Towards the Next Generation of Reactive Model Transformations on Low-Code Platforms: Three Research Lines

Benedek Horváth
IncQuery Labs Ltd.
Budapest, Hungary
Johannes Kepler University Linz
Linz, Austria
benedek.horvath@incquerylabs.com

Ákos Horváth
IncQuery Labs Ltd.
Budapest, Hungary
akos.horvath@incquerylabs.com

Manuel Wimmer
Johannes Kepler University Linz
Linz, Austria
manuel.wimmer@jku.at

ABSTRACT

Low-Code Development Platforms have emerged as the next-generation, cloud-enabled collaborative platforms. These platforms adopt the principles of Model-Driven Engineering, where models are used as first-class citizens to build complex systems, and model transformations are employed to keep a consistent view between the different aspects of them. Due to the online nature of low-code platforms, users expect them to be responsive, to complete complex operations in a short time. To support such complex collaboration scenarios, the next-generation of low-code platforms must (*i*) offer a multi-tenant environment to manage the collaborative work of engineers, (*ii*) provide a model processing paradigm scaling up to hundreds of millions of elements, and (*iii*) provide engineers a set of selection criteria to choose the right model transformation engine in multi-tenant execution environments.In this paper, we outline three research lines to improve the performance of reactive model transformations on low-code platforms, by motivating our research with a case study from a systems engineering domain.

CCS CONCEPTS

• **Software and its engineering** → **Model-driven software engineering**.

KEYWORDS

Low-Code Development Platforms, Model-Driven Engineering

ACM Reference Format:
Benedek Horváth, Ákos Horváth, and Manuel Wimmer. 2020. Towards the Next Generation of Reactive Model Transformations on Low-Code Platforms: Three Research Lines. In *ACM/IEEE 23rd International Conference on Model Driven Engineering Languages and Systems (MODELS '20 Companion), October 18–23, 2020, Virtual Event, Canada.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3417990.3420199

1 INTRODUCTION

In recent years a new generation of cloud-enabled collaborative software development tools, Low-Code Development Platforms (LCDPs), have emerged allowing citizen developers with no or little prior programming experience to design and implement full-fledged applications in a couple of hours, instead of weeks, and thus allow shorter time-to-market and time-to-value cycles.

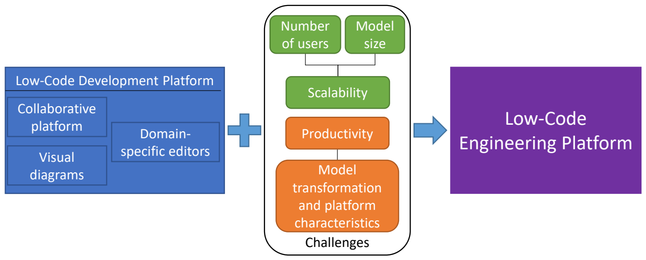

Figure 1: From LCDP to LCEP

LCDPs adopt the recent theoretical and practical advancements of Model-Driven Engineering (MDE). On these platforms citizen developers [45] build models of the software, by refining its operation on diagrams with different levels of abstraction, using domain-specific editors. Moreover, they use model transformations to derive platform-specific source code, tests, or configuration artifacts from the models to realize their systems as fully operational applications.

At the same time, Model-Based Systems Engineering (MBSE) has been successfully applying MDE techniques in designing and implementing complex systems. In recent years, several industrial vendors started to develop cloud-enabled products for MBSE [23]. Their functionality covers the whole life-cycle of modeling, combines multi-domain models from different repositories, and offers a collaborative platform for engineers from various disciplines.

In order to benefit from the cloud-enabled collaborative properties of LCDPs in systems engineering, these platforms need to be adapted for MBSE needs and challenges, and their performance should be improved in two respects: *number of users*, who concurrently and collaboratively work on different parts of the model, and scalability w.r.t. the *size of models* in the range of hundreds of millions of elements. The *characteristics of model transformations on collaborative platforms* should be studied to derive several selection criteria that can be used to assess state-of-the-art model transformation engines, to find the most suitable one for the given transformation context and execution environment. As illustrated by Figure 1, addressing these challenges will contribute to the advancement of LCDPs to Low-Code Engineering Platforms (LCEPs) [45] with advanced model processing and transformation capabilities.

Table 1: Model transformation and query approaches

	Model Transformation	Model Query
Lazy	[47]	[30, 44]
Incremental	[13, 49]	[6]
Reactive	[5, 36]	-
Parallel	[10, 11, 31, 46]	[7, 30]
Distributed	[3, 11]	[42]

In this paper, we outline three research lines by mapping the aforementioned challenges into research directions motivated by previous experiences from an industrial project. The paper is structured as follows: Section 2 introduces the related work, Section 3 shows a motivating example for the research, Section 4 outlines the research lines, including the main directions of advancement, Section 5 concludes the paper.

