations of data lakes using different technologies, including time series databases, too. We hope that our evaluation will help to shed some light on the advantages and limitations of each solution, and to identify situations where one type of solution outperforms the other.

## References


[1] Digital Twin Consortium, Glossary of digital twins, https://www.digitaltwinconsortium.org/glossary/index.htm, 2021.

[2] F. Bordeleau, B. Combemale, R. Eramo, M. van den Brand, M. Wimmer, Towards model-driven digital twin engineering: Current opportunities and future challenges, in: Proc. of ICSMM'20, volume 1262 of CCIS, Springer, 2020, pp. 43–54.

[3] F. Tao, H. Zhang, A. Liu, A. Y. C. Nee, Digital twin in industry: State-of-the-art, IEEE Trans. Ind. Informatics 15 (2019) 2405–2415. doi:10.1109/TII.2018.2873186.

[4] M. Grieves, J. Vickers, Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems, Springer, 2017, pp. 85–113.

[5] R. Hai, C. Quix, M. Jarke, Data lake concept and systems: a survey, CoRR abs/2106.09592 (2021). arXiv:2106.09592.

[6] P. Muñoz, J. Troya, A. Vallecillo, Using UML and OCL Models to Realize High-Level Digital Twins, in: Proc. of ModDiT2021@MODELS'21, IEEE, 2021, pp. 212–220.

[7] A. Mazak, S. Wolny, A. Gómez, J. Cabot, M. Wimmer, G. Kappel, Temporal models on time series databases, J. Object Technol. 19 (2020) 3:1–15.

[8] I. Robinson, J. Webber, E. Eifrem, Graph Databases: New Opportunities for Connected Data, 2 ed., O'Reilly Media, Inc., 2015.

# Modeling Linked Open Data (Poster)

Adiel Tuyishime[1], Javier Luis Cánovas Izquierdo[2], Maria Teresa Rossi[1] and Martina De Sanctis[1]

[1]Gran Sasso Science Institute (GSSI), Viale Francesco Crispi 7, L'Aquila, 67100, Italy
[2]IN3 – UOC, Barcelona, Spain

## 1. Introduction

Nowadays, many entities (e.g., business companies, government institutions) move toward sharing their data online for reuse [1], leading to the incremental growing of *Open Data* [2]. In public administrations, Open Data increases transparency [3] and allows citizens access to valuable information. Usually, the information provided by a unique dataset coming from Open Data sources are very limited as they are not integrated with other data sources. We propose to leverage the concept of Linked Open Data (LOD) [2] which, besides the benefits of Open Data, exploits Linked Data best practices for publishing and connecting structured data on the Web [4]. Thus, LOD supports knowledge sharing and information enrichment by adding links to both properties and values of a data object. In the literature, traditional modeling approaches exploit semantic Web features to model LOD. For instance, Alaoui *et al.* [5] propose data modeling in the context of enterprise applications development in a semantic-oriented perspective by using RDF and OWL ontologies. Meanwhile, Jamil *et al.* [6] present an approach for the semantic modeling of events using the case study of refugee registration and repatriation. Although ontologies offer powerful solutions, they are specialized in conceptual modeling and inferring new knowledge. This does not facilitate the development of various software artifacts (e.g., automatically generated code, APIs or libraries). Moreover, using ontologies implies good knowledge of the domain and the understanding of the used technologies, requiring a considerable effort and leading to a steep learning curve. For this reason, we propose a novel approach to model LOD based on Model-Driven Engineering (MDE) as it presents a wide range of tools and techniques supporting not only conceptual modeling but also the development and generation of different software artifacts, easy integration, and non-functional requirements analysis. MDE has been successfully applied in different domains and has proven to be a promising approach to follow due to the benefits it offers (e.g., code generation or model transformation). In addition, MDE enables the linking between models through the exploitation of *weaving models*. Thus, weaving models can be exploited in the scenario of LOD to integrate different models (e.g., Open Data models), while maintaining the separation of concerns and avoiding the construction of large and monolithic models for LOD, which could be difficult to handle, maintain and reuse [7].

## 2. Modeling LOD exploiting MDE Techniques

In LOD, data elements are linked to each other in such a way that they can be effectively navigated to provide additional context. In a smart city context, LOD together with

*MeSS'22: International workshop on MDE for Smart IoT Systems, July 04–08, 2022, Nantes, France*
adiel.tuyishime@gssi.it (A. Tuyishime); jcanovasi@uoc.edu (J. Cánovas Izquierdo); mariateresa.rossi@gssi.it (M. T. Rossi); martina.desanctis@gssi.it (M. De Sanctis)
0000-0002-2326-1700 (J. Cánovas Izquierdo); 0000-0003-0273-7324 (M. T. Rossi); 0000-0002-9417-660X (M. De Sanctis)

