CEUR-WS.org/Vol-3250/bxpaper3.pdf

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

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

[1]IAV GmbH, Germany

[2]Paderborn University, Germany

Abstract

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

Keywords

Bidirectional Transformations, Automotive Engineering, Traceability Maintenance

1. Introduction

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

Tenth International Workshop on Bidirectional Transformations (BX 2022)

tony@anjorin.de (A. Anjorin); nils.weidmann@upb.de (N. Weidmann); katharina.artic@iav.de (K. Artic)

© 2022 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)

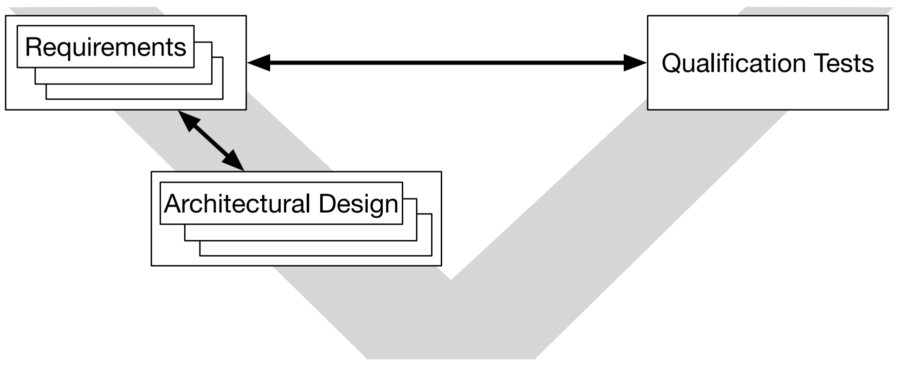

Figure 1: Our focus in the automotive domain

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

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

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

2. A characterisation of the problem domain

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

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

R1 Traceability links between requirements and architectural elements can be created, navigated, versioned, and generally used productively in either the requirements management tool or the architectural modelling tool (or sometimes in both). Convincing end users to use an additional tool solely for traceability maintenance is – in our experience – difficult.

R2 Traceability links that are clearly broken by changes that were made to one or both of the models are marked but preserved in a way that they can be reviewed and handled manually.

R3 A set of "suspect" traceability links can be determined for a manual review in some configurable manner (e.g., links connected to elements that have changed). This point is often optional as (R2) already requires a considerable amount of manual work.

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

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

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

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

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

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

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

3. Current solution strategies

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

3.1. A single tool for all models

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

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

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

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

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

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

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

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

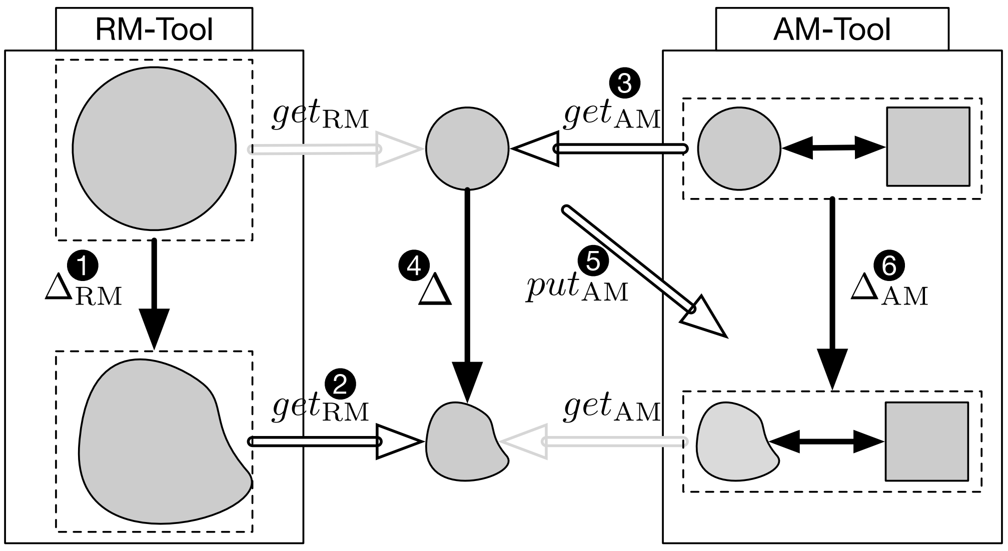

Figure 2: Using bx to synchronise a common overlap for traceability maintenance

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

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

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

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

3.3. An extra (backend) tool for traceability maintenance

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

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

4. How can future bx research help?

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

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

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

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

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

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

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

5. Conclusion

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

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

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

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

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

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

References

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

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

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

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

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

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

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

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

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

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

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

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