DAGSTUHL
REPORTS

Volume 8, Issue 12, December 2018

Multidirectional Transformations and Synchronisations (Dagstuhl Seminar 18491)
Anthony Cleve, Ekkart Kindler, Perdita Stevens, and Vadim Zaytsev 1

Algebraic Coding Theory for Networks, Storage, and Security (Dagstuhl Seminar 18511)
Eimear Byrne, Martin Bossert, and Antonia Wachter-Zeh 49

Dagstuhl Reports, Vol. 8, Issue 12 ISSN 2192-5283

ISSN 2192-5283

Published online and open access by
Schloss Dagstuhl – Leibniz-Zentrum für Informatik GmbH, Dagstuhl Publishing, Saarbrücken/Wadern, Germany. Online available at
http://www.dagstuhl.de/dagpub/2192-5283

Publication date
April, 2019

Bibliographic information published by the Deutsche Nationalbibliothek
The Deutsche Nationalbibliothek lists this publication in the Deutsche Nationalbibliografie; detailed bibliographic data are available in the Internet at http://dnb.d-nb.de.

Digital Object Identifier: 10.4230/DagRep.8.12.i

Aims and Scope
The periodical *Dagstuhl Reports* documents the program and the results of Dagstuhl Seminars and Dagstuhl Perspectives Workshops.
In principal, for each Dagstuhl Seminar or Dagstuhl Perspectives Workshop a report is published that contains the following:
- an executive summary of the seminar program and the fundamental results,
- an overview of the talks given during the seminar (summarized as talk abstracts), and
- summaries from working groups (if applicable).

This basic framework can be extended by suitable contributions that are related to the program of the seminar, e. g. summaries from panel discussions or open problem sessions.

Contact
Schloss Dagstuhl – Leibniz-Zentrum für Informatik
Dagstuhl Reports, Editorial Office
Oktavie-Allee, 66687 Wadern, Germany
reports@dagstuhl.de
http://www.dagstuhl.de/dagrep

Report from Dagstuhl Seminar 18491

Multidirectional Transformations and Synchronisations

Edited by

Anthony Cleve[1], Ekkart Kindler[2], Perdita Stevens[3], and Vadim Zaytsev[4]

1 **University of Namur, BE,** `anthony.cleve@unamur.be`
2 **Technical University of Denmark, DK,** `ekki@dtu.dk`
3 **University of Edinburgh, GB,** `perdita.stevens@ed.ac.uk`
4 **Raincode Labs, BE,** `vadim@grammarware.net`

Abstract

Bidirectional transformations (bx) are a mechanism for maintaining the consistency of two (or more) related sources of information, such as models in model-driven development, database schemas, or programs. Bx technologies have been developed for practical engineering purposes in many diverse fields. Different disciplines such as programming languages, graph transformations, software engineering, and databases have contributed to the concepts and theory of bx.

However, so far, most efforts have been focused on the case where exactly two information sources must be kept consistent; the case of more than two has usually been considered as an afterthought. In many practical scenarios, it is essential to work with more than two information sources, but the community has hardly started to identify and address the research challenges that this brings.

Driven by the practical needs and usage scenarios from industry, this Dagstuhl Seminar aimed to identify the challenges, issues and open research problems for multidirectional model transformations and synchronisations and sketch a road map for developing relevant concepts, theories and tools.

The report contains an executive summary of the seminar, reports from its working groups, as well as descriptions of industrial and academic case studies that motivated the discussions.

Seminar December 2–7, 2018 – http://www.dagstuhl.de/18491
2012 ACM Subject Classification Information systems → Extraction, transformation and loadingSoftware and its engineering → Synchronization
Keywords and phrases bidirectional transformation, synchronisation
Digital Object Identifier 10.4230/DagRep.8.12.1

1 Executive Summary

Perdita Stevens (University of Edinburgh, GB)
Ekkart Kindler (Technical University of Denmark, DK)

The Dagstuhl Seminar on "Multidirectional Transformations and Synchronisations" was the latest on a sequence of events [1] (coordinated by the Bx Steering Committee [2]) on bidirectional transformations (abbreviated **bx**) and related topics. Broadly speaking, the concern of the growing community interested in this topic is the *maintenance of consistency* between multiple data sources, in the presence of change that may affect any of them. The focus of this Dagstuhl meeting, in particular, was the special issues that arise when one considers more than two data sources at one time. Technical definitions of bx have always allowed for there to be more than two, but in practice, most work to date has focused on maintaining consistency between two data sources. We abbreviate "multidirectional transformation", hereinafter and generally, to **multx** or **mx**[1].

We began the week with a presentation of case studies and introductory tutorials, and towards the end of the week we had several plenary talks, in cases where someone was able to discuss work that seemed likely to be of general interest. Overall, though, this was a highly interactive Dagstuhl: most of our time during the week was spent either in working groups or, synergistically, discussing in plenary session the outcomes of those working groups and what else needed to be addressed. Reports from each of those working groups, and descriptions of the case studies, are found in the body of this report. Here we briefly introduce them. Inevitably, the topics of the groups overlapped, and some topics that were proposed for working groups were not reached during the week. We encouraged participants to move freely between groups to foster cross-fertilisation of ideas. The names given are those of the authors of the brief reports. Collectively, these topics comprise a research roadmap for the subject.

- WG1: *Whether Networks of Bidirectional Transformations Suffice for Multidirectional Transformations.*
  This group began what turned out to be a recurring theme of the week: see below.
- WG2: *Partial Consistency Notions.*
  This relates to handling situations in which consistency is not perfectly restored, but only improved to some extent.
- WG3: *Semantics of Multidirectional Transformations.*
  This group raised questions about definitions of syntactic and semantic consistencies, vertical and horizontal propagation, etc. After creating enough awareness of the importance of this topic, the working group dissolved and its efforts were merged into others, in particular WG4, WG8 and WG12.
- WG4: *Multiple Interacting Bidirectional Transformations.*
  This group started with an intention of providing a good example of a "truly" multidirectional transformation and defined scenarios where several multx and bx work together towards restoring consitency.
- WG5: *Mathematical Backgrounds for Multidirectional Transformations.*
  Following on from WG1, this group considered, from a theoretical perspective, handling multx by the use of a common "federated" supermodel related by spans of asymmetric lenses.
- WG6: *Synchronisation Policy.*
  Separate from the issue of what the mechanism is to restore consistency, when should the mechanism be used, and whose decides that?
- WG7: *Use Cases and the Definition of Multidirectional Transformations.*
  When are multx really necessary in practice, and how?
- WG8: *Human Factors: Interests of Transformation Developers and Users.*
  Sometimes in our focus on technical aspects we lose the human element – who are the humans involved and what do they need from whatever languages, tools and techniques are developed for multx?
- WG9: *Provenance in Multidirectional Transformations.*

[1] Consensus on just one of those two options was not achieved by this workshop!

Information about what changed and why – provenance and traceability – are crucial to trust in multx; how can that information be provided and handled?
- WG10: *Living in the Feet of the Span.*
  Following on from WG1 and WG5: what happens when one conceptually uses a common supermodel, but does not wish to materialise it?
- WG11: *Programming Languages for Multidirectional Transformations.*
  This group discussed the challenges that need to be met to produce such languages, with a focus on their type systems.
- WG12: *Verification and Validation of Multidirectional Transformations.*
  What needs to be verified or validated about multx, and in what ways do these needs challenge the state of the art in verification and validation?

A recurring theme of the week, turning up in one guise or another in most of the working groups, was the question of the extent to which excellent solutions to the two-source bx problems would, or would not, automatically solve the multx problems. Do problems involving multx really introduce new issues, or are they just more complicated than problems involving bx, perhaps organised in networks? The bx problem is far from solved – we do not yet, for example, have widely adopted and well-supported specialist bx languages – and so there was some feeling that we lacked a firm foundation on which to address multx. More positively, considering multx has the potential to help bx research make progress, by motivating areas that still require more study in order to support the multx case. For example, heterogeneity of the languages in which the data sources and the changes to them are expressed clearly points to a need for bx, and hence multx, approaches that do not need to materialise edit histories for all the sources, nor a common supermodel for them – even if the theory that addresses them might still call on such things conceptually.

The following case study descriptions are also included in the report:
- Multidirectional Transformations for Microservices
- Multidirectionality in Compiler Testing
- Bringing Harmony to the Web
- A Health Informatics Scenario

Perhaps the most important observation from this Dagstuhl meeting, though, is how broad the scope of the research necessary to address multx concerns is. The issues and examples discussed by participants went far beyond multidirectional versions of issues and examples already raised in earlier bx meetings. For example, microservices, the focus of the case study presented by Albert Zündorf (§4.1), would not traditionally have fallen under the bx umbrella, yet is clearly related.

This widening of scope is natural. As IT systems become more interdependent and more important to our everyday lives, it is inevitable that data, and the (often separately developed) behaviour it supports, reside in many places. They are coupled, in the sense that changes in one place may mean that changes in another place are necessary, in order to maintain all of these systems in useful operation. Making all such changes manually does not scale: some degree of automated maintenance of consistency is inevitably required. Multx thus subsumes much of software engineering and inherits its concerns.

Readers of this document may wish to join the Bx community by subscribing to its mailing list and/or consulting the Bx wiki: see http://bx-community.wikidot.com/start. There is also a catalogue of examples of bx and multx, including some that were discussed at this Dagstuhl [3].

References

1 Bidirectional Transformations Wiki: *Bx Events*, http://bx-community.wikidot.com/bx-events, 2012.
2 Bidirectional Transformations Wiki: *Bx Steering Committee*, http://bx-community.wikidot.com/bx-steering-committee, 2012.
3 Bidirectional Transformations Wiki: *List of pages tagged with* multxdagstuhl, http://bx-community.wikidot.com/system:page-tags/tag/multxdagstuhl, 2018.

2 Table of Contents

Executive Summary
Perdita Stevens, Ekkart Kindler 1

Working Groups

WG1: Whether Networks of Bidirectional Transformations Suffice for Multidirectional Transformations
Michael Johnson 6

WG2: Partial Consistency Notions
Anthony Anjorin, Anthony Cleve, Sebastian Copei, Zinovy Diskin, Jeremy Gibbons, Hsiang-Shang Ko, Nuno Macedo, James McKinna, Andy Schürr, Bran V. Selic, Perdita Stevens, Jens Holger Weber, and Nils Weidmann 8

WG4: Multiple Interacting Bidirectional Transformations
Holger Giese, Gabor Karsai, and Vadim Zaytsev 10

WG5: Mathematical Backgrounds for Multidirectional Transformations
Hsiang-Shang Ko 12

WG6: Synchronisation Policy
Jeremy Gibbons and James McKinna 13

WG7: Use Cases and the Definition of Multidirectional Transformations
Fiona A. C. Polack, Anthony Cleve, Davide Di Ruscio, and Martin Gogolla 14

WG8: Human Factors: Interests of Transformation Developers and Users
Matthias Tichy and Heiko Klare 16

WG9: Provenance in Multidirectional Transformations
Nils Weidmann 21

WG10: Living in the Feet of the Span
Jeremy Gibbons and Michael Johnson 21

WG11: Programming Languages for Multidirectional Transformations
Kazutaka Matsuda, James Cheney, and Soichiro Hidaka 23

WG12: Verification and Validation of Multidirectional Transformations
Perdita Stevens 25

Case Studies

Multidirectional Transformations for Microservices
Sebastian Copei, Marco Sälzer and Albert Zündorf 26

Multidirectionality in Compiler Testing
Vadim Zaytsev 42

Bringing Harmony to the Web
James Cheney 43

A Health Informatics Scenario
Harald König 46

Participants 48

3 Working Groups

3.1 WG1: Whether Networks of Bidirectional Transformations Suffice for Multidirectional Transformations

Michael Johnson (Macquarie University, AU)

Introduction

A fundamental question about multx, which can be asked even before structures for multidirectional transformations have been precisely defined, is whether a new contentful notion of multidirectional transformations is necessary, or whether its intended outcomes can be achieved with interlinked bidirectional transformations forming something that might be called a network of bidirectional transformations. Accordingly, Working Group 1 was allocated this question.

On Stevens' counterexample

One answer to this question had already been proposed by Stevens [4] who presented an example of a ternary relation R on sets A, B and C which cannot be defined by any collection of binary relations among the three sets. Thus if R were the consistency relation for a multidirectional transformation among A, B and C, then no collection of *bi*directional transformations between A, B and C, could properly maintain that consistency relation.

So, in one sense, bidirectional transformations cannot, of themselves suffice.

But it should be noted that this assumes that we restrict ourselves to bidirectional transformations just between the three given systems, A, B and C. Of course, if we permit ourselves to consider a fourth system, S say, then the ternary relation among A, B and C can be obtained from binary relations, indeed functions, between S and A, S and B, and S and C – the tabulation of the relation as a set of triples with projections onto A, B and C provides a trivial example.

This reopens the question: Are there multidirectional transformations that cannot be obtained from the interaction of arbitrarily many bidirectional transformations?

Working Group 1 approach

Working Group 1 decided to approach this problem by attempting again to construct an example of a multidirectional transformation that could *not* be obtained from bidirectional transformations.

A promising approach seemed to be to look for a collection of three or more systems which would be required to satisfy an inter-model constraint that depended in a fundamental way on the current state of all three systems. One of the working group participants, Harald König, had already developed such an example in a health informatics scenario [2], see also the case study later in this report (§4.4).

Working Group 1 initial outcomes

Working Group 1 developed a detailed analysis of König's multimodel constraint with the aim of locating the anticipated difficulties in building bidirectional transformations to support the constraint.

To our surprise, no difficulties were found. It seems that one can build sufficient complexity into a system S which is only bidirectionally related to each of the systems A, B and C, to encapsulate the intermodel constraint that needs to be maintained among A, B and C. In a sense this parallels the response to Stevens' counterexample – if we allow ourselves to introduce extra systems, it seems possible to interact with them only via bidirectional transformations, but to manipulate in them the multidirectional aspects of a given collection of systems.

Work plan

These observations led to a plan of work which, with the agreement of workshop participants over the next two days, was carried through in working groups WG5 and WG10.

Working Group 5 looked into the mathematical foundations of multidirectional transformations as wide spans of bidirectional transformations (the topology that seemed in our initial analysis to side-step the expected difficulties).

Working Group 10 was entitled "Living in the feet of the span": It was premised on the thought that if wide spans of bidirectional transformations do suffice to capture multidirectional transformations in principle, they might nevertheless be undesirable to build in engineering terms. Having such wide spans as theoretical constructs helps in the mathematical analysis of multidirectional transformations, but experience in building transformations among systems strongly suggests that there are substantial benefits to be obtained from *co*-spans [1, 3] – small systems that capture the essential shared data among other systems, rather than spans which seem to correspond to large federated systems of systems. Working Group 10 would therefore analyse the interactions needed between the extant systems to efficiently *build* the multidirectional transformation that might theoretically be described by a wide span.

There are separate reports from each of WG 5 (§3.4) and WG 10 (§3.9).

To what extent do networks of bx suffice?

This might be a suitable place to reflect upon the overall outcome of all three groups.

WG1: Surprisingly the explored multimodel constraint did not yield any difficulties that carried us beyond bx.

WG5: Mathematical foundations based on wide spans seemed to suffice.

WG10: Several approaches to building the interactions among the feet of a wide span hold promise.

But it is in the results of Working Group 10 that the truly multidirectional aspects resurfaced, and they present a body of future work that should be explored by the multidirectional transformation community.

Summary

- It seems likely that networks of bidirectional transformations suffice for specifying multidirectional transformations and analysing some of their properties.
- Indeed, wide spans of bidirectional transformations seem able to capture at least a wide range of multidirectional transformations
- But engineering such multidirectional transformations among extant systems in a minimally invasive and reasonably efficient manner raises the kinds of questions that present a body of work for future consideration by the multidirectional transformation community including questions such as

- Atomicity
- Concurrency
- Sequentialisation
- Side effects, and, in particular
- Intermodel constraints – the problems that began Working Group 1's deliberations may necessitate cross model querying in calculating how to complete what would otherwise be a propagation in a standard bidirectional transformation such as a symmetric lens.

References

1 S. Copei and A. Zündorf. *MX for Microservices.* https://materials.dagstuhl.de/files/18/18491/18491.SWM.Preprint.pdf, 2018. See also §4.1.

2 H. König. *Commonality Specifications, Merged Models, and Partial Morphisms* (slides). https://materials.dagstuhl.de/files/18/18491/18491.HaraldK%C3%B6nig.Slides.pdf. Example in the BX Repository at http://bx-community.wikidot.com/examples:patientappointmentbloodtest, 2018. See also §4.4

3 M. Johnson and R. Rosebrugh. *Cospans and symmetric lenses.* Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming. pp. 21–29. https://doi.org/10.1145/3191697.3191717, 2018.

4 P. Stevens. *Bidirectional Transformations In The Large.* 20th International Conference on Model Driven Engineering Languages and Systems (MoDELS), ACM/IEEE, pp. 1–11. https://doi.org/10.1109/MODELS.2017.8, 2017.

3.2 WG2: Partial Consistency Notions

Anthony Anjorin (Universität Paderborn, DE), Anthony Cleve (University of Namur, BE), Sebastian Copei (Universität Kassel, DE), Zinovy Diskin (McMaster University, CA), Jeremy Gibbons (University of Oxford, GB), Hsiang-Shang Ko (National Institute of Informatics, JP), Nuno Macedo (University of Minho, PT), James McKinna (University of Edinburgh, GB), Andy Schürr (TU Darmstadt, DE), Bran V. Selic (Malina Software Corp., CA), Perdita Stevens (University of Edinburgh, GB), Jens Holger Weber (University of Victoria, CA), and Nils Weidmann (Universität Paderborn, DE)

This working group discussed *partial consistency* for multx. We spent most of our time discussing different reasons why partial consistency is relevant, especially in a multx setting. Each of these "dimensions" explained in the following is not only a source of motivation for supporting some notion of partial consistency, but also represents an initial understanding of what partial consistency could mean to different people:

Generalising the way we measure and represent the "value" of consistency. Especially in a bx network representing a multx, there can be local as well as global measures of consistency. The "consistency as a surface" metaphor/model deliberately draws an analogy with continuous dynamic systems, and their differential geometry [1]; the type-theoretic account [2] emphasises a proof-relevant account of consistency. Each attempts to expand our notion of "how consistency is measured" beyond a mere `tt/ff` distinction. Stevens [3] has explicitly considered partial consistency as measured lattice-theoretically;

other models, in metric spaces, try to capture the 'least'ness of 'least change'. In all cases, being able to "measure" consistency in some way implies a notion of *partial* consistency ranging from inconsistent to fully consistent.

An approximated, simplified, or incomplete notion of consistency can be viewed as an emergent, provisional property of a modelling/development process. Especially in a multx context, it might even be crucial (from an engineering point of view) to be able to develop the multx incrementally and iteratively, producing, for example, an ordered collection of restorers going from very tolerant to strict as the development process matures and eventually concludes. To be able to work with and test the provisional versions of the multx, therefore, some form of partiality is necessary, e.g., this or that "part" of the models is not yet fully covered.

Accomodating a process where consistency can only be observed (and enforced) at specific synchronisation points requires a means of tolerating continued model evolution in the presence of (unobserved) "inconsistency" and thus partial consistency. Especially in a multx setting, conflicts must be allowed as concurrent updates are unavoidable. It might also be possible to check for *local* consistency after every change, while *global* checks can only be conducted, e.g., when everyone else checks in a final version of their models at some agreed upon point in time.

Enabling a decomposition of consistency restoration into phases or parts can be useful. Examples include restoring consistency for more "important", or the "simplest", non-contentious parts first. This might not only be necessary to make consistency restoration feasible in a practical setting, but can also be used to simplify an integration with an external component (a user, an optimiser, etc), as these parts can be postponed until the external component provides the required input. As with all other dimensions, such a staggered form of consistency restoration requires being able to handle (temporary) inconsistencies (and thus partial consistency).

The working group concluded by briefly discussing what can constitute an inconsistency. The general consensus was that this can be very different, depending on the approach used to define consistency; It can range from (i) a violation of a constraint, (ii) a violation of a rule application due to deletion of context or violation of application conditions, (iii) existence of structure not described by any rule, or (iv) values that are not in the domain of a restorer.

References

1. Anthony Anjorin. *An Introduction to Triple Graph Grammars as an Implementation of the Delta-Lens Framework*, Tutorial Lectures of the International Summer School on Bidirectional Transformations, pp. 29–72, https://doi.org/10.1007/978-3-319-79108-1_2, 2016.
2. James McKinna. *Complements Witness Consistency*, Proceedings of the 5th International Workshop on Bidirectional Transformations (BX), pp. 90–94, http://ceur-ws.org/Vol-1571/paper_10.pdf, 2016.
3. Perdita Stevens. *Bidirectionally Tolerating Inconsistency: Partial Transformations*, Proceedings of the 17th International Conference on Fundamental Approaches to Software Engineering (FASE), pp. 32–46, https://doi.org/10.1007/978-3-642-54804-8_3, 2014.