2 RELATED WORK

In this section, we summarize how different model transformation and query approaches address the scalability challenge for very large and complex models (Section 2.1), what methods researchers used to assess the performance of model transformation engines (Section 2.3). Besides, we introduce some related work on multi-tenant patterns to leverage scalable cloud architectures while maintaining separation between the tenants (Section 2.2). Finally, we conclude the section by positioning ourselves to research directions that have not been explored yet (Section 2.4).

2.1 Model transformation and query approaches

Both Kolovos et al. in [27] and Bucchiarone et al. in [8] have listed scalability of model transformations as one of the key challenges in MDE. Efficiently running model transformations, in terms of memory use and execution time, on models with hundreds of millions of elements is a challenging task. The challenge is multiplied in the case of distributed models and distributed execution environments. Although model transformations is a widely researched area, state-of-the-art tools lack in performance even for mid-sized models [8].

In order to address scalability in model transformations, several execution models have been proposed and implemented, as Table 1 shows it. In lazy mode, transformation rules are executed only if they are used by other rules [47]. In *target incremental* transformations target models are updated according to changes in the source model [13, 49]. Incremental transformations are usually executed faster, than batch transformations, which recompute the whole target model. Reactive model transformations are executed as reactions for events emitted by event sources, and they can combine incrementality with lazy evaluation [5, 36].

Model transformations are computation heavy operations. In order to make them more scalable, different parallelization techniques have been proposed.

Tisi et al. implemented a *task-parallel* engine for ATL in which each thread executes a different transformation rule and works on the whole source and target models [46]. Due to several constraints of the ATL language, the application of transformation rules is highly independent of each other, which is beneficial for the parallel execution of transformations. However, due to technical restrictions of the Eclipse Modeling Framework they used, target element creation, adding values to multi-valued collections, creating and reading traceability links need to be synchronized.

Burgueno et al. introduced the LinTra framework for model-to-model (M2M) transformations [10, 11]. The framework adopts the principles of the Linda coordination language that follows the Blackboard approach. In this approach, processes communicate via tuples in shared memory. LinTra cuts the source model into partitions and transforms them in a *data-parallel* way in a master-slave architecture. The master process coordinates the work of the slaves that execute the transformation rules on the model partitions [9].

Mezei et al. proposed a parallelization approach based on the offline dependency check of transformation rules [31]. Two transformation rules are in *metaconflict* if their match may conflict according to the metamodel items used in the rules. Those rules that are not in *metaconflict* can be grouped into independence blocks because they can be executed in parallel. Consecutive blocks may not be conflict-free, thus they implemented several heuristics to avoid and resolve conflicts in VMTS [29].

In order to overcome the memory limitations of a single machine, and leverage the benefits of a cluster, consisting of many processing nodes, several distributed model transformations have been proposed.

Benelallam et al. implemented distributed M2M transformations in ATL on MapReduce [3]. They equally distributed the source model among the worker nodes in the map phase. Each node applies the full transformation code for the subset of the source model they are assigned to. They call this phase *local match-apply*. In the reduce phase, worker nodes are responsible for building the partial target models into a full target model, by resolving the missing references in them (*global resolve*).

Burgueno et al. adapted the LinTra framework for distributed operation [11]. They experimented with different task and data allocation strategies in the cluster of two nodes. E.g., source and target models, and transformation processing nodes are on the same or different machines.

Model query approaches. A model transformation rule consists of a precondition, that can be a graph pattern or model query, whose match on the source model activates the transformation action which translates the source model elements into target model elements. In order to achieve scalable model transformations for very large models, scalable model queries should be employed. To this end, several approaches have been developed in the literature.

Tisi et al. proposed the lazy evaluation of OCL collection query operations, by providing their lazy execution algorithms and implementation [44]. Madani et al. proposed a lazy implementation of chained OCL queries in Epsilon Object Language based on the

Java Stream API [30]. The benefits of lazy evaluation are twofold. On one hand, they postpone the computation until needed. On the other hand, they enable the processing of infinite collections.

To further improve the evaluation of OCL queries in Epsilon, Madani et al. provided parallel implementations for first-order OCL operations in a data-parallel approach by creating a job for each model element and submitting it to a thread pool executor [30]. Furthermore, they extended these operations with short-circuiting thus further improving the processing time.

Bergmann et al. proposed incremental graph pattern matching on EMF models [6] by implementing the Rete algorithm [21]. In the Rete algorithm, a network of nodes is constructed from the graph pattern. In the network, each node caches the matches of the subpattern they are assigned to. The benefit of the algorithm is the incrementality and the ability to react to changes in the source model. However, intensive caching causes a large memory footprint, and updating the Rete network has computation complexity also. In order to improve it, Bergmann et al. proposed a parallel implementation of incremental pattern matching [7]. They split the Rete network into containers, where each of them is responsible for matching a set of subpatterns. Each container runs on a separate thread and communicates via message queues. The advantage of this approach is, the update propagation of the network is spread between the containers, thus the computation can complete faster, than in the single-threaded implementation.

All the aforementioned model query techniques can leverage only the computational power of a single machine. To overcome this limitation, Szárnyas et al. proposed IncQuery-D, a distributed incremental model query framework in the cloud [42]. The framework implemented a distributed Rete network, where each machine stores a subset of the Rete nodes which communicate with each other to update their local caches. They proposed a distributed termination protocol to know if a model change has propagated through the whole network.

