Pushing the Boundaries of Planned Reuse with Concern Specific Modelling Languages

Maximilian Schiedermeier
DISL & SCORE Labs, McGill University
Montréal, Québec, Canada
max.schiedermeier@mcgill.ca

ABSTRACT

Model-Driven Engineering (MDE) reduces complexity, improves Separation of Concerns and promotes reuse by structuring software development as a process of model production and refinement. A representative methodology is Concern-Oriented Reuse (CORE), an Aspect-Oriented Modelling (AOM) derivative that sets on partial models and model composition techniques to reach planned reuse. In CORE, proven solutions are bundled into concerns, reuse of which is then guided by a series of interfaces. CORE interfaces set on General Purpose Modelling Languages (GPML), therefore concern integration and reuse can be hindered by accidental complexity, arising out of a semantic mismatch between a concern's nature and GPML concepts. The established MDE answer to counter accidental complexity is Domain Specific Modelling Languages (DSML). However, it is unclear how DSMLs can be combined with partial model and model composition methodologies, and if such a combination could effectively redefine the boundaries of planned reuse. In this extended abstract I present findings on the nature of DSMLs, eligible for this combination and argue why they form a category on their own: *Concern Specific Modelling Languages* (CSML). I present a reliable framework for systematic integration of CSMLs into reusable concerns, and elaborate a representative novel concern. Finally, I describe experiments that allow measuring the effects of this novel concern on software design, implementation and planned reuse.

CCS CONCEPTS

• **Software and its engineering → Domain specific languages**; **Software design engineering**; **Reusability**; *Source code generation.*

KEYWORDS

Model-Driven Engineering, Concern-Oriented Reuse, Concern-Specific Modelling

ACM Reference Format:
Maximilian Schiedermeier. 2022. Pushing the Boundaries of Planned Reuse with Concern Specific Modelling Languages. In *ACM/IEEE 25th International Conference on Model Driven Engineering Languages and Systems (MODELS '22 Companion), October 23–28, 2022, Montreal, QC, Canada.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3550356.3552375

MODELS '22 Companion, October 23–28, 2022, Montreal, QC, Canada

ACM ISBN 978-1-4503-9467-3/22/10.
https://doi.org/10.1145/3550356.3552375

1 RESEARCH PROBLEM AND MOTIVATION

Modern software needs to cope with the ever increasing complexity of systems [4], and hence *reducing complexity* is a primary objective of software engineering. In *Model-Driven Engineering* (MDE) system complexity is reduced with the help of modelling languages, which each focus on a given level of abstraction. Multiple modelling languages can also be combined, to allow the developer to express the properties of a system from various points of view, thus promoting *Separation of Concerns*. On top of Separation of Concerns, MDE further counters system complexity by model reuse. While opportunistic reuse of entire models is a common practice, reuse of partial (and hence incomplete) models is more challenging. Still, with aspect-oriented modelling (AOM) techniques it is possible to compose partial models from one context to another. This practice enables planned reuse, where models are intentionally reduced to partial models, resulting in higher genericness and potential for reuse.

Concern-Oriented Reuse (CORE) [1] is an approach based on AOM that streamlines model reuse by encapsulating common solutions as partial models inside a reusable unit called a *concern*. Concern users can then apply model transformations to connect partial models across levels of abstraction, effectively *reusing architectural and design knowledge*, or *platform-specific development expertise*. From the perspective of a concern user, this reuse process is experienced as three sequential stages [6]:

(1) A *variation interface* (VI), which exposes the different variants offered by the concern, and their impacts on high-level system qualities.
(2) A *customization interface* (CI), where the concern designer exposes the generic entities in the concern that have to be adapted to a specific reuse context.
(3) A *usage interface* (UI), which defines how the functionality encapsulated by a concern may be used.

CORE streamlines the reuse process by allowing a *concern user* to a) choose a desired variant (from the VI), b) adapt the chosen models to the reuse context (with the CI), and c) use the structure and behaviour encapsulated by the concern (exposed in the UI).

In the above workflow, the concern user relies on *General Purpose Modelling Languages* (GPML) throughout *Customization* and *Usage* stages. GPMLs such as UML mostly cover the typical structural and behavioural modelling needs for software development, therefore for many concerns this provides an adequate level of abstraction. However, the Separation of Concerns power of MDE is limited when it comes to development concerns that do not align with the levels of abstraction of the MDE process and the GPMLs used.