3.3 WG4: Multiple Interacting Bidirectional Transformations

Holger Giese (Hasso-Plattner-Institut, DE), and Gabor Karsai (Vanderbilt University, US), and Vadim Zaytsev (Raincode Labs, BE)

Most of the work of the working group was shaped, motivated and conceptually linked to two case studies presented earlier during the week of the seminar: the cyber-physical system case study by Holger Giese (cf. [2]) and the compiler case study by Vadim Zaytsev (§4.2, also cf. [3]).

It seemed to us during the numerous discussions that syntactic consistency (achievable with bx) can be viewed as a precondition for enforcing some more global constraints (name uniqueness, deadlock freedom, simulation, model checking, etc.) with multx. This idea could potentially serve as a framework to combine bx and multx into one network. Whether this or another composition framework should be used in the future, it should be related to the scheme proposed earlier in 2018 by Diskin, König and Lawford [1].

3.3.1 Taxonomy

During one of the sessions the working group has sketched a taxonomy that can be, with sufficient domain research and linking to existing literature, worked on in the future and expanded into a real taxonomy of multidirectional transformations.

- Arity (of multx)
  - 1 (internal consistency)
  - 2 (bx)
  - > 2 (“true” multx)
- Model relations/hierarchy
  - List or sequence
  - Tree
  - DAG
  - Graph
  - Hypergraph
- Consistency
  - Representation
  - Structural consistency (“syntax”)
  - Behavioural consistency (“dynamic semantics”)
  - Static semantics
- Versioning
- Change action
  - Central
  - Distributed
- Megamodel of multx
- Authority
- Conflict resolution
- Policy or strategy
  - Lossy
  - Preserving
- Precondition
  - bx–multx
  - multx–bx

3.3.2 Case Study

In an attempt to come up with a simpler case study that would still be capable of serving as an example to demonstrate, we came up with the following. (NB: it was technically impossible to design a truly simple scenario because truly simple scenarios are handled by bx and other simpler frameworks and do not require multx).

Our domain is that of vending machines. The entity we would like to model, is of a machine serving coffee and tea: when enough coins are inserted, and a choice is made by pressing the right button, the machine produces the desired beverage to the best of its ability. Consider three models: L, P and S.

L is a labelled transition system. Its metamodel concerns states and transitions between them, with labels attached to both. L is good at modelling such aspects of the system as various kinds of buttons that the machine may have, as well as consequences of activating them. We also assume the labels have a way of representing sending and receiving messages if buttons are to be understood as channels.

P is a Petri net. Its metamodel concerns places, transitions, arcs between them and tokens that show which transitions can be fired according to the number of incoming arcs matched against the number of available tokens. P is good at modelling counting of all kinds, such as the number of inserted coins or the number of servings of coffee that can be made from available coffee beans before the machine needs to be refilled.

S is a sequence diagram. Its metamodel concerns lifelines, agents, messages and their relative timings. S is good at modelling sequences of events without knowing the internal workings of involved agents. It can represent either valid scenarios of combining L and P, or invalid ones, or even constraints such as "there should always be a beverage served after requesting it". In general, $L \parallel P \models S$.

Tasks that are naturally implementable with bx, are of syntactic nature: for example, if one is to edit L to add new actions, such changes need to be propagated to P. Such a bx will contain, among other things, the alignment in the sense of matching elements of L to elements of P, name-based or otherwise.

Tasks that cannot be handled naturally and gracefully by bx, are global and behavioural: for example, simulating execution of L in parallel with P to see if they are capable of producing the sequence of events specified by S.

References

1 Zinovy Diskin, Harald König, Mark Lawford. *Multiple Model Synchronization with Multiary Delta Lenses*, Proceedings of the 21st International Conference on Fundamental Approaches to Software Engineering (FASE), LNCS 10802, pp. 21–37, https://doi.org/10.1007/978-3-319-89363-1_2, Springer, 2018.

2 Holger Giese, Bernhard Rumpe, Bernhard Schätz, Janos Sztipanovits. *Science and Engineering of Cyber-Physical Systems*, Dagstuhl Seminar 11441. Dagstuhl Reports 1(11), https://doi.org/10.4230/DagRep.1.11.1, 2011.

3 Vadim Zaytsev. *An Industrial Case Study in Compiler Testing*. Proceedings of the 11th International Conference on Software Language Engineering (SLE), pp. 97–102, ACM, https://doi.org/10.1145/3276604.3276619, 2018.

3.4 WG5: Mathematical Backgrounds for Multidirectional Transformations

Hsiang-Shang Ko (National Institute of Informatics – Tokyo, JP)

Continuing with Harald König's example (§4.4) about medical record systems from Working Group 1, this working group discussed the idea of modelling multidirectional transformations (multx) using a *(wide) span of asymmetric lenses*, treating systems/models to be synchronised as views of a federated model incorporating all data in the relevant models and multiary inter-model constraints. The aim (also inherited from Group 1) was to explore to what extent multiary constraints can be handled using this span of lenses. It was emphasised a few times during the session that the span, in particular the federated model, which is monolithic and possibly gigantic, does not need to be actually constructed, and exists mainly for theoretical purposes, namely providing a space where reasoning about the whole system can happen. Some possible ways to avoid constructing the span were subsequently discussed in Working Group 10, whose participants were mostly from this group.

The group first spent some time recapping and clarifying the medical system example, which involved three models respectively responsible for bed assignments to patients (M_1), appointments for patients to meet doctors (M_2), and blood tests taken by patients (M_3). There was a ternary inter-model constraint: "any patient who is assigned a bed (in M_1) and has a severe blood test (in M_3) must have an appointment (in M_2)". The federated space could then be constructed by merging the three models, making them share the same list of patients, and adding some auxiliary queries that find all patients assigned a bed and having a severe blood test (henceforth "qualifying patients") so that the constraint can be formulated, checked, etc. Quite a few remarks were made about making the models more precise and comparing their categorical representation with UML, but these remarks did not directly affect the rest of the session.

We went on to discuss the behaviour of *put*ting an updated model into the federated model, and an interesting case was the *put* transformation for M_2 (where appointments are managed). It is easy to add an appointment, whereas deleting an appointment is potentially dangerous since that could be the only remaining appointment for a qualifying patient, violating the constraint. One way to avoid the danger is to forbid deletion of appointments, but Zinovy Diskin's plenary talk on *multiary delta lenses* [1], in particular the idea of *reflective updates*, inspired a new solution: further modifying M_2 by making a new appointment for the patient. While this opened up a new possibility, some were worried about how to establish the new set of properties of multiary delta lenses and potential consequences of losing the old and fundamental well-behavedness properties (PutGet for example).

The session closed with some general remarks: we may not have the luxury to add an extra federated model, but on the other hand some kind of external solution is necessary when existing models/systems cannot be modified. We also started to think about avoiding the construction of the span, for example by maintaining the list of qualifying patients in M_2, whose *put* needs the list to determine whether the constraint is satisfied and what to do. A more thorough discussion continued in Working Group 10.

References

1 Zinovy Diskin, Harald König, Mark Lawford. *Multiple Model Synchronization with Multiary Delta Lenses*, Proceedings of the 21st International Conference on Fundamental Approaches to Software Engineering (FASE), LNCS 10802, pp. 21–37, https://doi.org/10.1007/978-3-319-89363-1_2, Springer, 2018.

3.5 WG6: Synchronisation Policy

Jeremy Gibbons (University of Oxford, GB) and James McKinna (University of Edinburgh, GB)

This working group discussed *synchronisation policy* for transformations—that is, consideration of *whose* responsibility it is to decide to restore consistency, *when* that should be done, and *which* participants should be considered authoritative and which should be updated. (In particular, we did not mean the question of how to choose between different options for restoring consistency when required to do so, which is sometimes also referred to as a "policy".)

One might take version control systems as an illustrative example: the `man` page for `git` describes which commands are available and what they do, but a novice user needs a separate tutorial that explains recommended *patterns of use* for the tool, and typical *workflows*. A developer working on one part of a shared system might legitimately not wish to get interrupted while "in the flow" of coding in order to mentally process repeated synchronisation of updates from other parts of the system. Similarly, Stevens [1] recommends that build systems for megamodels should be demand-driven, rather than attempting global consistency restoration. As another analogy, James Cheney pointed out that the field of computer security distinguishes between *policy* (what security properties to achieve) and *mechanism* (how to achieve them).

The group spent some time discussing a number of existing studies and problems. Zinovy Diskin described a *taxonomy* of 44 distinct bx synchronisation patterns [2]. Josh Ko described a problem of *coordinated consistency restoration* when reconciling refactorings of communication protocols and their abstractions as session types [3]. Albert Zündorf showed how to express Harald König's health informatics scenario [4] using synchronisation based on event-driven programming [5]. (The latter two might be viewed in terms of cospans of asymmetric lenses, synchronising on a communication channel that represents the information shared between two parties.) Jens Weber observed that messaging applications in complex environments (specifically, hospital information systems) absolutely need to manage their inbox, and want to have routine messages batched up for efficient handling, only synchronising immediately on truly urgent messages. The following day, Anthony Anjorin described some consistency management scenarios based on examples from the industry automation domain [6], which were also essentially synchronisation policies.

Most of our discussion applied as much to bidirectional transformations as to multidirectional. But the landscape of options for synchronisation is much more interesting for multidirectional transformations than for bidirectional, because there are more degrees of freedom. For one, there is a question of how many participants to fix, and how many may change in order to restore consistency; it is not all-or-nothing. For a second, one need not restore consistency among all participants at once; there is a non-trivial choice of schedules—perhaps one design approach for a multx language would be integrate a bx language for one-to-one transformations with a process algebra for coordination. For a third, whereas it makes sense to compose multiple spans of asymmetric lenses into a single wide span, it does not make sense to compose multiple cospans of lenses into a single cospan—the former represents the "union" of all data sources, and the latter their "intersection", which is typically empty.

References

1 Perdita Stevens. *Towards Sound, Optimal, and Flexible Building from Megamodels*. MoDELS, p. 301–311, https://doi.org/10.1145/3239372.3239378, 2018.
2 Zinovy Diskin, Hamid Gholizadeh, Arif Wider, and Krzysztof Czarnecki. *A Three-Dimensional Taxonomy for Bidirectional Model Synchronization*. Journal of Systems and Software 111:298–322, https://doi.org/10.1016/j.jss.2015.06.003, 2016.
3 Liye Guo, Hsiang-Shang Ko, Keigo Imai, Nobuko Yoshida, and Zhenjiang Hu. *Towards Bidirectional Synchronization between Communicating Processes and Session Types*. Second Workshop on Software Foundations for Data Interoperability (SFDI), 2019.
4 Harald König. *Commonality Specifications, Merged Models, and Partial Morphisms* (slides). https://materials.dagstuhl.de/files/18/18491/18491.HaraldK%C3%B6nig.Slides.pdf (2018). Example in the BX Repository at http://bx-community.wikidot.com/examples:patientappointmentbloodtest.
5 Sebastian Copei and Albert Zündorf. *MX for Microservices*. https://materials.dagstuhl.de/files/18/18491/18491.SWM.Preprint.pdf, 2018.
6 Anthony Anjorin, Enes Yigitbas, Erhan Leblebici, Andy Schürr, Marius Lauder, and Martin Witte. *Description Languages for Consistency Management Scenarios Based on Examples from the Industry Automation Domain*. Programming Journal 2(3):7, https://doi.org/10.22152/programming-journal.org/2018/2/7, 2018.

3.6 WG7: Use Cases and the Definition of Multidirectional Transformations

Fiona A. C. Polack (Keele University, GB), Anthony Cleve (University of Namur, BE), Davide Di Ruscio (University of L'Aquila, IT), and Martin Gogolla (Universität Bremen, DE)

In practical terms, multiple models (in the widest sense) are used to simplify — either because it is not possible, or because it is not efficient, to capture what is required in one notation, view, tool, etc.

When more than one representation is used, information is lost. That information is needed to establish or maintain consistency. In one sense, the technical debt incurred in decomposing into multiple models must be repaid, at least in part, when establishing cross-model features such as consistency.

We hypothesised that there are two sorts of case:

- A network of bx can be used if (likely only if) all the information needed to establish a particular feature (e.g. a specific form of consistency) is held in within the two linked models – we think this is the situation that Zinovy Diskin describes when asserting (informally) that multx are strictly unnecessary.
- A multx is needed when the information necessary to establish the desired feature is held separately from the two models between which the feature must be established.

Horizon 2020 project Typhon [1] has the example of a distributed data storage system. A single conceptual model (CM) links to many different components including relational and no-SQL databases, file stores, etc. Each component has its own logical schema (LS_i) using the internal concepts of that component (e.g. relational db; relational schema; rdb tables, rdbms types and operations). There is a bx between each LS_i and CM, which maintains

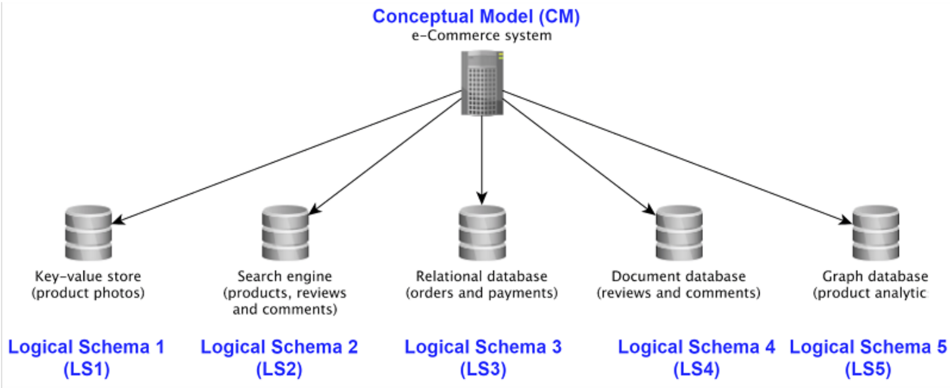

Figure 1 The conceptual model and logical schemata in Typhon [2].

consistency between CM and LS_i. However, to determine whether pairs or triples (up to all LS_i) of LS_i are consistent requires reference to the CM, which abstracts the relationships between all concepts. Thus, consistency between LS_1 and LS_2 references CM to establish how concepts in each LS_i are related in the overall system. This is a multx.

Albert's micro-system (§4.1) could be interpreted similarly. However, Albert's problem required that other notions of consistency be established. For example, protocols defined by outside committees or people needed to be enforced by both software (micro services based on customer oriented views and the "databases" that hold the specific data). It did not seem to be possible to define how overall consistency could be maintained because it was not possible / feasible to establish how the non-software components could become or maintain mutual consistency (i.e. where people change linked policies in such a way that software no longer correctly enforces both parts of the pair). Whilst some desired consistency elements could be multx-enforced because they could be expressed as constraints over the language(s) of the software-related models, it was not possible to express all the desired consistency in the same (set of) language.

Comparing the two examples, we see that the definition of the desired "feature" is critical. In real cases, the third element (which we called the black box, simply because it was drawn using a black pen) might be a collection of different, likely incompatible, boxes. It might also include the need for a person or human role to make a choice or resolve an inconsistency. The box (or collection of boxes) might also capture a synchronisation policy (human or logical).

Finally, we recognised that the black box – the multi part of the multx – was not, in practice, fixed. Even in the case of stable models, it is possible that new forms of consistency that need to be (re)established would be identified – for component models might have an element that is not obviously expressing the same "thing", but which is subsequently recognised as having a dependency or other constraint-expressive relationship that must be maintained. In real systems, it is also the case that the black boxes change over time. For instance, the complex multi-way relationship between a programming language and machine code would once have been a (or many) bx and multx, but has been replaced by a bx model called a *compiler*. There is an open question concerning whether a black box element is consistent with the element(s) that is replaces: is the set of rules defining compilation consistent with the behaviour of the new compiler, and if not, what are the consistency consequences elsewhere in the collection of models?

References

1 *Typhon: Polyglot and Hybrid Persistence Architectures for Big Data Analytics*, https://www.typhon-project.org/, 2018.
2 Davide Di Ruscio, Fiona A.C. Polack, Martin Gogolla, Anthony Cleve, Dominique Blouin, Nuno Macedo. *Conceptual Schema to Multiple Logical Schemas*, http://bx-community.wikidot.com/examples:conceptualschematomultiplelogicalschemas, 2019.

3.7 WG8: Human Factors: Interests of Transformation Developers and Users

Matthias Tichy (Universität Ulm, DE) and Heiko Klare (Karlsruhe Institute of Technology, DE)

3.7.1 Taxonomy

In order to provide a common terminology around the challenges w.r.t. users of multidirectional transformations, we derived a taxonomy. The taxonomy covers the following (not complete) set of aspects:

Different roles of users, e.g.:

- Developer of a transformation
- User of a modelling tool
- Developer of a modelling language and corresponding tool
- Developer of other aspects, e.g., Requirements Engineering, Architecture
- Domain expert

Person characteristics

- Experience
- Knowledge

Processes

- Tool Development
- Model Development

Quality Attributes

- Ease of Use
- Expressive Power
- Comprehensibility
  - Justification
  - Teaching
  - Understanding
  - Capability of Change
  - Number of people (individuals vs. group)
- Learnability

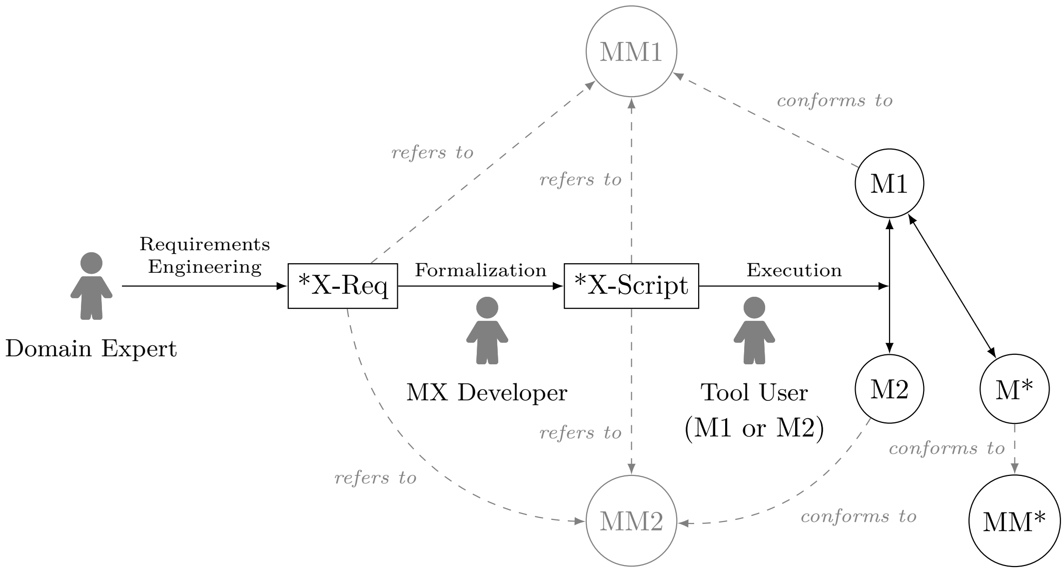

Figure 2 Roles and artefacts involved in a consistency scenario for two or more types of models.

Artefacts

- Transformation Script
  - Transformation Language
    - Declarative
    - Operational
  - Transformation Engine
- Domain-specific Language
  - Syntax
  - Semantics
- Model

For example, we have to distinguish between the developer of a modelling language, the developer of a model transformation, and the developer of other aspects of a modelling tool, as well as the user of the tool. Experience and knowledge of the different users w.r.t. to the domain, the modelling tool, and the modelled system also have to be considered.

Figure 2 shows the roles and artefacts involved in a scenario, in which two or more types of models shall be kept consistent with a bidirectional or multidirectional transformation (referred to as *X), according to the previously introduced taxonomy. Given two model types MM1 and MM2, a domain expert acquires the consistency relations between them in a requirements elicitation process, leading to the requirements artefact *X-Req. An *X-Developer then formalizes these requirements for a specific *X-Language. This leads to an *X-Script, i.e., a transformation specification for that specific *X-Language. That *X-Script can then be executed using the *X-Engine of the *X-Language by a tool user, who develops a model M1 or M2 (or potentially other models denoted as M*) of the model types MM1 or MM2, respectively, to check and restore consistency with a model of the other model type(s). We discussed interests and challenges for two central user roles in this scenario: the transformation developer, who specifies consistency in transformations, and the tool user, who uses those transformations for preserving consistency.

3.7.2 The Transformation Developer Perspective

We considered the scenario that a transformation developer wants to define a new transformation *X-Script, for example between model types MM1 and MM2. Other transformations, for example between model types MM1 and MM*, may already be defined.

The good case

In the best case, the transformation operates correctly, i.e. conforming to the requirements in *X-Req. It also interoperates correctly with other transformations, e.g. an existing transformation between MM1 and MM*. Such proper operation should be validated by test cases that indicate absence of errors to the transformation developer.

