MONDO: Scalable Modelling and Model Management on the Cloud

Dimitrios S. Kolovos[1], Antonio Garcia-Dominguez[1], Richard F. Paige[1],
Esther Guerra[2], Jesús Sánchez Cuadrado[2], Juan De Lara[2],
István Ráth[3], Dániel Varró[3], Gerson Sunyé[4], Massimo Tisi[4]
{dimitris.kolovos, antonio.garcia-dominguez, richard.paige}@york.ac.uk,
{Esther.Guerra, Jesus.Sanchez.Cuadrado, Juan.deLara}@uam.es,
{rath, varro}@mit.bme.hu, {gerson.sunye, massimo.tisi}@inria.fr

[1]University of York, United Kingdom,
[2]Universidad Autónoma de Madrid, Spain,
[3]Budapest University of Technology and Economics, Hungary,
[4]AtlanMod (Inria, Mines Nantes, LINA), France

Abstract. Achieving scalability in modelling and MDE involves being able to construct large models and domain-specific languages in a systematic manner, enabling teams of modellers to construct and refine large models in collaboration, advancing the state of the art in model querying and transformations tools so that they can cope with large models (of the scale of millions of model elements), and providing an infrastructure for efficient storage, indexing and retrieval of large models. This paper outlines how MONDO, a collaborative EC-funded project, has contributed to tackling some of these scalability-related challenges.

1 Project Identity

- **Project acronym:** MONDO
- **Project title:** Scalable Modelling and Model Management on the Cloud
- **Project partners:** The Open Group (Project Coordinator), University of York (Technical Coordinator), Autonomous University of Madrid, University of Nantes, Budapest University of Technology and Economics, IKERLAN, SOFTEAM, Soft-Maint, UNINOVA
- **Website:** `www.mondo-project.org`
- **Project start date/duration:** Nov 1, 2013 (30 months)

2 Introduction

As MDE is increasingly applied to larger and more complex systems, the current generation of modelling and model management technologies are being stressed to their limits in terms of their capacity to accommodate collaborative development, efficient management and persistence of models larger than a few hundreds of megabytes in size. As discussed in [1], achieving scalability in MDE involves:

- being able to construct large models and Domain Specific Languages (DSLs) in a systematic manner;
- enabling large teams of modellers to construct and refine large models in a collaborative manner;
- advancing the state of the art in model querying and transformations tools so that they can cope with large models (with millions of model elements);
- providing an infrastructure for efficient storage, indexing and retrieval of such models.

This paper discusses how the MONDO project has contributed to tackling these challenges. Section 3 provides an overview of the open-source MONDO platform and then Sections 4–8 concentrate on each of the platform's major components. Section 9 outlines the ongoing evaluation process and concludes the paper.

3 Platform overview

The MONDO platform currently consists of the following components:

1. A framework (DSL-tao & EMF-Splitter) for automated development of scalable Eclipse-based editors for DSLs;
2. Cloud-based (CloudATL) and Reactive (ReactiveATL) versions of the widely used ATL model-to-model transformation language;
3. A new version of the VIATRA model transformation engine, built on a reactive virtual machine architecture supported by the EMF-IncQuery incremental model query framework;
4. A framework for online and offline collaborative modelling that supports query-based access control based on IncQuery queries and VIATRA transformations;
5. A framework for indexing of heterogeneous models stored in file-based version control repositories such as Git or SVN.

These components are then integrated into five classes of artifacts:

1. Cloud server nodes, which implement web service APIs that expose the cloud-based tools in MONDO, and may host "golden" and "front" SVN/Git repositories for collaborative modelling (further discussed in Section 8).
2. Cloud worker nodes for CloudATL-based model transformations.
3. Eclipse workbenches, which include the Eclipse-based tools from MONDO. Some of these tools invoke the cloud API.
4. Standalone Java applications, which use the tools from MONDO as additional libraries that may invoke the cloud API.
5. Other clients, which invoke the cloud API or use the web UI of the collaboration framework.

Cloud frontends and Eclipse workbenches use OSGi to integrate the various components in MONDO in a controlled manner, ensuring they do not unintentionally interfere with each other and enabling their independent evolution beyond the lifetime of this project. The cloud API is largely implemented in Apache Thrift (`thrift.apache.org`), an open source library for efficient cross-language Remote Procedure Calls and serialization. Figure 1 illustrates how these components are interconnected and organised internally. In the following sections, we focus on the various ways in which the tools enable these integrations into the platform.

