Integration and Coordination of Health Data Systems: State-of-the-Art and Open Problems*

Harald König[1] and Patrick Stünkel[2]

[1] University of Applied Sciences FHDW Hannover, Germany
[2] Western Norway University of Applied Science, Norway
harald.koenig@fhdw.de
Patrick.Stunkel@hvl.no

Abstract. Software Engineering involves a multitude of different tools and concepts. These tools typically produce a wide variety of diverse artifacts such as code, design models, and requirements. Even though these artifacts are highly interdependent, consistency, e.g. semantical correctness, is in general only checked *locally* within one modeling environment and not *globally* across tools. A prominent example is consistency of class models and process models.
Only recently, research activities have started, which specify frameworks for *global* consistency maintenance. Such a framework must be capable of detecting, monitoring, and repairing global inconsistencies. But it should also be abstract enough to cover the detection and possible restoration of contradictory data, e.g. distributed but overlapping patient data of electronic health records.
We give an overview over recent and current research activities in global consistency management, from which the development and maintenance of *heterogeneous eHealth systems* will benefit. We outline a possible workflow in a multi-user environment, which aims at keeping all artifacts consistent, and sketch some of the biggest challenges in this area.

1 Introduction and Problem Statement

Landscapes of ICT-systems must provide a precise and undistorted representation of the real-world-domain under consideration. The ideal coexistence of a real-world entity and its electronic reproduction requires a single and unique electronic representation. However, reality is different, because the distributed nature of health-services, continuous changes of health records and the vast amount of existing legacy systems and different information models result in an entirely unclear distribution of data of a single real-world entity. Unfortunately, natural barriers like the value and importance of stored information, continuous requirement appearances as well as the permanent pressure to keep the systems in operation, prevent consolidation.

But even if electronic data would perfectly be centralized, its prescriptive and descriptive models in the sense of Model-Driven Software Engineering (MDSE) are highly inter-dependent and simultanously constrain each other: Requirements specifications,

* This work was supported by the Department of Computing, Mathematics, and Physics at HVL Bergen

design definitions, their implementations and documentations each represent the system from a particular perspective.

Understanding the ensemble of several interrelated heterogeneous models as one big concept is known as *multimodeling* [17]. Other research areas use the terms "megamodeling" [18], "feature-driven design" [40], "viewpoints" [22], or "domain-driven design" [32] for this holistic view. A common goal of all these areas is the discussion how to sustain high quality of multimodels. Because each involved local artifact (a component of a multimodel) is subject to perpetual change, this is only possible by continuous change propagation to the other components in order to re-establish *global consistency*.

Especially in e-health environments we are forced to cope with consistency maintenance of distributed information on different levels of the modeling hierarchy (data, models, and metamodels): Besides dependent modeling artifacts, also real-world objects are distributed over several storage facilities, e.g., some portions of the health record of a patient may be stored in the hospitals's database whereas other parts are in the file system of a general practitioner, yet common data like patient's name and address occur in both portions. Cross-storage dependencies (and hence inconsistencies) arise in the presence of constraints, e.g., if a visit to the general practitioner must always precede a visit to the hospital.

In view of these overall dependencies, it is a problem that supporting tools (e.g. relational database, database schema definition tool, file system explorer, UML modeling tool) are limited to *local* consistency checks. Since support of multimodel consistency maintenance is missing, and since the involved local artifacts are subject to accelerated evolution and change, these artifacts quickly become out-of-sync (i.e. inconsistent). This barrier decreases productivity and overshadows the advantages of MDSE. Formal concepts and architectures for the maintenance of overall consistency are required.

There are many different facets of global consistency[3]: One distinguishes between *vertical* dependencies (e.g., type conformance relations between data (e.g. XML document, database contents) and data model (e.g. XSD, database schema)) and *horizontal* dependencies. The latter arise between artifacts on the same modeling level, e.g., use-cases together with their textual descriptions must be in line with process models, which in turn must not contradict involved class models and activity diagrams. Finally and most importantly, these conceptual documents must yield *correctness* of solutions, i.e. consistency with program code.

Whereas data model updates may induce data migrations (restoration of vertical consistency), class models must consistently co-evolve with data models, data exchange formats, and process model descriptions (restoration of horizontal consistency). Finally, distributed data portions of a real-world object must at every point in time be consistent. Thus the main challenge can be stated as follows:

How to achieve cross-tool consistency maintenance and global consistency checking and restoration by means of an integrated conceptual and architectural framework?

From this main problem statement the following detailed questions can be extracted:

[3] Related work will be discussed in Sect. 3

1. *Workflow*: What are the relevant services for consistency maintenance? Which of them are automatic / semi-automatic / manual?
2. *(Conceptual) framework*: In view of the different facets of consistency: Is there an appropriate abstraction, which covers all the different types of dependencies? What is an suitable formalisation?
3. *Architecture*: What is a practical system topology for this framework?

The goal of the following sections is to report on the most important research activities w.r.t. the conception and implementation of frameworks which detect, monitor, and repair overall inconsistencies. Another objective is to show that the necessary activities cannot all be automatic, such that it is necessary to specify workflows in a multi-user environment. Finally, we sketch open problems and directions of future research.