2.2 Multi-tenant architectures

In multi-tenant architectures, multiple customers (tenants) use the same computation resource, application or database instance, while they see it as a highly configurable and isolated environment. The benefits of these architectures are the high utilization of computation resources and improved maintenance in the deployment of applications. Although multi-tenancy is a widely researched topic in Software as a Service (SaaS) applications [12, 20, 33], there is little research on their application in MDE, especially for model transformations. The nearest application areas we found were model checking and formal verification. Hu et al. proposed the Verification as a Service (VaaS) concept in a multi-tenant architecture, that hosts verification software in the cloud which can be composed to verify a software model on demand [24]. They store the verification software, models, and results in databases. They implemented a workflow to compose a verification application for each tenant by retrieving components from the databases, linking and compiling them together, and deploying the executable code on provisioned cloud infrastructure. The deployed environment can be used by the tenant to verify software models.

2.3 Model transformation performance evaluation

Researchers followed different strategies to assess the performance of model transformation engines. One strategy is to implement custom model transformations on custom metamodels [31, 36, 47]. The advantage of this approach is the quick implementation; however, the results are difficult to be compared.

Second strategy is to use a repository like ATL Transformation Zoo[1], that contains a collection of model transformation cases and metamodels [10, 46]. A similar strategy is to adopt cases from workshops and contests, like AGTIVE [39], GraBats [16] and Transformation Tool Contest (TTC, [39]). TTC is an annual contest of model transformation engines on constantly changing case studies [3, 11, 13]. The advantage of this strategy is, the performance of different implementations can be compared with each other. However, there is no benchmark workflow that would automatically evaluate the case studies on the transformation tools. To address this shortcoming, a benchmark can be designed that is able to execute a set of model transformation rules on source models with varying sizes and collect run-time metrics during the process on different engines. Calvar et al. used the VIATRA CPS benchmark to evaluate the performance of ATL incremental transformations [13]. Although the benchmark has some technical descriptions,[2] it has not been published in the literature yet.

Varró et al. proposed a graph transformation benchmark, a special case of model transformations [50]. They defined the paradigm features of graph transformations, e.g., pattern size, maximum degree of nodes (fan-out), number of matchings, length of transformation sequence, to describe the characteristics of a transformation rule. Besides, they investigated how tool features, e.g., parallel rule execution, 'as long as possible' rule application, multiplicity-based optimization, parameter passing between consecutive rule applications, influence the runtime of model transformations. Although they applied the benchmark in a transformation case on several tools, as runtime metrics they only measured the execution times of the pattern matching and transformation phases.

Benelallam et al. presented four model query and transformation benchmarks along three dimensions [4]: the context and objectives of the benchmark, the complexity of the models and metamodels that were used to assess the engines, the complexity and types of queries and transformations which were run on the engines. From the four benchmarks, one of them was only for model queries (the Train Benchmark framework [43]), one was for queries and transformations, and two of them were for transformations only. Besides, to the best of our knowledge, only one of the four benchmarks was published in the literature in detail.

Szárnyas et al. proposed the Train Benchmark framework [43] to assess the scalability of model query engines for validating graph patterns on large graph models. The framework focuses on the query performance evaluation of well-formedness validation constraints in the railway domain. Although the benchmark contained a model manipulation step, to correct the validation error in the model, it did not focus on measuring the model transformation itself, but the performance of the query.

[1]https://www.eclipse.org/atl/atlTransformations
[2]https://github.com/viatra/viatra-cps-benchmark/wiki

2.4 Summary

To achieve responsive low-code platforms, we need scalable reactive model transformations that are able to quickly react to events which occur on the platform, e.g., derived views of the model need to be updated due to a change in the model. On the one hand, these transformations are executed automatically, if a triggering event occurs. On the other hand, if many events concurrently occur, and the transformation actions are long-running tasks, then congestion in their processing can arise quickly, which hinders the performance of the platform. To achieve scalable reactive model transformations on LCDPs, state-of-the-art reactive transformation approaches need to be improved.

As previously shown, many model transformation approaches have been already researched, however, there is little empirical experience about their combinations, e.g., parallel or distributed approach in combination with reactive transformations. Parallel extension of reactive transformations could improve the congestion shortcoming of sequential execution of transformation rules and offer better throughput of the engine. Furthermore, reactive transformations could be combined with different data and task distribution strategies ([3, 11]), to leverage the benefits of multiple computation nodes.

Since LCDPs are cloud-native collaboration platforms, they have to adopt the multi-tenant architecture patterns to separate the tenants from each other, while maintaining high computation resource utilization. As shown before, multi-tenant patterns have been extensively researched for SaaS applications, however, to the best of our knowledge, there is little research done on their specialization for MDE, especially for model transformations.