The bad case

In practice, a transformation developer will make errors while specifying a transformation *X-Script. In that case, two types of errors can occur:

- *X-Script does not fulfil *X-Req: Such errors should be identified by failing test cases or at least through feedback from a tool user who experiences erroneous behaviour.
- *X-Script does not properly interoperate with other *X-Scripts (e.g. between MM1 and MM*): Such errors can be detected during development by test cases that investigate certain combinations of *X-Scripts, or by a tool developer who combines independently developed *X-Scripts in a specific scenario.

In both cases, an a-priori detection of errors by analysing or testing an *X-Script should be preferred over an a-posteriori detection by a tool user. To support the transformation developer in finding bugs during development, he requires support by the *X-Language and its engine to find out why consistency is not preserved correctly. This especially comprises tracing back a failure to the origin of the change that lead to the failure (provenance). Additionally, a transformation developer requires appropriate debugging support to step through an *X-Script execution, just like in ordinary code development.

3.7.3 The Tool User Perspective

We considered the role of a tool user, who performs modifications in one model and executes a transformation to detect and resolve introduced inconsistencies to other models. We identified two different scenarios that reveal different challenges that occur when a tool user applies a transformation. The first scenario deals with the “good case”, in which the formalization of consistency conforms to the actual consistency relations. The second scenario deals with the “bad case”, in which consistency relations are not properly represented in their formalization, leading to errors during execution of the transformation. For ease of understanding, the tool user in the following scenarios will always be responsible for M1.

The good case

We assume that *X-Req and its formalization *X-Script represent all actually existing consistency relations that exist between the model types MM1 and MM2 (and only those). In that case, if a user applies changes to M1, which potentially introduces an inconsistency regarding *X-Req to M2, the execution of *X-Script has to inform the user about potential inconsistencies, provide him information about where the specification in *X-Req is violated and provide options on how to make the models consistent again. When the user selects one option to restore consistency, this can be either directly applied or trigger a consistency restoration process, in which the modification is wrapped into a change request that has to be sent to a person that is responsible for modifications in M2. After applying the changes, they must be stored together with the decision rationale to ensure that developers understand the reasons for the modifications.

The bad case

In practice, *X-Req and the derived *X-Script will potentially not cover exactly those consistency relations that actually exist between the model types MM1 and MM2. If the tool user performs a modification in M1 that is covered completely and correctly by *X-Req and *X-Script, then still the previously presented "good case" applies. If that case is not correctly covered by *X-Req and *X-Script, two cases can be distinguished:

1. An inconsistency is detected by *X-Script execution, although no inconsistency exists (false positive)
2. No inconsistency is detected by *X-Script execution, although an inconsistency exists (false negative)

There are different reasons why this can happen:

1. *X-Req is wrong, i.e., it contains faulty consistency relations
2. *X-Script is wrong, i.e., it does not conform to *X-Req
3. *X-Req is underspecified, i.e., it does not represent all existing consistency relations
4. *X-Req is overspecified, i.e., it contains consistency relations that do actually not exist

While the first two options can lead to both false positives and false negatives, an underspecification of *X-Req will lead to false negatives, whereas an overspecification of *X-Req will result in false positives. Depending on the reason for the faulty inconsistency detection, it is either necessary to fix *X-Req or to fix *X-Script. A fix of those artefacts may take some time, as the error usually occurs when a tool user executes the *X-Script, but the *X-Script is implemented and thus fixed by different persons, the *X-Developers. Due to that, it will be necessary to allow a manual override of decision taken by the *X-Script execution to prevent delays in the development process. Afterwards the user can make a change request to have *X-Req and *X-Script updated.

3.7.4 Challenges

The scenarios reveal different challenges that require further investigation.

C1: How to deal with a lack of domain understanding of M1 tool user about changes in M2?
The M1 tool user has knowledge about MM1 and its instances but potentially not about other domains, especially MM2, and the relations between them. Because in general the execution of *X-Script can only support the user in making a decision on how to restore consistency by showing and rating potential effects or recommending options, the M1 tool user has to make the final decision on consistency restoration to M2. It is an open challenge how to overcome the lack of necessary knowledge to make that decision.

C2: How to integrate the *X-Script execution into organizational processes and tools?
The *X-Script execution will in general not be fully automated but involve user decisions, even of different roles. This may lead to a complex and time-consuming process that is even influenced by organizational processes or existing development processes. Finally, the consistency restoration process must be integrated into theses processes and especially tooling that supports them. It is an open challenge how this integration should be made.

C3: How to deal with confidential model parts in intra-organizational settings?
Models may contain parts that are only to be seen and modified by specific roles, e.g., because they contain confidential information. If the modification of another model requires decisions on restoring consistency with such confidential data, it must be ensured that only people with appropriate roles see that data and perform the appropriate decisions. It is an open challenge how to deal with confidential data or, in general, role-based access control during consistency restoration.

C4: When, whom and how to allow a manual override of decisions of the *X-Script execution?
Due to the fact that in practice the specification of consistency will never fully or correctly cover the actual consistency relations, it must be possible to overwrite decision of an *X-Script whenever they are wrong. On the other hand, it should not be possible to always make manual overrides and completely disable the defined consistency restoration process by *X-Script. It is an open challenge when to allow such overwrites and whether that depends on a certain role. In addition, it is yet unclear how to make the *X-Engine deal with ad-hoc, manual overrides of the specified consistency relations, while still preserving traceability of consistency-restoring decisions and their rationales.

C5: How much specification of *X-Req is enough? How to support users in understanding limitations of *X-Req and *X-Script?
The specifications of consistency relations in *X-Req and *X-Script will potentially not cover all relations that exist between two models types MM1 and MM2, either because some relations are missed or because they are too complex to express in an appropriate formalism. It is an open challenge to decide how much specification is enough and especially how to support the user in understanding the limitations of the specifications, as he has to deal with the rest of the relations that are not specified in *X-Script. Finally, this is a cost/benefit estimation that has to be made by the domain expert who specifies *X-Req.

C6: How to increase the trust of a tool user in the execution of *X-Script?
Due to the fact that *X-Script can be incomplete or erroneous, as discussed in the bad case scenario, the tool user can easily lose confidence in *X-Script if failures occur too often or are too severe. It is therefore a general challenge to increase the trust of a user in such an *X-Script and to find general ways how to achieve that. This is highly related to identifying the appropriate extent of a specification as discussed in challenge **C5**.

C7: How to deal with non-determinism w.r.t. user trust?
Non-determinism in *X-Script, e.g. due to different explanations for inconsistencies or reaching consistency in different ways, can reduce the user trust into *X-Script. Tool users will usually expect determinism from such an *X-Script and also assume that small differences in inputs models should result in small differences in the target models by executing *X-Script. It is an open challenge how to preserve user trust whenever non-determinism is inevitable or to reduce evitable non-determinism.

3.8 WG9: Provenance in Multidirectional Transformations

Nils Weidmann (Universität Paderborn, DE)

The role of provenance and traceability constitutes an important issue in a multx context. Some ideas where exactly and why provenance is an interesting research topic with respect to multx were discussed in this working group.

In the context of model-driven engineering (MDE), traceability is provided by the correspondence model of a triple graph grammar (TGG). Seminal work on the various roles of correspondence nodes in different TGG tools already exists. Depending on the concrete implementation, correspondence links can be attributed, as the original formalism does not say anything about that. However, a certain state defined by the source, target and correspondence model is not unique, e.g. in case of confluent TGGs. Therefore, the correspondence model itself does not provide enough traceability information. As a proof object, an additional protocol is required that tracks the sequence of rule applications. In the context of software engineering, the OSLC (Open Services for Lifecycle Collaboration) standard could provide software engineers with useful ideas how to keep multiple models consistent.

Another topic of interest is traceability in mega modeling for object-relational mapping (ORM). It is possible to have arbitrarily nested containers, whereas it is possible to have correspondences between those containers.

In general, traceability links provide more than a binary information about the relation between two model elements. The concrete use cases in a multx context have to be defined, though. On the one hand, they can be used to give evidence that some relationship exists in an abstract sense. On the other hand, they can also be used for a special purpose like navigation in data models. In this case, the importance of a link can be measured in terms of how often it is used for navigation.

When restoring consistency, the multx context poses far more challenges with respect to provenance than the binary case. The possibility of consistency restoration within one step is a common case in bx, but very unlikely in multx situations. Therefore, versioning gets more and more important, which means that each rule application creates a new version of the data model.

References

1 James Cheney. *Provenance – a Dagstuhl presentation*, https://materials.dagstuhl.de/files/18/18491/18491.JamesCheney1.Slides.pdf, 2018.

3.9 WG10: Living in the Feet of the Span

Jeremy Gibbons (University of Oxford, GB) and Michael Johnson (Macquarie University, AU)

This working group followed on from Group 1 discussing whether multx are necessary or whether networks of bx suffice (§3.1), and Group 5 discussing mathematical foundations of multx (§3.4). Those two groups had already spent some time discussing Harald König's health informatics scenario [1] (§4.4): three independent systems, M_1 recording patients and their possible assignment to beds, M_2 recording appointments between patients and doctors, and M_3 recording blood tests, synchronised on the sets of patients, becoming subject to the additional inter-model integrity constraint I that any patient assigned a bed and in receipt of a severe test result must also have a doctor's appointment.

In particular, as discussed in Group 5, one implementation strategy for incorporating I into M_1, M_2, M_3 is to build a federated system M recording all three collections of data, with asymmetric lenses reconstructing M_1, M_2, M_3 as views of M; this forms a three-legged *span of lenses*. Without considering I, it is easy to implement *put*s and *get*s for patients etc, since the individual systems are just projections from the federated system. To address I as well, *put*s require a bit more work: for example, when assigning a patient to a bed in M_1, if that patient already has a severe test result in M_3 then they should also be given an appointment in M_2.

The particular focus in this working group was to discuss alternative implementation strategies, in case one chooses not actually to implement M itself but simply to treat it as a specification of the required behaviour of M_1, M_2, M_3 and I. What alternative implementation strategies are there? We identified four, in two groups of two:

1. modify one or more of the "feet" M_1, M_2, M_3 of the span to implement the constraint I (for example, in our scenario, modifying the appointments database to know about beds and severe test results);
2. as (1), but adapt the foot by applying some kind of wrapper (a "sock") to it rather than modifying it;
3. add a coordinating *façade* component [3], which intercepts *put*s to the components M_1, M_2, M_3, and applies the additional *get* and *put* operations required to re-establish I;
4. as (3), but materialise as a separate component M_4 the queries required for re-establishing I.

Alternative (1) may be unacceptable–perhaps the components to be modified are proprietary systems, or otherwise unchangeable. Alternatives (1) and (2) both awkwardly duplicate some of the logic required to implement I; in our scenario, the behaviour of M_1 will need to change to possibly create an appointment when assigning a patient to a bed, and the behaviour of M_3 similarly when recording a severe test result.

Alternative (3) entails several additional operations on the separate components to implement a *put*; this at least introduces issues of atomicity and transactional updates. Moreover, when the components M_1, M_2, M_3 are distributed, *put* must be allowed to perform I/O actions rather than being a pure function, becoming an *effectful lens* [4]. Alternative (4) saves the additional queries on other components for determining what reconciliation is required, but cannot help with the additional *put*s; it also introduces an additional component, which must itself be synchronised.

References

1 Harald König. *Commonality Specifications, Merged Models, and Partial Morphisms* (slides). https://materials.dagstuhl.de/files/18/18491/18491.HaraldK%C3%B6nig.Slides.pdf, 2018. Example in the BX Repository at http://bx-community.wikidot.com/examples:patientappointmentbloodtest. See also §4.4.

2 Sebastian Copei and Albert Zündorf. *MX for Microservices*. https://materials.dagstuhl.de/files/18/18491/18491.SWM.Preprint.pdf, 2018. See also §4.1.

3 Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

4 Faris Abou-Saleh, James Cheney, Jeremy Gibbons, James McKinna, and Perdita Stevens. *Notions of Bidirectional Computation and Entangled State Monads*. MPC, LNCS 9129, pp. 187-214, https://doi.org/10.1007/978-3-319-19797-5_9, 2015.

3.10 WG11: Programming Languages for Multidirectional Transformations

Kazutaka Matsuda (Tohoku University, JP), James Cheney (University of Edinburgh, GB), and Soichiro Hidaka (Hosei University, JP)

This working group has discussed language supports for multi-directional transformations, mainly on the two topics: representation of consistency relation and coordination languages.

3.10.1 Consistency Relations by Types

A consistency relation, a relation that any updates must fulfil (after consistency resolution if necessary), is one of the important notions in bidirectional and multi-directional transformations. Thus, a representation of a consistency relation in a programming language is an important research question.

It would be a natural idea to represent such constraints as types in a programming language. In response to the discussions in WG 5, which discussed a category-theoretic solution to the hospital example presented by Harald König, we have observed that a type system must be able to represent the following constraints to handle the hospital example:

- Functional dependencies and key dependencies
- Join dependencies

We find that refinement types and dependent types (see Baltopoulos et al. [2] and Xi et al. [9] for examples) would be useful for addressing the problem of representing consistency relation. Here, by a refinement type, we mean a type refined by a predicate, such as $\{a : \mathit{Int} \mid a > 0\}$. In a dependent type system, a type can depend on values, by special forms of types called dependent products (also known as $\prod$ types) and dependent sums (also known as $\sum$ types).

Among the two, a dependent sum is especially useful for representing what is called inter-model constraints in the software engineering community. A dependent sum type $\sum_{(a:\sigma)} \tau$ represent a set of pairs (u, v) such that u has type σ and v has the type obtained from τ by replacing a by u. For example, the type $\sum_{a:\mathit{Int}}\{b : \mathit{Int} \mid a < b\}$ represents a set of pairs (a, b) of Int values satisfying $a < b$.

Also, it is convenient if type check can be done locally (i.e., on each component independently for a dependent sum) instead of globally (i.e., on both components of a dependent sum). To do so, for a dependent sum $\sum_{\{a:A|\phi(a)\}}\{b : \psi(a, b)\}$, it is convenient if we can find a weak enough constraint $\phi'(b)$ such that $\phi'(b)$ implies $\phi(a, b)$, so that we can check the global constraint $\phi(a, b)$ only when the local check $\phi'(b)$ fails.

Thus, a research question is: *can we design a refinement type system that satisfies the above criteria*? A good start would be extending the relational lens framework [3], which does not have dependent sums but a sort of refinement types that consider functional dependencies.

As a side note, a sophisticated type system would sometimes be useful for users to give valid updates especially when they support holes. A successful example is a dependent programming language Agda [1], which allows users to specify a well-typed term interactively to fill a hole with editor's support.

3.10.2 Coordination Language

One of the lessons that we have learned from this Dagstuhl seminar is that there should be a separation of concerns, especially when we realize multi-directional transformations by combining bidirectional transformations; one concern is how to *specify* bidirectional transformations between two objects, and the other is how to *use* bidirectional transformations to make a whole system in sync. So far, though there have been a lot of studies addressing the former issue, the latter has been overlooked in the programming language context.

Thus, a research question is: *how can we design a programming language for using bidirectional transformations*? It is expected that such a language should support:

- describing where and when to apply bidirectional transformations (cf. synchronization policies discussed in WG 6),
- declaring datatypes for "virtual" data that do not correspond to any concrete target data in sync, such as spans [6], and
- mixing bidirectional transformations defined in different systems such as the lens [4] and the triple graph grammar [8].

We may call this kind of languages *coordination languages* or *strategy languages*. This needs of coordination languages may be similar to how a multi-tier programming language [7] is about session types [5].

References

1 Agda, *The Agda Wiki*, https://wiki.portal.chalmers.se/agda/pmwiki.php, 2008.
2 Ioannis G. Baltopoulos, Johannes Borgström, Andrew D. Gordon: *Maintaining Database Integrity with Refinement Types*. ECOOP, pp. 484–509, https://doi.org/10.1007/978-3-642-22655-7_23, 2011.
3 Aaron Bohannon, Benjamin C. Pierce, Jeffrey A. Vaughan: *Relational lenses: a language for updatable views*. PODS, pp. 338–347, https://doi.org/10.1145/1142351.1142399, 2006.
4 J. Nathan Foster, Michael B. Greenwald, Jonathan T. Moore, Benjamin C. Pierce, Alan Schmitt: *Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem*. ACM Transactions on Programming Languages and Systems (TOPLAS), Volume 29, Number 3, p. 17, https://doi.org/10.1145/1232420.1232424, 2007.
5 Kohei Honda, Vasco Thudichum Vasconcelos, Makoto Kubo: *Language Primitives and Type Discipline for Structured Communication-Based Programming*. ESOP, pp. 122–138, https://doi.org/10.1007/BFb0053567, 1998.
6 Michael Johnson, Robert D. Rosebrugh: *Spans of lenses*. EDBT/ICDT Workshops, pp. 112–118, http://ceur-ws.org/Vol-1133/paper-18.pdf, 2014.
7 Matthias Neubauer, Peter Thiemann: *From sequential programs to multi-tier applications by program transformation*. POPL, pp. 221–232, https://doi.org/10.1145/1040305.1040324, 2005.
8 Andy Schürr: *Specification of Graph Translators with Triple Graph Grammars*. WG, pp. 151–163, https://doi.org/10.1007/3-540-59071-4_45, 1994.
9 Hongwei Xi, Frank Pfenning: *Eliminating Array Bound Checking Through Dependent Types*. PLDI, pp. 249–257, https://doi.org/10.1145/277650.277732, 1998.

3.11 WG12: Verification and Validation of Multidirectional Transformations

Perdita Stevens (University of Edinburgh, GB)

The purpose of this working group was to discuss issues relating to verification and validation that will arise when multidirectional transformations (hereinafter: multx) are used.

Verification and validation of software, let alone of bx, is not a fully solved problem. We tried to focus on the specific new issues raised by having multx as opposed to bx, but it was hard to do so and much of what we discussed applies equally to bx.

We talked both about validating the multx themselves, and about validating their implementations (i.e. on the assumption that the specification of the multx was clear) – sometimes it is important to separate these. Most of our discussion focused on V&V of the multx themselves (i.e., whether what is written in the multx formalism is what was intended and has the properties intended, assuming that it is then "correctly" implemented according to whatever semantics the multx formalism has).

We discussed different aspects of what we need to do, including:

- Validate that the definition of consistency we have is the one we want. (Part of validating Bx is to give examples (e.g., examples of consistent and of inconsistent model pairs). This helps students to understand what they have written. We would expect it to extend (by be harder) for multx.)
- Verify that a particular collection of models is consistent
- Verify that consistency restoration "worked" on a particular occasion – meaning not only that the resulting set of models is consistent, but perhaps also that any intended properties such as variants of "least change" were obeyed (see below for more on properties);
- At the next level up, verify that the consistency restoration procedure embodied by a particular bx will always work (in all the above senses). Depending on the formalism used this might include checking that consistency restoration is a function (e.g. terminates; for rule-based systems, is confluent).
- Perhaps: at the next level up, verify that any multx written in a particular language/following a particular method/using a particular toolset will "work".
- Check the "ilities" of the multx, e.g.,
  - feasibility, both in terms of
    - use of computing resource e.g. where a SAT solver is used – no point in defining a multx if it can't be applied in less than the lifetime of the universe
    - use of human resource (e.g. it won't work to present too many decisions to the human user)
  - scalability, considering the sizes of instances that must be allowed for
  - in control theory senses: observability, controllability

We briefly discussed coverage issues, relating to the question of how we know, in a test-based verification scenario, when we have enough tests. In a bx setting, with TGGs, one sometimes looks for coverage of each rule as a basic standard, going on to look at combinations of two rules.

People are often unwilling to invest in verifying or validating their early *models* because they don't trust the subsequent tool chain and transformations – so such V&V work would be duplicating work that will have to happen later anyway. To look at it positively, this suggests that getting better at automatable V&V of transformations and their tools has potential to move manual validation work earlier in the process with consequent benefits.

There are also practical issues that need to be solved and become worse in a multx setting than they are in a bx setting: if, for example, our conceptual multx is embodied in a distributed collection of models and transformations, we may need ways to mock out parts of the system.

We discussed properties of multx that we might want to verify. These often depend on the specific setting, but might include:

- correctness, i.e. consistency really is restored
- hippocraticness, i.e. if the current state is already consistent, restoring consistency does nothing
- least change/least surprise properties of some kind: but NB there are many, many possibilities for such properties
- in rule-based systems, properties of being terminating/deterministic/confluent (i.e. causing consistency restoration to be a function
- some notion of reachability of states, or progress: in some settings we need to be sure to rule out the trivial consistency restoration which is "discard recent changes and return to a previous known-consistent state"
- some notion of non-redundancy, e.g. in a rule-based system, that each rule will actually sometimes need to be invoked

Examples of settings include

- systems of MGGs [1]
- a megamodel-based network of models connected by bx with consistency restoration in the network done with reference to an orientation model [2]

References

1 Frank Trollmann and Sahin Albayrak. *Extending Model Synchronization Results from Triple Graph Grammars to Multiple Models*. ICMT, LNCS 9765, pp. 91–106, Springer, https://doi.org/10.1007/978-3-319-42064-6_7, 2016.
2 Perdita Stevens. *Towards Sound, Optimal, and Flexible Building from Megamodels*. MoDELS, pp. 301–311, ACM, https://doi.org/10.1145/3239372.3239378, 2018.