2 Example

In this section, we present a simple example (see also [58]), which already shows some important challenges in the area. Clearly, real-world examples are far more complex but can, in principle, be treated in the same way. Consider Fig. 1, which contains two data models: In the system of some general practitioner, patient's blood pressure values and their name are stored, whereas in some central register, possible contacts of certain persons are maintained. Suppose now that there is an overall constraint, possibly due to some legal health insurance regulations, requiring that ...

Patients with hypertension[4] *must have at least one contact person.*

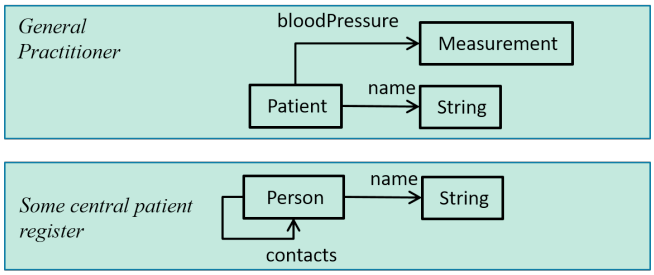

Fig. 1: Two data models

This regulation is a typical example of a global constraint, since it concerns data which is typed over both data models. Any tool, which maintains only one of the data models, cannot check consistency w.r.t. this global constraint, because the first tool (in the practitioner's application) knows nothing about contact persons and the second (maintaining the central register) is not aware of blood pressure values.

[4] Systolic and diastolic blood pressure over 140 and over 90

We can analyse the questions of Sect. 1 with regard to this example: Let's assume that John is a patient of the general practitioner Dr. Livingstone and that, some months ago, John's blood pressure values turned out to be $(125, 85)$[5]. Since these values are normal, there is currently no contact person registered for John. Suppose further that John's wife Mary is also a patient of Dr. Livingstone and that she suffers from hypertension, because her blood pressure is $(150, 100)$. According to the legal requirement above, Mary possesses contact person Ben, the son of John and Mary. Thus, in this example, the constellation is globally consistent w.r.t. to the above constraint. But the situation changed yesterday, because John visited Dr. Livingstone's office for some routine examination and, unexpectedly, his blood pressure values have increased to $(145, 95)$.

Question 1: The recent blood pressure measurement is an event, which must trigger a workflow, in which the following services are needed: (a) *detection of global inconsistencies*, (b) *consistency restoration*. If all relevant data is stored electronically, (a) can be carried out automatically, whereas (b) can at best be performed semi-automatically, because it is not clear which person becomes John's new contact: Is it his wife (although she also suffers from hypertension), is it his son (he is already the contact of Mary), or is it some other appropriately qualified person? It is unlikely that an automatic algorithm is able to determine a solution which always fits the participants' expectations.

Question 2: How can we formalise the structures of class diagrams in Fig.1 and their respective object states (i.e. John, Mary, Ben together with their medical observations), such that these structures are also a basis for handling inconsistencies between other artifacts like use-case descriptions, activity diagrams, process models, etc? How does a machine-readable *common language* for defining global constraint looks like?

Furthermore, Dr. Livingstone's data model differs from the central patient model. Yet both models share common aspects, because PATIENT and PERSON refer to the same business concept. Stored objects (PATIENT John in Dr. Livinsgtone's data and PERSON John in the central register) represent the *same* real-world object. Conceptually, this requires a *unification of logically separated structures* and the formal cross-model *identification of identical objects*.

Question 3: The practitioner's database may or may not be located in his office. The general patient register may physically be on a remote server in the cloud. In general, systems are located in different geographic areas, composed of smaller components (e.g., processes, services, and backends), and are constraint by special hardware properties. Hence, a system architecture has to be provided, which enables (direct or indirect) communication and data exchange of *physically separated components*.

3 Solutions

In this section, we report on current methodologies for consistency maintenance of multimodels. Each subsection refers to one of the three above posed questions.

[5] Blood pressure values measured in mmHg

3.1 Workflow Services

In Sect.2 we observed that the two crucial workflow services of heterogeneous multimodeling are *detection of global inconsistencies* and *consistency restoration*. For both services, the research community distinguishes between several fundamentally different strategies: Amongst many others, [51] surveys multiview consistency checking and describes advantages and disadvantages of several approaches. Merging these insights with a recent survey [42] on UML-based techniques, which pursues the goal to determine a heterogeneous, multi-view language for UML-based multimodeling, it turns out that there are 4 fundamentally different "consistency techniques":

- System Model Approach: Heterogeneity is managed by unifying all local models to one global model. Consistency is checked w.r.t. this global model.
- Heterogeneous Transformations: Separated models are related via additional satisfiability or dependency rules accompanied by optional model translations.
- Universal Logic: Separated but interdependent models can be investigated by translating them into a common logic.
- Extended Metamodels: Model integration is supported by extending the abstract syntax of the modeling language, with additional semantic concepts.

In view of these 4 techniques, implementation proposals of the two services can be surveyed as follows:

Inconsistency detection: *System Model* approaches require computation of a comprehensive model which comprises all artifacts under consideration, e.g., a data model with PERSONs, MEASUREMENTs, and CONTACTs in Fig 1. This calculation has been described e.g. in [55,20] for the case of two interrelated models and in [44,58] for many models and can be carried out automatically. The original idea, however, can be dated back to the contributions of Goguen [35] and Ehrig [25]. Afterwards, classical (single-model) consistency checking methods can be applied.