data provided by IoT devices could contribute to the creation of new knowledge about a city. For instance, multiple information are provided by IoT devices installed around a smart city, such as air quality, traffic, noise, etc. In this context, anybody interested in pollution, could not only see the information about pollution but could also navigate towards other factors that can have an impact on pollution, such as road traffic, or that can be impacted by it, such as health, among others. In Figure 1 we report an example as an abstract representation of our proposal to model LOD exploiting MDE techniques. On the left-side (a) we report an example of LOD in which we have three different domains, namely *Population*, *Mobility*, and *Pollution* that could be expressed as data objects defined by different classes and relationships. In the figure, we report at least one class for each domain representing how the data is linked to each other to enable information sharing. For instance, in the *Pollution* domain we have a class *AirQuality* which is linked with a class *Health* in the *Population* domain with a relationship (*has impact on*) that indicates the impact *AirQuality* has on *Health*. On the right-side (b) of the figure, we report how the example can be modeled by using MDE techniques, by means of the typical structure of data objects in the three domains through metamodeling. As can be seen, we propose to use three models (i.e., Population Model, Pollution Model and Mobility Model) which are conforming to their corresponding metamodels (i.e., Population metamodel, Pollution metamodel and Mobility metamodel). To establish the link between these three models a Weaving Model is introduced. This way, we integrate different modeling domains that contributes to the megamodelling by enabling an ecosystem of models.

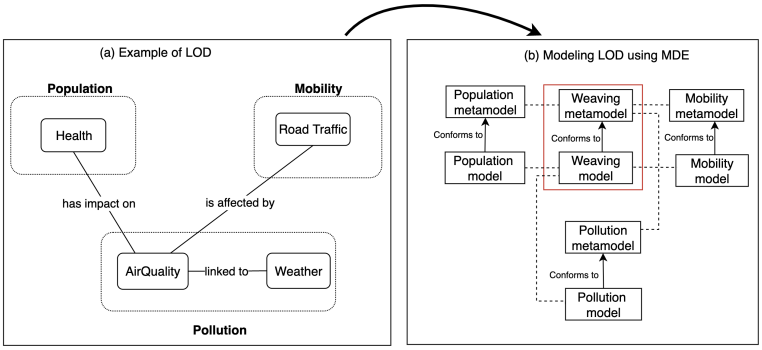


**Figure 1:** Example of modeling LOD exploiting MDE Techniques.

## References


[1] E. Kalampokis, E. Tambouris, K. Tarabanis, A classification scheme for open government data: Towards linking decentralized data, International Journal of Web Engineering and Technology 6 (2011) 266–285.

[2] F. Bauer, M. Kaltenböck, Linked Open Data: The Essentials - A Quick Start Guide for Decision Makers, edition mono/monochrom, Vienna, Austria, 2012. ISBN: 978-3-902796-05-9.

[3] P. P. M. J. Anneke Zuiderwijk, Mila Gascó, Special issue on transparency and open data policies: Guest editors'introduction, volume 9, 2014.

[4] C. Bizer, T. Heath, T. Berners-Lee, Linked data - the story so far, Int. J. Semantic Web Inf. Syst. 5 (2009) 1–22.

[5] K. Alaoui, M. Bahaj, Semantic Oriented Data Modeling for Enterprise Application Engineering Using Semantic Web Languages, International Journal of Advanced Trends in Computer Science and Engineering 9 (2020).

[6] S. Jamil, S. Noor, I. Ahmed, N. Gohar, Fouzia, Semantic modeling of events using linked open data, Intelligent Automation Soft Computing 29 (2021) 511–524. doi:10.32604/iasc.2021.017770.

[7] D. Di Ruscio, Specification of Model Transformation and Weaving in Model Driven Engineering, Universitá di L'Aquila, PhD Thesis, 2007.

# Model-Driven Development of Digital Twins for Supervision and Simulation of Sensor-and-Actuator Networks (Extended Abstract)

Gaël Pichot[1], Jérôme Rocheteau[1,2] and Christian Attiogbé[2]

[1]Icam Ouest, 35 avenue du champ de manœuvres, 44470 Carquefou, France
[2]LS2N, 2 chemin de la Houssinière, BP 92208, 44322 Nantes Cedex 3, France

Digital Twins are defined as virtual counterparts of physical objects, processes or systems that make it possible to supervise, control and simulate these objects, processes or systems. They are made of sets of software components interacting with physical devices. Unfortunately, there is currently no real standard neither on the very definition of Digital Twins nor on their method of development. This leads to empirical development of Digital Twin applications. In a previous work we had, firslty, proposed a meta-model able to represent Sensor-and-Actuator Networks and had, secondly, defined a model transformation that generates IoT applications for the supervision and the control of such networks. In this paper we extend this previous work: Firstly, the Sensor-and-Actuator-Networks meta-model is improved in order to better represent computations within such networks. This allows us to declare total or partial simulations of such networks. Secondly, we extend the model transformation in order to generate data-driven simulation programs that can interoperate with the IoT application for supervision-and-control of a given Sensor-and-Actuator Network. This can be seen as a digital twin of a Sensor-and-Actuator Network as it makes it possible to control the actuators by the means of computation units that provide their setpoints from the own measurements of the sensors of this network. Finally, we model and develop a digital twin of a connected plastic extruder.

