Towards a Structured Workflow Language for Model Management*

Sahar Kokaly

Department of Computing and Software
McMaster University, Hamilton, ON, Canada
kokalys@mcmaster.ca

Abstract. In Model Driven Engineering (MDE), models and mappings play a key role in system design. However, in practice, models and mappings do not exist in isolation, but are combined to form *systems of interrelated models*. We call the trace of operations, such as model transformations or model merges, between an initial configuration of a system of interrelated models to a final one, *a workflow*. Current approaches for using workflows in MDE exist, but are generally informal and do not properly address traceability and verification. In this work, we propose a structured method for defining workflows for model management, which automatically ensures traceability and inherently enables verification. This approach also sets the stage for defining a declarative workflow language, which we believe can aid in validation. Through this framework, comparison and optimization of workflows is possible, as they are represented as algebraic terms in a mathematically defined language. Finally, the framework gives rise to multiple levels of abstraction, making it flexible enough to be used at different stages of the system design, while enabling better workflow readability and maintainability.

1 Problem

The need for workflows in MDE.

Model Driven Engineering (MDE) focuses on the use of models and mappings between them to drive the software development cycle. However, models and mappings do not exist in isolation, but are combined to form *systems of interrelated models*, that are chained through the use of operations, to form *workflows* that fulfill a given intention in the software design. Forming such chains is therefore a natural step in MDE to enable the description of the composition of activities in software construction and provide explicit means for MDE automation [8]. Consider the Epsilon family of languages [1], which is comprised of various domain-specific languages that are used for tasks such as model transformation (Epsilon Transformation Language (ETL)), model validation (EVL), model merge (EML), model comparison (ECL) and code generation (EGL). In order to combine the outputs of these languages and form a chain of model management operations, a workflow language must be used.

* This work is being done as part of the NECSIS project, funded by Automotive Partnership Canada and NSERC. I would like to thank Dr. Tom Maibaum, Dr. Zinovy Diskin, and Dr. Richard Paige for their supervision in this work.

Workflows should be modelled.

However, in order to stay faithful to the MDE philosophy, workflows should also be modelled before implementation [9], making them amenable to analysis and independent of platform specific details. But, as supported in [8], to the best of our knowledge, little work is devoted to understanding the underlying structure of such workflows when they are used in MDE.

Models of workflows should be structured.

As was recently stated in an article by Whittle et al.: *"It turns out that the main advantages are in the support that MDE provides in documenting a good software architecture. Most would agree that a clearly described software architecture is one of the key ingredients for successful software development"*[10]. In order for MDE to provide this advantage, workflows in MDE should not only be modelled, but be defined in a *structured* way that makes their specification serve as a good documentation method, and as a basis for sound implementation. Structured approaches have historically demonstrated their value in managing the complexity of systems. For example, the shift from assembly programming to the structured programming paradigm has proved to be a good design decision, and we believe the same is true for the way workflows should be defined in MDE.

The right level of abstraction.

The current way workflows are modelled is via diagrams that contain only unstructured nodes and unstructured arrows. However, this makes the workflow definitions too abstract, which does not lend itself to their disciplined implementation, or checking their correctness. To further motivate the importance of this problem, consider the following model management scenario:

User: "I have two models: M1 and M2. I would like to merge them and then generate code from the result. I would then like to trace back to see what part of the code came from M1."

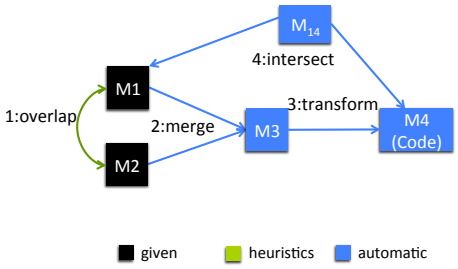

Fig. 1. A simple workflow...

```
import models M1, M2
import overlap O=O(M1, M2)
merge (M1,M2) into M3 using O
import transformation t
apply t to M3 to produce Code
Intersect (M1,Code) into M14
```

Fig. 2. ...and the script implementing it.

This can be depicted graphically in Fig. 1, and a script implementing this workflow scenario is shown in Fig. 2. We start with two input models $M1$ and $M2$, define their **overlap** in the first step of the workflow, then **merge** these two models into a third model $M3$ via a merge operation. Afterwards, some **transform** operation is used to generate code ($M4$). In order to find which part

of the code came from $M1$, an **intersect** operation is used, which yields M_{14}. The main issue with the current way the workflow is represented is that there exists a very large gap between the diagram and the code that would implement it. We would therefore like to propose a way to model the workflow that is still abstract, but not too abstract to be too far from the semantics of the operations and how they are composed.

Traceability mappings as first class citizens.