The most direct (and most well-known) approach to consistency checking is via monitoring satisfiability of consistency rules. These rules are specified in a special language "understanding" all local models, although these models remain separated but may be homogenised by *Heterogenous Transformations* as is proposed in [6]. [50] invents the language *xlinkit* which provides XML-based link generation amongst web content. In a similar way, cross-model "affects"-links are often used. Recipes to increase the efficiency of the checking algorithms can be found in [24,16,44]. Based on the theory of "Triple-Graph-Grammars" (TGG) [56], consistency rules are formulated as relations between (patterns in) graphical structures, consistency checking then extends the model parts under consideration by these "consistency" links, and, if all areas of the models have been matched, they are considered consistent [26]. This approach would specify a rule which searches for CONTACT patterns, if it detects a hypertension pattern in the related model.

[46] is a typical representative of *Universal Logic*: Global constraints are investigated in the context of feature-oriented software development and software product lines. Inconsistency detection is performed by mapping feature models to propositional logic. In our example, this means that not only the obligation to have a contact for people with hypertension is formulated as a logical directive, but also the models themselves

are represented as logical formulas. In the same spirit, the "Consistency Workbench" by Küster, Engels, et al. [28] translates models into a logic-based semantic domain in which consistency verification is carried out.

Extended Metamodels: [27] connects several UML metamodels by intelligent links. E.g., the union of the metamodels for object diagrams and for statecharts is extended by an associaton "current" from OBJECT (in the first metamodel) to STATE (in the second metamodel) thus establishing a semantic link between them. In such a way, global (e.g., OCL-)constraints can restrict the set of possible state changes after certain operation calls. A similar approach, though not intended solely for UML is used in the GEMOC initiative for globalising modelling languages [14].

Consistency Restoration: Conceptual approaches for fixing inconsistencies can be divided into (1) rule-driven, (2) implementation-driven, (3) choice set computation, (4) universal solutions.[6]

A typical rule-driven approach are production rules of TGG's, see above. These rules already contain the necessary information to fix an inconsistent state of two related models. In such a way, a detected hypertension pattern is propagated across models by creating some (initially incomplete) contact. Additionally, linear optimization techniques support a ranking of several repair rules [45]. Implementations (i.e. "TGG-engines") have been provided, e.g. (e)MOFLON [2,3], GraTraM [45].

Local changes that caused global inconsistencies can, in general, not uniquely be propagated to other models in order to restore consistency. This is only possible, if a change occurs in an artifact, from which the others can be generated by means of model transformations (e.g. code generation or object-relational (OR) mappings) with languages like ATL or operational QVT[7]. If, in the example, a woman's name is updated after her marriage, this can uniquely be propagated to the other model.

In the general case, additional *update propagation policies* are needed. Whereas these policies remain abstract in the basic theory of bidirectional transformations (BX) [15] and especially in the theory of (delta-)lenses [34,21], possible policies can be derived from inverses of unidirectional transformation encodings with bidirectional programming languages like, e.g., JTL [13,30], BiGUL [43], or even QVT-R[7]. A typical example would be, if a complete name without separating symbols (e,g., "John Henderson") is generated from first name and last name ("John", "Henderson"). This is in general not invertible (e.g. "Ho Chi Min"). Although *minimality* policies like minimal distance (of a consistent structure from the one under repair) w.r.t. to classical metrics have not been successful [11], other optimisation approaches like e.g. the policy of *maximal preservation* [36] may be pursued further.

A complete generalisation of this setting would be a specification of change propagations and amendments after (possibly concurrent and conflicting) updates of several artifacts under consideration, e.g., if hypertension has been detected and the last contact has been deleted simultaneously.

Automatic change propagation and amendments of original updates seem to be possible only in a setting of certain functional (in fact: functorial) update processes [38],

[6] Since we are aiming at a conceptual grouping of approaches, we do not pursue the more technology-driven classification of model repair approaches in [48].

[7] http://www.omg.org/spec/QVT/

which reveal universal properties such as, for instance, adjoint situations (of functors). This is by far not possible in general as could already be seen in the simple example of Sect.2 (we will, additionally, discuss other inherent natural barriers for unique model repair in Sect.4). Therefore, practical approaches, which do not use additional bidirectional programming languages, first have to determine all possible restoration choices, then restrict this choice space, such that the proposals do not cause new inconsistencies, and then present the remaining choices (possibly prioritised) to the respective modelers and leave to them the final selection, see e.g. [23]. Furthermore, other heuristics for repair proposals, e.g., "extension precedes deletion" or "amendment of manual changes only after user approval" can guide this prioritisation.

Other promising approaches to control and present different restoration choices in a reasonable way is "Answer Set Programming" [29], CLP-based[8] model completion [57], and completion rules [52], the latter two in combination with the above-mentioned construction of a unified global model. Last but not least [41] proposes a practical approach in the field of model co-evolution [12,53], where proposals of model adaptions after a metamodel change are presented to the modeler as a ranked list, cf. the remarks in the last paragraph. In this work, top proposals are chosen according to maximal cosine similarities with the old model and minimal number of edits needed to restore consistency. This also yields a pragmatic taxonomy for characterizing changes in one model w.r.t. to possible update propagation by "unbreaking", "resolvable", and "breaking".