**Extension of the SAN meta-model** The meta-model[1] of Sensor-and-Actuator Networks, or SAN for short, has been designed (1) to represent sensors, actuators and computation units of sensor-and-actuator networks and (2) to encompass concepts from other meta-models like SensorML, SOSA, SSN and SMM. The latter address the issue of modeling sensor networks. It merely allows us to declare elements of SAN models by specifying what kind of data is exchanged with such networks. It doesn't allow us to define these elements i.e. how data is processed. Moreover, a model-to-text transformation[2] makes it possible to obtain an IoT-based supervision-control-and-data-acquisition application, or SCADA for short, from SAN models by generating a set of HTTP requests that configure a previously developed IoT platform called

*STAF 2022 Workshop - International Workshop on MDE for Smart IoT Systems*


[1]The SAN meta-model is currently provided by the means of the following Xtext grammar: https://git.icam.fr/jerome.rocheteau/san/-/blob/main/fr.icam.san/src/fr/icam/san/SensorActuatorNetwork.xtext.

[2]This model transformation generates the following Java program that benefits from the EMIT Java HTTP client library: https://git.icam.fr/jerome.rocheteau/recyplast/-/blob/main/src/test/java/fr/icam/recyplast/Extruder.java.

EMIT[3] once executed.

We extend this SAN meta-model in two ways. Firstly, we introduce another kind of *process trigger*s, called *task*s, that represent computations repeatedly launched at fixed-rate. This addition completes the SAN modelling of *process trigger*s; *process*es represent computations and *trigger*s represent when and how the latter are computed. More than anything, *task*s make it possible to represent computations of simulations programs that continously listen to exchanged data into networks and that can send valuable data back at defined time spans. Secondly, we introduce another way to bind values to *process parameter*s while declaring *trigger*s. In fact, *process*es are (parametric) components with formal parameters and *trigger*s are (parameterized) components with real parameters i.e. values. Values can be provided either as raw *value*s (i.e. string, integers, floating point numbers, etc) as previously in SAN modelling, or as *measurement*s i.e. references to given data channels. It then makes it possible to bind *process parameter*s with values that can vary during the time but which type and source are ensured by the model compliance with the system.

**Generation of simulation programs from SAN models** We extend the model transformation from SAN models to *task*s by the means of the Xtext code generation feature. It generates a Java abstract class per *task*[4]. The latter consists in a Java class that implements both interfaces Runnable and MQTTCallback. The Runnable interface is used to process the collected data and fire the output results. The MQTTCallback interface is used to collect the data. In fact, the model transformation associates a Java variable per *parameter* of the *trigger process*. This Java variable is initialized according to the corresponding *argument* of its associated *parameter*: it is initialized once with the specified value if this *argument* consists of a *value*; it is initialized each time a message arrived on the topic that corresponds to the *measurement* if this *argument* consists of a *measurement*. The way data is processed and results are computed is delegated to the inherited classes of the Java abstract class that correspond to the *task*. For instance, this makes it possible to implement several simulation algorithms from a single *task*.

This model transformation extension leads to a « 2-level application ». The first level is defined by the SCADA application whereas the second level is defined by the data-driven simulation programs. In fact, such simulation programs are based on and driven by the data that is exchanged and collected between sensors, actuators and computational units within the network.

**Development of the digital twin of a plastic extruder** We had starting updating the SAN meta-model by modelling the plastic extruder used for the Recyplast-Demo project at the Icam school factory[5]. An extruder is a machine that transforms input plastic balls into output plastic products of various shapes depending on the trailing equipment: the die. It consists (1) in feeding the machine with input material, (2) in pushing the material along the extruder by

[3]The EMIT platform consists of web-services that allows us to edit and manage MQTT clients and callbacks. Its repository is publicly available at https://github.com/jeromerocheteau/emit.

[4]The generated Java program for the *task* declared in the following extruder SAN model can be found at: https://git.icam.fr/jerome.rocheteau/recyplast/-/blob/main/src/test/java/fr/icam/recyplast/ExtruderSimulator.java.

[5]The SAN model of the extruder is available: https://git.icam.fr/jerome.rocheteau/recyplast/-/raw/main/src/main/resources/extruder.san.

the means of a motorized endless screw, (3) in heating this material along the extruder by the means of several heating barrels equipped with a heating resistance actuator and a temperature sensor such that the input material changes from solid state to liquid state and (4) in shaping this output material thanks by the means of the die. The latter has a pressure and a temperature sensor that monitor the output material. The screw motor speed and the barrels temperatures are servo measurements i.e. setpoints are automatically adjusted according to their respective measurements. A torque measurement of the screw motor is also provided by a dedicated sensor. In practice, the motor speed is surprisingly set to a constant value, even if it could be adjust at runtime, as well as the setpoints of the heating barrels. Moreover, the setpoints of the heating barrels are all set to the same value, even if it could be possible to provide different setpoints to each heating barrel. These setpoints of motor speed and barrel temperature are mainly specified by the plastic material providers. The SAN model transformation allows us to configure the EMIT platform such that the result corresponds to the SCADA application of the plastic extruder[6].