4 Case Studies

4.1 Multidirectional Transformations for Microservices

Sebastian Copei (Universität Kassel, DE), Marco Sälzer (Universität Kassel, DE) and Albert Zündorf (Universität Kassel, DE)

4.1.1 Introduction

We would like to use theory and techniques from bidirectional transformations (BX) [6] for microservices. Especially, we look for microservices that share a certain amount of data with several other microservices and that need to synchronize on this data regularly. While our microservices share some common data, we want to be able to develop them as independently as possible, i.e. we want independent release cycles and separate internal data models. This allows to build these microservices by independent development teams and to have a data model that fits the purpose of each microservice, specifically. To achieve collaboration and data synchronization for our microservices, we need reliable techniques that require only minimal interaction between the development teams.

For our microservices we adapt two main ideas from the area of Domain Driven Design [3, 7] i.e. *Bounded Contexts* and *Event Sourcing*. Bounded Contexts propose that each microservice may use its own internal data model that fits its needs best. For example the Students' Office of Kassel University may have some data model for student data. The Students' Office stores your name and student id and your major topic and the history of courses and exams that you have completed. Some other microservice e.g. within the software engineering group of Kassel University might also deploy some student data. However, the SE Group is not interested in the other courses a student is doing or has done. Instead, the SE Group needs to store your achievements in various assignments during the term. While the SE Group manages Students' achievements, the Theory Group might run seminars and needs to manage presentations that have a different grading scheme. In addition, different research groups of Kassel University may share some data about which students work for which groups as Teaching Assistants (TA). In our example we use a dedicated TA Pool microservice where the TA data is synchronized.

In former times we would have used a huge monolith system with an enterprise or university wide common data model. This common data model would have to cover the needs of the central Students' Office as well as the needs of all research groups and also all the TA data. We might end up with a *Student* table that has some hundred columns and no one knows which of these columns are used for which purpose. To avoid this mess, Evans and Szpoton [3] propose bounded contexts. This means, instead of one common enterprise wide data model each group within the university just creates its own small data model that fits its own Bounded Context. While this avoids the problems of one big global data model, it creates the new problem of integrating all these Bounded Contexts' models.

To address the integration problem for bounded context, [3, 7] propose an event based communication between bounded contexts. To make this successful, [3, 7] also propose that each bounded context uses events internally to log all state changing operations. If you use such an event log also for persistence, i.e. for restoring the state of your system after restart we call this mechanism Event Sourcing.

The idea of this paper is to come up with a formalization for microservices that use Event Sourcing and with a formalization of bidirectional synchronization mechanisms for multiple microservices that share common data via Event Sourcing. From this formalization we derive design and implementation guide lines that shall result in reliable synchronization mechanisms.

In Section 4.1.2 we come up with a formalization of our ideas. In Section 4.1.3 we derive some guidelines for the design and implementation of event sourcing and event sharing for microservices. This is followed by a case study description in Section 4.1.4. There are still many open question to be discussed at our Dagstuhl seminar. Some of these questions are outlined in Section 4.1.5.

4.1.2 Formalizing Event Sourcing

First let us set up some basic notations: like [6] we use capital letters such as M, N for metamodels i.e. for sets of models (that adhere to a common class diagram). O denotes an empty model. We denote events with e and a set of events with E. An event $e = (t, x_1, ..., x_n)$ has the event type $t \in T$ and a number of parameters $x_i \in \mathbb{R} \cup \mathbb{CHAR}^*$, i.e. parameters are arbitrary numbers or strings. We will also use series of events $\overline{e} = (e_1, ..., e_n)$ and denote by $\overline{E}$ the set of all possible event series with events from E.

Events may be applied to (or synchronized with) models via function $\mathit{apply}(e, m)$, which generates a possibly new model m'. Furthermore, we define the application of an event series $\overline{e} = (e_1, \ldots, e_n)$ to a model m as $\mathit{apply}(\overline{e}, m) = \mathit{apply}(e_n, \ldots, \mathit{apply}(e_2, \mathit{apply}(e_1, m)) \ldots)$.

Now we are able to define Event Sourcing and some conditions on these. First of all we require that any valid model can be constructed by a series of events:

▶ **Proposition 1** (completeness). *For all models $m \in M$ exists a series of events $\overline{e} \in \overline{E}$ such that $m = \mathit{apply}(\overline{e}, O)$.*

This simply means, that for every possible model there exists a generating event series. With this we can understand *apply* formally as: If m is generated by an event series $\overline{e}$, then applying e on m means appending e to $\overline{e}$, written as $\overline{e} + e$, which in turn generates a possibly new model m'. Next, we require that you can go from one model to another via events:

▶ **Proposition 2** (undoability). *For all models $m_1, m_2 \in M$ exists a series of events $\overline{e} \in \overline{E}$ such that $m_2 = \mathit{apply}(\overline{e}, m_1)$.*

Note, this corresponds to *undoability* known for BX transformations: We can always find events that undo a modification and we can always go back to the empty model.

Two events that modify the same elements of a model m cancel each other, i.e. when both are applied, the effects of the second event overwrites all effects of the first event:

▶ **Definition 1.** Let $\overline{e_1}, \overline{e_2} \in \overline{E}$ be event series with $e \in \overline{e_1}$ and $\overline{e_2} = \overline{e_1} \setminus e$. We say that e is ineffective in $\overline{e_1}$ if $\mathit{apply}(\overline{e_1}, O) = \mathit{apply}(\overline{e_2}, O)$. Otherwise, e is called effective in $\overline{e_1}$.

Furthermore, we say that an event e' overwrites an event e in an event series $\overline{e}$ if e is effective in $\overline{e}$ (or respectively a corresponding model m) and ineffective in $\overline{e} + e'$.

Using modern messaging services, it is easy to guarantee that an event send from one microservice to another is received, but you usually have to prepare for multiple deliveries. Thus, if we apply an event a second time, it should have no effect:

▶ **Proposition 3** (hippocraticness). *For all models $m \in M$ and all events $e \in E$ holds that $\mathit{apply}(e, m) = \mathit{apply}(e, \mathit{apply}(e, m))$*

This corresponds to *hippocraticness* in BX.

Now we want events that keep models consistent: For the definition of consistency we introduce common event types $T_{com} \subseteq T$. An event s with type $t \in T_{com}$ is called *shared event*. The set of all possible shared events is denoted by S.

▶ **Proposition 4** (uniqueness). *For all shared events $s \in S$, for all models $m \in M$ and all $\overline{e} \in \overline{E}$ with $m = \mathit{apply}(\overline{e}, O)$ holds that $m = \mathit{apply}(s, m)$ if and only if s is effective in $\overline{e}$.*

We call Prop. 4 the *uniqueness* property of shared events: Each effective, shared event s is associated with a unique element in model m and only s is able to create that unique model element and if we apply s on m with no effect, than the only possible reason for not modifying m is that m already contains the corresponding unique effect due to an earlier application of s. In addition Prop. 4 requires that the application of s on m must have an effect, if (the effect of) s is not already contained in m. Thus, shared events are not veto-able.

Overall, we require that for any shared model element there shall be a unique shared event that creates or modifies it and all these shared events must be contained in the Event Source of the corresponding model. While these are pretty strong requirements (and we seek for some weaker condition here), it turned out to be quite straight forward to implement these requirements as discussed in Section 4.1.3.

In contrast to shared events all other events shall affect only the source model or only the target model:

▶ **Proposition 5.** *Let $m \in M$ be a source model and $n \in N$ be a target model. For all events $e \in E \setminus S$ follows that: If $apply(e, m) \neq m$ then $apply(e, n) = n$ and if $apply(e, n) \neq n$ then $apply(e, m) = m$.*

With $\triangle(\overline{e})$ we denote the set $\{e \mid e \in \overline{e}$ and e is effective $\}$ and $\triangle_s(\overline{e})$ the set $\{e \mid e \in \overline{e}$ and e is effective and shared$\}$. This means that $\triangle(\overline{e})$ is the set of effective events in $\overline{e}$ and $\triangle_s(\overline{e})$ is the subset of effective events in $\overline{e}$ that are also shared. Furthermore we understand the expression $\triangle_s(m)$ as the set of shared and effective events of a model $m \in M$, this means if $m = apply(\overline{e}, O)$ than $\triangle_s(m) = \triangle_s(\overline{e})$.

The set $\triangle_s$ is unique for a model m, i.e. any series of events that constructs m contains the same set of shared (effective) events:

▶ **Theorem 2.** *For all models $m \in M$ and all event series $\overline{e_1}, \overline{e_2} \in \overline{E}$ with $m = apply(\overline{e_1}, O)$ and $m = apply(\overline{e_2}, O)$ follows that $\triangle_s(\overline{e_1}) = \triangle_s(\overline{e_2})$.*

Proof. Let $m \in M$ be a model and $\overline{e_1}, \overline{e_2} \in \overline{E}$ be event series with $m = apply(\overline{e_1}, O)$ and $m = apply(\overline{e_2}, O)$. From Prop. 4 follows for all effective $s \in \triangle_s(\overline{e_1})$ that $m = apply(s, m)$. With the fact that $m = apply(\overline{e_2}, O)$ and the reverse direction of Prop. 4 follows that $s \in \triangle_s(apply(\overline{e_2}, O))$ which means $s \in \triangle_s(\overline{e_2})$. That all $s \in \triangle_s(\overline{e_2})$ are elements of $\triangle_s(\overline{e_1})$ can be shown in the exact same way. ◀

▶ **Definition 3.** Let $m \in M$ be a model with $m = apply(\overline{e_1}, O)$, and $n \in N$ be a model with $n = apply(\overline{e_2}, O)$. We say that m and n are consistent and write $m \simeq n$ if and only if $\triangle_s(\overline{e_1}) = \triangle_s(\overline{e_2})$.

Note that with Th. 2 the set $\triangle_s(m)$ is unique independent of the chosen $\overline{e}$, which means that this is well-defined. In other words, we consider two models consistent if and only if they contain (the effects of) the same set of shared events.

▶ **Definition 4.** Let $m \in M$ be a source model and $n \in N$ be a target model. We define $Get : M \times N \to N$ with $Get(m, n) := apply(\triangle_s(m), n)$ and $Put : M \times N \to M$ with $Put(m, n) := apply(\triangle_s(n), m)$.

Thus, *Get* computes the shared (effective) events contained in the source model m and applies them to the target model n. Similarly, *Put* applies the shared (effective) events of the target to source.

With these defintions we can show that our events fulfill the *GetGet* and *PutPut* rules:

▶ **Theorem 5.** *For all source models $m \in M$ and all target models $n \in N$ holds that $Get(m, Get(m, n)) = Get(m, n)$ and that $Put(Put(m, n), n) = Put(m, n)$.*

Proof. Let $m \in M$ be a source model and $n \in N$ a target model. Per Def. 4 follows that $Get(m, Get(m, n))$ equals $Get(m, apply(\triangle_s(m), n))$, which in turn equals $apply(\triangle_s(m), apply(\triangle_s(m), n))$. Due to our hippocraticness Prop. 3 this reduces to $apply(\triangle_s(m), n)$, which is per definition the same as $Get(m, n)$. The *PutPut* case can be shown in the exact same way. ◀

To show that our events guarantee the *GetPut* and *PutGet* rules we need the following lemma.

▶ **Lemma 6.** *For all models $m_1, m_2, m_3 \in M$ with $m_3 = apply(\triangle_s(m_1), m_2)$ holds that $\triangle_s(m_3) = \triangle_s(m_1) \cup (\triangle_s(m_2) \setminus \triangle_s(m_1))$.*

Proof. To proof Lem. 6 we show the cases $\subseteq$ and $\supseteq$ separately. Let $s \in \triangle_s(m_3)$, which means that s is an effective shared event in m_3. With the fact that m_3 was generated by applying all shared and effective events from m_1 to m_2 follows that $s \in \triangle_s(m_1)$ or $s \in \triangle_s(m_2) \setminus \triangle_s(m_1)$, because it was either in the model m_1 or in the model m_2 and not overwritten by the application because it is still effective. This obviously means that $s \in \triangle_s(m_1) \cup (\triangle_s(m_2) \setminus \triangle_s(m_1))$.

For the other direction let $s \in \triangle_s(m_1) \cup (\triangle_s(m_2) \setminus \triangle_s(m_1))$. Per assumption m_3 is generated by $\mathit{apply}(\triangle_s(m_1), m_2)$. This means that $\triangle_s(m_3)$ includes all effective, shared events that were in m_1 or m_2 and not overwritten in this application. Therefore $s \in \triangle_s(m_3)$, independent of its origin $\triangle_s(m_1)$ or $\triangle_s(m_2) \setminus \triangle_s(m_1)$. $\square$ ◀

With this we can proof that our events guarantee the *GetPut* and *PutGet* rules:

▶ **Theorem 7.** *For all source models $m \in M$ and all target models $n \in N$ holds that $\mathit{Put}(m, \mathit{Get}(m, n)) \simeq \mathit{Get}(m, n)$ and that $\mathit{Get}(\mathit{Put}(m, n), n) \simeq \mathit{Put}(m, n)$.*

Proof. To prove that $\mathit{Put}(m, \mathit{Get}(m, n)) \simeq \mathit{Get}(m, n)$ holds we have to show that $\triangle_s(\mathit{Put}(m, \mathit{Get}(m, n))) = \triangle_s(\mathit{Get}(m, n))$. By Def. 4 $\triangle_s(\mathit{Put}(m, \mathit{Get}(m, n)))$ is the same as $\triangle_s(\mathit{apply}(\triangle_s(\mathit{Get}(m, n)), m))$ and with Lem. 6 it can be seen that $\triangle_s(\mathit{apply}(\triangle_s(\mathit{Get}(m, n)), m))$ is the same as $\triangle_s(\mathit{Get}(m, n)) \cup (\triangle_s(m) \setminus \triangle_s(\mathit{Get}(m, n)))$. Here we can replace the second $\triangle_s(\mathit{Get}(m, n))$ with the use of Def. 4 and Lem. 6, which results in $\triangle_s(\mathit{Get}(m, n)) \cup (\triangle_s(m) \setminus (\triangle_s(m) \cup (\triangle_s(n) \setminus \triangle_s(m))))$. It can be seen, that the right subterm is equivalent to O, because we subtract from $\triangle_s(m)$ the set $\triangle_s(m) \cup (\triangle_s(n) \setminus \triangle_s(m)$, which is obviously a superset of $\triangle_s(m)$. But this means that $\triangle_s(\mathit{Get}(m, n)) \cup (\triangle_s(m) \setminus (\triangle_s(m) \cup (\triangle_s(n) \setminus \triangle_s(m)))) = \triangle_s(\mathit{Get}(m, n))$. Thus if you go back to the beginning of this proof it says $\triangle_s(\mathit{Put}(m, \mathit{Get}(m, n)))$ equals $\triangle_s(\mathit{Get}(m, n))$. The GetPut rule can be proven analogous. $\square$ ◀

Note, $\mathit{Get}(m, n)$ and $\mathit{Get}(\mathit{Put}(m, n), n)$ are usually not equal as the *Put* may overwrite some shared events in m and thus a subsequent *Get* applies fewer events to n. Thus, if you have two models synchronizing with *GetPut* is not equal to synchronizing with *PutGet*. In both cases the models will be synchronized but in the *GetPut* case the source model will still have all its old shared events (plus some from the target model). In the *PutGet* case the source model will loose some shared events that are overwritten by events of the target model but it will get all shared events from the target model. Altogether we consider two models as consistent if they contain equivalent sets of shared events in their Event Sourcing history.

For BX the differences between GetPut and PutGet just correlate to preferences for the handling of conflicting changes (i.e. shared events that overwrite each other). For MX we need to take additional care: let us assume we have three microservices named O (for the Students' Office, E (for the Software Engineering Group), and T (for the Theory Group). Let us now assume there is a student with *studentId* $m4242$ and microservice E has named this student *Alice* while microservice T has named this student *Alexa*. If we do a $\mathit{Get}(\mathit{Put}(O, E), E)$ microservices O and E share the name *Alice* while T still believes in *Alexa*. If we now do a $\mathit{Get}(\mathit{Put}(O, T), T)$ O and T agree on *Alexa* while E still believes in *Alice*. We may do these two synchronizations as often as we want, we do not reach global consistency.

We could achieve global consistency in above case easily, if one or both synchronizations would use a *PutGet* instead of a *GetPut*. This would introduce some kind of priority ordering for the microservices such that conflicting events do not travel in opposite order. However such a global organization scheme for a (large) set of microservices requires a lot of coordination of the different development teams that develop the individual microservices, independently.

In order to avoid the need for a global coordination scheme, in our implementation we use time-stamps in order to number the events within each Event Source, cf. Section 4.1.4. Then, before applying an (external) event, we check our Event Source for a conflicting events. In case of a conflict, the latest event wins, i.e. if an event arrives late, it is ignored. We still need to incorporate this strategies into our formalization.

From our consistency conditions we now derive properties for the implementation of our Event Sourcing and synchronization mechanisms.

4.1.3 Implied Properties and Guidelines

Proposition 1 (completeness) requires that any valid model can be created by applying a series of events. In implementing a microservice A this requires that our microservice offers a dedicated API (application programmer's interface) for creating and modifying its internal data model. The internal data model shall be modified only via this API and each API method (invocation) that actually modifies the internal model shall raise an event with an event type and parameters corresponding to the invocation of the API method. Applying an event thus corresponds to the invocation of the corresponding API with the contained parameters.

Proposition 2 (undoability) requires that for any change done our API shall provide appropriate reverse methods that allow to undo that change. Thus, if we have an API operation that e.g. adds a student to a university, we shall also provide a possibility to remove that student from that university. While this is a reasonable property for APIs, it is not mandatory for consistency.

Proposition 3 (hippocraticness or idem potent) requires that any API method has to check whether the internal model already contains the intended model changes and the API method executes the change only if it is not yet contained. Note that our events have only parameters of types number or string. We may not pass references to model objects as parameters. This allows us to use events for persistence and to send events via some messaging service. Having only parameters of number or string types require that these parameters have to form some kind of *primary key* that allows to identify the model elements to be modified or that have already been modified, uniquely. Generally, each API method must implement some kind of *getOrCreate* semantics thus calling it the first time new model elements are created and thereafter these elements are just retrieved.

Proposition 4 (uniqueness) requires that two microservices A and B that share some events of certain event types c_i must provide corresponding API methods $A :: m_i()$ and $B :: n_i()$ that use exactly the same parameters as provided by the events of the corresponding types and these API methods raise exactly the events of the corresponding types and with the corresponding parameters (when they result in an model modification). Although the shared API methods need to stick to a common parameter signature they may implement the corresponding changes differently in their internal models. We only require that the uniqueness Proposition 4 is met, i.e. when we receive a shared event we must be able to tell whether we already contain the corresponding changes or not. This means, we need to guarantee the one to one correspondence between shared events and affected model elements.

Proposition 4 (non veto-able) also requires that the shared events must always work. Thus, if we have e.g. a shared event *(examCreated, "modelling", "2019-03-12")* this creates an exam for a modelling course. If the modeling course not yet exists, we must create it on the fly by calling the appropriate API method. In order to do this, the *examCreated* event must provide sufficient parameters to denote the course uniquely and sufficient parameters to be able to call the corresponding API method. Note, in our example the creation of the modeling course will internally generate an *(courseCreated, "modeling")* event. When this event arrives later, we ignore it as its effects have already been achieved.

From our experience, the discussed implementation requirements provide a pretty good guide line for the design and implementation of a micro service and the API of its model. The developers have quite a freedom in designing their models and their API and to develop their microservices, independently. When integrating two microservices the developers have to agree *only* on the signatures of the shared API methods.

Usually, the microservices have a customer supplier relationship [3], i.e. the supplier microservice predefines the shared API and the customer microservice adapts the shared API methods. Ideally, the customer microservice just adapts some of its existing API methods to meet the signature of the shared supplier API methods. Sometimes, the customer will also need to extend or restructure its API and to extend and restructure its internal model, accordingly.

For multiple microservices, we also need to take care of event conflict handling strategies that guarantee global consistency. A simple way is to apply events not just in order of arrival but ordered by some time stamp. Such a functionality is easily provided by a generic Event Source implementation.

In practical cases, we also need to take care of access rights, not every microservice might be allowed to call every API method on every other microservice (or to send the corresponding Event) and not every microservice might have the right to read all data (or to receive all Events) of all other microservices. In our example implementation, each microservice has a REST interface that handles *Get* and *Put* requests. This REST interface is the place, where we handle all access rights. By controlling which events we send out to whom and which of the received events we apply to our internal data model, the REST interface as a good place to handle all access right issues for a given microservice.