Choice set computation raises the question, *when* to select or at least narrow (a) resonable solution(s). One can distinguish between two viewpoints: Least change policies as described above and further investigated in the next chapter, see also [47] and [36], but also the possibility to carry on with uncertain models, i.e. leave update propagtation uncertain: [39,4].

Finally, a future research direction is to find repair proposals by means of "cognifications" [9], e.g. methods where machines learn the modeler's behaviour and are thus able to predict reasonable decisions. Whereas Eclipse's Code Recommender abilities [9] are a first simplistic approach of "cognified" MDSE, [10] even proposes to carry out model transformations by means of hierarchical neural networks. Learning update propagation policies may also be possible with "Reinforcement Learning", a first approach has been proposed for single models in [5].

3.2 Conceptual Framework

Formal system integration must abstract away from the different modeling languages and from the modeling level of the problems under consideration and it must be able to uniformly specify commonalities of artifacts and their dependencies. Application-based "complete product models" were proposed in [8], but the well-established concept of directed multi-graphs, which are able to encode most of the artifacts under consideration, turned out to be more feasible. Binary lenses of BX and (recently) multi-ary lenses [19] replace the category[10] of graphs with an arbitrary category and provide an abstract

[8] Constraint Logic Programming

[9] http://www.eclipse.org/recommenders/

[10] For basic explanations on *Category Theory for Software Engineering, see [33]*

interface, in which operations for (forward and backward) propagations are specified. In the same spirit, conceptual work in the area of "megamodeling" focusses on the specification of relevant operations without detailing inner structures, e.g. [18].

Typical common languages for the specification of commonalities and other global constraints are sentences of first-order-logic, the (more application-oriented) "description logic" (DL), OCL (Object-Constraint-Language), and the Diagrammatic Predicate Framework (DPF) [54]. The latter prevents mismatches between grafical artifacts and string-based formulas imposed on them. For (an extension of) the example of Sect. 2 this framework has been used in [58].

A sound and fully theoretical underpinning are institution-based multimodeling laguages [7], which enable homogeneous specifications of models and constraints.

3.3 Architecture

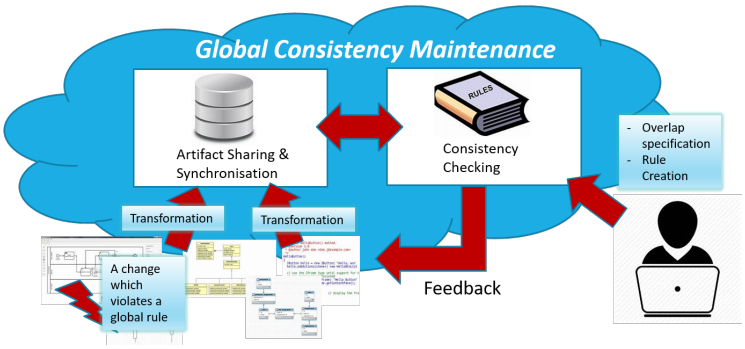

Fig. 2: Global Consistency Maintenance

The most expedient system topology, which simultaneously specifies the above-mentioned workflow, is depicted in Fig.2. If there are various artifacts (local models) for the description of data or concepts from several perspectives (left bottom in Fig. 2), then in a preprocessing phase these artifacts must be transformed into a common language, e.g. into directed graphs, cf. Sect. 3.2, and then stored in an artifact repository, which may be located somewhere in a trusted and secured network. This storage may keep the artifacts separated or it may compute a unified model. Moreover, a special modeler defines consistency rules and maintains commonality specifications.

If a change event occurs in a local model, its delta between previous and current version must immediately and transactionally be synchronized into this repository. If a unified model is used, it must be re-calculated on the change's affected parts. After having re-checked global consistency by the respective service (cf. Sect. 3.1), a detected inconsistency must be repaired. For this, the restoration service (cf. Sect. 3.1) may narrow the solution set, which is then sent back to the respective modelers by means of

work items (notifications). As mentioned above, the final selection, how to react on the original update, is then left to these users.

This framework is also able to repair inconsistencies of data. In the example of Sect. 2, the inconsistency detection due to John's hypertension takes place in the consistency maintenance space. Then prioritised proposals are contained in the notification of the responsible user of the central register (e.g. John's daughter may be the best proposal for the new contact, because both Mary and Ben are no appropriate choices, see the discussion in Sect. 2).

4 Open Problems and Summary

Let's revisit the main problem statement in Sect. 2: We illustrated how one can ensure cross-tool consistency maintenance and global consistency checking by means of an integrated conceptual and architectural framework, if one chooses from possible implementations of the two main workflow services, see Sect. 3.1, and if these services are embedded into the depicted architecture of Fig. 2 of Sect.3.3. We defined the scope of automatisation depending on the different basic approaches (cf. Question 1) and have delineated the necessary abstraction level (Question 2) for the conceptual aspects of the framework (Sect. 3.2).

Obviously the biggest challenge in this area is *intelligent consistency restoration*[11]. There is the intricacy to find the right amount of automatisation while simultaneously not surprising the modeler(s) too much with strange repair proposals.