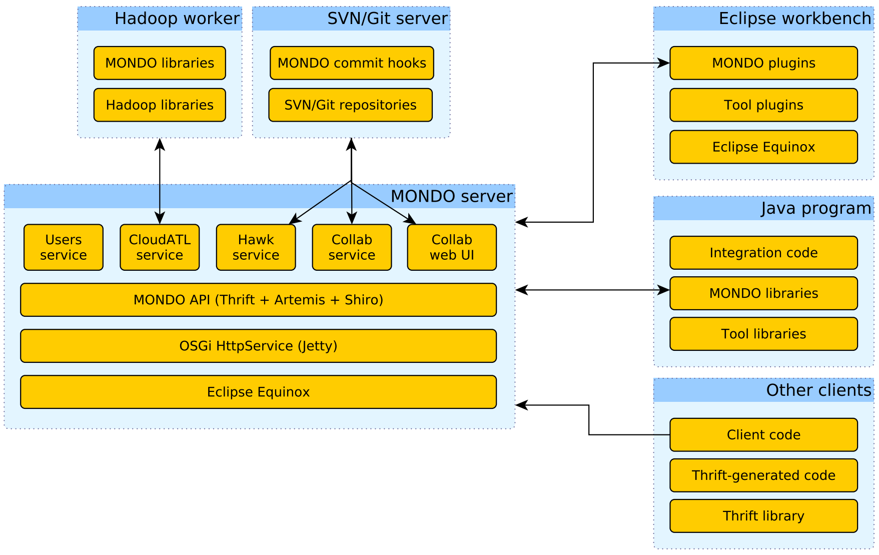


Fig. 1. Architecture of the MONDO cloud platform

4 Heterogeneous Model Indexing Framework (Hawk)

Hawk [2] is a model indexing framework that can monitor large collections of models stored in regular file-based version control systems (e.g. Git or SVN) and maintain a high-performance graph database with a snapshot of the latest version of the models. Hawk can speed up advanced queries using its support for indexed and derived attributes, and the API is designed for managing its configuration and querying the indexed models.

Hawk has been integrated with the rest of the platform in two ways: by integrating new components into Hawk, and by developing compatibility layers so other tools can use Hawk with minimal changes. The main components of Hawk are:

– **Client components**: Hawk provides local and remote (Thrift-based) client components to manipulate Hawk indexes with the same UI.
– **Model parsers**: Hawk provides components that can parse Ecore XMI, Modelio XMI/EXML, IFC2x3 STEP/XML, IFC4 STEP/XML and BPMN models. The Ecore XMI component allows Hawk to index models developed using the scalable modelling tools discussed in Section 7, produced by the transformation engine outlined in Section 5, or developed collaboratively with the framework presented in Section 8.
– **Version control systems**: Hawk provides components for monitoring local folders, Git/SVN version control systems, workspaces and HTTP locations. The Git/SVN version control systems can be one of the "front" repositories maintained by the collaboration framework discussed in Section 8.
– **Backends**: Hawk can use Neo4j or OrientDB for storing its graphs. Neo4j is the current market leader for graph databases, and OrientDB is another high-performance open source graph database with a more permissive license that simplifies redistribution in certain scenarios.

Hawk also provides abstractions that allow the tools developed in the other work packages to treat a local or a remote Hawk index as a standard EMF-compliant model, without requiring any changes. CloudATL can use Hawk indexes as the source of a transformation, IncQuery can query Hawk indexes and update the query results as their contents change, and DSL-tao can use a Hawk index for faster discovery and editing of cross-file references.

5 CloudATL and ReactiveATL

CloudATL is an extended version of the ATL transformation engine that can distribute large-scale transformations over a Hadoop cluster of worker nodes. The engine operates in a transparent manner to the user, who does not need to have any knowledge about distributed programming. The engine shares input data among the slaves, except for the the input model that is equally divided among them. The engine also relies on the MapReduce communication protocols to aggregate the intermediate results and provide the final output model.