Multi-tenant platforms have different load characteristics than non-multi-tenant ones, since tenants should be isolated from each other, while the underlying computation resources are shared between them. Besides, reactive transformations are executed as reactions for events and so they are expected to complete quickly, to have a reactive and responsive platform. To the best of our knowledge, there is no benchmark in the literature, that could be used to assess the performance of reactive model transformation engines on multi-tenant platforms. Benchmarks have the advantage that they define a uniform methodology along which engines can be evaluated. The methodology includes the source and target metamodels, the source models with varying size and complexity, the transformation rules that are executed on the models upon the occurrence of the triggering events described by a reactive scenario. Several runtime metrics are collected that characterize the behavior of the transformation engines in the different transformation cases. Moreover, if the benchmark framework orchestrates the execution of the measurements, then the evaluation can be automatically repeated which raises the validity of the measurements.

3 MOTIVATING EXAMPLE

In safety-critical domains the correct behavior of the systems is crucial, otherwise, a malfunctioning system can cause accidents that may harm human lives and result in major financial losses. To minimize this risk, Model-Based Systems Engineering (MBSE) has been successfully applying MDE techniques in designing and implementing complex safety-critical systems. Such systems are realized by the collaborative work of engineering teams, each focusing on different aspects of the system: requirements analysis, system design, validation, and verification. In the systems engineering V-model [22], model transformations are frequently used means to derive formal models from the high-level design models to prove their correctness and adherence to the system requirements and specifications.

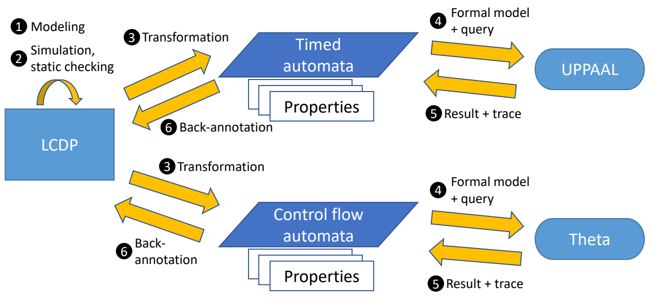

Figure 2: Validation and verification workflow

In this paper, as a motivating example, we use an industrial case study from the aerospace domain. In this domain, engineers use the Systems Modeling Language (SysML, [34]), the standard modeling language in systems engineering. The language has been used to design, analyze, and validate complex systems in safety-critical domains [15, 37]. Both structural, and dynamic, behavioral, timing aspects of the system can be modeled in this language. Recent advancements in modeling tools have enabled the testing and simulation of models. However, due to the safety-critical nature of the aerospace domain, model-based testing is not enough to prove the correctness of the system, because it can only show the presence of errors in the models, not their absence. Thus, in these cases, formal methods (model checking) have to be applied. Formal methods use precise formalisms to prove the correctness of behavioral models by evaluating formal expressions (properties) on every execution trace of the model. If the model violates the property then a counterexample trace is returned from the model checker.

Figure 2 illustrates how LCDPs can be used as cloud-based, collaborative modeling tools in a validation and verification workflow of systems engineering models. Systems engineers design the behavioral models (state machines, activity diagrams, sequence diagrams) on LCDP ❶, and run static checks or simulate them ❷. If the models are structurally correct, then engineers prepare them for formal verification. They define properties to be checked on the model and transform the properties and the models for the input formalisms of different model checkers ❸, e.g., UPPAAL [2] and Theta [48]. The formal models (e.g., timed automata, control-flow automata) and queries (e.g., CTL or LTL expressions [14]) are forwarded to model checkers ❹. The model checkers evaluate the query on the model and return the verification result ❺. If the model violates the checked property, then a counterexample trace is returned as well. These artifacts are back-annotated to the original formalism and returned to the user ❻, who can inspect them and correct the engineering model accordingly.

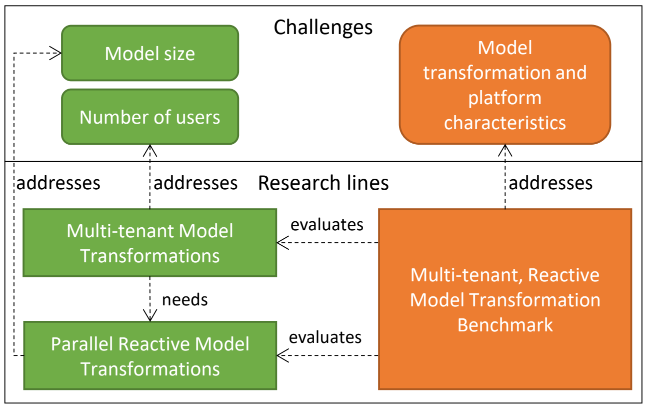
Challenges
Model size
Number of users
Model transformation and platform characteristics
addresses
addresses
Research lines
addresses
Multi-tenant Model Transformations
evaluates
Multi-tenant, Reactive Model Transformation Benchmark
needs
Parallel Reactive Model Transformations
evaluates

Figure 3: Mapping of challenges to research lines

Due to the inherent complexity of safety-critical domains, the models of such systems are very complex. The challenge is multiplied in the case of product lines where products are constructed from shared component models. However, their specializations and unique combinations can result in very large and complex models in the range of hundreds of millions of elements. To guarantee the correctness of the system and provide short validation and verification feedback loop, only those parts of the models should be revalidated ❷ that are influenced by the model update. Moreover, in step ❸, only the changed parts of the source model should be transformed to the formal target model. Although several reactive model transformation engines have been developed in the past, they are not able to scale for models in this range [8]. Thus further research is needed to address this challenge and provide a scalable continuous validation and verification workflow for engineers.