[47] proposes to measure the impact of consistency restorations and thus find least changes (in the sense of minimal distances). However, [11] points out drawbacks of the idea to let a formal measure, e.g. the edit distance between grafical structures, guide an automatic repair procedure. It turns out that in many cases the *principle of least surprise* is violated. This principle, originally an HCI design guideline [37], requires deliberate modeler's decisions not to be disrupted by automatic amendments and it demands that modeling data not shared with other artifacts is not touched. Similarly, [1,49] state a *principle of least change*: "The action taken by the maintainer of a constraint after a violation should change no more than is needed to restore the constraint." The following examples, however, show that it is hard to comply with the latter requirement[12].

Example 1: Suppose, in an OR-Mapping scenario, class *C* of some Ecore-UML-diagram possesses attribute *a*. Accordingly, the respective database table *C* contains column *a*. A modeler may now delete column *a* in the DB table. What would be the propagated change to the class model, which changes no more than is needed? We think that many of the readers would say "delete attribute *a* in the class, too". But if a certain update propagation operation may adhere to the (important) principle to preserve as much information as possible, it would consequently just set the property "isTransient" of attribute *a* to TRUE. The UML modeler may or may not be surprised!

Example 2: In the next scenario, suppose clubs of your hometown together with its members are stored in a database owned by the town's administration department.

[11] Because we learned that *automatic* restoration is not realistic in the majority of use-cases, we prefer to use the term "intelligent".

[12] The examples are taken from [11]

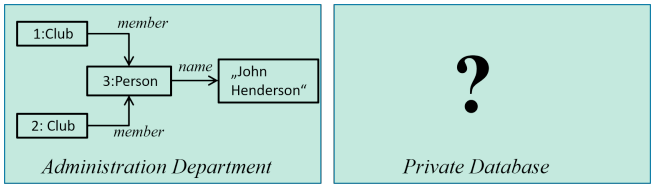

Fig. 3: Club memberships before synchronizing with the private database

Since you are interested in people that are members of your local clubs and not always have access to this database, you also create a local database with the same schema. Fortunately, you find an open source tool, which guarantees to transform the contents of the administration database (the source) into your private environment (the target) based on the following consistency rule:

For each membership of a person with name N in club C in the source there is the same membership of a person with name N in club C in the target.

It sounds as if this perfectly fits your copying requirements. The left part of Fig. 3 shows an excerpt of the administration database indicating that John Henderson is a member of both depicted clubs. According to the constraint definition, the two models considered as a multimodel are inconsistent as long as your local database is empty. The open source transformation promises to restore consistency. What result of the transformation would you expect? We guess, you would say that ideally ”?” should be replaced by an exact copy of the left-hand side in Fig. 3. However, surprisingly, the open source tool produces a different result, see Fig. 4.

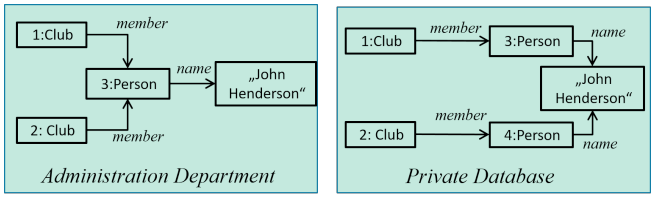

Fig. 4: Club memberships after synchronizing the private database

The open source tool may act along the following paradigm (which is the common QVT-R-”enforce”-semantics[7]): Since the object structure in the source (the left-hand side in Fig. 3, this structure is shown again on the left-hand side of Fig. 4) is hierarchical, the transformation traverses the hierarchy from top (clubs) to bottom (names). At each level in the source an object on the respective level in the target is constructed except for elements of primitive data types such as strings: If there is already a respective string constructed in the target, it is reused. Furthermore, several elements on the same level

in the source are traversed sequentially. And, as a matter of course, determination of person's names in the target is according to the given consistency rule.

Therefore for each club a new person with name "John Henderson" is created, which yields two different persons (with the same name) in the target. The paradigm of the algorithm follows the hierarchical structure of XML-documents, i.e. it assumes subelements not to be contained in several parent elements. It is not unusual for two persons to have the same name, such that, subjectively, this does not produce a surprising result. Moreover, the given constraint is not violated. But this result possibly surprises *you*!

We conclude with the remark that some outcome of an update propagation may surprise one human user and may not surprise another one. Even more may an automatic restoration algorithm produce unexpected results (in the opinion of one actor) simultaneously being a desired result (from the point of view of a colleague). We conjecture that, in the near future, research on *intelligent* consistency restoration operations will reveal a tremendous amount of exciting techniques competing with more practical solutions, where the user selects from possible resulting models. Especially, the health care industry may hopefully benefit from each new idea in the field.