CloudATL can be submitted to a standard Hadoop cluster through the Hadoop APIs as well as through standalone console-based tools. The latter use a service-oriented API that turns a MONDO server into a frontend for a dedicated Hadoop cluster. The API provides services to launch, monitor and stop CloudATL transformations on the Hadoop cluster, and CloudATL jobs can read remote Hawk indexes through the EMF resource abstractions mentioned above. For validation and experimentation purposes, a virtualized Hadoop cluster (based on Docker[1]) has been made publicly available, and automated scripts for launching, stopping, and resizing the cluster are provided[2].

[1] `https://www.docker.com/`
[2] `https://github.com/atlanmod/hadoop-cluster-docker/`

Initial versions of CloudATL ran into scalability limitations of the standard XMI model serialization format, especially the lack of support for concurrent reads and writes. To solve this issue, a decentralized persistence backend (NeoEMF/HBase [3]) was developed and integrated with CloudATL. NeoEMF/HBase is transparent to standard EMF tools, since it exposes its persistence functionality through the usual EMF interfaces. A persistence manager communicates with the underlying data nodes through a persistence driver, and supports a pluggable caching strategy. The design of the persistence manager decouples the high level EMF-based code from the low-level data structures and code accessing the database engine. Maintaining these uniform APIs between the different levels allows including additional functionality on top of the persistence driver by using the decorator pattern, such as different caching strategies.

NeoEMF/HBase offers lightweight on-demand loading and efficient garbage collection. Model changes are automatically reflected in the underlying storage, making changes visible to all the clients. In the case of CloudATL, it guarantees that only the model elements needed to perform the partial transformation are loaded by each CloudATL worker. Moreover, NeoEMF/HBase has explicit support for ACID properties. This allows the use of NeoEMF/HBase as a scalable persistence backend for distributed and concurrent model transformations.

ReactiveATL[3] is a reactive engine for the ATL transformation language. It is designed to perform on-demand transformation in model-driven applications, by activating only the strictly needed computation in response to updates or requests of model elements. Computation is updated when necessary, in an autonomous and optimized way by using incrementality and lazy evaluation.

6 IncQuery and VIATRA

VIATRA [4] is a *reactive, event-driven and incremental model transformation platform.* It integrates IncQuery [5] as a graph query engine over EMF models with support for RETE-based incremental and local-search based query evaluation [6]. Within the MONDO project, initial support was provided for *distributed incremental graph queries* [7] deployed over a cloud infrastructure to scale complex graph queries for models with over 100 million elements.

Incremental queries enable to uniformly handle elementary model changes (e.g. change of an attribute) and aggregate model changes (e.g. disappearance of a match in the query result set) as atomic events. To identify specific sequences of such events, BME introduced *complex event processing to model transformations* in VIATRA [8], which is especially suitable for *streaming transformations* [9].

Reactive model transformations [4] following the paradigm of reactive programming [10] may trigger reactions to atomic and complex events where rules can be immediately fired upon certain changes are detected (live mode). This also enables to incrementally chain multiple views and transformations [11].

MONDO partners collaborated to allow IncQuery to query Hawk indexes directly, and through the EMF resource abstractions of Hawk. An optimised

[3] `https://github.com/atlanmod/org.eclipse.atl.reactive`

version of IncQuery was developed that can load only the part of the model that is required. It accesses all instances of a type using direct edge traversal instead of having to iterate through the entire model. The optimised version also passes the Train Benchmark [12] test suite of the IncQuery implementation.

7 Scalable DSL Modelling Tools

DSL-tao [13] is a meta-modelling tool which enables the description of DSLs and their graphical modelling environments by means of patterns. It provides an extensible catalogue of recurrent patterns that can be instantiated to contribute to the DSL's abstract syntax (e.g., typical domain patterns like state machines or workflows, and meta-modelling design patterns like different realizations for tree-like data structures), the DSL's concrete syntax (e.g., graphical or tabular) and infrastructure (i.e., functionalities of the modelling environment).

Of special relevance for scalability are infrastructure patterns related to DSL modularity. While modularity mechanisms have been developed and are well supported for programming languages, this is not so for DSLs. In order to obtain more scalable DSL modelling tools, DSL-tao includes a modularity pattern which permits customising model fragmentation strategies for DSLs. Hence, similar to how programming languages organize software projects, one can identify the meta-model elements playing the roles of project, package and unit. The generated modelling environment will then fragment and physically organize models into packages (folders) and units (files) of smaller size, so that they can be more efficiently managed. As an example, Figure 2 shows a modelling environment that instantiates the modularity pattern for wind-turbine models, enabling the definition of each subsystem in a different package, and its constituent components and state machines as files inside the package.