Furthermore, these systems are designed by the collaborative work of multiple systems engineering teams, who work in a complex, cloud-based low-code environment. Although these platforms need to be customized for the users' needs, they are usually deployed on the same physical machine in the cloud. Therefore, the computational resources are shared among the users who may perform resource-intensive operations e.g., model transformation or formal verification. In order to achieve fair resource allocation, while maintaining users' isolation, multi-tenant architectures for model transformations have to be adopted.

Finally, to achieve high-productivity engineering platforms, that can run transformations efficiently, in terms of model size, change frequency, and the number of users using the same transformation engine, Key Performance Indicators (KPI) that characterize reactive model transformations on multi-tenant architectures should be identified. From the KPI metrics, several selection criteria can be constituted, that can be used to assess the different reactive model transformation engines and choose the most suitable one in the given transformation context and execution environment.

4 RESEARCH LINES

In order to address the challenges that were introduced in Section 1 and highlighted in the motivating example, we map them to research lines, illustrated by Figure 3.

In order to raise the number of users who collaboratively run model transformations on the platform, while maintaining high resource utilization, and keeping users isolated from each other, we will explore *multi-tenant model transformations*. To further advance the scalability of model transformations for models in the size of hundreds of millions of elements, we aim to improve the performance of *reactive model transformations* by extending them for *parallel execution*. Finally, we will define a *multi-tenant, reactive model transformation benchmark* to evaluate the performance of state-of-the-art reactive model transformation engines by analyzing the characteristics of reactive model transformations on multi-tenant platforms and deriving selection criteria from them to be used for the assessment of the engines. The goal is to have a multi-tenant collaborative platform where engineers can efficiently run parallel reactive model transformations, using the most suitable engine for their transformation cases.

4.1 Multi-tenant Model Transformations

Collaborative modeling has become an emerging topic. In systems engineering, many engineers from different disciplines work together to design and build the system. During the collaborative work, similarly to our running example, each team focuses on different aspects of the system, thus maintaining a coherent, consistent picture of the models is essential.

Traditionally in collaborative modeling, rich client applications are used to design, analyze, validate, and verify the models. Model repositories are used to keep track of changes in the models and to make them available for the whole team. Due to the advancements in cloud infrastructures and supporting technologies, there has been a paradigm shift in recent years; the functionality of rich client modeling applications is gradually moving to the cloud. The advantages of the new paradigm are:

(1) resource-intensive operations are run in an environment, where computational resources are elastically scalable,
(2) heavy-weight applications can be deployed in the cloud,
(3) platform is accessible for everyone with a lightweight client.

Although the cloud offers elastically scalable computation resources, these resources are on one hand finite, on the other hand, they are shared among multiple customers, tenants, that led to the development of multi-tenant patterns. Fehling et al. proposed three multi-tenancy patterns that differ in the level of isolation they offer between the tenants: *shared component*, *tenant-isolated component*, *dedicated component* [20].

The *shared component* pattern offers the least degree of isolation between the tenants. Every tenant uses the same instance of the application or machine, and only minor configuration differences might be between them. The application is unaware that multiple tenants use it, thus if one of them puts a heavy workload on the application, then it negatively influences the experience of others.

In the *tenant-isolated component* pattern, tenants share the same application instance, however, an identifier is used to differentiate and isolate them from each other. They receive a highly customized application for their needs, with performance monitoring to avoid unequal resource consumption, and with isolated data access to ensure each tenant uses the application as if they were its sole user.

In the *dedicated component* pattern every tenant receives a separate application stack with separate data stores, thus ensuring complete isolation and no resource sharing between them.

Mietzner et al. proposed the horizontal and vertical combinations of multi-tenancy patterns in cloud infrastructures [32]. Horizontal combinations in the application layer mean, how tenant-aware and non-tenant-aware services can communicate with each other, whereas vertical combinations mean how tenant-aware applications can be deployed on non-tenant-aware providers.

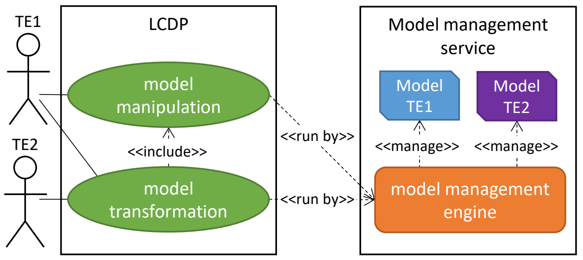

Figure 4: Tenant-isolated model management service

In order to advance LCDPs and make them scalable for the number of users and the number of tenants, we are planning to adopt the *tenant-isolated component* and *dedicated-component* patterns for LCDPs to support various model manipulation use cases with many tenants on the same platform.