Traceability is increasingly required in software development at the stakeholder level (e.g., to ensure a given requirement has been implemented in the system), but also at the software development level (e.g., to ensure traceability as high level models are refined along the development process) [8]. Because model management workflows explicitly model the relations between the several steps of an MDE process, traceability is a natural consequence of using such workflows. But, as in any system, in order to be able to reason about it, it should be modelled by a correct mathematical model. If the model is incomplete, reasoning is not possible [5]. Based on that, traceability mappings should be considered first class citizens in the workflow model; if they are left implicit, they cannot be directly used for reasoning and analysis. To do this, traceability mappings should be included in the arity of the model management operations (i.e., the typing of their parameters), and then, the composition of these operations to construct workflows will also have explicit traceability.

Verification and Validation of Workflows.

Finally, it is known that the presence of errors in models and model management operations risks both the reliability of MDE-based processes and the soundness of the resulting products. For this reason, there is a great need for mechanisms to ensure quality and the absence of errors in models and model management operations. This propagates to the workflow level as well, creating a need for mechanisms to ensure quality and absence of errors in the model management workflows. To do this, verification and validation techniques are typically used, but as far as we know, none exist in the area of modelling workflows in MDE. For verification, what is needed is a mechanism to ensure that the workflows satisfy one or more correctness properties. For validation, mechanisms to check whether the workflows meet the user requirements are needed. The latter is typically done through testing, but some formal methods, like model checking, can also aid in validation.

In this work, we propose a method that we argue will improve how workflows are modelled in MDE. We see this work being useful in both theory and practice and of interest to audiences from both academia and industry that are generally interested in specifying their model management tools and reasoning about their model management chains.

2 Related Work

In this section we provide an overview of current approaches in the area of workflows for MDE, and summarize with a list of problems.

Epsilon is a suite of languages that are used for modeling and applying operations on models, such as comparison, validation, transformation, merge, etc. [1]. In order to make use of the power of these languages and build a complex system, a workflow needs to be defined that executes tasks, prescribed in the different languages, in a predefined order. The current tool of choice to do this in Epsilon is the ANT tool [1]. What ANT does is similar to what a "make file" does. In ANT, each workflow is captured as a project, and each project consists of a number of targets. There is a default target that represents the starting state, and each target contains a number of tasks that depend on other targets that must be executed before it. In this manner, what ANT does is simply define a control flow with provision for branching and looping.

The Formalism Transformation Graph (FTG) and its complement, the Process Model (PM) as presented in [8], together form a framework for explicitly describing model transformation chains in MDE. The FTG describes the different languages that can be used at each stage of model development. The PM models the control flow and data flow between each transformation action in the chain. Though traceability is claimed in this work, it is not clearly shown.

Model Transformation Chain (MTC) Flow is a tool that allows MDE developers to design, develop, test and deploy MTCs [2]. It offers a graphical editor for defining any MTC and suggests alternative execution paths for MTCs. One obvious disadvantage to this tool is that it currently only supports sequential transformation chains, and does not enable activities like merging or comparing between models within a chain.

UML Activity Diagrams are used for specifying workflows and takes into account both data and control flow [6]. There has been work based on using UML activity diagrams as a workflow modeling language [9], but it has been shown in the same work that although UML's activity diagrams are well-suited for this problem, they are not ideal. Traceability in UML activity diagrams can be ensured at a high level (i.e., between interfaces of activities and operations), and not at the lower level desired in MDE (i.e., the model and mapping level).

In summary, below are the issues we have identified and plan to address:

- **Modeling**: Workflows are not always modelled before implementation, and if they are, it is not in a structured way that enables reasoning.
- **Expressiveness**: Many languages/tools support sequential composition only and do not support more interesting workflow combinators such as parallel composition and branching.
- **Traceability**: As far as we understand, traceability is either implicit or non-existent. A way to automatically ensure traceability, or use it for reasoning and analysis, is not possible with current approaches.
- **Verification**: This cannot be done without the notion of data flow, and requires a grammar from which to build terms and the ability to check whether a term is correct or not.
- **Validation**: In general, this is not straightforward, but we think it can be achieved with the help of a proper declarative workflow language.

3 Proposed Solution

This section provides a high level overview of our proposed solution to modelling workflows in MDE. We propose a structured approach, and for the purposes of this paper, we use diagrams to exhibit this structure.

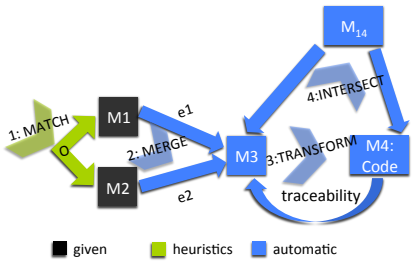

Fig. 3. Structured workflow through diagrams.