Altogether, we believe that our consistence conditions provide very good and easy to follow guide lines for the implementation of microservices that share some common data. We would like to extend this work with compile time checks and validation mechanisms that allow to guarantee data consistency between microservices.

In the next section we discuss a simple case study that uses our ideas.

4.1.4 Student Affairs Sample Implementation

Our our Student Affairs case study employs four microservices that all deal with student data at Kassel University, cf. Fig. 3. The *Students' Office* deals with course programs and all the examinations of the students. The *SE Group* deals e.g. with assignments in the modelling course. The *Theory Group* provides a specific grading scheme for seminar presentations. And the two research groups exchange data on Teaching Assistant students via the *TA Pool*. Each microservice is developed independently and uses its own bounded context data model [3]. As shown in Fig. 3 the microservices use Event Sourcing [3] to store data persistently. Each microservice also provides an API that is used by the corresponding GUIs as well as for loading and logging events as well as for the synchronization of the microservices. For example, each time a student enrolls for a certain examination within the Students' Office, a corresponding event is raised and added to the Students' Office's Event Source. At any time, e.g. the SE Group may issue a *getEvents* request causing the Students' Office to respond with all *studentEnrolled* events referring to courses run by the SE Group. The SE Group may now do the grading of these students with the help of the students' performance data gathered locally. Each grading operation will raise and record a *studentGraded* event within the SE Group microservice. After the grading, the SE Group

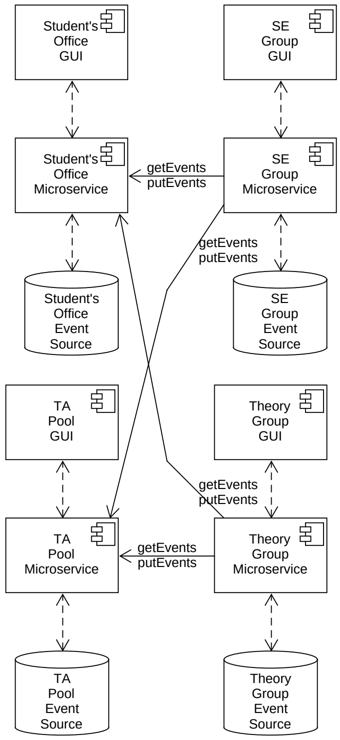

Figure 3 Example Architecture.

may submit the *studentGraded* events (that of course include the achieved grades) to the Students' Office via a *putEvents* request.

Similarly, the SE Group may hire some of its (excellent) students as Teaching Assistant. The corresponding *studentHired* event may then be send to the TA Pool. Then the Theory Group may retrieve all *studentHired* events. Thus the research groups may avoid to hire the same student twice.

Instead of a class diagram, Figure 4 shows a simple object diagram for an example state / model of our Students' Office microservice. Our Students' Office deploys objects of type *UniStudent* to represent students, e.g. object *alice4*. *alice4* has *CS* as major subject. *StudyProgram CS* contains courses on *math* and *modeling*. For the *modeling* course the Students' Office has scheduled an examination *e6* on *2019-03-19*. *alice4* has enrolled for this examination as indicated by object *e9*.

Figure 5 shows an example object model for our SE Group microservice. In this situation, the data from our Students' Office has already been integrated and the SE Group has already graded the single student of its *modeling* class. The SE Group microservice does not store student's names but only their *studentId*. Thus our student Alice is represented by object *s3* of type *SEStudent*. In the SE Group model, students have *Achievement* objects like *a7* that model their participation in an *SEClass* like the *modeling* class *s2* of term *2018-10*. The *SEClass s2* has two *Assignments a5* and *a6*. Students hand in their *Solutions* like *s8* and *s9*. The *Solutions* of one student are collected under the corresponding *Achievement* object. In our example the *Solutions* have already been graded by some achieved points. From the points achieved by their *Solutions*, a grade is computed and stored within the corresponding *Achievement* object. In our case Alice has scored an *A* in *modeling*.

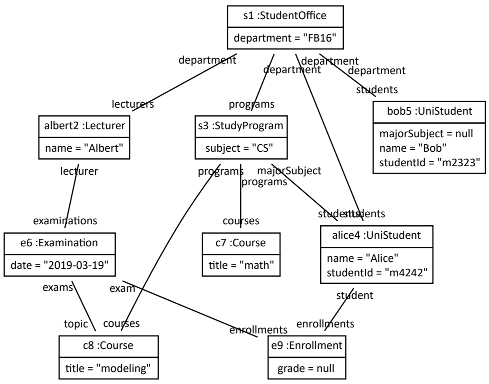

Figure 4 Object Diagram for Student's Office.

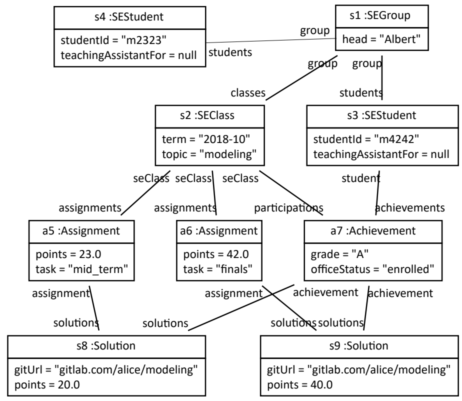

Figure 5 Object Diagram for SE Group.

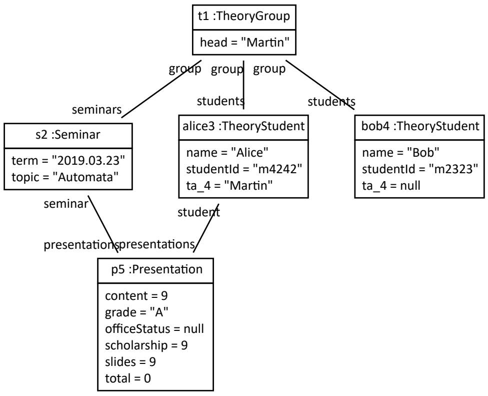

Figure 6 Object Diagram for Theory Group.

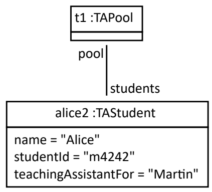

Figure 7 Object Diagram for Theory Group.

The Theory Group of Kassel University runs e.g. a seminar on *Automata* at *2019.03.23*, cf. Figure 6. Our student Alice has given a *Presentation* in this seminar and scored 9 points for the *content*, the *scholarship*, and the *slides*, each. *Martin*, the head of the Theory Group has hired *Alice* as Teaching Assistant for his group, cf. attribute *ta_4* of object *alice3*.

Finally, Figure 7 shows the object model of the TA Pool microservice used by the research groups in order to exchange data on Teaching Assistants. Currently the TA Pool only records that *Alice* works for *Martin*.

As you may have noticed, our four microservices deploy quite different data models. There is some data in the Students' Office that SE Group and Theory Group do not care about e.g. study programs. Vice versa, the SE Group deploys *Assignments* and *Solutions*, that do not bother the Students' Office. For some other data there are correspondences, e.g. *Course*s and *Examinations* from the Students' Office correspond to *SEClass*es within SE Group and to *Seminar*s in the Theory Group. And of course, *UniStudents* correspond to *SEStudents* corresponds to *TheoryStudent* corresponds to *TAStudent*. A less obvious correspondence exists between *Enrollment* and *Achievement* objects and *Presentation* objects.

The challenge for our case study is to keep the corresponding data of the four microservices consistent. Listing 1 shows method *applyEvents* of our Students' Office microservice. Listing 2 shows some example Yaml [2] string for shared events as it may be passed to the *applyEvents* method. (Actually, this Yaml string has been derived from shared events raised through the *getOrCreateStudent* and *enroll* method of the Students' Office's API.)

Method *applyEvents* uses a *yamler* to *decode* the *yaml* string into a *List* of *Map*s. Each such *map* contains the tag values of one event. Thus, the for loop iterates over all events and for each event we run through a chain of if else if statements. As soon as the *eventType* matches a certain case e.g. *studentCreated*, the corresponding API operation is called, e.g. *getOrCreateStudent*. The additional event parameters are passed to the API method in order to denote the exact model element to be affected.

In case of a *studentEnrolled* event, the Students' Office wants to call API method *enroll(student, exam)* passing an *UniStudent* and an *Examination* object as parameters, cf. last statement of Listing 1. To retrieve the *student* parameter we use the *studentId* passed within the *studentEnrolled* event. Similarly, we use the *courseName* to call *getOrCreateCourse* and *lecturerName* in order to get or create the responsible lecturer. With *course* and *lecturer* and *date* we call *getOrCreateExamination* to create or retrieve the required examination object. Finally, we are ready to call *enroll(student, exam)*.

Note, if methods *getOrCreateStudent*, *getOrCreateExamination*, or *enroll* actually create internal objects, then the corresponding event is raised and added to the internal event source of our microservice. Similarly, if one of the API methods is called via the graphical user interface, the corresponding events are automatically added to the event source of our microservice. A careful reader may have noticed that Listing 1 and Listing 3 employ two variants of method *getOrCreateStudent*, one with *studentId* and student *name* as parameters and one with only *studentId* as parameter. The reason is that some shared events refer to the affected student only via the *studentId*. If the corresponding *studentCreated* event did not yet reach us, we still need to create a student with the corresponding *studentId*. Usually, creating a student requires to provide the student's name, too. If we do not have the student *name* at hand, we just create an half done student object assigning only the *studentId*. When later on the *studentCreated* event with the corresponding *studentId* and with the required *name* arrives, we finish the construction of the student object and only then we raise the internal *studentCreated* event to be added to the local Event Source.

Listing 3 shows method *applyEvents* of our SE Group microservice. The SE Group microservice handles *studentCreated* events quite similar to the Students' Office. However the *getOrCreateStudent* API method of SE Group internally creates an *SEStudent* object and drops the student's *name*. Despite SE Group does not store the student's name, internally, the method handling the *studentCreated* event within method *applyEvents* of SE Group requires a *name* as parameter. Having the student's name here became mandatory due to our requirements on shared events: If we create a new *SEStudent* object within our SE Group microservice we must eventually raise the corresponding shared *studentCreated* event that will be used to synchronize with the Students' Office. As the Students' Office stores student's names, the corresponding shared event needs to provide this information and thus the SE Group has to collect this information. Thus, although the SE Group does not store student's names in its *SEStudent* objects, it must store student's names in its internal event source in order to retrieve valid *studentCreated* events for the synchronization with the Students' Office.

The SE Group microservice deals with shared events in quite the same way as the Students' Office does. Mainly, the SE Group uses other names for similar things. However, the first version of the SE Group API did not have an *enroll* operation nor did it support *studentEnrolled* events, at all. This part has been added for synchronization purposes later on. SE Group already had a *getOrCreateClass* method and when it came to microservice integration it became clear that *SEClass* objects corresponds to *Examination* objects, somehow. Both represent that a certain course is given in a certain term. Similarly, SE Group *Achievements*

```
public void applyEvents(String yaml){
    Yamler yamler = new Yamler();
    List<Map<String, String>> list
       = yamler.decodeList(yaml);
    for (Map<String, String> map : list) {
        ...
        else if (STUDENT_CREATED
                .equals(map.get(EVENT_TYPE))) {
            getOrCreateStudent(map.get(NAME),
                map.get(STUDENT_ID));
        }
        else if (STUDENT_ENROLLED
                .equals(map.get(EVENT_TYPE))) {
            UniStudent student =
                getOrCreateStudent(
                    map.get(STUDENT_ID));
            Course course =
                getOrCreateCourse(
                    map.get(COURSE_NAME));
            Lecturer lecturer =
                getOrCreateLecturer(map,
                    LECTURER_NAME);
            Examination exam =
                getOrCreateExamination(course,
                    lecturer, map.get(DATE));
            enroll(student, exam);
        }
        else if (EXAMINATION_GRADED
                .equals(map.get(EVENT_TYPE))) {
            UniStudent student =
                getOrCreateStudent(
                    map.get(STUDENT_ID));
            Course course =
                getOrCreateCourse(
                    map.get(COURSE_NAME));
            Lecturer lecturer =
                getOrCreateLecturer(map,
                    LECTURER_NAME);
            Examination exam =
                getOrCreateExamination(course,
                    lecturer, map.get(DATE));
            Enrollment enrollment =
                enroll(student, exam);
            gradeExamination(enrollment,
                map.get(GRADE));
            }
        ...
    }
}
```

Listing 1 applyEvents method of our Student's Office.

```
- eventType: studentCreated
  studentId: m2323
  name: Bob
  .eventNumber: 7

- eventType: studentCreated
  studentId: m4242
  name: Alice
  .eventNumber: 8

- eventType: studentEnrolled
  studentId: m4242
  courseName: modeling
  lecturerName: Albert
  date: "2019-03-19"
  .eventNumber: 10
```


Listing 2 Yaml Encoded Shared Events.

correspond to *Enrollments* within the Students' Office. However, due to regulations in Kassel University, students may attend a class and do some assignments in one term and they may enroll for the official examination within another term. Thus, the SE Group may have *Achievement* objects that do not (yet) correspond to *Enrollment* objects and we shall not send *studentEnrolled* events to the Students' Office each time an *Achievement* is created. We solved this problem by extending our *Achievement* objects with an *officeStatus* attribute, cf. Figure 5. This attribute is *open* on creation of an *Achievement* and it changes to *enrolled* when the corresponding *studentEnrolled* event is applied.

The SEGroup also has a *hireStudent(stud, lecturerName)* API method that marks the corresponding student as Teaching Assistant for the given lecturer. This API method raises a shared *studentHired* event that may be send to the TA Pool. The Theory Group has a similar implementation of method *applyEvents* that matches events to the API of the Theory Group.

To meet the requirements of Section 4.1.2 we need to guarantee that methods are Hippocratic or idem potent, i.e. if we call a *getOrCreate* operation twice with the same parameters the second call will only retrieve the corresponding model element but not modify it. To achieve this we use certain event parameters as primary keys. For example, if we call *getOrCreateStudent("Alexa", "m4242")* then the student id *m4242* serves as unique key. If we call *getOrCreateStudent("Alexa", "m4242")* again, method *getOrCreateStudent* will find the student object with *studentId m4242* and return it. In addition, method *getOrCreateStudent* will assign the name *"Alexa"*, again. As this does not change the underlying student object, that is OK. If we call *getOrCreateStudent("Alice", "m4242")* now, again the underlying student object is retrieved and the new name is assigned. However, this time the underlying model element is changed and thus a new *studentCreated* event is raised and added to the internal event source. This new *studentCreated* event overwrites the previous *studentCreated* event that has the same *studentId* but the former name was *"Alexa"*. With this implementation we meet Prop. 4 and this guarantees data consistency between our two microservices. Note, our implementation achieves uniqueness for *studentIds* by construction. Whenever you use *studentId m4242* it refers to the student object. We never create two student objects with the same *studentId*.

```
public void applyEvents(String yaml){
   Yamler yamler = new Yamler();
   List<Map<String, String>> list
      = yamler.decodeList(yaml);
   for (Map<String, String> map : list) {
      ...
      else if (STUDENT_CREATED
            .equals(map.get(EVENT_TYPE))) {
         getOrCreateStudent(map.get(NAME),
            map.get(STUDENT_ID));
      }
      else if (STUDENT_ENROLLED
            .equals(map.get(EVENT_TYPE))) {
         SEStudent student =
            .getOrCreateStudent(
               map.get(STUDENT_ID));
         SEClass seClass =
            .getOrCreateSEClass(
               map.get(COURSE_NAME),
               map.get(DATE));
         Achievement achievement =
            getOrCreateAchievement(student,
               seClass);
         enroll(achievement);
      }
      else if (EXAMINATION_GRADED
            .equals(map.get(EVENT_TYPE))) {
         SEStudent student =
            getOrCreateStudent(
               map.get(STUDENT_ID));
         SEClass seClass =
            getOrCreateSEClass(
               map.get(COURSE_NAME),
               map.get(DATE));
         Achievement achievement =
            getOrCreateAchievement(student,
               seClass);
         gradeExamination(achievement);
      }...
   }
}
```

Listing 3 applyEvents method of our SE Group.

Similarly, the *title* of a *Course* serves as primary key for *Course* objects and the *name* of a *Lecturer* serves as key for *Lecturer* and *SEGroup* objects. As there may be multiple *Examinations* scheduled for the same day and as a *Course* may have multiple *Examinations* in different terms, you need the *title* of the corresponding *Course* and a *date* in order to identify an *Examination*. We assume that there is only one *Examination* per *Course* and term, thus the SE Group uses the *Examination* date to find the *SEClass* for that term i.e. we use the *SEClass* with a *term* attribute that is less than half a year in front of the *Examination*. Thus, we are able to synchronize *Examinations* and *SEClasses* without sharing the exact

Examination date or *term* start. While this is unnecessary complicated, it exemplifies how two microservices may choose their own model design and how we achieve bidirectional model synchronization for models with varying design.

In our Student Affairs example our microservices synchronize via REST APIs offered by the Students' Office and the TA Pool. At any time, the SE Group or the Theory Group service may initiate an http based *getEvents* call to e.g. the Students' Office, cf. Figure 3. Then the Students' Office goes to its Event Source and retrieves the current list of shared events. This list is encoded as Yaml string and returned to the calling micro service. The calling micro service then calls its *applyEvents* method and the *getEvents* call is completed. Now e.g. the SE Group may ask its own Event service to retrieve the shared events known by SE Group. At this time, this will include all just received and applied shared events that have had an effect on the model of the SE Group. Now we decode the SE Group shared events as Yaml string and send it over to the Students' Office via an http based *putEvents* call. This causes the Students' Office to call its *applyEvents* methods. The shared events that have been send to the SE Group and that have now been send back will not cause any modifications within the Students' Office data model. Only the shared events that have been raised by the SE Group genuinely will be incorporated into the Students' Office data model (and Event Source). Now both microservices are synchronized. Guaranteed. (If we did the implementation right.)

In order to minimize communication, our Event Sources number all events with time stamps (milliseconds). Thus, once we received (and incorporated) an event with e.g. time stamp *42424242*, the next time we do a *getEvents* call we are interested only in new shared events. Therefore, our http based *getEvents* call has a *lastKnownTimeStamp* parameter telling the Students' Office to retrieve and send only later events. Similarly, the SE Group keeps track of the *lastKnownTimeStamp* it has send to the Students' Office and sends only newer Events on subsequent *putEvents* calls. Note, if we receive e.g. a *studentCreated* event with time stamp *42424223* at the SE Group and we do not yet have a student with that *studentId*, the SE Group will create a new *SEStudent* and raise a new *studentCreated* event that will get a new time stamp e.g. *42424246* in the Event Source of the SE Group. This re stamping of events works as we use two different *lastKnownTimeStamp*s: one for the last event that the SE Group has received and another for the last event that we have send. To avoid the problem that the SE Group and the Theory Group send different name updates for student *m4242* in an alternating fashion, before applying an event we additionally ask our Event Source if it already has an event of the same type with the same primary keys that has a larger / later time stamp. In this case the earlier event shall be overwritten by the already arrived later event and thus we do not apply the earlier event. Thus, our conflict is resolved in favor of the event that has the larger / later time stamp or in other words the last edit wins. To make this work, in our implementation each shared event needs an explicit key that identifies the object that is finally edited and two events with the same key overwrite each other with the policy 'last edit wins'.

To summarize, in our implementation it was surprisingly easy to meet the pretty strong requirements of Section 4.1.2. Actually, the formal requirements gave us clear design guidelines for the development of our microservices. First of all, Prop.1 and Prop. 3 forced us to develop a model API totally with *getOrCreate* methods. This also forced us to design our model and events such that we have sufficient primary keys for all relevant model elements. Here it was very helpful, that the *getOrCreate* methods implement a very constructive approach to primary keys. The *getOrCreate* methods just deliver only one element for the primary key information you provide. For example, if we use only the *date* to denote an *Examination*, it is not possible to have different *Examination*s on the same day any more. This becomes evident in test scenarios pretty soon. Thus you may add the *Course name* to the primary key of an examination. This suffices to denote the *Examination*, uniquely. In our example we also use a *Lecturer name* within the *examinationCreated* event. This is used to connect the new *Examination* to the responsible *Lecturer*. It may also be used to filter *studentEnrolled* events to be send only to the research group that is responsible for the corresponding class in that term (or to the research group with the head named as lecturer).

The need to solve alternating changes of the same attribute caused us to add another event key that tells the Event Source to overwrite the earlier event with the later event. This again is easy to test by doing alternating changes to a certain attribute and by then asking the Event Source for the list of all stored events. If the conflict is detected only the later event is retrieved.