In this case the semantic mismatch introduces what is called *accidental complexity*. In the context of CORE this translates to

concerns that cannot be easily reused at CI and UI stage, due to their inherent mismatch on GPML concepts. In general, the semantic gap between a specific application domain and the concepts offered by GPMLs can be bridged with a *Domain Specific Modelling Language* (DSML) [3]. As such the idea to integrate tailored DSMLs into a planned reuse process is promising. Such combination could resolve the problem of accidental model complexity within concern reuse interfaces, and hence simplify planned reuse significantly. Implicitly, such a DSML would be part of the concern unit of reuse, and hence also showcase partiality - the full potential of adherent models is by design meant to unfold only when combined with a composition specification, toward an application context. In the following I refer to this kind of DSMLs as *Concern Specific Modelling Language*s (CSML).

While CSMLs greatly simplify the reuse process for concern users, CSMLs are unfortunately challenging to develop and integrate with the standard MDE process for the concern user. Designing a purposeful concern is already challenging by nature: A concern designer must express thorough domain knowledge into versatile models, for maximized reusability and convenience. Adding the design and placement of a CSML into the concern integration process renders this an even greater challenge. We generalized the described trade-off into a first research question:

RQ1: Are DSMLs an adequate way to bridge the semantic gap that retrains partial model reuse, when the nature of the unit of reuse does not align well on GPML concepts?

We believe that a single anecdotal CSML-enabled concern is insufficient to declare general feasibility. Ideally, the concern integration procedure itself is subject to a clear, modular plan of action that guides toward expressive CSML-enabled concerns.

RQ2: To which extent can we formulate the generic stages of a successful CSML-concern integration into an assistive methodology?

Finally, even if CSMLs are a technical possibility it remains unclear if adherent concerns bring a measurable benefit to the MDE community, comparable to classic CORE concerns.

RQ3: Does the reuse of CSML concerns measurably stand up to the benefits of classic CORE?

2 BACKGROUND AND RELATED WORK

The above research questions touch multiple well-studied fields. In the following, I present fundamental concepts and contributions, which serve as building blocks. Common to all related work is a relevance for integrating DSMLs with existing planned reuse methodologies.

2.1 MDE, Accidental Complexity and DSMLs

Model-Driven Engineering (MDE) [5, 12] is a unified conceptual framework in which the whole software life cycle is seen as a process of *model production*, *refinement* and *integration*. Models are built representing different views of a software system using different formalisms, i.e. modelling languages. Given an appropriate language choice, a model can concisely express the properties important at the current level of abstraction.

GPMLs such as UML mostly cover typical structural and behavioural modelling needs for *software development*. However, their general purpose nature causes a semantic gap between a specific application domain and GPML concepts. This leads to *accidental complexity*, effectively hindering the software life cycle. DSMLs target a specific application domain and can therefore provide a better match (see 1). However, the introduction of DSMLs has itself an effect on the MDE life cycle. Existing model transformations e.g. code generators, are in most cases incompatible to novel languages. Hence countering accidental complexity with DSMLs introduces a need for additional model transformations. This is a known DSML related challenge and existing research, such as *Melange* [2] intends to counter this overhead by a set of reusable engine related tools, that enable reusing languages, their metamodels and even associated transformations.

2.2 DSMLs in the context of planned reuse

Reuse is central to DSMLs and MDE in general, the main unit of reuse being the modelling language. A modeller using a modelling language is reusing knowledge of the language engineer. Modelling languages typically come with a tool that ensures consistency between views of the system at the same level of abstraction. Additionally, reusable model transformations automate the refinement of models when moving between levels of abstraction, thus *reusing architectural and design knowledge* or *platform-specific development expertise*.

In the light of my research questions, I am particularly interested in existing planned reuse techniques that could benefit from integrated DSMLs. A fitting study context is AOM approaches that showcase practical limitations due the described effects of accidental complexity.

In AOM, a modelling language is augmented with advanced language features that enable the modularization and composition of model fragments. These are models that are not necessarily viable in isolation. A model weaver is a special model transformation that takes as an input two models and a composition specification, to produce a new *composed* output model in which the two input models have been merged.

As foreshadowed in 1, CORE is a fitting representative. CORE used to set uniquely on GPMLs, which in return meant that any non alignment of a concern on the available GPML concepts resulted in unmanageable accidental complexity, ultimately restricting concern integration and reuse.