The Recyplast-Demo project aims at using recycled plastic in extrusion process. This leads to feeding the extruder with material which properties can vary during the time whereas properties of current plastic material used in extrusion doesn't. The underlying challenge then consists in ensuring a constant quality of the output plastic products despite the variable properties of the input materials. The investigated solution corresponds to slave the screw motor speed setpoint and the heating barrels temperature setpoints from the material pressure and temperature measurements of the die. In fact, the output products quality are correlated to the material pressure and temperature measurements of the die. This leads us to develop a program that monitors the different measurements at runtime – screw motor speed and torque, heating barrels temperature, material pressure and temperature – and that provides suggestions for the screw motor speed and heating barrels temperature setpoints. Adding such a program to the SCADA application directly corresponds to a Digital Twin, or DT for short, of the extruder. DTs differ from SCADA applications in the way that a DT is not just defined by a bidirectionnal exchange of data between the DT and its physical system: it also includes algorithms, like simulations such that it can be seen as the next level simulation based on collected data which is obviously the case in this context.

In this work[7], we prove the feasability of developing IoT-based full Digital Twins from models of Sensor-and-Actuator Networks i.e. enabling both the Supervision-Control-And-Data-Acquisition feature and the data-driven simulation one by the means of a IoT application. This has been achieved by a tooling based on the Xtext support for language engineering that relies on the Ecore meta-model and the EMF framework. Moreover, the way data-driven simulation programs are generated offers a valuable balance between the support provided by the code generation and the flexibility of finely customizing the simulation algorithms.

[6] A bridge from OPC-UA to MQTT has been developed in front of the extruder PLC in order to ensure communications between the extruder PLC and the EMIT platform.

[7] This work has received fundings under the Regional Council of Pays-de-la-Loire and the European Regional Development Fund (ERDF) programs for the Recyplast-Demo project under the grant agreement n°2021-15465.

# Speak Well or Be Still: Solving Conversational AI with Weighted Attribute Grammars (Poster)

Vadim Zaytsev

*Formal Methods and Tools, University of Twente, The Netherlands*

There is a growing need to specify models of possible conversations with non-human entities. Such models have a difficult task to set the bar for correctness yet tolerate conversations with only partial conformance to it, and accommodate computations that non-human entities perform during the conversation on several possibly independent emergent models with unrelated flows of information in them. As it turns out, this is possible to specify formally in a fairly concise way with weighted attribute grammars [6]. In this paper, a variant of those is presented, called **WAGIoT**, that combines the power of analytic, generative, attribute, weighted and probabilistic grammars in one DSML.

Conversation entities are an essential part of smart IoT systems. They come in many forms, largely synonymous: conversational AI, virtual digital assistants, interactive agents, smart bots, chatbots, etc. In the presence of the Turing test [17] as the ultimate goal of artificial intelligence, conversation programs became an iconic example of an AI system very early. Notable milestones shaping and eventually commoditising the trend, were ELIZA [19], PARRY [4], A.L.I.C.E. [18], Wolfram Alpha [20], IBM Watson [9], Siri [3], Cortana [12], Alexa [2], Google Assistant [7], Alisa [22] and Bixby [15]. Conversation agents are needed in many areas from smart homes to game design. We refer to the excellent recent overview of this research direction by Adamopoulou and Moussiades [1] and focus now on the relation between the linguistic component and the operational logic of the conversation entity.

Looking at the problem linguistically, the conversation can be encoded as an automaton with states representing internal states of the conversation component, and transitions annotated with inputs (coming from the user or an edge device) and outputs (being sent to the user or to actuator). In the computation theory such automata are called Mealy machines [11] if they are deterministic and have a finite number of states. Both limitations are unfortunately too crippling, so all the substantial body of research on Mealy machines cannot be applied directly. What might be theoretically more feasible, is an input/output extension of pushdown transition systems or process rewrite systems [10], that can handle both infinite/uncountable number of states or transitions, have enough memory to handle complex tasks intelligently, and still represent a strict subclass of Turing machines such that reachability is decidable.

The lack of available theories pushed people to consider hybrid setups. For instance, actual derivations can be handled by a grammar, but the grammar rules that get applied, must follow

*MeSS'22: International Workshop on MDE for Smart IoT Systems, 5 July 2022, Nantes, France*
vadim@grammarware.net (V. Zaytsev)
https://grammarware.net/ (V. Zaytsev)
0000-0001-7764-4224 (V. Zaytsev)

an associated coloured graph [21] or a path in a Petri net [5]. The contemporary systems used in the industry, like RiveScript [13], are also hybrid in nature: the conversation is specified as request-response pairs (akin to the event-based paradigm in grammarware [23]) with ad hoc computations on global data. Powerful off-the-shelf AI packages like Dialogflow [8], lifting natural language understanding tasks to intents and their fulfillment, and using a smart knowledge connector to incorporate existing data, allow these computations to be arbitrarily complex. For example, Salvi et al build Jamura [14], a conversational smart home assistant using Telegram API for collecting user input, Dialoglflow Agents for processing it and the ThingSpeak platform to connect to the server and the clients.