Prop. 2 challenged our implementation as you are not allowed to reject (veto) a shared event. If you receive a shared event you must incorporate its effects in your model in order to stay consistent. This may become a problem if events build upon each other e.g. if the application of a *studentEnrolled* event requires that the corresponding *SEClass* must already exist. To avoid such a dilemma our implementation just uses the *getOrCreate* methods to retrieve the required *Examination* or *SEClass*. Thus, if the *Examination* (or *SEClass*) does not yet exist, we create it on the fly. On the downside, this requires the *studentEnrolled* event to provide sufficient parameters to create an *Examination* (or an *SEClass*) on the fly, if necessary. This is the reason for including the *Lecturer name* in the *studentEnrolled* event.

After studying the Student Affairs case there are still a lot of

4.1.5 Open Questions

At the Transformation Tool Contest 2017 we studied the Families 2 Persons case [1, 8]. We would really like to go through this case and variants of this case from the perspective of Event Sourcing. We expect that our formal requirements give good guidelines on how the case needs to be extended or modified in order to achieve guaranteed consistency. For example, the Families to Persons case uses family names to identify family objects. However, there may be multiple families with the same family name. According to our requirements this is not allowed. Thus, you need to extend the primary key for families by some additional information. You may e.g. add the parents' first names to the key for families. Thus it would be family *Homer and Marge Simpson*. Accordingly a person like *Bart* would have *Bart (Homer and Marge) Simpson* as full name. However, it would become pretty complicated when Bart marries and creates his own family.

On the 2017 BX Workshop in Shonan, Michael Johnson presented a case where a manufacturing enterprise wanted to exchange customer data with a marketing enterprise. The main concern in this case is security: The manufacturing enterprise does not want to give the marketing enterprise full access to its technical documents. However, access to a cutout of customer data is OK. We would like to investigate whether our shared event based collaboration is an easy means to control which data is accessed by other microservices. Actually, the *getEvents* method provided by our Students' Office does already a lot of filtering. For example, you cannot access the grades of students of courses that have not been given by your group. From a software engineering point of view, method *getEvents* is the single access point for other microservices and it is also a good place in order to control access rights and to control which information is send to which partner.

We believe that there is a close correspondence between our shared events and triple graph grammar rules [5]. Each shared event is implemented by two methods one in the source microservice and one in the target microservice. These two methods correspond to the left and right rule of a triple graph grammar rule and the shared event itself corresponds to the middle rule. With triple graph grammars a forward transformation requires to collect all occurrences of the left sides of the triple rules that have not yet been transformed towards the target model and then to apply the corresponding right sides of that rules. This is quite similar to collecting shared events in our Event Source and to send those shared events to the target model and to apply them there. This correlations needs further investigation.

We really want to use the Dagstuhl meeting in order to discuss the relationship of our Event Sourcing based theory with traditional BX theory in further detail. How does our approach fit into the existing literature and what are related ideas. Somehow, our formalization proposes that the API of a microservice is a set of functions that together may be interpreted as a kind of meta model for the microservice's internal states. Each API function forms an editing operation and altogether these editing operations describe the set of all reachable models. Similarly, each model is created by a series of API function calls. And these function calls have algebraic properties like idem potent and commutative (if all function calls are effective). And together these algebraic properties guarantee BX behavior. To come up with more reliable MX behavior we may need to add time stamps to our theory, but we are not sure that this finally solves all these issues.

We would really like to provide compile time verification means in order to ensure that the implementations of two microservices that share some events are correct and do guarantee consistency. As verification is probably difficult within plain old Java code, we might require that the event handling API is implemented using some model transformation language. A high level model transformation language might simplify the verification of properties like hippocraticness and uniqueness. We might also add some explicit declaration of primary keys to the specification of shared events in order to leverage this information in correctness verification tasks.

We are looking forward to the discussions at Dagstuhl.

References

1. A. Anjorin, T. Buchmann, and B. Westfechtel, *The Families to Persons Case*, in Proceedings of the 10th Transformation Tool Contest, a part of the Software Technologies: Applications and Foundations (STAF 2017) federation of conferences (A. Garcia-Dominguez, G. Hinkel, and F. Krikava, eds.), CEUR Workshop Proceedings, CEUR-WS.org, http://ceur-ws.org/Vol-2026/paper2.pdf, 2017.
2. Oren Ben-Kiki, Clark Evans, and Brian Ingerson. *YAML: YAML Ain't Markup Language (yaml^{TM}) version 1.1.*, Tech. Rep: 23, https://yaml.org, 2005.
3. Eric Evans and Rafał Szpoton. *Domain-Driven Design*. Helion, 2015.
4. Michael Johnson, Perdita Stevens, *Confidentiality in the process of (model-driven) software development*. Programming 2018: 1–8, https://doi.org/10.1145/3191697.3191714, 2018.
5. Andy Schürr. *Specification of graph translators with triple graph grammars*. International Workshop on Graph-Theoretic Concepts in Computer Science. https://doi.org/10.1007/3-540-59071-4_45, Springer, 1994.
6. Perdita Stevens. *Bidirectional model transformations in QVT: Semantic issues and open questions*. Software and System Modeling 9(1): 7-20, https://doi.org/10.1007/s10270-008-0109-9, 2010.
7. Vaughn Vernon. *Implementing domain-driven design*. Addison-Wesley, 2013.
8. Zündorf, Albert, and Alexander Weidt. *The SDMLib Solution to the TTC 2017 Families 2 Persons Case*, http://ceur-ws.org/Vol-2026/paper9.pdf, 2017.

4.2 Multidirectionality in Compiler Testing

Vadim Zaytsev (Raincode Labs, BE)

A classical software language processor can be viewed as a chain of transformations, most of them even unidirectional, going through most of the following intermediate artefacts [1]:

- Program text
- Preprocessed program text
- Parse tree as a structural model of a program
- Abstract syntax graph as a conceptual model
- Annotated graph with types and other information
- Code model suitable for optimisations
- Executable code
- Computation result

Each of these artefacts/models conforms to a different metamodel. Examples of bidirectional transformations in this chain, are:

- Error correction facilities [2], where a "later" and more rich artefact can be used to point out errors in an "earlier" and more primitive artefact, such as misplaced punctuation or parenthesis in the text of the program.
- Semantic-driven disambiguation [3], where the structure or a model of a program can only be decisively determined after semantic analysis. The need and necessity for such techniques is caused by having constructs like "`x * y`" in C (which can either mean declaring a variable `y` typed as a pointer to a value of type `x`, or a multiplication of two variables named `x` and `y`), dangling clauses in COBOL (a language where it is not always straightforward to determine where one statement ends and the next one begins), offside rule in languages like SASL, Python or Haskell (where statement affiliation with a block depends on the indentation of a piece of code), as well as various ambiguities in 4GLs caused by bad language design [4].
- Incremental techniques where the change that needs to be propagated in either direction, is several orders of magnitude smaller than the entire model. For example, many legacy systems have flat hierarchies of interlinked and intercommunicating entities spanning over millions of lines of code, but the evolution they undergo on a daily basis, covers small scale bug fixes, rarely even multiline. Implementations of incremental synchronisation techniques usually involve some sort of bx.

At Raincode Labs, which is commonly employed as a team of compiler mercenaries, we are being asked to implement some of these features regularly, so having some bx is a norm rather than something exotic.

A typical compiler test is a tuple, which elements correspond to some of the artefacts listed in the beginning of this section. In the simplest case, it is a tuple with a program text and its expected execution results. However, such simplistic test cases are only useful with mature projects [5]. Compilers under active development require a much more elaborate framework for testing, capable of forming hypotheses, crystallising them as specifications and testing them differentially on available oracles (such as working legacy implementations or remaining living domain experts). It is not uncommon for such a test spec to include all or almost all of the artefacts, allowing for testing whether the parser could recognise the input as correct, whether it succeeded building a proper parse tree, whether in its turn a corresponding syntax graph was constructed correctly, etc, all the way to the execution of the compiled code and comparing the result with the baseline [4, 5]. In practice it helps enormously to have the capability to locate the exact point of failure.

So, since a test case is an n-tuple, a collection of them (known as a test suite) can be seen as a specification of an n-ary relationship. When it gets broken (by a change in a compiler, or, even more commonly during development, by the customer providing additional information that conflicts with the contemporary understanding of the intended language semantics), it needs to be restored, and that can/should be done by a multx. In general, all connected artefacts are needed as inputs to make a consistency restoration decision, and all of them have a chance to be changed as its result.

Unfortunately, the state of the art is to accomplish this with a combination of manual programming and bespoke proprietary tools. The main intention behind exposing this case study during the seminar as well as in this report, is to provide a somewhat detailed description of an open problem that seems suitable to be solved with multx.

References

1 Vadim Zaytsev, Anya Helene Bagge. *Parsing in a Broad Sense*, Proceedings of the 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS), LNCS 8767, pp. 50–67, https://doi.org/10.1007/978-3-319-11653-2_4, Springer, 2014.

2 Maartje de Jonge, Lennart C. L. Kats, Eelco Visser, Emma Söderberg. *Natural and Flexible Error Recovery for Generated Modular Language Environments.* ACM Transactions on Programming Languages and Systems, 34(4): 15:1-15:50, https://doi.org/10.1145/2400676.2400678 2012.

3 Mark van den Brand, Steven Klusener, Leon Moonen, Jurgen J. Vinju. *Generalized Parsing and Term Rewriting: Semantics Driven Disambiguation.* Proceedings of the Third Workshop on Language Descriptions, Tools and Applications (LDTA), ENTCS 82(3), https://doi.org/10.1016/S1571-0661(05)82629-5, 2003.

4 Vadim Zaytsev. *Open Challenges in Incremental Coverage of Legacy Software Languages.* Proceedings of the Third Edition of the Programming Experience Workshop (PX/17.2), pp. 1–6, https://dl.acm.org/citation.cfm?id=3167105, 2017.

5 Vadim Zaytsev. *An Industrial Case Study in Compiler Testing.* Proceedings of the 11th International Conference on Software Language Engineering (SLE), pp. 97–102, https://doi.org/10.1145/3276604.3276619, ACM, 2018.

4.3 Bringing Harmony to the Web

James Cheney (University of Edinburgh, GB)

A driving vision of the Harmony project [1] was to support synchronization of (differently) structured data to bridge and integrate data on the Web. A substantial amount of data is stored in relational databases (or graph databases that are gradually reinventing relational databases). Synchronizing such data in a principled way requires foundations. Schema mappings [2] are one well-explored approach to relational data integration, while bidirectional techniques have seen much less attention, including initial steps such as relational lenses by Bohannon et al. [3]; however, that work constituted a theoretical development without an efficient implementation. In recent work, Horn et al. [4] showed how to implement relational lenses efficiently using incrementalization. However, much remains to be done to build this

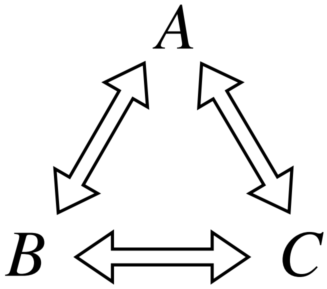

Figure 8 A multx with pairwise relations.

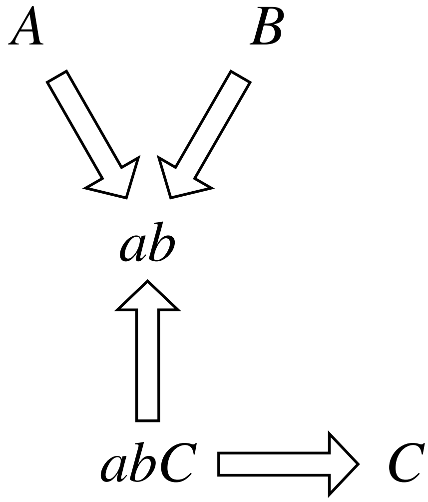

Figure 9 A multx with a T-shape.

approach into a principled and efficient approach to synchronizing multiple large-scale Web data sources. Among the many challenges are:

- extending (incremental) relational lenses from the asymmetric to symmetric (incremental) case, or further to "webs" of symmetric bx constituting a multidirectional bx
- understanding how (invertible, composable) schema mappings and bx relate: are (some) schema mappings (underdetermined) relational lenses? are (some) relational lenses schema mappings?
- can we compose lenses over other formats (text, graph, tree) with relational lenses?
- given that timeliness, history/archiving, citation/attribution, and provenance are considered critical for Web data to assess its quality, can some of these requirements be integrated into a BX-based formalism?

Concretely, suppose there are three databases, no two of which are controlled by the same administrator / community, for example:

(**A**) Wikipedia (HTML/XML text)
(**B**) DBpedia (RDF triples)
(**C**) some organization's relational knowledge base, e.g. a social science database project.

There is considerable overlap between the first two and some overlap between their common data and the third: for example, perhaps C wants to import some information about cities and populations from Wikipedia and/or DBPedia (and it is hoped that these will remain consistent with each other too).

We could imagine (at least) two multidirectional BXs relating these three (where $\Rightarrow$ denotes an asymmetric lens):

$$A \Leftarrow AB \Rightarrow B \Leftarrow BC \Rightarrow C \Leftarrow AC \Rightarrow A$$

that is, an equilateral triangle of (spans of) lenses, with each "edge" AB, BC, AC a database containing the aligned *union* of each pair of databases (Figure 8); or

$$A \Rightarrow ab \Leftarrow B$$

$$ab \Leftarrow abC \Rightarrow C$$

i.e. a "T" shape where the top is a cospan centered on ab, which involves only the information common to A and B, and there is a span $ab \Leftarrow abC \Rightarrow C$ (maintained by C) that explicitly aligns (the relevant parts of) ab with C (Figure 9).

In either case, criteria for success might include:

- When one data source changes, it can publish its changes to the others and some synchronization process takes place that restores the overall network to consistency.
- Consistency restoration need not take place in a synchronized way. One data source doesn't have to wait for all of the others to complete synchronizing before allowing further local changes.
- Users of the system have a way to determine whether the version of the data they looked at was up-to-date (or more accurately, how out-of-date it was), and revisit results later.
- The amount of shared/duplicated data and coordination between the sources is manageable; small changes to one source are translated to small changes to another source when this is possible.
- Each source has the capability to monitor and reject ill-founded or catastrophic changes, possibly according to an independently-specified access control policy (e.g. a change to Wikipedia/DBpedia source inadvertently requiring the deleting of all of C should be rejected). Ideally, users can inspect and accept/reject individual changes.
- Each system should be able to work with its own native data model, including any associated query or update languages.

These requirements suggest the need for capabilities that go well beyond the bare-bones round-tripping laws of BX, for example to retain history, coordinate distributed systems, and map between different data models. Some of these issues are also well-explored in the conventional relational data integration literature too, and it may be that existing solutions can be transferred to a BX-based setting without difficulty in some cases.

This case study aspires to assess the state of the art of different parts of the BX and data integration landscape, understand what subproblems are solved and what are the open subproblems, and understand whether solutions to the high-level problems are feasible yet (perhaps stipulating solutions to well-defined subproblems) or whether integrating these different approaches introduces new challenges.

References

1 Harmony project. https://alliance.seas.upenn.edu/~harmony/old/index.html

2 AnHai Doan, Alon Halevy, and Zachary Ives. *Principles of Data Integration*, http://research.cs.wisc.edu/dibook/, Morgan Kaufmann, 2012.

3 Aaron Bohannon, Benjamin C. Pierce, and Jeffrey A. Vaughan. *Relational Lenses: A Language for Updatable Views*, PODS, pp. 338–347, https://doi.org/10.1145/1142351.1142399, 2006.

4 Rudi Horn, Roly Perera, and James Cheney. *Incremental Relational Lenses*, ICFP, pp. 74:1-74:30, https://doi.org/10.1145/3236769, http://arxiv.org/abs/1807.01948, 2018.

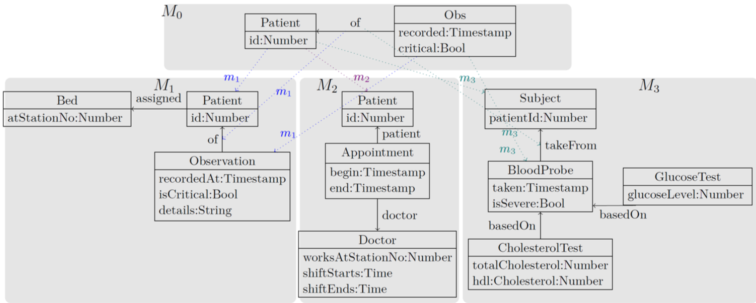

Figure 10 Three Systems for Medical Data with Common Terminology.

4.4 A Health Informatics Scenario

Harald König (University of Applied Sciences FHDW Hannover, DE)

Figure 10 shows (excerpts of) underlying data models of three medical record systems (M_1, M_2, and M_3) together with a correspondence specification scheme M_0. The data models represent a typical arrangement of a hospital's information system architecture, which leads to the problem of *heterogenous data integration*. M_1 stores patient records. It aims at providing caregivers with assigned beds and stations and with simplified descriptions of medical observations. M_2 is a web application tracking appointments between patients and doctors, and M_3 maintains blood test results.[2]

Rules for joint data consistency usually contain terminology from all three systems. It is thus essential to specify common terminology of the systems, i.e. correspondences between same terms or concepts in the models. In the example, M_0 specifies these correspondences grafically: Patient represents sameness of M_1.Patient, M_2.Patient, and M_3.Subject (despite different names, concepts may coincide). Similarly, Obs represents the fact that each M_3.BloodProbe is an M_1.Observation. Moreover, arrow of in M_0 specifies sameness of properties M_1.of and M_3.takenFrom.

Now consider the following rule for joint consistency:

If there is a cholesterol test based on a severe blood probe, and if there is already a bed assigned to the patient the probe is taken from, then there must be an appointment scheduled for this patient.

At first glance, validity of this so-called *inter-model constraint* [1] can not be treated with bidirectional transformations among the models M_1, M_2, M_3 alone, because the dependencies arising from the constraint cannot be reduced to a family of binary relations: The variation in severity of the cholesterol test's blood probe yields two different sets of valid bed-appointment constellations, an assignment of a bed influences correctness of appointment-severity occurences, and statements on valid bed-bloodprobe relations depend on the existence of certain appointments.

[2] Note that there are usually many more distributed sources of patient's data, e.g. in file systems of general practitioners or medical specialists as well as in databases of health insurances.

Thus several research questions arise: Let $M_1, \ldots, M_n$ be a set of models with common terminology M_0 and C be a set of inter-model constraints imposed on some (or all) of these models. Given database states $A_1, \ldots, A_n$, such that each A_i conforms to M_i:

1. How can we conceptually delineate an extension of bx, with which validity of inter-model constraints C can be checked after a local update of some A_i.
2. In case of a violation of some constraints: how can joint consistency be restored?
3. Do we need a more sophisticated "multx"-framework or is a framework sufficient that is based on a network of bx?
4. If a network of bx is enough, can we avoid introduction of extra systems?

Solutions: A conceptual consistency checking framework for the case of an arbitrary number $n \geq 2$ of model spaces was presented in [2]: Correspondences are formalized as (possibly) partial morphisms in an appropriate catagory of graph-like structures, see the dashed arrows in Fig. 10: Each m_i highlights existing commonalities in M_i ($i \in \{1, 2, 3\}$). Whereas m_1 and m_3 are total, m_2 is properly partial, because Observations do not exist in M_2. In practice, morphisms $m_i : M_0 \rightarrow M_i$ are specified with statements of a respective DSL, e.g.

relate ($\mathsf{M_1}$.Observation, $\mathsf{M_3}$.BloodSample) as Obs

for the assignments $m_1(\mathsf{Obs}) = \mathsf{Observation}$ and $m_3(\mathsf{Obs}) = \mathsf{BloodProbe}$, such that there is no need to introduce a database for M_0. A category-theoretic solution is to encode the complete picture in Fig. 10 as a "diagram" in a category. Any inter-model constraint is then imposed on the *colimit* of this diagram. This colimit can be interpreted as the *merge* or the *union* of models M_1, M_2, and M_3 modulo their commonalities. To check whether database states (snapshots) A_1, A_2, and A_3 satisfy an inter-model constraint, we must also compute the colimit of these snapshots modulo their commonalities (if the same real-world object is simultaneously recorded in two or more databases). Clearly, reasoning about properties of this validation can be carried out on a "virtually" computed colimit. Moreover, it can be shown that it is sufficient to investigate only the colimit of the data portion, which is affected by the constraints under consideration [3], such that the complete colimit must not physically be computed. Thus there is no need to introduce a fourth extra database.

Open problems: Up to now, however, there is no general solution (based on bx-methods) for appropriate update propagation and consistency restoration, if $n > 2$. "Multidirectional Transformations and Synchronisations" seems to be a promising research direction, which may provide means to answer this open research question. At best a methodology can be proposed, which enables consistency restoration without introducing extra systems.

References

1 Zinovy Diskin, Yingfei Xiong, Krzysztof Czarnecki. *Specifying Overlaps of Heterogeneous Models for Global Consistency Checking*, Proceedings of the First International Workshop on Model-Driven Interoperability (MDI MoDELS), https://doi.org/10.1007/978-3-642-21210-9_16, 2010.