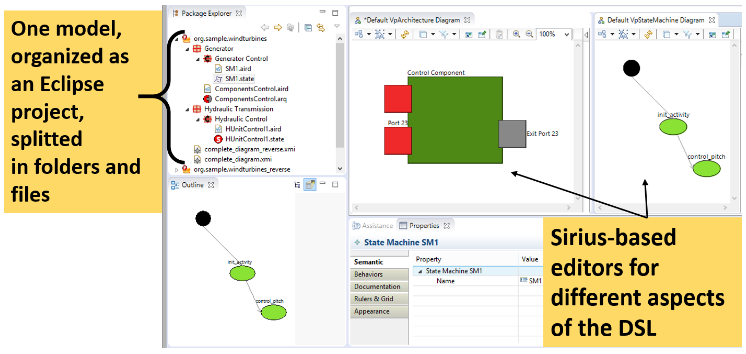
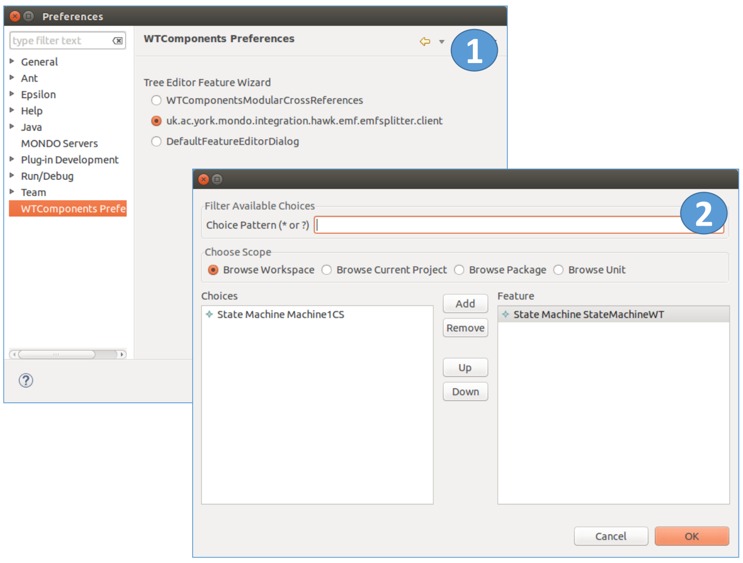


Fig. 2. Screenshot of generated DSL modelling tool supporting fragmented models.

The modularity pattern is realised by EMFSplitter [14], a framework that generates domain-specific Eclipse-based model editors according to the defined model fragmentation strategy. As a model is fragmented across the workspace, EMFSplitter needs an efficient way to find candidate values for non-containment references, according to a given scope. For this purpose, EMFSplitter relies on Hawk. The final version of Hawk can monitor the generated local workspace and provide efficient local and global queries over the fragmented models.

The user only needs to open the Eclipse preferences page produced by EMF-Splitter and select Hawk to find candidate values for non-containment references (window labelled 1 in Figure 3). The next time that the EMFSplitter dialog for editing a non-containment reference is opened, the Hawk+EMFSplitter integration will create a Hawk index (backed by OrientDB) that monitors over all the models in the workspace, and will use it to efficiently find the candidate values. The same index will be reused on later queries, and will be kept up to date automatically by Hawk.



WTComponents Preferences

Tree Editor Feature Wizard
( ) WTComponentsModularCrossReferences
[✓ uk.ac.york.mondo.integration.hawk.emf.emfsplitter.client]
( ) DefaultFeatureEditorDialog

Filter Available Choices
Choice Pattern (* or ?)

Choose Scope
[✓ Browse Workspace] ( ) Browse Current Project ( ) Browse Package ( ) Browse Unit

Choices
State Machine Machine1CS

Feature
State Machine StateMachineWT

Fig. 3. (1) EMFSplitter preferences page for selecting a component to find candidate values for non-containment references. (2) EMFSplitter editor dialog for non-containment references.