One of the heterogeneous approaches is binary context-free grammars [16] which combine two generative grammars into one mathematical object. Essentially our proposal is to combine attribute grammars (that can propagate and share data in a very controlled fashion), weighted grammars (that provide highly controllable nondeterminism), analytic grammars (for parsing user inputs) and generative grammars (for producing answers).

Our domain-specific language for conversation models is called WAGIoT, and is in fact a tailored implementation of Weighted Attribute Grammars [6]. It allows conversation designers to specify interactions with actuators, sensors and users. A typical WAGIoT model can be seen on Figure 1. There are many model transformations that infer enough information from this provided model in order to make it formally executable. For instance:

- a synthesized attribute $\dot{n}$ in `getname` and an inherited attribute $\dot{n}$ in `time` have the same name; the given grammar is represented as a directed graph of nonterminals, in which WAGIoT normaliser finds the shortest path between these two places and makes sure the information is properly propagated;
- analytic nonterminals such as `greet` and `stop` have multiple branches, which get assigned counters; these are increased each time a branch is chosen, and this information is always available elsewhere—wherever there is a need to build a model of the human user's behaviour/speech patterns;
- all the unassigned weights are calculated based on available information; for instance, `time` branch in rule 10 gets a probability $[1 : 1 + w]$ because $w$ is a dynamic value, so the only option is to add the default 1; if rule 11 had the weight $[1 : 3]$ instead of $[w]$, then rule 10 would get a weight $[2 : 3]$ to compensate for the missing static part;
- types of attributes are inferred: $n$ is determined to be a string due to a built-in nonterminal `Id`, and $w$ is an integer because it is used as weight;
- code blocks between ⟦ and ⟧ are expected to conform to a special interface with methods responding to their use in generative and analytic contexts, and otherwise are lowered to strings, which is what happens in rules 10–11.

Our preliminary experiments show that these transformations hide details that otherwise overwhelm those who are supposed to write such models. Although we claim that the simplicity of RiveScript [13] pushes its users to combine it with informal hacks, exposing too much of the complexity at once is just as damaging. More work is required to find the best balance, as well as to pursue the obvious enhancements like replacing fixed terminals with signals from a NLP model. Our ongoing endeavours are made public at https://github.com/grammarware/wagl.

```
                S  ←  setup activity* stop.                                          (1)
            setup  ←  greet? getname.                                                (2)
            greet  ←  'hello'.                                                       (3)
            greet  ←  'good morning'.                                                (4)
            greet  →  'greetings, human!' 'what is your name?'.                      (5)
          getname  ←  'I am' ṇ := Id.                                                (6)
          getname  →  'nice to meet you,' ṇ.                                         (7)
         activity  ←  time | ⋯.                                                      (8)
             time  ←  'what time is it?'.                                            (9)
             time  →  'it is' ⟦DateTime.Now⟧ ',' ṅ ▶ w := 5.                         (10)
             time  →  [w] 'it is' ⟦DateTime.Now.Hour⟧ 'o′clock' ▶ w := w − 1.        (11)
             stop  ←  'stop' | 'off'.                                                (12)
```

**Figure 1:** A simplified WAGIoT grammar for a holistic example showcasing the notation. All ←-rules are analytic, all →-rules are generative. Having multiple analytic rules means that any of them can be applied—as in (3–4). Having multiple generative rules means one of them is chosen at random with probabilities following rule weights—as in (10–11). Rule (1) contains a regular right part (a Kleene star). Rule (12) contains a shorthand notation for multiple rules of the same kind for the same nonterminal (a Backus bar). The ellipsis in rule (8) designates that this is but a fragment of a larger grammar. := in rules (6) and (10–11) denotes attribute assignment. $\dot{x}$ are inherited attributes (propagated downwards in the tree); $\underset{\cdot}{x}$ are synthesized attributes (propagated upwards). Weight is printed in square brackets.