2 Patrick Stünkel, Harald König, Yngve Lamo, Adrian Rutle. *Multimodel Correspondence through Inter-model Constraints*, Seventh International Workshop on Bidirectional Transformations (BX), Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, pp. 9–17, https://doi.org/10.1145/3191697.3191715, 2018.

3 Harald König and Zinovy Diskin. *Efficient Consistency Checking of Interrelated Models*, Proceedings of the 13th European Conference on Modelling Foundations and Applications (ECMFA), pp. 161–178, https://doi.org/10.1007/978-3-319-61482-3_10, 2017.

Participants

- Anthony Anjorin
Universität Paderborn, DE
- Gabor Bergmann
Budapest Univ. of Technology & Economics, HU
- Dominique Blouin
Telecom ParisTech, FR
- James Cheney
University of Edinburgh, GB
- Anthony Cleve
University of Namur, BE
- Sebastian Copei
Universität Kassel, DE
- Davide Di Ruscio
University of L'Aquila, IT
- Zinovy Diskin
McMaster University – Hamilton, CA
- Jeremy Gibbons
University of Oxford, GB
- Holger Giese
Hasso-Plattner-Institut – Potsdam, DE
- Martin Gogolla
Universität Bremen, DE
- Soichiro Hidaka
Hosei University – Tokyo, JP
- Michael Johnson
Macquarie University – Sydney, AU
- Tsushima Kanae
National Institute of Informatics – Tokyo, JP
- Gabor Karsai
Vanderbilt University, US
- Ekkart Kindler
Technical University of Denmark – Lyngby, DK
- Heiko Klare
KIT – Karlsruher Institut für Technologie, DE
- Hsiang-Shang Ko
National Institute of Informatics – Tokyo, JP
- Harald König
FHDW – Hannover, DE
- Ralf Lämmel
Facebook – London, GB
- Yngve Lamo
West. Norway Univ. of Applied Sciences – Bergen, NO
- Théo Le Calvar
Université d'Angers, FR
- Nuno Macedo
University of Minho – Braga, PT
- Kazutaka Matsuda
Tohoku University – Sendai, JP
- James McKinna
University of Edinburgh, GB
- Fiona A. C. Polack
Keele University – Straffordshire, GB
- Blake S, Pollard
Carnegie Mellon University – Pittsburgh, US
- Robert Rosebrugh
Mount Allison University – Sackville, CA
- Bernhard Rumpe
RWTH Aachen, DE
- Andy Schürr
TU Darmstadt, DE
- Bran V. Selic
Malina Software Corp. – Nepean, CA
- Friedrich Steimann
Fernuniversität in Hagen, DE
- Perdita Stevens
University of Edinburgh, GB
- Matthias Tichy
Universität Ulm, DE
- Frank Trollmann
TU Berlin, DE
- Jens Holger Weber
University of Victoria, CA
- Nils Weidmann
Universität Paderborn, DE
- Bernhard Westfechtel
Universität Bayreuth, DE
- Vadim Zaytsev
RainCode Labs, BE
- Albert Zündorf
Universität Kassel, DE

Report from Dagstuhl Seminar 18511

Algebraic Coding Theory for Networks, Storage, and Security

Edited by

Eimear Byrne[1], Martin Bossert[2], and Antonia Wachter-Zeh[3]

1 **University College Dublin, IE,** ebyrne@ucd.ie
2 **Universität Ulm, DE,** martin.bossert@uni-ulm.de
3 **Technical University of Munich, DE** antonia.wachter-zeh@tum.de

Abstract

This report documents the program and the outcomes of Dagstuhl Seminar 18511 "Algebraic Coding Theory for Networks, Storage, and Security". The ever increasing traffic in networks and the growth of distributed storage systems require advanced techniques based on algebraic coding to meet user demand. Private access to such services is a major concern for consumers and is still a new field in the context of distributed storage. The topics of this workshop are very relevant for a number of emerging industrial research fields concerned with efficient and reliable storage and transmitting large files through large networks.

Seminar December 16–21, 2018 – http://www.dagstuhl.de/18511
2012 ACM Subject Classification Mathematics of computingSecurity and privacy
Keywords and phrases Coding theory, infomation theory, distributed storage, cryptography, error-correction, private information retrieval, private computation, adversarial channel
Digital Object Identifier 10.4230/DagRep.8.12.49
Edited in cooperation with Giuseppe Cotardo

1 Executive Summary

Eimear Byrne
Martin Bossert
Antonia Wachter-Zeh

Algebraic Coding Theory for Networks, Storage, and Security was the fourth in a series of seminars exploring applications of coding theory in modern communications theory (see also Dagstuhl Seminars 16321 (2016), 13351 (2013) and 11461 (2011)). The seminar brought together 50 mathematicians, engineers and computer scientists with expertise in coding theory, network coding, storage coding, cryptography and code-based security to participate in dissemination and collaboration within the seminar themes.

The main focus of this workshop was to explore novel results in coding theory for application in data storage management, cryptography and privacy. The impact of novel coding techniques across these domains was discussed and explored. Particular emphasis was placed on new applications of coding theory in public key cryptography, coding techniques for privacy in distributed storage and on practical schemes using coding theory for content delivery. These novel coding applications continue to have a significant impact on changing focus and broadening of coding theory fundamentals.

Overview talks were given by Philippe Gaborit (*Recent Results for Cryptography Based on Rank Metric*), Emina Soljanin, (*Service Rates of Codes*), Eitan Yaakobi, (*Private Proximity Retrieval*), Sacha Kurz, (*Multisets of Subspaces and Divisible Codes*), Heide Gluesing-Luerssen (*On Ferrers Diagram Codes*) and Salim El Rouayheb (*GASP Codes for Secure Distributed Matrix Multiplication*). In addition, several short talks were given by other participants based on current research interests with a view to stimulating collaboration. Presentations on system cybersecurity, private information retrieval, locally recoverable codes, adversarial channels and various aspects of rank metric codes were given. The remaining seminar time was allocated to discussion groups, including those in code-based cryptography, private computation, service rates of codes, algebraic geometry codes and adversarial channels. Aside from the working group discussions, particpants took the opportunity to engage in specific collaborations with co-authors.

We summarize some of the content of the working group discussions below. It has been well documented that redundancy is a basic requirement for stability of distributed data storage systems. Algebraic codes have been identified as having applications in providing efficiency in this domain far exceeding replication. Coding theory methods allow information retrieval minimizing disc access, storage size, local recoverability, data repair and data retrieval. Consequently, the area of storage coding has seen an exponential growth. An important aspect of user access in distributed storage is privacy of information retrieval so that users who are remotely accessing files can do so without storage servers knowing what they have accessed. Attempts to efficiently solve this problem come from coding theory.

An important application of secret sharing schemes is distributed storage of private data, where each party is a storage node and all parties wish to store a secret securely and reliably. Secret sharing is a fundamental cryptographic primitive and is used as a building block in numerous secure protocols. In our discussions we focussed on secret sharing schemes for the threshold access structure and on secret sharing with errors/attacks in a broader context. Fuzzy vaults and secret sharing over networks were discussed. A motivation for this area is for example biometric authentication in the presence of adversaries.

Another aspect of distributed storage is the service rate of codes. Emerging applications, such as distributed learning and fog computing, add yet another use for coding. In these applications, the goal is to maximize the number of users that can be simultaneously served by the system. One such service is simultaneous download of different jointly coded data blocks by many users competing for the system's resources. Here, coding affects the rates at which users can be served. The achievable service rate region is the set of request rates for each file that can be supported by the system. A variety of approaches to open problems about service rate were discussed. In particular, we addressed the question of code constructions that serve all requests for fixed rate constraints on file and the problem of how to determine the achievable service rate region for certain families of codes.

Privacy and security present formidable challenges in our modern connected world. Public-key cryptography is the foundation of multi-party communication as well as for key exchange of symmetric cryptosystems. With the increasing likelihood of a capable quantum computer, post-quantum secure systems have recently turned into the research focus, especially for devices that are hard to update and have very long life cycles. Code-based cryptography provides post-quantum secure public-key systems.

The working group on code-based-cryptography discussed the importance of security reduction arguments and went through several examples of these in relation to coding theory in cryptography. This discussion was a great benefit to the participants, many of whom have expertise in coding theory and keen to broaden their understanding of cryptography.

The group also focussed on McEliece-like systems based on quasi-cyclic moderate density parity-check (QC-MDPC) codes and on low-rank parity-check (LRPC) codes. Distinguisher attacks were discussed, as well as possible modifications to the broken Gabidulin based cryptosystem.

Reliable communication across a channel in the presence of an adversary is a very general channel model that arises in many applications. Coding strategies for data transmission and authentication across the arbitrarily varying channel (where an adversary may alter the channel statistics) and for covert communication were discussed. A framework for linear systems under attack, such as the scenario where a restricted number of sensor measurements is vulnerable to adversarial attacks, was introduced and coding theoretic arguments used for attack detection and correction strategies.

There were about 20 PhD and postdoctoral researchers in attendance, who reported a very positive experience and satisfaction at being give the opportunity to explore new collaborations with more senior researchers and to get exposure to new problems in coding theory. All participants welcomed the time made available to them to take part in discussion groups and in more focussed collaborations. All were very pleased with the quality of the facilities and administrative support offered by staff at Schloss Dagstuhl, which made for a very productive meeting. Andreas Lenz and Rawad Bitar organised an afternoon excursion to Trier for the group. Giuseppe Cotardo collected and compiled data for the final published report.

2 Table of Contents

Executive Summary
Eimear Byrne, Martin Bossert, and Antonia Wachter-Zeh . . . 49

Overview of Talks

Recent results on rank based cryptography
Philippe Gaborit . . . 54

The Covering Radius of Rank-Metric Codes
Albertog Ravagnani . . . 54

How general is the rank decoding algorithm based on rank one elements?
Anna-Lena Trautmann-Horlemann . . . 54

Achievable Service Rates in Coded Distributed Storage
Emina Soljanin, Carolyn Mayer . . . 54

New Bounds and Generalizations of Locally Recoverable Codes with Availability
Alexey Frolov . . . 55

General adversarial channels
Sidharth Jaggi . . . 55

Private Proximity Retrieval
Eitan Yaakobi . . . 56

Private Function Computation for Coded Databases
Jörg Kliewer . . . 56

On decoding Folded and/or Derivative Codes over any field
Daniel Augot . . . 57

(Multi-)Sets of subspaces and divisible codes
Sascha Kurz . . . 58

A new rank metric codes based encryption scheme
Pierre Loidreau . . . 58

Minimal linear codes from functions over finite fields
Sihem Mesnager . . . 58

On Ferrers Diagram Codes
Heide Gluesing-Luerssen . . . 59

Constructions and Classifications of MRD codes
John Sheeky . . . 59

Resilient LTI systems with redundant signals
Margreta Kuijper . . . 59

Parametrizing Systematic Gabidulin Codes and Applications
Alessandro Neri . . . 60

Spectral Methods for Coding in a Non-Commutative Setup
Marcus Greferath . . . 60

Functional PIR and batch codes
Yiwei Zhang . . . 60

Coding for Stochastic Gradient Descent
Rawad Bitar 61

The problem of constructing complete MDP convolutional codes over small fields
Julia Lieb 61

Working groups

Secret sharing in the presence of adversaries 61

Code-Based-Cryptography 63

Adversarial channel 64

Algebraic geometry codes 65

Service Rates of Codes 65

Participants 67

3 Overview of Talks

3.1 Recent results on rank based cryptography

Philippe Gaborit (University of Limoges, France)
gaborit@unilim.fr

We propose an overview of recent results in rank based cryptography.

3.2 The Covering Radius of Rank-Metric Codes

Alberto Ravagnani (University College Dublin, Ireland)
alberto.ravagnani@ucd.ie

The covering radius of a rank-metric code is the maximum distance between the code and a matrix from the ambient space. In this talk, I will discuss some structural properties of matrix codes with the rank metric, and relate them to the covering radius. In particular, I will present new bounds on this parameter obtained with different combinatorial methods.

3.3 How general is the rank decoding algorithm based on rank one elements?

Anna-Lena Trautmann-Horlemann (University of St. Gallen, Switzerland)
anna-lena.horlemann@unisg.ch

The covering radius of a rank-metric code is the maximum distance between the code and a matrix from the ambient space. In this talk, I will discuss some structural properties of matrix codes with the rank metric, and relate them to the covering radius. In particular, I will present new bounds on this parameter obtained with different combinatorial methods.

3.4 Achievable Service Rates in Coded Distributed Storage

Emina Soljanin, Carolyn Mayer (Rutgers University - Piscataway, US)
emina.soljanin@rutgers.edu, cdmayer@wpi.edu

Coding has traditionally been used in transmission and storage of data to provide reliability in a more efficient way than simple replication. The traditional performance indicators of codes are the minimum distance and the code rate. More recently, special codes have been developed that also provide efficient maintenance of storage under node failures. In addition to the traditional metrics, the properties of codes that matter in such scenarios are the code locality and availability. Emerging applications, such as distributed learning and fog computing, are adding yet another use for coding. In these applications, the goal is to maximize the number of users that can be simultaneously served by the system. One such service is simultaneous download of different jointly coded data blocks by many users competing for the system's resources. Here, coding affects the rates at which users can be served. Interestingly, the best schemes often combine replication and coding. This talk will define the service rates of codes as new performance metrics, survey the existing literature, and show a connection between optimizing the code service rates to a graph vertex cover problem.

3.5 New Bounds and Generalizations of Locally Recoverable Codes with Availability

Alexey Frolov (Skolkovo Institute of Science and Technology (Skoltech) – Moscow, Russia)
al.frolov@skoltech.ru

We investigate the distance properties of linear locally recoverable codes (LRC codes) with all-symbol locality and availability. New upper and lower bounds on the minimum distance of such codes are derived. The upper bound is based on the shortening method and generalized Hamming weights (GHWs) that are fundamental parameters of any linear codes with many useful applications. This bound improves existing upper bounds. To reduce the gap in between upper and lower bounds we do not restrict the alphabet size and propose explicit constructions of codes with locality and availability via rank-metric codes. The first construction relies on expander graphs and is better in low rate region, the second construction utilizes LRC codes developed by Wang et al. as inner codes and is better in high rate region. We also suggest one possible generalization of LRC codes in which the recovering sets can intersect in a small number of coordinates. This feature allows us to increase the achievable code rate and still meet load balancing requirements. We derive upper and lower bounds on the parameters of such codes and present explicit constructions of codes with such a property.

3.6 General adversarial channels

Sidharth Jaggi (The Chinese University of Hong Kong, China)
jaggu@ie.cuhk.edu.hk

The question of when communication is possible in an adversarial jamming context is intimately connected to the question of high-dimensional packings – for instance, communication over a binary-input binary-output channel where the adversary can flip up to pn bits is equivalent to designing packings of pn-radius Hamming balls in n-dimensional Hamming space.

We consider a fairly general class of adversarial channels, and:

- show that each adversarial channel has a bijection with a certain "confusability polytope" embedded in the simplex of all distributions of joint-types of pairs of inputs to the channel;
- precisely characterize when a positive rate is possible (i.e. exponential-size packings are possible). Sufficiency is characterized in terms of codes where each pair of codewords has joint-type given by a "completely positive distribution" outside the confusability polytope. Necessity follows by a Ramsey theoretic argument showing that each large code must have a sufficiently large subcode where each pair of codewords has roughly the same type-class, followed by a Plotkin-type argument, and a separate Fourier analytic argument to handle asymmetric type-classes.

3.7 Private Proximity Retrieval

Eitan Yaakobi (Technion – Haifa, Israel)
yaakobics.technion.ac.il

A private proximity retrieval (PPR) scheme is a protocol which allows a user to retrieve (an approximation of) the set of indices of all records in a database that are within some distance r from the user's record x. The privacy at each server is given by the fraction of the record x that is kept private. The distortion of a PPR scheme measures how accurately the user can calculate this set of indices. We assume that each servers stores a copy of the database. While it is possible to achieve perfect privacy by studying a related private information retrieval problem, it is unclear how to do so without pre-computing the answers for every possible user record x and radius r. We therefore focus on protocols that trade perfect privacy for a massive reduction in computation and storage.

In this paper, we initiate this study while focusing on case when the records are binary vectors together with the Hamming distance. In particular, for a given privacy level, we investigate the minimum number of servers that guarantee a prescribed distortion value. We also consider collusions of pairs of servers and investigate other distance measures.

3.8 Private Function Computation for Coded Databases

Jörg Kliewer (New Jersey Institute of Technology - Newark, US)
jkliewer@njit.edu

We consider the problem of private computation in a distributed storage system. In private computation, a user wishes to compute a function of f messages stored in non-colluding databases while revealing no information about the computation result to the databases. We first employ computation of a linear function of the messages, where linear codes are used to encode the information on the databases. We show that this private linear computation capacity, which is the ratio of the desired linear function size and the total amount of downloaded information, matches the maximum distance separable (MDS) coded capacity of private information retrieval for a large class of linear codes that includes MDS codes.

Our converse result is valid for any number of messages and linear combinations, and the capacity expression depends on the rank of the coefficient matrix obtained from all linear combinations. Finally, we also present initial results how our linear computation approach can be extended to computing arbitrary multivariate polynomials of the messages.

3.9 On decoding Folded and/or Derivative Codes over any field

Daniel Augot (INRIA Saclay - Île-de-France, France École Polytechnique, France)
daniel.augot@inria.fr

Sudan [5] and Guruswami-Sudan [3] algorithms are very general algorithms for decoding Reed-Solomon and algebraic-geometry codes, using the interpolation approach. It can be shown that the algorithms can recover the list of codewords in a ball of relative radius $1-\sqrt{R}$, for Reed-Solomon codes of rate R. One can see that the finiteness of the field does not play any more in these decoding algorithms (although, admittedly, there are variants of these algorithms, adapted to the finite field case, which give an even better decoding radius).

A breakthrough was obtained by Guruswami and Rudra [2] for decoding folded Reed-Solomon, for which, when the parameters are properly set, a decoding radius of $1-R-\varepsilon$ can be achieved (for long codes, and large alphabet). Crucially, this methods relies on a relation

$$X^q \equiv \gamma X \mod E(X)$$

for well chosen $\gamma \in GF(q)$ and $E(X) \in GF(q)[X]$. Using this trick, the root-finding algorithm has to deal with a polynomial of degree q^{s-1}, where s is the interpolation order. Later, Guruswami [1] proposed another algorithm for decoding folded Reed-Solomon codes, where an exhaustive search in an affine space of dimension $s-1$ has to be done, leading to a q^{s-1} factor in the complexity. Also, Guruswami and Wang consider derivate codes, but still face a q^{s-1} complexity wall [4].

It is striking that, for decoding these codes there is no "algebraic algorithm" which would work over any field, finite or not. This talk will discuss this issue.

References

1 "Linear-algebraic list decoding of folded Reed-Solomon codes", Venkatesan Guruswami, *Computational Complexity (CCC), 2011 IEEE 26th Annual Conference on*, 77–85, IEEE, 2011.
2 "Explicit capacity-achieving list-decodable codes", Venkatesan Guruswami and Atri Rudra, "Proceedings of the thirty-eighth annual ACM symposium on Theory of computing", 1–10, ACM, 2006.
3 "Improved decoding of Reed-Solomon and algebraic-geometric codes", Venkatesan Guruswami and Madhu Sudan, *Foundations of Computer Science, 1998. Proceedings. 39th Annual Symposium on*, 28–37, IEEE, 1998.
4 "Optimal rate list decoding via derivative codes", Venkatesan Guruswami and Carol Wang, *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*, 593–604, Springer, 2011.
5 "Decoding of Reed Solomon codes beyond the error-correction bound", Madhu Sudan, *Journal of complexity*, 13(1):180–193, 1997.

3.10 (Multi-)Sets of subspaces and divisible codes

Sascha Kurz (University of Bayreuth, Germany)
sascha.kurz@uni-bayreuth.de

A multi set of subspaces in $GF(q)^v$ gives rise to a q^r divisible linear code if the dimensions of the subspaces are at least $r+1$. This connection has implications e.g. for the existence of vector space partitions, packing or covering problems for subspaces, or subspace codes. Several optimal linear codes in the Hamming metric are divisible. The cylinder conjecture of Ball is actually a classification statement for divisible codes. Extendability results for partial spreads concluded from minihypers can be obtained via divisible codes. Generalizations also permit extendability results for codes in the rank metric or subspace codes.

The aim of this talk is to give a brief introduction to the connection between multisets of subspaces and divisible codes, survey some results and applications of q^r divisible codes, and, most importantly, to encourage collaboration from other participants.

3.11 A new rank metric codes based encryption scheme

Pierre Loidreau (DGA MI, France Université de Rennes 1, France)
pierre.Loidreau@univ-rennes1.fr

We present new approaches for the design of code-based rank metric public-key encryption schemes based on Gabidulin codes. Compared to the rank metric code based NIST proposals using probabilistic decoding of LRPC codes, the use of algebraic codes enables to design schemes no decoding errors. Hence as in the original McEliece cryptosystem but contrarily to MDPC and lattice based cryptography, an attacker does not get any statistical advantage of a decoding failure to attack the scheme.