References

1. Abou-Saleh, F., Cheney, J., Gibbons, J., McKinna, J., Stevens, P.: Introduction to bidirectional transformations. In: Bidirectional Transformations - International Summer School, Oxford, UK, July 25-29, 2016, Tutorial Lectures. pp. 1–28 (2016), https://doi.org/10.1007/978-3-319-79108-1_1
2. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: A standard-compliant metamodeling framework with graph transformations. In: Rensink, A., Warmer, J. (eds.) Model Driven Architecture - Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006, Proceedings. pp. 361–375. Lecture Notes in Computer Science, Springer (2006), https://doi.org/10.1007/11787044_27
3. Anjorin, A., Lauder, M., Patzina, S., Schürr, A.: Emoflon: leveraging EMF and professional CASE tools. In: Informatik 2011: Informatik schafft Communities, Beiträge der 41. Jahrestagung der Gesellschaft für Informatik e.V. (GI), 4.-7.10.2011, Berlin, Deutschland (Abstract Proceedings). p. 281 (2011), http://subs.emis.de/LNI/Proceedings/Proceedings192/article314.html
4. Bak, K., Diskin, Z., Antkiewicz, M., Czarnecki, K., Wasowski, A.: Partial instances via subclassing. In: Erwig et al. [31], pp. 344–364, https://doi.org/10.1007/978-3-319-02654-1_19
5. Barriga, A., Rutle, A., Heldal, R.: Automatic model repair using reinforcement learning. In: Proceedings of AMMoRe@MoDELS'18 (to appear)
6. Bézivin, J., Büttner, F., Gogolla, M., Jouault, F., Kurtev, I., Lindow, A.: Model Transformations? Transformation Models! In: Nierstrasz, O. (ed.) MoDELS 2006, Genova, Italy. Lecture Notes in Computer Science, vol. 4199, pp. 440–453. Springer (2006), http://dx.doi.org/10.1007/11880240_31
7. Boronat, A., Knapp, A., Meseguer, J., Wirsing, M.: What is a multi-modeling language? In: Corradini, A., Montanari, U. (eds.) Recent Trends in Algebraic Development Techniques, 19th International Workshop, WADT 2008, Pisa, Italy, June 13-16, 2008, Revised Selected Papers. Lecture Notes in Computer Science, vol. 5486, pp. 71–87. Springer (2008), https://doi.org/10.1007/978-3-642-03429-9_6

8. Broy, M., Feilkas, M., Herrmannsdoerfer, M., Merenda, S., Ratiu, D.: Seamless model-based development: From isolated tools to integrated model engineering environments. Proceedings of the IEEE 98(4), 526–545 (2010), https://doi.org/10.1109/JPROC.2009.2037771
9. Cabot, J., Clarisó, R., Brambilla, M., Gérard, S.: Cognifying model-driven software engineering. In: Software Technologies: Applications and Foundations - STAF 2017 Collocated Workshops, Marburg, Germany, July 17-21, 2017, Revised Selected Papers. pp. 154–160 (2017), https://doi.org/10.1007/978-3-319-74730-9_13
10. Chen, X., Liu, C., Song, D.: Tree-to-tree neural networks for program translation. CoRR abs/1802.03691 (2018), http://arxiv.org/abs/1802.03691
11. Cheney, J., Gibbons, J., McKinna, J., Stevens, P.: On principles of least change and least surprise for bidirectional transformations. Journal of Object Technology 16(1), 3:1–31 (2017), https://doi.org/10.5381/jot.2017.16.1.a3
12. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: Automating co-evolution in model-driven engineering. In: 12th International IEEE Enterprise Distributed Object Computing Conference, ECOC 2008, 15-19 September 2008, Munich, Germany. pp. 222–231 (2008), https://doi.org/10.1109/EDOC.2008.44
13. Cicchetti, A., Ruscio, D.D., Eramo, R., Pierantonio, A.: JTL: A bidirectional and change propagating transformation language. In: Software Language Engineering - Third International Conference, SLE 2010, Eindhoven, The Netherlands, October 12-13, 2010, Revised Selected Papers. pp. 183–202 (2010), https://doi.org/10.1007/978-3-642-19440-5_11
14. Combemale, B., DeAntoni, J., Baudry, B., France, R.B., Jézéquel, J., Gray, J.: Globalizing modeling languages. IEEE Computer 47(6), 68–71 (2014), https://doi.org/10.1109/MC.2014.147
15. Czarnecki, K., Foster, J.N., Hu, Z., Lämmel, R., Schürr, A., Terwilliger, J.F.: Bidirectional transformations: A cross-discipline perspective. In: Proceedings of the 2Nd International Conference on Theory and Practice of Model Transformations. pp. 260–283. ICMT '09, Springer-Verlag, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-02408-5_19
16. Demuth, A., Riedl-Ehrenleitner, M., Egyed, A.: Towards flexible, incremental, and paradigm-agnostic consistency checking in multi-level modeling environments. In: Proceedings of the Workshop on Multi-Level Modelling co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages & Systems (MoDELS 2014), Valencia, Spain, September 28, 2014. pp. 73–82 (2014), http://ceur-ws.org/Vol-1286/p8.pdf
17. Diskin, Z.: Towards generic formal semantics for consistency of heterogeneous multimodels. Tech. Rep. 2011-07, The University of Waterloo (2011), http://gsd.uwaterloo.ca/node/330
18. Diskin, Z., Kokaly, S., Maibaum, T.: Mapping-aware megamodeling: Design patterns and laws. In: Erwig et al. [31], pp. 322–343, https://doi.org/10.1007/978-3-319-02654-1_18
19. Diskin, Z., König, H., Lawford, M.: Multiple model synchronization with multiary delta lenses. In: Fundamental Approaches to Software Engineering, 21st International Conference, FASE 2018, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2018, Thessaloniki, Greece, April 14-20, 2018, Proceedings. pp. 21–37 (2018), https://doi.org/10.1007/978-3-319-89363-1_2
20. Diskin, Z., Xiong, Y., Czarnecki, K.: Specifying overlaps of heterogeneous models for global consistency checking. In: Models, LNCS, vol. 6627, pp. 165–179. Springer (2011)

21. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From state- to delta-based bidirectional model transformations: The symmetric case. In: Whittle, J., Clark, T., Kühne, T. (eds.) Model Driven Engineering Languages and Systems, 14th International Conference, MODELS 2011, Wellington, New Zealand, October 16-21, 2011. Proceedings. Lecture Notes in Computer Science, vol. 6981, pp. 304–318. Springer (2011), https://doi.org/10.1007/978-3-642-24485-8_22
22. Easterbrook, S.M., Chechik, M.: A framework for multi-valued reasoning over inconsistent viewpoints. In: ICSE. pp. 411–420 (2001)
23. Egyed, A.: Fixing inconsistencies in UML design models. In: 29th International Conference on Software Engineering (ICSE 2007), Minneapolis, MN, USA, May 20-26, 2007. pp. 292–301 (2007), http://dx.doi.org/10.1109/ICSE.2007.38
24. Egyed, A.: Automatically detecting and tracking inconsistencies in software design models. IEEE Trans. Software Eng. 37(2), 188–204 (2011), https://doi.org/10.1109/TSE.2010.38
25. Ehrig, H., Mahr, B.: Fundamentals of Algebraic Specification 1: Equations and Initial Semantics. Springer-Verlag Berlin, Heidelberg (1985)
26. Ehrig, H., Ehrig, K., Hermann, F.: From model transformation to model integration based on the algebraic approach to triple graph grammars. ECEASST 10 (2008), http://journal.ub.tu-berlin.de/index.php/eceasst/article/view/154
27. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: A graphical approach to the operational semantics of behavioral diagrams in UML. In: ≪UML≫ 2000 - The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2-6, 2000, Proceedings. pp. 323–337 (2000), https://doi.org/10.1007/3-540-40011-7_23
28. Engels, G., Heckel, R., Küster, J.M.: The consistency workbench: A tool for consistency management in uml-based development. In: Stevens, P., Whittle, J., Booch, G. (eds.) ≪UML≫ 2003 - The Unified Modeling Language, Modeling Languages and Applications, 6th International Conference, San Francisco, CA, USA, October 20-24, 2003, Proceedings. Lecture Notes in Computer Science, vol. 2863, pp. 356–359. Springer (2003), https://doi.org/10.1007/978-3-540-45221-8_30
29. Eramo, R., Malavolta, I., Muccini, H., Pelliccione, P., Pierantonio, A.: A model-driven approach to automate the propagation of changes among architecture description languages. Software and System Modeling 11(1), 29–53 (2012), https://doi.org/10.1007/s10270-010-0170-z
30. Eramo, R., Pierantonio, A., Tucci, M.: Enhancing the JTL tool for bidirectional transformations. In: Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France, April 09-12, 2018. pp. 36–41 (2018), http://doi.acm.org/10.1145/3191697.3191720
31. Erwig, M., Paige, R.F., Wyk, E.V. (eds.): Software Language Engineering - 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26-28, 2013. Proceedings, LNCS, vol. 8225. Springer (2013), http://dx.doi.org/10.1007/978-3-319-02654-1
32. Evans, E.: Domain-Driven Design – Tackling Complexity in the Heart of Software. Addison-Wesley (Sep 2003)
33. Fiadeiro, J.L.: Categories for Software Engineering. Springer (2005)
34. Foster, J.N., Greenwald, M.B., Moore, J.T., Pierce, B.C., Schmitt, A.: Combinators for bidirectional tree transformations: A linguistic approach to the view-update problem. ACM Trans. Program. Lang. Syst. 29(3), 17 (2007), http://doi.acm.org/10.1145/1232420.1232424
35. Goguen, J., Ginali, S.: A categorical approach to general systems theory. In: Category Theory and Computer Programming. pp. 257–270. Plenum, New York (1978)