# References


[1] E. Adamopoulou, L. Moussiades, An Overview of Chatbot Technology, in: AIAI, Springer, 2020, pp. 373–383.
[2] Amazon, Amazon Alexa Voice AI, https://developer.amazon.com/en-US/alexa, 2014.
[3] Apple, Siri, https://www.apple.com/siri/, 2011.
[4] K. M. Colby, S. Weber, F. D. Hilf, Artificial Paranoia, Artificial Intelligence 2 (1971) 1–25.
[5] J. Dassow, S. Turaev, $k$-Petri Net Controlled Grammars, in: LATA, LNCS 5196, Springer, 2008, pp. 209–220.
[6] M. Gerhold, V. Zaytsev, Towards Weighted Attribute Grammars, Under review, 2022.
[7] Google, Google Assistant, your own personal Google, https://assistant.google.com, 2016.
[8] Google, Google Cloud: Dialogflow, https://cloud.google.com/dialogflow/docs/, 2017.
[9] IBM, IBM Watson, https://www.ibm.com/watson, 2011.
[10] R. Mayr, Process Rewrite Systems, Information and Computation 156 (2000) 264–286.
[11] G. H. Mealy, A Method for Synthesizing Sequential Circuits, Bell System Tech Journal 34 (1955) 1045–1079.
[12] Microsoft, Your Personal Productivity Assistant, https://www.microsoft.com/en-us/cortana, 2014.
[13] N. Petherbridge, RiveScript, https://www.rivescript.com, 2018.
[14] S. Salvi, Geetha V, Sowmya Kamath S, Jamura: A Conversational Smart Home Assistant Built on Telegram and Google Dialogflow, in: TENCON, IEEE, 2019, pp. 1564–1571.
[15] Samsung, Bixby | Apps & Services, http://bixby.samsung.com/, 2017.
[16] S. Turaev, R. Abdulghafor, A. A. Alwan, A. A. Almisreb, Y. Gulzar, Binary Context-Free Grammars (2020).
[17] A. M. Turing, Computing Machinery and Intelligence, Mind 59 (1950) 433–460.
[18] R. S. Wallace, The Anatomy of A.L.I.C.E., in: Parsing the Turing Test, Springer, 2009, pp. 181–210.
[19] J. Weizenbaum, ELIZA — A Computer Program for the Study of Natural Language Communication between Man and Machine, CACM 9 (1966) 36–45. doi:10.1145/365153.365168.
[20] Wolfram Research, Wolfram|Alpha: Computational Intelligence, https://www.wolframalpha.com, 2009.
[21] D. Wood, A Note on Bicolored Digraph Grammar Systems, IJCM 3 (1973) 301–308.
[22] Yandex, Alisa — Voice Assistant from the Yandex company, https://yandex.ru/alice, 2017.
[23] V. Zaytsev, Event-Based Parsing, in: REBLS, 2019. doi:10.1145/3358503.3361275.

# Smart Home Model Verification with AnimUML (Poster)

Frédéric Jouault[1], Ciprian Teodorov[2] and Matthias Brun[1]

[1]*ESEO, Angers, France*
[2]*Lab-STICC, ENSTA Bretagne Brest, France*

**Keywords**
UML, model verification, smart home,

Model verification techniques, such as model checking, generally require relatively advanced expertise. They are therefore typically used in applications where their usefulness is especially appreciated, if not necessary, and can offset their costs. Critical system design has, for instance, been one of the main consumers of such techniques. They could however bring benefits to many other domains. Development times can be shortened by the drastically reduced number of mistakes in verified design models. Moreover, they can help reduce the number of bugs remaining in shipped products. Lowering barriers to entry for the application of these techniques should therefore have a significant impact. In this work, we show how model checking can be applied to UML models in the smart home context. The models were created with AnimUML, which makes them markedly easier to create than with traditional tools. Furthermore, this tool's direct model analysis support at the UML level makes it relatively simple to verify properties. It was able to detect several corner case issues, which would have been much harder to detect, and especially diagnose, with testing only.

Besides being time consuming, testing reaches its fundamental limits, checking *what the system should not do*. Besides, in the home automation context, the problem is even more complex due to the distributed nature of the problem [1]. The situation can certainly be improved by using formal verification approaches, like model-checking. These techniques, naturally geared towards distributed systems, allow the verification of properties expressing *what the system should do*, which naturally completes the correctness specification of a system. During the last decade, tremendous progress was achieved on this axis [2, 3], however most of the proposed approaches and tools require a high-degree of sophistication from the home automation designer. Moreover, the marketing target of home automation solutions, like Google Smart Home[1], is wide and targets non-expert users. Nevertheless, the modeling community started a push towards lively verification environments [4], which enables seamless user interaction during the design and debugging process. The AnimUML environment [5, 6, 7] pushes the frontiers of this approach by allowing not only early debugging of high-level specifications, but also

*STAF 2022 Workshop MESS'22: International workshop on MDE for Smart IoT Systems, July 5, 2022, Nantes, France*
frederic.jouault@eseo.fr (F. Jouault); ciprian.teodorov@ensta-bretagne.fr (C. Teodorov); matthias.brun@eseo.fr (M. Brun)
0000-0002-2395-9623 (F. Jouault); 0000-0002-0722-5857 (C. Teodorov)

[1]https://developers.google.com/assistant/smarthome/overview

formal verification of partial (under development) specifications. This work presents a case study[2] where a LYWSD03MMC[3] temperature and humidity sensor is integrated with the Google Smart Home automation platform using local fulfillment[4]. We leverage the capabilities offered by AnimUML both to better understand the overall architecture, through lively user-model interactions, and to allow for the verification of non-trivial properties. The two main limitations of the approach are the following. 1) Although the tooling significantly helps, it is still necessary to learn to use AnimUML. 2) In addition to modeling the app, the designer must also model the behavior of the Smart Home API, and of the device. Regarding the first issue, we plan to keep improving the tool to make it easier to learn and use. As for the second issue, ideally manufacturers should provide such models. Moreover, we already provide a reusable Google Smart Home API model as part of our case study.