First of all, we will separate the model management engine from the low-code platform, due to the high resource need of model management operations. The model management engine is responsible for loading the model from model repositories into memory, performing transformations on them, and persisting the results in model stores. We introduce a model management service, in which different model management engines can be used. The service is horizontally combined with the low-code platform using either the (1) *dedicated component* or the (2) *tenant-isolated component* multi-tenancy pattern, as depicted in Figure 4.

In scenario (1) the service deploys a dedicated management engine for each tenant which offers a high degree of isolation. The engines are started exclusively for the tenants without any resource sharing between them. It is beneficial for the tenants because they are the sole users of the engine. However, the underlying resources may not be utilized the best and economies of scale may not be achieved [20], if the tenant does not use the full capacity.

In scenario (2) the service deploys management engines that are shared between the tenants. Tenants use the management engines as if they were their sole users, while the underlying computation resources are shared between them. In this setting various model management optimizations can be performed in order to avoid degraded performance caused by a spike in the workload of a tenant. E.g., load identical models only once in memory, track changes in the model according to which tenant made it, off-load infrequently used models to model repository. Moreover, with cloud provisioning and decommissioning, the available computation resources can be elastically scaled, however, their cost implications should be covered by the tenants. Besides, these operations may have severe impacts on ongoing model manipulation processes, thus they have to be scheduled accordingly.

Since LCDPs are collaborative platforms, multiple engineers use them concurrently which imposes several challenges:

- keeping the model consistent among the users in various views, analysis and edit operations,
- maintaining short response time and high throughput of user operations,
- maintaining the high availability of the platform.

The challenges are manifold in the case of model transformations. In model transformations, the intermediate results of the transformations can be kept in memory to achieve a shorter execution time to react to changes in the source model. Besides, if the source model is simultaneously accessed by multiple users and operations (e.g., model analysis, code generation, transformation), then it must be synchronized between the operations.

We are planning to investigate how different locking (e.g., [17]) and lock-free concurrent access mechanisms, researched by the databases community, can be applied for model management, especially for model transformations, to achieve short response time and high throughput, while keeping the models consistent.

Moreover, we are planning to experiment with the aforementioned multi-tenancy patterns, to see how they can be horizontally combined between the LCDP and the model management service.

4.2 Parallel Reactive Model Transformations

Reactive programming provides abstractions to express *event-driven* applications in which data and computation dependencies are managed automatically [1]. These applications react to *events* emitted by external *event sources* without an explicit notion of time or prior knowledge of the sequence of events. Paton and Díaz surveyed active database systems, that provide a knowledge model and execution strategy for supporting reactive behavior in databases [35].

Reactive model transformations adopt principles from reactive programming for model transformations. In the modeling environment, *events* are created from changes in the model, which cause new *matches* for the transformation *precondition* (pattern), which in turn *activates* the transformation *action*.

As for our motivating example, reactive model transformations offer a quick and efficient way to continuously derive formal models from the behavioral engineering models. It is due to the fact that reactive transformations are automatically executed upon changes in the source model, and they update the target model following the traceability links between the two models. In this way, engineering and formal models are synchronized and can be continuously verified to find design errors sooner.

To justify it, let us consider the SysML state machine to Timed Automata transformation. A SysML state machine includes several `Regions` that contain `States` and `Transitions`. `Transitions` connect the different `States`, and can have guard `Constraints`, while `States` may have state invariant `Constraints`, that must be true as long as the `State` is active. A timed automaton is a `Template` that consists of several `Locations` and `Edges`. `Edges` connect the `Locations` and they can have guard, while `Locations` can have invariant `Expressions` that must be true as long as the `Location`

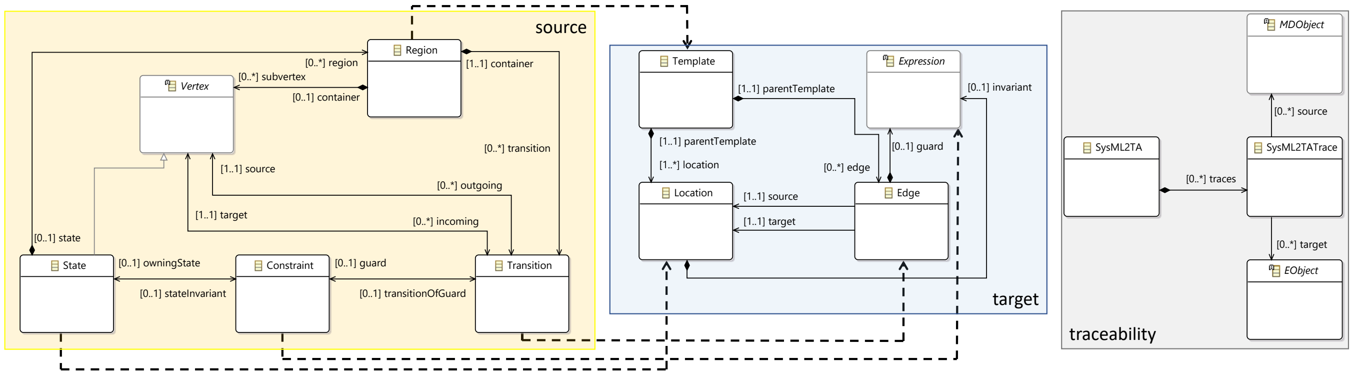

Figure 5: Source, target and traceability metamodels excerpt

is active [38]. The translation of the corresponding elements is illustrated by dashed arrows in Figure 5. During each transformation action, a traceability link (SysML2TATrace) is created, in order to update the target model according to changes in the source model.