36. Habel, A., Sandmann, C.: Graph repair by graph programs. **To appear** in Proc. of GCM@STAF, 2018 (2018)
37. James, G.: The Tao of Programming. InfoBooks (1987), https://books.google.de/books?id=idkNAAAACAAJ
38. Johnson, M., Rosebrugh, R.D., Wood, R.J.: Lenses, fibrations and universal translations. Mathematical Structures in Computer Science 22(1), 25–42 (2012), https://doi.org/10.1017/S0960129511000442
39. Kästner, A., Gogolla, M., Selic, B.: From (imperfect) object diagrams to (imperfect) class diagrams: New ideas and vision paper. In: Proceedings of the 21th ACM/IEEE International Conference on Model Driven Engineering Languages and Systems, MODELS 2018, Copenhagen, Denmark, October 14-19, 2018. pp. 13–22 (2018), http://doi.acm.org/10.1145/3239372.3239381
40. Kästner, C., Apel, S.: Feature-Oriented Software Development, pp. 346–382. Springer Berlin Heidelberg, Berlin, Heidelberg (2013), https://doi.org/10.1007/978-3-642-35992-7_10
41. Kessentini, W., Sahraoui, H.A., Wimmer, M.: Automated metamodel/model co-evolution using a multi-objective optimization approach. In: Modelling Foundations and Applications - 12th European Conference, ECMFA 2016, Held as Part of STAF 2016, Vienna, Austria, July 6-7, 2016, Proceedings. pp. 138–155 (2016), https://doi.org/10.1007/978-3-319-42061-5_9
42. Knapp, A., Mossakowski, T.: Multi-view consistency in UML: A survey. In: Graph Transformation, Specifications, and Nets - In Memory of Hartmut Ehrig. pp. 37–60 (2018), https://doi.org/10.1007/978-3-319-75396-6_3
43. Ko, H., Zan, T., Hu, Z.: Bigul: a formally verified core language for putback-based bidirectional programming. In: Proceedings of the 2016 ACM SIGPLAN Workshop on Partial Evaluation and Program Manipulation, PEPM 2016, St. Petersburg, FL, USA, January 20 - 22, 2016. pp. 61–72 (2016), http://doi.acm.org/10.1145/2847538.2847544
44. König, H., Diskin, Z.: Efficient consistency checking of interrelated models. In: Modelling Foundations and Applications - 13th European Conference, ECMFA 2017, Held as Part of STAF 2017, Marburg, Germany, July 19-20, 2017, Proceedings. pp. 161–178 (2017), https://doi.org/10.1007/978-3-319-61482-3_10
45. Leblebici, E.: Inter-Model Consistency Checking and Restoration with Triple Graph Grammars. Ph.D. thesis, Darmstadt University of Technology, Germany (2018), http://tuprints.ulb.tu-darmstadt.de/7426/
46. Lopez-Herrejon, R.E., Egyed, A.: Detecting inconsistencies in multi-view models with variability. In: 6th European Conference on Modelling Foundations and Applications (ECMFA), Paris, France. pp. 217–232 (2010)
47. Macedo, N., Cunha, A.: Least-change bidirectional model transformation with QVT-R and ATL. Software and System Modeling 15(3), 783–810 (2016), https://doi.org/10.1007/s10270-014-0437-x
48. Macedo, N., Tiago, J., Cunha, A.: A feature-based classification of model repair approaches. IEEE Trans. Software Eng. 43(7), 615–640 (2017), https://doi.org/10.1109/TSE.2016.2620145
49. Meertens, L.: Designing constraint maintainers for user interaction. In: Mu, C. (ed.) Third Workshop on Programmable Structured Documents, PSD Laboratory, Tokyo University. pp. 1–3 (2005)
50. Nentwich, C., Emmerich, W., Finkelstein, A.: Consistency management with repair actions. In: Proceedings of the 25th International Conference on Software Engineering, May 3-10, 2003, Portland, Oregon, USA. pp. 455–464 (2003), http://computer.org/proceedings/icse/1877/18770455abs.htm

51. Paige, R.F., Brooke, P.J., Ostroff, J.S.: Metamodel-based model conformance and multiview consistency checking. ACM Trans. Softw. Eng. Methodol. 16(3), 11 (2007), http://doi.acm.org/10.1145/1243987.1243989
52. Rabbi, F., Lamo, Y., Yu, I.C., Kristensen, L.M.: A diagrammatic approach to model completion. In: Dingel, J., Kokaly, S., Lucio, L., Salay, R., Vangheluwe, H. (eds.) Proceedings of the 4th Workshop on the Analysis of Model Transformations co-located with the 18th International Conference on Model Driven Engineering Languages and Systems (MODELS 2015), Ottawa, Canada, September 28, 2015. CEUR Workshop Proceedings, vol. 1500, pp. 56–65. CEUR-WS.org (2015), http://ceur-ws.org/Vol-1500/paper7.pdf
53. Ruscio, D.D., Iovino, L., Pierantonio, A.: Coupled evolution in model-driven engineering. IEEE Software 29(6), 78–84 (2012), https://doi.org/10.1109/MS.2012.153
54. Rutle, A., Wolter, U., Lamo, Y.: A Diagrammatic Approach to Model Transformations. In: Proceedings of the 2008 Euro American Conference on Telematics and Information Systems (EATIS 2008). pp. 1–8. ACM (2008)
55. Sabetzadeh, M., Nejati, S., Liaskos, S., Easterbrook, S., Chechik, M.: Consistency checking of conceptual models via model merging. In: RE. pp. 221–230 (2007)
56. Schuerr, A.: Specification of graph translators with triple graph grammars. Lecture Notes in Comput. Sci. 903, 151–163 (1994)
57. Sen, S., Baudry, B., Precup, D.: Partial model completion in model driven engineering using constraint logic programming. In: Proceedings of the International Conference on the Applications of Declarative Programming (2007), http://www.irisa.fr/triskell/publis/2007/sen07b.pdf
58. Stünkel, P., König, H., Lamo, Y., Rutle, A.: Multimodel correspondence through inter-model constraints. In: Conference Companion of the 2nd International Conference on Art, Science, and Engineering of Programming, Nice, France, April 09-12, 2018. pp. 9–17 (2018), http://doi.acm.org/10.1145/3191697.3191715