## References


[1] A. Demeure, S. Caffiau, E. Elias, C. Roux, Building and Using Home Automation Systems: A Field Study, in: ISEUD 2015, Madrid, Spain, 2015. doi:10.1007/978-3-319-18425-8_9.

[2] A. Souri, M. Norouzi, A state-of-the-art survey on formal verification of the internet of things applications, Journal of Service Science Research 11 (2019) 47–67. doi:10.1007/s12927-019-0003-8.

[3] C.-J. M. Liang, L. Bu, Z. Li, J. Zhang, S. Han, B. F. Karlsson, D. Zhang, F. Zhao, Systematically debugging iot control system correctness for building automation, in: Proceedings of the 3rd ACM International Conference on Systems for Energy-Efficient Built Environments, BuildSys '16, Association for Computing Machinery, 2016, p. 133–142. doi:10.1145/2993422.2993426.

[4] D. Ingalls, The lively kernel: Just for fun, let's take javascript seriously, in: Proceedings of the 2008 Symposium on Dynamic Languages, DLS '08, Association for Computing Machinery, 2008. doi:10.1145/1408681.1408690.

[5] F. Jouault, V. Besnard, T. L. Calvar, C. Teodorov, M. Brun, J. Delatour, Designing, animating, and verifying partial uml models, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS '20, Association for Computing Machinery, 2020, p. 211–217. doi:10.1145/3365438.3410967.

[6] F. Jouault, V. Sebille, V. Besnard, T. L. Calvar, C. Teodorov, M. Brun, J. Delatour, Animuml as a uml modeling and verification teaching tool, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 615–619. doi:10.1109/MODELS-C53483.2021.00094.

[7] M. Pasquier, F. Jouault, M. Brun, J. Pérochon, Evaluating tool support for embedded operating system security: An experience feedback, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, Association for Computing Machinery, 2020. doi:10.1145/3417990.3420048.


[2] Case study material is available on GitHub at https://github.com/fjouault/SmartHomeCaseStudy.

[3] https://esphome.io/components/sensor/xiaomi_ble.html#lywsd03mmc

[4] https://developers.google.com/assistant/smarthome/concepts/local

# SimulateIoT-Federations: Domain Specific Language for designing and executing IoT simulation environments with Fog and Fog-Cloud federations (Poster)

José A. Barriga[1], Pedro J. Clemente[1]

[1]*Quercus Software Engineering Group. http://quercusseg.unex.es. Department of Computer and Telematic Systems Engineering. University of Extremadura, Av. Universidad s/n, 10003, Cáceres (Spain)*

The Internet of Things (IoT) is being applied to areas such as smart-cities, home environment, agriculture, industry, etc. These application areas are very different from each other, thus requiring IoT systems with specific performance in terms of quality of service (QoS), delay, bandwidth or energy consumption. For instance, new IoT paradigms such as the Internet of Vehicles (IoV), or classic IoT systems such as healthcare, are latency sensitive application areas that need ultra-low latency infrastructure to make the application of IoT feasible. On the other hand, applications such as video analytics, or massively multiplayer online gaming involves high bandwidth requirements and an efficient management of the network [1]. In this context Cloud Computing is the common Computing paradigm applied, however it could be a bottleneck and a single point of failure. As part of the solution to these challenges and issues, fog computing has taken on a major role. Fog computing is defined by the OpenFog Consortium as "a horizontal system-level architecture that distributes computing, storage, control and networking functions closer to the users along a cloud-to-thing continuum". As a layer located between the Cloud layer and the Edge layer, it is closer to the end-devices than the Cloud, thus reducing latency, increasing bandwidth, enabling greater energy savings, better management of network load balancing, in short, offering greater QoS at an affordable cost [2].

However, the IoT is constantly evolving. According to the International Data Corporation, by 2025 the number of devices connected to the Internet will be around 42 billion, and a total of 80 zettabytes of data will be generated in the same year. The rapid growth of internet-connected things, and thus the increase in data generated, brings new opportunities but also new challenges (e.g. IoV). Therefore, even though Fog computing has helped a number of organisations and corporations to meet their IoT goals, further progress is needed in the development of infrastructures capable of meeting these new challenges. In this sense, both

*MeSS 2022: International Workshop on MDE for Smart IoT Systems, Nantes, France, July, 2022*

*Corresponding author: José A. Barriga

† This work was funded by the Government of Extremadura, Council for Economy, Science and Digital Agenda under the grant GR21133 and the project IB20058 and by the European Regional Development Fund (ERDF). These authors contributed equally.

jose@unex.es (J. A. Barriga); pjclemente@unex.es (P. J. Clemente)