Some state-of-the-art transformation engines provide reactive model transformation modes, e.g., the Reactive ATL [36], and Event-driven Virtual Machine (EVM) [5] in VIATRA [49].

Bergmann et al. proposed the EVM concept for reactive model transformations in VIATRA [5]. EVM contains a set of *rule specifications*, which consists of the model transformation *action* and the *precondition* of that *action* as graph pattern. If a precondition matches the model, then it is an *activation*. The EVM contains a scheduler which fires the transformation actions, depending on the life-cycle of the *activation*.

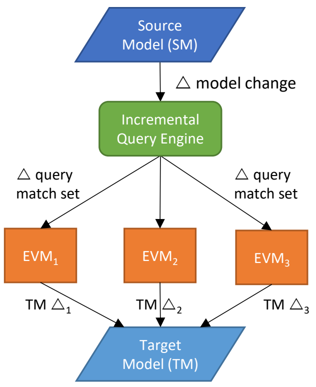

Figure 6: Parallel reactive model transformations

In order to achieve better execution time of reactive model transformations in parallel, we are going to extend the EVM for task-parallel execution mode. The incremental query engine, depicted in Figure 6, receives changes from the model and updates its cache of the partial graph pattern matches, following the Rete algorithm. After that, the engine forwards the updated query match sets (appeared, disappeared, updated pattern matches) to the EVM instances. The *rule specifications* (tasks) are distributed among the EVM instances and so each of them is waiting for matches of different preconditions. If an *activation* appears, then the corresponding *action* is fired, which results in a change in the target model.

```
val transition2EdgeRule = createRule()
 .name("transitionInstanceRule")
 .precondition(transitionInstance)
 .action(ActivationStateEnum.CREATED) [
  // create a new edge in the target model
  val edge = createChild(template, Template.Edge, Edge);
  val source = transition.source;
  val target = transition.target;
  edge.source = getTrgTrace(source) as Location;
  edge.target = getTrgTrace(target) as Location;
  // create traceability link
  createTrace(transition, edge);
 ].action(ActivationStateEnum.UPDATED) [
  // find edge in target model
  val edge = getTrgTrace(transition) as Edge;
  val source = transition.source;
  val target = transition.target;
  // update locations
  edge.source = source == null ?
    null : getTrgTrace(source) as Location;
  edge.target = target == null ?
    null : getTrgTrace(target) as Location;
 ].action(ActivationStateEnum.DELETED) [
  // find edge in target model
  val edge = getTrgTrace(transition) as Edge;
  val template = edge.parentTemplate;
  // remove edge from container
  template.edge.remove(edge) ;
  val trace = getTrace(transition);
  // remove traceability link
  removeTrace(trace);
 ].build();
```

Listing 1: Transition's reactive transformation rule

Listing 1 shows the rule of the `Transition` to `Edge` reactive transformation (`transition2EdgeRule`). As a precondition to the rule, a simple graph pattern is referred, that matches if a `Transition` in the source model is created, removed or its `source` or `target` `States` are updated. To illustrate the proposed method above, this rule can be assigned to an EVM instance running on a separate thread. The instance is going to be monitoring the `Transition` pattern's match set and run the corresponding actions. A similar rule allocation can be done for the `State` to `Location` transformation (`state2LocationRule`) as well. Since the create and the update transformation actions of the `transition2EdgeRule` depend on the `state2LocationRule`, thus their executions should be scheduled accordingly. Besides, the target model should be handled in transactions to guarantee its consistency. Finally, the imperative code shown in Listing 1 may be derived from a declarative representation that may allow for more effective static analysis.

Challenges of task-parallel reactive transformations are:

- dependencies between the model transformation rules should be discovered to find the independent ones [18],
- the concurrent editing of the target model from multiple EVM instances requires transactional model processing and locking mechanisms to guarantee its consistency,
- alternatively, different lock-free mechanisms should be adopted for reactive transformations [40, 41].

4.3 Multi-tenant, Reactive Model Transformation Benchmark

Although many model transformation engines have been developed in the last decades [19, 26] there is no de-facto benchmark to compare their performance. The Transformation Tool Contest (TTC) aims to address this shortcoming by organizing an annual contest to evaluate the performance, verifiability, conciseness and understandability of transformation tools in several challenging case studies. Although some of them evaluate *incremental* transformations, they are not always *reactive* transformations that are automatically activated for events created by the application or external sources.