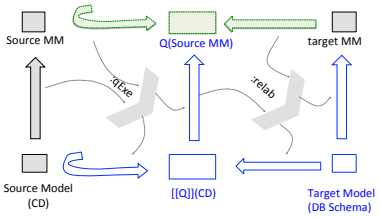

Fig. 4. Transformation diagrammatically.

Back to our example from Section 1, we model our workflow once again using a diagram as depicted in Fig. 3. Now, the operations appearing on the chevrons are themselves defined diagrammatically. For example, consider the transformation operation, which was first proposed in [3] and is shown in Fig. 4. The transformation is defined through the tiling (composition via a common edge) of two operations: `qExe`, which specifies query execution, and `relab`, which relabels the query result in terms of the target metamodel. Note that the output of one operation (`qexe`), namely the vertical blue arrow, becomes part of the input to the next operation (`relab`). Also, notice two types of output mappings from the generated target model; the vertical blue arrow which represents a typing mapping to the target metamodel, and the horizontal blue arrow which represents a traceability mapping. This way of defining the transformation automatically ensures traceability in the system, as the traceability mapping is explicit in the *arity* of the operation, or, in other words, the typing of its parameters and outputs; when the parameter or output is a graph, the type is a graph shape. We can then define a typing system for the operations (i.e., prescribing their (graphical) arities), which is also diagrammatic in its nature. This is shown for a subset of model management operations, namely, `match`, `merge`, `transform` and `intersect`, in Fig. 5.

Name	Input Arity	Output Arity
Match		
Merge		
Transform		
Intersect		

Fig. 5. Typing system for a subset of model management operations.

Workflow definitions defined in this manner can then be parsed into a Directed Acyclic Graph (DAG) which can be used to verify correctness of the terms. Some properties that can be checked are lack of cycles in the DAG and that each operation satisfies its typing, presented in Fig. 6.

Finally, we can construct a workflow language ($W = (\Sigma_{MMT}, \Sigma_C)$), where Σ_{MMT} is the signature of model management operations and Σ_C is the signature of workflow combinators. For our current simple scenario, a workflow would be structured as an algebraic term $W = (match; merge; transform; intersect)$, where $match, merge, transform, intersect$ are from Σ_{MMT} and ; is sequential composition from Σ_C. This algebraic term represents the intent of Fig. 3, and the meaning of sequential composition is given by the tiling explained in [3] and shown in Fig. 4.

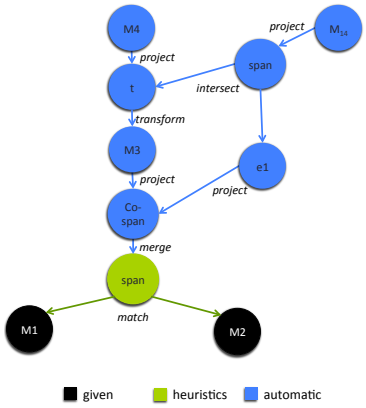

Fig. 6. Parsing the algebraic workflow using a DAG.

We envision our framework supporting the full range of model management operations supported by standard model management tools, and containing workflow combinators that we will derive from common model management scenarios.

In summary, we specify our system of interrelated models via graphs and graph mappings, constraints on such a system via diagram predicates, and operations on these systems via diagram operations. The latter are then composed (tiled) to form complex chains which define workflows in our language.

4 Preliminary Work

This work started by examining the emerging concept of "megamodeling" in MDE. We quickly realized that the way megamodeling is discussed in most papers lacks a theoretical foundation and therefore is not amenable for verification or reasoning. We decided to look at megamodeling from a more formal view, and we presented "Mapping-Aware Megamodels" in [4]. In the mentioned work, we proposed that megamodels are models of systems of interrelated models and operations over them. We showed how to specify this system of interrelated models, some operations over such a system and composition of such operations. The highlight of our approach is that all of the above was specified in a non-ad hoc way, meaning, we proposed a general approach for specifying megamodels in MDE that is built on mathematical foundations borrowed from Category Theory. What we ended up with was a library of design patterns and laws for megamodelling, that we believe act as building blocks important for laying the foundations of a systematic engineering approach to this field.

In related work done within our group [7], a declarative approach to model transformations that is query structured is presented. The initial results show that such an approach when compared to procedural model transformation approaches makes the transformation definition more structured, which leads to better readability, maintainability and verifiability of the transformations. This way of defining a model transformation is an example of how we would like to structure our model management operations, allowing us to compose them in

such a way that guarantees correctness of the chain by construction, and providing better readability, maintainability and verifiability at the workflow level.