The dialog for editing non-containment references is the same whether Hawk is used or not (window labelled 2 in Figure 3). Depending on the scope (workspace, project, package, or unit), Hawk will be used in a different way. In a first stage, Hawk will be queried to either find all the instances of the *EType* of the reference across the workspace, or find only the instances that are contained within a specific project, package, or unit. In the second (and optional) stage, the retrieved instances can be filtered by project nature.

8 Collaboration Framework

The MONDO Collaboration Framework provides support for both online and offline collaborative modeling scenarios by using graph queries for defining *fine-grained model-level access control policies* and *locks*, and bidirectional model transformations to derive *filtered views for each collaborator* and to propagate changes introduced in these views back to a server. The transformation uniformly enforces *high-level fine-grained access control policies* during the derivation of views and the *back-propagation* of changes.

In an **offline collaboration** scenario (see Fig. 4), models are stored in a *gold repository* of an off-the-shelf version control system (VCS) that is not directly accessible by end users. Instead, the VCS server instead hosts a separate *front repository* dedicated to each collaborator (group) that contains a copy of the gold repository filtered according to read access privileges of that user (using the *get* operation of the bidirectional transformation). This front repository of the user, which contains complete version history, can also be used to commit his changes. Then the MONDO Collaboration Framework tries to propagate these changes to the gold repository (by a *putback* operation), but these changes may be denied based on write permissions.

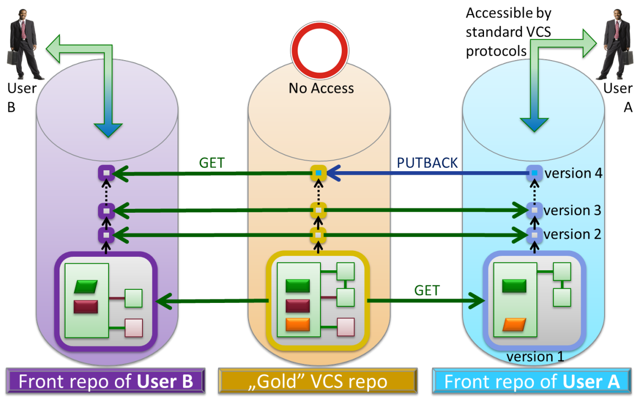


Fig. 4. Access control by bidirectional transformations (offline collaboration).

In an **online collaboration** scenario, users can connect via their web browsers to open, view and edit models stored in the backends in live sessions. Multiple users can collaborate on the same model simultaneously along the same access control policy (used for offline collaboration as well). The editor is provided as an Eclipse RAP-based web application, which can optionally be co-deployed with other server-side components. The main conceptual difference with the offline case is that the live detection of violating access control policies by incremental queries. At the end of an online collaborative session, the result is persisted back in the VCS as in the offline case.

Conflicting model changes need to be resolved by collaborators prior to a successful commit, which is carried out by *search-based automated model merge* [15] where rule-based design space exploration [16] is used to search the space of solution candidates that represent conflict-free merged models. Our method allows to easily incorporate domain-specific knowledge into the merge process to provide better solutions. The merge process automatically calculates multiple merge candidates to be presented to domain experts for final selection.

9 Evaluation and Conclusions

In this paper we provided an outline of important scalability challenges in the context of MDE, and MONDO's technical contributions for addressing them. MONDO has already contributed novel techniques and several prototype implementations in all four identified key-areas. Currently, the research contributions of MONDO are under assessment in the context of four industrial case studies.

The first case study (provided by UNINOVA[4]) comes from the construction domain and involves collaborative development and automated management of large computer-aided design (CAD) models of buildings. The second case study (provided by Soft-Maint[5]) involves exploration and automated regeneration of code from large models, which have been reverse-engineered from existing legacy codebases. The third case study (provided by IKERLAN[6]) involves multi-device collaborative development of models for offshore wind power generators, and the fourth case study involves managing large collections of UML models captured in a proprietary format supported by Softeam's[7] Modelio[8] tool. To assess the usefulness and impact of the technologies produced by MONDO, industrial partners specified a set of concrete requirements and measures with reference to existing state-of-the-art technologies during the first six months of the project. We intend to present the evaluation results in a follow-up publication as soon as the evaluation reports have been produced by the respective MONDO partners.