3.12 Minimal linear codes from functions over finite fields

Sihem Mesnager (Universities of Paris XIII (LAGA), France Paris VIII, France)
smesnager@univ-paris8.fr

Minimal linear codes have significant applications in secret sharing schemes and secure two-party computation. In the literature, there are two generic constructions of linear codes. The first one is based on functions over finite fields and second one is based on defining sets (which could be the support or the complementary of the support of functions over finite fields). We shall review some recent constructions of minimal linear codes based on the generic constructions and present those obtained from weakly regular bent functions (2017) and from weakly regular plateaued functions (2018).

3.13 On Ferrers Diagram Codes

Heide Gluesing-Luerssen (University of Kentucky, US)
heide.gl@uky.edu

Ferrers diagram codes are rank-metric codes where all matrices have support in a given Ferrers diagram. With the aid of a lifting construction such codes can be used to construct large subspace codes. In this talk I will report on progress in the construction of maximal Ferrers diagram codes (for any given Ferrers diagram and rank distance). While various methods exists, the general conjecture about the maximum possible dimension of a Ferrers diagram codes with given rank distance remains widely open. In the second part of the talk I will discuss the proportion of maximum Ferrers diagram codes in the space of all Ferrers diagram codes with the same shape and the according dimension.

3.14 Constructions and Classifications of MRD codes

John Sheekey (University College Dublin, Ireland)
john.sheekey@ucd.ie

We will present the known constructions and classifications for maximum rank distance codes, and survey their connections to objects such as spreads and semifields. We will discuss recent results which show that MRD codes consisting of square matrices over the binary field are surprisingly rare in some cases.

3.15 Resilient LTI systems with redundant signals

Margreta Kuijper (University of Melbourne, Australia)
mkuijper@unimelb.edu.au

This talk looks at linear time-invariant (LTI) systems, such as autopilot systems. Such systems involve input, output and state signals. In recent years there have been several events where such systems have been cyber attacked, resulting in malfunctioning and damage. There is currently a need for an automated response as part of the resilience of the system.

In this talk I will explore several fundamental ideas around linear systems under attack. One of these involves the scenario where a restricted number of system outputs (sensor measurements) is vulnerable to adversarial attacks that take place over time. I will introduce the fundamental notion of a system's "security index" as an analogon of "minimal distance" in linear coding theory. I will show how ideas from coding theory can be used for the purpose of attack detection and automated attack correction.

3.16 Parametrizing Systematic Gabidulin Codes and Applications

Alessandro Neri (University of Zurich, Switzerland)
alessandro.neri@math.uzh.ch

Rank metric codes have been introduced in 1978 by Delsarte, and independently by Gabidulin in 1985 and Roth in 1991. These codes are linear subspaces of the space of $n \times m$ matrices over a finite field F_q, but they can also be seen as sets of vectors of length n over an extension field F_{q^m}. Codes that are optimal in this metric are called Maximum Rank Distance (MRD) codes. The first and most studied family of MRD codes is given by the so-called generalized Gabidulin codes, and they represent the analogue of generalized Reed-Solomon (GRS) codes for the rank metric.

In the talk we examine the structure of generalized Gabidulin codes, in comparison with GRS codes. We focus in particular on their encoders. In analogy with GRS codes, we show that the systematic generator matrices of Gabidulin codes have a Cauchy-like structure. Finally, some possible research directions are presented.

3.17 Spectral Methods for Coding in a Non-Commutative Setup

Marcus Greferath (Aalto University, Finland)
marcus.greferath@aalto.fi

In algebraic coding theory, the BCH bound is probably one of the most impressive examples, showing how a Fourier transform can be used in order to construct codes of prescribed minimum distance. So far, this spectral technique is basically restricted to cyclic codes, however there is no strict reason to keep it limited to this case. This talk is particularly interested in the scenario, where a non-commutative finite group describes the co-ordinate domain. We will sketch the successful development of a Fourier theory for this setting and observe a few strange facts.

3.18 Functional PIR and batch codes

Yiwei Zhang (Technion – Haifa, Israel)
ywzhangcs.technion.ac.il

Codes with locality and/or availability have been extensively studied in recent years, including locally repairable codes (LRC), PIR codes, batch codes, etc.

Usually in such a code of dimension s, we focus on the recovering sets only for the s information symbols. We propose a natural generalization of PIR and batch codes, named functional PIR codes and functional batch codes, by analyzing the recovering sets for arbitrary vectors of length s. In this talk we present some bounds and constructions for functional PIR and batch codes. This is a joint work with Prof. Tuvi Etzion and Prof. Eitan Yaakobi.

3.19 Coding for Stochastic Gradient Descent

Rawad Bitar (Rutgers University – Piscataway, US)
rawad.bitar@rutgers.edu

We tackle the problem of stragglers, slow or unresponsive machines, in distributed machine learning algorithms. We consider the setting in which a Master wants to run a machine learning algorithm on tremendous amount of information. The Master offloads the computational tasks to worker machines. Straggler workers are the bottleneck of such systems. Gradient Coding has been recently proposed to mitigate stragglers in distributed gradient descent. We propose stochastic gradient coding that allows graceful performance degradation with the number of stragglers when using stochastic gradient descent.

3.20 The problem of constructing complete MDP convolutional codes over small fields

Julia Lieb (University of Aveiro, Portugal)
jlieb@ua.pt

It has been shown that, transmitting over an erasure channel, maximum distance profile (MDP) convolutional codes have optimal recovery rate for windows of a certain length. Additionally, the subclass of complete MDP convolutional codes has the ability to reduce the waiting time during decoding.

The existence of (complete) MDP convolutional codes for arbitrary parameters has been shown for sufficiently large field sizes. Moreover, there exist basically two general construction techniques for these codes. However, these constructions require very large field sizes.

In this talk, I will show that it turns out to be hard to find constructions over small fields even for quite small parameters although we could show that these codes exist over much smaller fields than those of the known constructions. Finally, some very particular construction examples for moderate field sizes should be presented.

4 Working groups

4.1 Secret sharing in the presence of adversaries

4.1.1 Secret sharing

Consider the scenario that n parties wish to store a secret securely and reliably. To this end, a dealer distributes the secret into n shares, i.e. one share for each party, such that 1) (reliability) a collection $\mathcal{A} \in 2^{\{1,\dots,n\}}$ of "authorized" subsets of the parties can decode the secret, and 2) (secrecy) a collection $\mathcal{B}$ of blocked subsets of the parties cannot collude to deduce any information about the secret. A scheme to distribute the secret into shares with respect to access structure $(\mathcal{A}, \mathcal{B})$ is called a secret sharing scheme, initially studied in the seminal works by Shamir and Blakeley. An important application of secret sharing schemes is distributed storage of private data, where each party is a storage node. Besides, secret sharing is a fundamental cryptographic primitive and is used as a building block in numerous secure protocols. In our discussions we focus on secret sharing schemes for the threshold access structure, i.e. $\mathcal{A}$ contains all subsets of $\{1, \dots, n\}$ of size at least $n - r$ and $\mathcal{B}$ contains all subsets of $\{1, \dots, n\}$ of size at most z. In other words, the secret can be decoded in the absence of any r parties, and any z parties cannot collude to deduce any information about the secret. We define the threshold t of the scheme as $t = n - r - 1$, i.e. any set of at least $t + 1$ parties are authorized and any set of less parties are blocked.

4.1.2 Initial thoughts

In our initial discussions we looked at the 1981 McEliece & Sarwate paper on secret sharing via Reed-Solomon codes. Without errors/adversaries the secret sharing is accomplished as erasure decoding. However, some of the shares may be in error. The main result of the paper is that errors-and-erasures decoding can be used to still do secret sharing in the presence of errors. After discussing this paper we decided to focus on secret sharing with errors/attacks in a broader context. Fuzzy vaults and secret sharing over networks were discussed. A motivation for this area is for example biometric authentication in the presence of errors/attacks.

We then looked at a number of other papers. The main ones being

1. the 2016 Eurocrypt paper *Essentially optimal robust secret sharing with maximal corruptions* by Bishop, Pastro, Rajaraman and Wichs.
2. the 2015 ITW paper *Talking reliably, secretly, and efficiently: a "complete" characterization* by Zhang, Kadhe, Bakshi, Jaggi and Sprintson. This paper uses hash functions to identify which shares were corrupted by the adversary, and then erasure decoding in the valid shares to recover the secret. Compared to straight forward error decoding this trick allows for smaller rate codes, and hence to reach the channel capacity asymptotically.

4.1.3 Secret sharing over a network with adversaries

Problem Setting: A dealer S wants to share a secret among n parties, such that any(?) k out of these n can recover it. The source uses a random linear network channel to each participant. The adversarial has access to at most t links of the network, which he can both eavesdrop and corrupt and introduce adversarial errors into the network during the distribution of the shares. A similar model was considered in Distributed Secret Dissemination Across a Network by Nihar B. Shah, K. V. Rashmi and Kannan Ramchandran. They developed a protocol to distribute shares efficiently, assuming the network is sufficiently well connected (k-propagating). Their scheme uses predefined channel coefficients to distribute the shares amongst the parties, but assumes that all nodes in the network are honest-but-curious, i.e. they have access to all passing messages, but do not corrupt the messages.

We want to extend their model in two important ways (see Fig. 1):

1. Instead of assuming a given/known topology and precomputing coefficients for each node, we want to assume that each inner node forwards a random linear combination of their incoming information. As known from classical random network coding theory, in this setting errors can/will spread and possibly affect all shares. To be able to detect and correct these adversarial types of errors, we want to use rank-metric or subspace codes.
2. A limited number of links are assumed to be malicious and will introduce errors into the system.

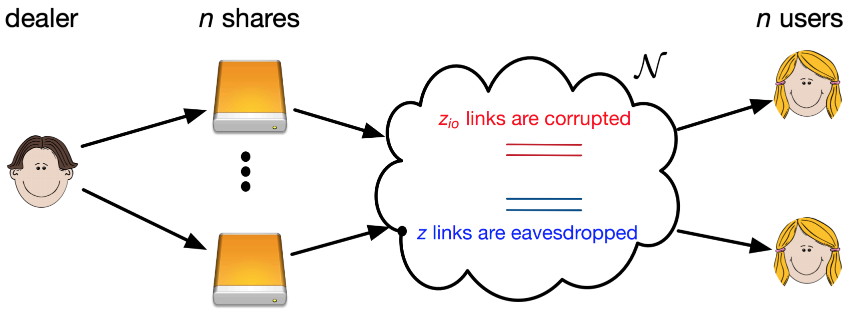

Figure 1 Proposed system model.

Things to be done:

- Establish sufficient conditions on the network to allow for a k-out-of-n secret sharing protocol.
- Find suitable codes for this setting, such that the shares are (related to) the codewords.
- Find means of identifying corrupted shares (and possibly do an analogue of erasure decoding in the network coding setting).
- Find a way to add randomness to each share, to weaken the adversary and to prevent any eavesdropper to gain information about the secret.
- ...

4.2 Code-Based-Cryptography

In this working group, we discussed cryptography based on quasi-cyclic moderate density parity-check (QC-MDPC) codes and based on low-rank parity-check (LRPC) codes. There are a several important considerations about these systems which may be interesting from a coding-theoretic perspective.

- The parity-check matrix for a QC-MDPC code of length $2n$ is defined by two polynomials, modulo $x^n - 1$. In the event that n is not chosen as a prime and $x^n - 1$ can be factored, is it possible to leverage this factorization in reducing the complexity/search space for low-weight codewords?
- MDPC codes are not interesting for classical coding purposes, and for this reason, relatively little analysis of decoding properties of these codes has occurred. Typically, a simple bit flip algorithm is implemented, and there is some risk of decoding failure. Simulation results for current parameters of interest indicate an error rate which is considered too high a rate for serious consideration in cryptographic application. Are there better decoders that can be easily implemented? Are there theoretic results to support improved performance or offer performance guarantees?
- Do there exist quantum algorithms which might break quasi-cyclic syndrome decoding (QCSD)?
- For certain parameters, there exists a distinguisher for LRPC codes based on a containment in (sums of) Gabidulin codes. Much like the Schur square distinguisher in the case of generalized Reed-Solomon codes, this distinguisher reveals a space of dimension much lower than expected for a random code. Is it possible to extend this distinguisher to other parameters using a strategy of puncturing or shortening?

- Fundamentally, the application of lattices, MDPC codes, or LRPC codes in cryptography represent the same framework under three different metrics: Euclidean, Hamming, and rank metric. Taking this global view is helpful in illustrating the advantages and disadvantages of each metric. We considered the specific application of hash functions, wherein the Euclidean system has many good parameters, the Hamming system has some good parameters, and the rank system has no good parameters. What other conclusions can we reach based on these differences? While theory may support the possibility of each perspective, what happens when we push to establish actual security parameters?
- What new types of codes are promising in cryptographic application? Consider bivariate Gabidulin "Reed-Muller" codes over general finite fields, as discussed in the working group on algebraic geometry codes.
- What algorithms exist for decoding random linear rank-metric codes?
- For serious consideration in post-quantum standardization, it is critical going forward to work toward security reductions for code-based cryptography.

4.3 Adversarial channel

Many areas of research fall under the category of adversarial channels. In this working group, we discussed several adversarial settings and interesting problems within each. Specifically, the following topics were identified:

- Reliable communication over the arbitrarily varying channel (AVC): In this setting, channel statistics may change according to the state imposed by an adversary. There are many variations in which the adversary has a varying amount of information regarding the transmitted signal. In particular, coding strategies for authentication over the AVC, and coding strategies to gain information about adversarial power were considered.
- Covert communication: Here, we wish to communicate without an adversary identifying that a transmission is being sent. The signal should approximately mimic the noise of the channel. One potential direction is to adapt the code construction techniques of Ahlswede and Dueck in [1].
- Confusability graphs and zero-error capacity: We briefly discussed the confusability graph framework for studying the zero-error capacity of adversarial channels. Several resources discussed are [3], [4], and[2].
- Network reconstruction: We wish to reconstruct a network given some information from each of the nodes (e.g. the structure of a network of friends on social media), where a node or some subset of nodes are either malicious or (unintentionally) untrustworthy. Several parameters to consider are the nature of the shared information and a metric for measuring the accuracy of a reconstruction. Given precise definitions for each of these things, the question is whether we can characterize the amount of damage an untrustworthy node is capable of.

References

1 "Good codes can be produced by a few permutations", Rudolf Ahlswede and Gunter Dueck, *IEEE transactions on information theory*, 28(3), 1982 .
2 "The zero error capacity of a noisy channel", Claude Shannon, *IRE Transactions on Information Theory*, 2(3):8–19, 1956.
3 "On the Shannon capacity of a graph", László Lovász, *IEEE Transactions on Information theory*, 25(1):1–7, 1979.
4 "Adversarial network coding", Alberto Ravagnani and Frank R. Kschischang, *IEEE Transactions on Information Theory*, 65(1):198–219, 2019.

4.4 Algebraic geometry codes

The first two days we reviewed algebraic plane curves over finite fields, the notions of divisors, Riemann-Roch space and all the tools needed to construct algebraic geometry codes (AG codes for short). We pointed out the connection between the discrete logarithm problem and the elliptic curves. Finally, as an exercise, we found the equivalent version of the Berlekamp-Welch decoding algorithm for Reed-Solomon codes in the AG setting.

The last day we dealt with the case of a Galois group which is the product of cyclic groups, like $C_l \times \ldots \times C_l$ and to try to define Reed-Muller codes, by defining multivariate skew polynomial rings. The exercise was to try to define the codes and compute their minimum distance.

4.5 Service Rates of Codes

Consider a system storing K files $(f_1, \ldots, f_K)$ over N nodes (labelled $1, \ldots, N$) using an (N, K) code. Suppose that file f_i has t_i recovery sets, $R_1^{(i)}, \ldots, R_{t_i}^{(i)}$, and let μ_l be the service rate of node l (i.e. the average rate at which node l resolves received file request). Denote by λ^i the rate of requests for file f_i and $\lambda_j^{(i)}$the portion of requests for file f_i that are assigned to $R_j^{(i)}$.

Then the achievable service rate region of such a system is the set of vectors $(\lambda^1, \ldots, \lambda^K)$ such that, for every $1 \leq i \leq K$, there exist $\lambda_j^{(i)}$, $1 \leq j \leq t_i$, satisfying the following:

$$\sum_{j=1}^{t_i} \lambda_j^{(i)} = \lambda^i, \qquad 1 \leq i \leq K \tag{1}$$

(The demand for file f_i is served.)

$$\sum_{i=1}^{K} \sum_{j:l \in R_j^{(i)}} \lambda_j^{(i)} \leq \mu_l, \qquad 1 \leq i \leq N \tag{2}$$

(No node is sent requests in excess of its service rate.)

$$\lambda_j^{(i)} \geq 0, \qquad 1 \leq i \leq K, 1 \leq j \leq t_i \tag{3}$$

(Requests send to repair groups are nonnegative.)

Given $K-1$ arrival rates $\lambda^1, \ldots, \lambda^{K-1}$ we would like to find the maximum

$$\lambda^K = \sum_{j=1}^{t_K} \lambda_j^{(K)}$$

subject to the constraints described in (1)-(3).

In this group, we discussed a variety of approaches to open problems about service rate. Two problems that arose are introduced below.

1. Given constraints on file request rates, how can we construct a code serving all requests in the described region?

When constructing a code, we may be interested in minimizing the number of nodes used. Note that a minimal length code serving a given set of requests does not always have the minimum length. Other possible considerations include the number of coded nodes and the utilization of each node in a coding scheme.

1. How can we determine the achievable service rate region for given families of codes?

Results about the service rate regions for Simplex Codes and MDS codes rely on the structure of the repair groups. Classifying the structure of repair groups of other families of codes may help determine the service rate region.

These problems can be considered in various settings. Settings proposed within our group use methods or structures including (i) linear optimization, (ii) batch codes, (iii) graph covers, or (iv) generator matrices.

Developing tools to investigate service rate regions from several perspectives will make this emerging area accessible to a wider research community.

Participants

- Gianira Nicoletta Alfarano
Universität Zürich, CH
- Iryna Andriyanova
University of Cergy-Pontoise, FR
- Daniel Augot
INRIA Saclay – Île-de-France, FR
- Allison Beemer
Arizona State University – Tempe, US
- Rawad Bitar
Rutgers University – Piscataway, US
- Jessalyn Bolkema
SUNY – Oswego, US
- Martin Bossert
Universität Ulm, DE
- Eimear Byrne
University College Dublin, IE
- Giuseppe Cotardo
University College Dublin, IE
- Marwa Dammak
University of Cergy-Pontoise, FR
- Salim El Rouayheb
Rutgers University – Piscataway, US
- Tuvi Etzion
Technion – Haifa, IL
- Alexey Frolov
Skoltech – Moscow, RU
- Philippe Gaborit
University of Limoges, FR
- Heide Gluesing-Luerssen
University of Kentucky, US
- Oliver Wilhelm Gnilke
Aalborg University, DK
- Marcus Greferath
Aalto University, FI
- Daniel Heinlein
Aalto University, FI
- Lukas Holzbaur
TU München, DE
- Anna-Lena Horlemann-Trautmann
Universität St. Gallen, CH
- Sidharth Jaggi
The Chinese University of Hong Kong, HK
- Jörg Kliewer
NJIT – Newark, US
- Stanislav Kruglik
Skoltech – Moscow, RU
- Margreta Kuijper
The University of Melbourne, AU
- Sascha Kurz
Universität Bayreuth, DE
- Julien Lavauzelle
Ecole Polytechnique – Palaiseau, FR
- Hunter Lehmann
University of Kentucky, US
- Andreas Lenz
TU München, DE
- Julia Lieb
University of Aveiro, PT
- Pierre Loidreau
University of Rennes, FR
- Felice Manganiello
Clemson University, US
- Carolyn Mayer
Worcester Polytechnic Institute, US
- Sihem Mesnager
University of Paris VIII, FR
- Alessandro Neri
Universität Zürich, CH
- Cornelia Ott
Universität Ulm, DE
- Mario Osvin Pavcevic
University of Zagreb, HR
- Sven Puchinger
TU München, DE
- Alberto Ravagnani
University College Dublin, IE
- Julian Renner
TU München, DE
- Joachim Rosenthal
Universität Zürich, CH
- Ronny Roth
Technion – Haifa, IL
- John Sheekey
University College Dublin, IE
- Carmen Sippel
Universität Ulm, DE
- Emina Soljanin
Rutgers University – Piscataway, US
- Razane Tajeddine
Aalto University, FI
- Violetta Weger
Universität Zürich, CH
- Eitan Yaakobi
Technion – Haifa, IL
- Yiwei Zhang
Technion – Haifa, IL
- Jens Zumbrägel
Universität Passau, DE