5 Expected Contributions

The main contributions of this work will be:

- A structured specification of megamodels: Models and mappings as first class citizens modelled via graphs and graph mappings, respectively, the constraints on them modelled via diagram predicates, and the operations on them modelled via diagram operations.
- A workflow language for megamodeling that is built via the composition (tiling) of diagram operations.
- Evaluation of the workflow language and a set of suggested improvements to the state-of-the-art in the area of workflows in MDE.

We expect that our work will serve as a framework on which model management languages, and model transformation chain tools can build specifications for their tools and verify correctness of their chains. In addition to addressing the issues in the state-of-the-art (refer to Section 2), we believe our work will provide the following advantages to the way workflows are modelled in MDE.

- **Readability and Maintainability**: Workflows that are easier to read and write, and are therefore more maintainable. Early work showing this with regards to using a declarative model transformation approach shown in [7].
- **Multiple levels of abstraction**: Workflows that can be modelled at different levels of abstraction, making them usable at various stages of the design, from the initial stage of communicating with the domain expert to get the requirements for the workflows, down to code generation.
- **Comparison and Optimization**: Representing workflows as algebraic terms will enable us to *compare* workflows to check if they are semantically equivalent, and *optimize* workflows to find more efficient ones.

6 Plan for Evaluation and Validation

For validation, we plan to build our workflow language by example. What this means is that we will start with simple case studies to help define what model management operations and workflow combinators will form our language, and then work our way up to more complex examples while expanding our language.

For evaluation, the plan is to compare our approach to similar approaches by taking a practical model management scenario[1], modelling it in the various approaches (including ours), and performing analysis with respect to predefined criteria. For example, readability, ease of use and maintainability of the workflow definitions can be achieved through a usability study that would reveal how much the various approaches aid in improving communication and help in different

[1] We plan to conduct an industry case study from the automotive domain which has been used in the NECSIS project, namely the power window case study used in [8]. We would also like to aim for another industry case study which we hope to define as the work progresses.

design stages. Other criteria include how traceability is defined and correctness of the methods. As an outcome of the evaluation step, we hope to identify a set of weaknesses in the related approaches and come up with a set of suggestions for improving the way workflows are modelled in MDE.

7 Current Status and Timeline

The remainder of the PhD will take our work in [4], expand the work on diagram operations and workflows by adding more operations and combinators, and implement the parsing approach to work with these operations and combinators. As explained in Section 6, we plan to build our workflow language "by example" which means the case study work will be performed in parallel with the workflow language specification. We expect that this approach will help reveal the model management operations and combinators to be included as part of the signatures that define our workflow language, as a primary step, and help evaluate our language as a secondary step. The planned timeline for this work is as follows:

- More detailed literature review. To be completed by Oct'14.
- Workflow language specification and case studies. To be completed by Dec'15.
- Evaluation. To be completed by Mar'16.
- Workflow comparison and optimization. To be completed by Jun'16.
- Writeup, submission, and defence. To be completed by Dec'16.

References

1. The Epsilon Book. Dimitris Kolovos, Louis Rose, Antonio García-Domínguez, Richard Paige (2014)
2. Alvarez, C., Casallas, R.: MTC Flow: A Tool to Design, Develop and Deploy Model Transformation Chains. In: ACME. pp. 7:1–7:9. ACME '13, ACM (2013)
3. Diskin, Z.: Model Synchronization: Mappings, Tiles, and Categories. In: GTTSE. pp. 92–165 (2009)
4. Diskin, Z., Kokaly, S., Maibaum, T.: Mapping-Aware Megamodeling: Design Patterns and Laws. In: SLE. pp. 322–343 (2013)
5. Diskin, Z., Xiong, Y., Czarnecki, K., Ehrig, H., Hermann, F., Orejas, F.: From State- to Delta-Based Bidirectional Model Transformations: The Symmetric Case. In: MoDELS. pp. 304–318 (2011)
6. Dumas, M., Hofstede, A.H.M.t.: UML Activity Diagrams As a Workflow Specification Language. In: UML. pp. 76–90. Springer-Verlag, London, UK, UK (2001)
7. Gholizadeh, H., Diskin, Z., Maibaum, T.: Providing a well-formed structure for model transformation. In: AMT@MoDELS (In submission)
8. Lucio, L., Mustafiz, S., Denil, J., Vangheluwe, H., Jukss, M.: FTG+PM: An Integrated Framework for Investigating Model Transformation Chains. In: SDL Forum. pp. 182–202 (2013)
9. Vanhooff, B., Baelen, S.V., Hovsepyan, A., Joosen, W., Berbers, Y.: Towards a Transformation Chain Modeling Language. In: SAMOS. pp. 39–48 (2006)
10. Whittle, J., Hutchinson, J., Rouncefield, M.: The State of Practice in Model-Driven Engineering. IEEE Software 31(3), 79–85 (2014)