3 APPROACH AND UNIQUENESS

In this section I present how our previous and scheduled contributions align on the primary research questions (see 1). I then delineate the general uniqueness of our advancement.

3.1 Approach

Ultimately all realized and ongoing experiments were designed to provide insight on the presented research questions. We therefore study feasibility and effects of CSML integrations into existing partial model reuse methodologies, on the example of CORE (see 2).

We ran a thorough analysis of CORE's established workflow and extracted a methodology, to outline how concerns with integrated CSML could best make use of the existing CORE toolchain. In [10],

we presented *FIDDLR*, a methodology that defines clear separable tasks for this goal. *FIDDLR*'s contribution is twofold: It demonstrates the general feasibility of crafting CSML-enabled concerns, and provides hands-on instructions for all associated steps. The latter are perfectly in the spirit of Separation of Concerns: The three main tasks, namely *Design of a fitting DSML*, *Definition of model transformations to translate to GPML models*, and *Provision of concern design models* are separated tasks that can be easily fulfilled by individual experts.

Yet we did not stop at a methodology definition. We applied *FIDDLR* to elicit a novel concern, that due to semantic mismatch on GPML concepts would have been impossible to integrate otherwise. We followed the plan of action suggested by *FIDDLR*. Although this process induced several framework refinements, the outcome was an operational novel concern named *RESTify* [10]. *RESTify* is a refactoring concern that allows the assisted refactoring of legacy functionality into a RESTful service. For this purpose we designed a novel "*partial*" DSML: *ResTL* [8]. *ResTL* is a pertinent example of a CSML, since unlike existing related languages it is intentionally designed to be insufficient for the purpose of a standalone REST interface description (see Fig. 1). However, when *ResTL* models are joined with a composition specification on existing legacy models, that combination provides the semantic equivalent to existing wholesome REST interface DSMLs.

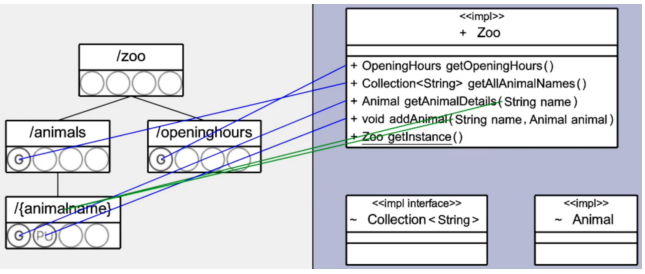

Figure 1: *ResTL* model (left) and composition specification (lines) with a class diagram (right). ResTL defines resources and enabled HTTP operations, but *in isolation* is no semantic equivalent to a REST API specification language.

However the integration of *RESTify* was not only motivated by a practical validation of *FIDDLR*. The concern itself can be considered a representative for a new category of reusable units: Concerns that bundle integrated CSMLs and the required model transformations to maximise reuse of the existing CORE methodology, and that without *FIDDLR* could not have been easily integrated. Next we investigated whether this novel concern category is easily reused, as well as the measurable effects of such reuse on MDE. To test *RESTify*'s potential on representative software artifacts, we advanced CORE's reference implementation TouchCORE: we added multi-languages support [11] and a convenient generic split view to specify compositions specifications between CSML and GPML concepts.

We then tested whether we could conveniently turn sample software [9], archetypal for a venue, an e-commerce and a gaming context, into operational RESTful services. Next we systematically tested the outcome with unit tests, to validate the behaviour of the generated and deployed code. Although this practical validation once more led to subsequent refinements and minor bugfixes, ultimately all applications were successfully converted into operational RESTful services. We likewise tested whether this reference concern would readily support fundamental CORE advantages, such as fast reconfiguration of the concerns feature selection (see 1). In the context of *RESTify* this translates to the concern user being able to switch between different REST technologies, e.g. *Spring Boot* and *Apache CXF*, and regenerate deployable service code within a matter of seconds. Since the integrated CSML and composition specification capture the semantics of the RESTifcation process, reduced of any technological selection, we were able to ensure direct support for four alternative target REST technologies.