References

1. Dimitrios S. Kolovos, Louis M. Rose, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan de Lara, István Ráth, Dániel Varró, Gerson Sunyé, and Massimo Tisi. MONDO: scalable modelling and model management on the cloud. In *Projects Showcase*, pages 44–53, 2015.
2. Konstantinos Barmpis and Dimitrios S. Kolovos. Towards scalable querying of large-scale models. In Jordi Cabot and Julia Rubin, editors, *Modelling Foundations and Applications*, volume 8569 of *LNCS*, pages 35–50. Springer, 2014.

[4] `http://www.uninova.pt/`
[5] `http://www.sodifrance.fr/`
[6] `http://www.ikerlan.es/`
[7] `http://www.softeam.fr/`
[8] `https://www.modeliosoft.com/`

3. Abel Gómez, Amine Benelallam, and Massimo Tisi. Decentralized Model Persistence for Distributed Computing. In *3rd BigMDE Workshop*, L'Aquila, 2015.
4. Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. Road to a reactive and incremental model transformation platform: Three generations of the VIATRA framework. *Software and Systems Modeling*, 2016. In press.
5. Zoltán Ujhelyi, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, Benedek Izsó, István Ráth, Zoltán Szatmári, and Dániel Varró. EMF-IncQuery: An Integrated Development Environment for Live Model Queries. *Science of Computer Programming*, 98, 2015.
6. Márton Búr, Zoltán Ujhelyi, Ákos Horváth, and Dániel Varró. Local search-based pattern matching features in emf-incquery. In *Graph Transformation - 8th Int. Conf., ICGT 2015, L'Aquila, Italy*, volume 9151 of *LNCS*, pages 275–282. Springer.
7. Gábor Szárnyas, Benedek Izsó, István Ráth, Dénes Harmath, Gábor Bergmann, and Dániel Varró. Incquery-d: A distributed incremental model query framework in the cloud. In *ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, MODELS 2014*, Valencia, Spain, 2014. Springer.
8. István Dávid, István Ráth, and Dániel Varró. Streaming model transformations by complex event processing. In Juergen Dingel and Wolfram Schulte, editors, *ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, MODELS 2014*, Valencia, Spain, 2014. Springer.
9. Jesús Sánchez Cuadrado and Juan Lara. Streaming model transformations: Scenarios, challenges and initial solutions. In Keith Duddy and Gerti Kappel, editors, *Theory and Practice of Model Transformations*, volume 7909 of *LNCS*, pages 1–16. Springer Berlin Heidelberg, 2013.
10. Engineer Bainomugisha, Andoni Lombide Carreton, Tom Van Cutsem, Stijn Mostinckx, and Wolfgang De Meuter. A survey on reactive programming. *ACM Computing Surveys*, 2012.
11. Csaba Debreceni, Ákos Horváth, Ábel Hegedüs, Zoltán Ujhelyi, István Ráth, and Dániel Varró. Query-driven incremental synchronization of view models. In *2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling (VAO 2014)*, pages 31:31–31:38. ACM, 2014.
12. Benedek Izsó, Gábor Szárnyas, István Ráth, and Dániel Varró. MONDO-SAM: A framework to systematically assess MDE scalability. In *BigMDE 2014 2nd Workshop on Scalable Model Driven Engineering*, page 40. ACM, 2014.
13. Ana Pescador, Antonio Garmendia, Esther Guerra, Jesús Sánchez Cuadrado, and Juan de Lara. Pattern-based development of domain-specific modelling languages. In *MoDELS*, pages 166–175. IEEE, 2015.
14. Antonio Garmendia, Esther Guerra, Dimitrios S. Kolovos, and Juan de Lara. EMF splitter: A structured approach to EMF modularity. In *XM@MoDELS*, volume 1239 of *CEUR Workshop Proceedings*, pages 22–31. CEUR-WS.org, 2014.
15. Csaba Debreceni, István Ráth, Dániel Varró, Xabier De Carlos, Xabier Mendialdua, and Salvador Trujillo. Automated model merge by design space exploration. In *Fundamental Approaches to Software Engineering - 19th International Conference (FASE 2016)*, volume 9633 of *LNCS*, pages 104–121. Springer, 2016.
16. Ábel Hegedüs, Ákos Horváth, and Dániel Varró. A model-driven framework for guided design space exploration. *Automated Software Engineering*, 22:399–436, 2015.