0000-0001-8377-1860 (J. A. Barriga); 0000-0001-5795-6343 (P. J. Clemente)

corporations and academia are focusing their efforts on the development of new computing paradigms, such as Edge-Cloud computing, Cloudlet computing, Mobile Cloud Computing or Mobile Ad-hoc Cloud computing [1]. These efforts are also being focused on the improvement of existing computing paradigms, such as Fog or Cloud computing (e.g. Fog Federations, Fog-Cloud federations, task scheduling or offloading algorithms and policies improvements). To do this, IoT systems need to be developed, deployed and tested, requiring high investments on devices, fog nodes, cloud nodes, analytic nodes, hardware and software. However, in order to decrease the cost associated with developing and testing the system, the IoT system can be simulated. Thus, simulating environments help to model the system, reasoning about it, and take advantage of the knowledge obtained to optimise it. Designing IoT simulation environments has been tackled focusing on low level aspects such as networks, motes and so on more than focusing on the high level concepts related to IoT environments. Additionally, the simulation users require high IoT knowledge and usually programming capabilities in order to implement the IoT environment simulation [3]. The concepts to manage in an IoT simulation includes the common layers of an IoT environment including Edge, Fog and Cloud computing and heterogeneous technology.

Model-driven engineering is an emerging software engineering area which aims to develop the software systems from domain models which capture at high level the domain concepts and relationships, generating from them the software artefacts by using code-generators. In this respect, SimulateIoT [3] is a model-driven engineering approach to define, generate code and deploy IoT systems simulations. In this paper, SimulateIoT has been extended taking into account the requirements and new challenges of current IoT systems.

In this sense, the first contribution is based on the addition of the federated Fog concepts to the IoT domain and SimulateIoT metamodel. The federation of Fogs allows the different fog nodes to act as one entity rather than as isolated nodes. In this way, the user has the possibility to analyse the impact (usually on delay) of the application of new task scheduling or offloading algorithms and policies, using geographic distributions of Fog nodes, the addition or subtraction of certain nodes, etc.

The second contribution is based on the concept of Fog-Cloud federation. IoT systems are heterogeneous in infrastructure as a response to the heterogeneity (requirements) of their tasks and processes. In this respect, the cooperation between the different layers of an IoT system is essential to optimise the system, and a current research area. For instance, there are tasks that may be computationally complex and also have latency requirements in some parts of their processes, or applications that generate several kinds of tasks, such as delay sensitive tasks and complex computational tasks (e.g. a stream processing application). In order to achieve optimal execution of such tasks, federation between the Cloud layer and the Fog layer is a key element. In this way, the Fog layer should carry out the latency-sensitive processes, and the Cloud layer should carry out the computationally complex ones. In this sense, and as in the first contribution, end-users will be able to test the impact of algorithms and policies that manage the orchestration between Fog and Cloud in terms of performance in the execution of this kind of tasks.

The third contribution is carried out as a complement to the previous ones. The possibility of modelling IoT applications is added. IoT applications are the ones that generate different tasks and processes (with different requirements), thus making use of the new infrastructure included and allowing end-users to test their task scheduling algorithms, offloading policies, etc.

The last contribution focuses on the need to create a feasible latency model for the end-user of the simulator. SimulateIoT allows the Cloud and Fog nodes to be deployed on different machines, thus emulating a real system, otherwise the simulation results would not be realistic in terms of delay. To this end, we have included the possibility to model the latency that each Edge node or IoT application would hypothetically experience when interacting with the Fog/Cloud layers. In this way, the end-user can model the maximum and minimum latency, as well as the latency distribution (e.g. Gaussian) that each of the nodes might experience when interacting with each other.

In short, these extensions allow the modelling of a federated Fog and Cloud layer that can support critical applications with critical requirements for QoS, latency (e.g. ultra-low latency), bandwidth, energy consumption etc. Thus, end-users of the simulator can design, test, analyse and optimise IoT systems according to current and future IoT scenarios in terms of infrastructure and services requirements.

**Keywords:** IoT IoT systems simulation Model-driven Engineering Fog federation Fog-Cloud federation

## References

[1] A. Yousefpour, C. Fung, T. Nguyen, K. Kadiyala, F. Jalali, A. Niakanlahiji, J. Kong, J. P. Jue, All one needs to know about fog computing and related edge computing paradigms: A complete survey, Journal of Systems Architecture 98 (2019) 289–330, ISSN 1383-7621, doi:\let\@tempa\bibinfo@X@doihttps://doi.org/10.1016/j.sysarc.2019.02.009, URL https://www.sciencedirect.com/science/article/pii/S1383762118306349.

[2] F. Bonomi, R. Milito, J. Zhu, S. Addepalli, Fog computing and its role in the internet of things, in: Proceedings of the first edition of the MCC workshop on Mobile cloud computing, 13–16, 2012.

[3] J. A. Barriga, P. J. Clemente, E. Sosa-Sánchez, A. E. Prieto, SimulateIoT: Domain Specific Language to Design, Code Generation and Execute IoT Simulation Environments, IEEE Access 9 (2021) 92531–92552, doi:\let\@tempa\bibinfo@X@doi10.1109/ACCESS.2021.3092528.