Yet we acknowledge that in our role as concern designers we are too biased to estimate the practical value of *RESTify* ourselves. We therefore currently conduct a controlled experiment with independent developers of various backgrounds. Goal is to fairly test reuse of the *RESTify* concern, by asking external developers to refactor our sample applications. To eliminate learning effects and provide insight on the practical contribution of *RESTify*, we designed the study to have four control groups, which allows fair cross validation. We captured primary metrics such as "*time required per refactoring task*" and measured the "*quality of the outcome*" with automated tests to get an understanding of the concern's immediate impact on SE. However, we expect deeper insight from a parallel analysis of the participant's task activities, documented by screen recordings. The latter will provide us with a better understanding on how the concern impacts developer habits and task oriented advancement. Since the experiment is still running we can not yet draw final conclusions, however the data gathered so far suggests a substantial positive effect on both, the refactoring process and the quality of the outcome, is to say the produced RESTful service code.

3.2 Uniqueness

Although our approach is heavily based on existing work, to the best of our knowledge there is no related approach that combines the building blocks in a comparable way. We believe the idea of packaging CSMLs within a unit of reuse, alongside all required model required transformations, is genuine and new. Although it is not uncommon to pair DSMLs with tailored tooling, (c.f. IDEs with integrated DSMLs such as XCode's Storyboard [7]), existing tools rarely apply CSMLs to forward planned reuse in form of an encapsulated unit.

Also, we are not aware of another methodology than *FIDDLR*, that delineates clear and separable tasks for the streamlined integration of CSML-enabled concerns.

Finally, we believe that *RESTify* is by itself unique. Not only in terms of its composition, that comprises amongst others *ResTL*, a novel CSML that perfectly reflects the minimal concepts required for the associated refactoring task - *RESTify* is also unique in its capacities: Although there is a variety of tools that support stub code generation based on a provided REST interface description, we are not aware of any industrial or academic tool that supports generating of readily deployable service code. Contrary to other tools, *RESTify* omits the need for a manual merging of generated REST stubs with legacy code. On top there seems to be no other

approach that also supports the dynamic selection of the target REST technologies. *RESTify* covers the latter by means of an integrated feature tree. We believe these characteristics to be exciting and unique features that further underline the potential of CSMLs.

4 RESULTS AND CONTRIBUTION

In this last section I recapitulate acquired results and match them on the initial research questions. The paper then concludes with an interpretation of our contributions, followed by a short outlook on upcoming future work.

4.1 Results

Our research is motivated by a present MDE challenge, namely planned reuse being hindered and restricted by accidental complexity. We compiled arguments why the use of DSMLs - the standard MDE way to bridge semantic mismatch - is not easily applied in this context. We identify as main reason the disrupting nature of supplementary languages, regarding existing model transformation workflows. The assumed trade-off between common DSML advantages and the challenges anticipated in this context then served as motivation for **RQ1**. Subsequently we argued how CORE is a fitting study object, as it exemplifies this challenge.

Since concern integration is a challenging task we further argued how a successful integration of DSML-enabled concerns almost inevitably requires a reliable methodology. Yet for a start it was not clear whether such a methodology exists, and to which extent the concern integration process can be generalized, leading to **RQ2**. Careful analysis of the existing workflow and model composition techniques inspired the design of *FIDDLR*, a modular framework that guides the integration of DSML-enabled concerns.

Further evaluation of *FIDDLR* suggested that the framework is a viable methodology and provides the necessary guiding. We were able to validate *FIDDLR*'s plan of action, by integration of a novel concern: *RESTify*. Throughout the concern integration we advanced CORE's reference implementation, *TouchCORE*, and extended it to practically support reuse of DSML-enabled concerns (see 3).

This work contributes to our understanding of the language nature that renders a DSML candidate for concern integration. We observed how integrated languages would substantially differ from existing DSMLs - namely them being partial by design, as exemplified by *ResTL*. This observation is coherent to fundamental AOM principles, where crafted models are intentionally partial and only gain full expressiveness through composition specifications, combined with additional artifacts. We hence coined the term *Concern Specific Modelling Languages* (CSMLs) for this category of DSMLs.

From our practical experiments we also retained a strong coupling between CSMLs and the associated transformations. Although implicitly suggested by *FIDDLR*, we noted this coupling justifies a combined packaging. We believe this is a fundamental consequence of the CSMLs' partial nature.

Finally, in pursue of **RQ3** we investigated the effects of CSML concern reuse in the light of classic core. We designed and conducted a controlled experiment around *RESTify*, to test reuse of a representative novel concern. Although not yet fully obtained and evaluated, the collected data suggest beneficial MDE effects, notably on behalf of code correctness and development times.

4.2 Contribution

Although the initial research questions are not yet entirely answered to our satisfaction, the presented observations do serve as ground for a preliminary evaluation.

First of all we deem that the general feasibility of combining DSMLs with the power of planned reuse proven. *FIDDLR* provides a plan of action, and also demonstrates its effectiveness by means of a sample concern, *RESTify*. We also believe the merits of CSML-enabled concerns are at this point free of doubt. Our observations around *RESTify* include SE features that in understanding are unprecedented. We see this as evidence for the potential of CSML-enabled concerns. We believe the data gathered around our ongoing controlled experiment underlines this impression. Based on the above evaluation, we believe that this suggests a positive answer for the primary research question (**RQ1**). The envisioned technological combination is feasible (**RQ2**) and beneficial (**RQ3**), thus we also consider it viable. We want to orient future research on two goals: Firstly, to collect evidence of CSML-concern merits, using a second CSML-enabled reference concern, integrated with *FIDDLR*. Secondly, to examine the compatibility of CSMLs in other planned reuse contexts than CORE. Ultimately we expect that this will strengthen a general understanding of CSML characteristics and how partial languages are best packaged with complementing model transformations, to redefine the boundaries of planned reuse.

REFERENCES

[1] Omar Alam, Jörg Kienzle, and Gunter Mussbacher. 2013. Concern-Oriented Software Design. In *Proceedings of the 16th International Conference on Model-Driven Engineering Languages and Systems - MODELS 2013 (Lecture Notes in Computer Science, Vol. 8107)*. Springer, Berlin, Heidelberg, 604–621.

[2] Thomas Degueule, Benoit Combemale, Arnaud Blouin, Olivier Barais, and Jean-Marc Jézéquel. 2015. Melange: A Meta-Language for Modular and Reusable Development of DSLs. In *Proceedings of the 2015 ACM SIGPLAN International Conference on Software Language Engineering* (Pittsburgh, PA, USA) *(SLE 2015)*. Association for Computing Machinery, New York, NY, USA, 25–36. https://doi.org/10.1145/2814251.2814252

[3] Jeff Gray, Juha-Pekka Tolvanen, Steven Kelly, Aniruddha Gokhale, Sandeep Neema, and Jonathan Sprinkle. 2007. Domain-Specific Modeling. In *Handbook of Dynamic System Modeling*. CRC Press, Boca Raton.

[4] M. Jamshidi. 2008. *System of systems engineering? New challenges for the 21st century*. Wiley, Hoboken, NJ. 616 pages.

[5] Stuart Kent. 2002. Model Driven Engineering. In *International Conference on Integrated Formal Methods – IFM*. Springer-Verlag, London, UK, 286–298.

[6] Jörg Kienzle, Gunter Mussbacher, Omar Alam, Matthias Schöttle, Nicolas Belloir, Philippe Collet, Benoit Combemale, Julien Deantoni, Jacques Klein, and Bernhard Rumpe. 2016. VCU: the three dimensions of reuse. In *International Conference on Software Reuse*. Springer, Berlin, Heidelberg, 122–137.

[7] Rory Lewis, Yulia McCarthy, and Stephen M Moraco. 2012. *Beginning IOS Storyboarding: Using Xcode*. Apress.

[8] Maximilian Schiedermeier. 2020. A concern-oriented software engineering methodology for micro-service architectures. In *Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*. 1–5.

[9] Maximilian Schiedermeier. 2022. RESTify context software bundle. https://github.com/kartoffelquadrat/RestifySoftwareBundle.

[10] Maximilian Schiedermeier, Jörg Kienzle, and Bettina Kemme. 2021. Internation Conference on Software Language Engineering. In *FIDDLR: Streamlining Reuse with Concern-Specific Modelling Languages*. ACM, New York, NY, USA, 81–88.

[11] Maximilian Schiedermeier, Bowen Li, Ryan Languay, Greta Freitag, Qiutan Wu, Jörg Kienzle, Hyacinth Ali, Ian Gauthier, and Gunter Mussbacher. 2021. Multi-Language Support in TouchCORE. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE, 625–629.

[12] Douglas C. Schmidt. 2006. Model-Driven Engineering. *IEEE Computer* 39 (2006), 41–47.