As it was shown in the motivating example, in order to find the transformation engine that performs the best according to the needs of systems engineers on collaborative platforms, state-of-the-art reactive transformation engines should be compared along a set of selection criteria that are derived from the characteristics (Key Performance Indicators, KPIs) of parallel reactive model transformations on multi-tenant platforms.

KPIs of reactive model transformations in single-threaded execution mode are: (*i*) initial memory need to cache the source model, due to the underlying incremental model queries, (*ii*) model change type (add, remove, update), (*iii*) size of the model change (number of elements), (*iv*) frequency of model changes, (*v*) complexity of the transformation (memory and CPU footprint). Further KPIs to examine, due to the proposed parallel execution mode of reactive transformations: (*vi*) degree of independence between the transformation rules, (*vii*) number of transformations running in parallel, (*viii*) model synchronization time overhead between conflicting rule applications. Some further KPIs to consider in the multi-tenant execution environments are: (*ix*) number of tenants using the same transformation engine, (*x*) degree of isolation between them, (*xi*) load profile of model transformation engines under concurrent transformation requests, (*xii*) maintenance overhead of provisioning operations on model transformation engines, to leverage the elastic resource allocation benefits of the cloud.

From these KPIs, several evaluation criteria can be constructed to assess the performance of model transformation engines in parallel reactive execution mode, on multi-tenant collaborative platforms. Some example evaluation criteria are: (*i*) how do the size and frequency of source model changes influence the memory and CPU consumption of the transformation engine, (*ii*) how quickly is the engine able to react to changes in the model, (*iii*) how many transformations can run in parallel per second, (*iv*) how much synchronization overhead does the resolution of conflicting rule applications cause, (*v*) how many tenants can the same model transformation engine serve, (*vi*) how do cloud provisioning operations on model transformation engines affect their performance.

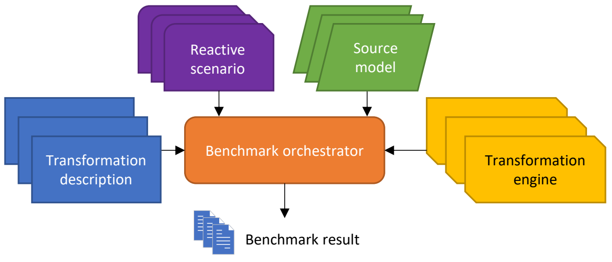

Figure 7: Reactive model transformation benchmark

In order to support the evaluation of reactive model transformation engines, we propose a high-level overview of a benchmark, depicted in Figure 7. The architecture consists of:

- the *transformation descriptions*, which describe the transformation cases in a unified format,
- the *reactive scenarios* that describe the frequency, size and type of changes that should be made on the source models,
- the *source models* on which the model transformations are executed,
- the *transformation engines*, which execute the given transformation on the source model according to the transformation rule and the changes made in the model,
- the *benchmark orchestrator* that controls the workflow. Selects the source model and transformation in a format the transformation engine expects it, runs the reactive scenario on the model to exercise the reactivity of the transformation engine, and measures the run-time metrics of the engine (e.g., transformation execution time, memory and CPU use, number of parallel transformations per second, etc.),
- the *benchmark results* summarize the run-time metrics of the transformation cases on the engines.

In order to design and implement the benchmark, we are going to build on the previous work of Izsó et al. with MONDO-SAM, an extensible MDE benchmark framework [25]. We are going to adopt the workflow stages and primitives for our needs and will reuse its support for metrics calculation and visualization on diagrams.

Similar to Langer et al. [28], we are going to implement a generic API for the workflow and implement adapters for each transformation engine. Similar to the Train Benchmark framework [43], we will design models and transformations with growing size and complexity to exercise the model transformation engines.

The goals of the benchmark are:

1. to define a set of evaluation criteria to compare reactive model transformation engines on multi-tenant platforms,
2. to support the selection of the most suitable engine in the given transformation and application context,
3. to provide means to repeatably measure runtime metrics of state-of-the-art reactive model transformation engines.

Table 2: Summary of research lines and challenges

	Addressed challenge and summary of the derived research line	Research objectives
Multi-tenant model transformations (RL1)	*Improve the scalability of model transformation engines for multi-tenant platforms, while keeping the tenants isolated.* Adopt and experiment with different horizontal combinations of multi-tenancy patterns for model management engines in conjunction with LCDPs.	(RO1.1.) Isolation of tenants while maintaining fair resource distribution between them. (RO1.2.) Tenant-aware model management services with optimized memory and model access. (RO1.3.) High-throughput lock-free and lock-based concurrent model access mechanisms.
Parallel reactive model transformations (RL2)	*Improve the scalability of model transformations for frequently changing models with hundreds of millions of elements.* Adopt reactive model transformations for parallel execution mode in order to be able to quickly react to frequent changes in very large models.	(RO2.1.) Find the independent model transformation rules that can be executed in a task-parallel execution mode to achieve high degree of parallelism. (RO2.2.) Transactional model management with model synchronization primitives to allow parallel model editing and guarantee model consistency. (RO2.3.) Alternatively, adopt different lock-free mechanisms for reactive transformations. (RO2.4.) If the task-parallel execution does not provide satisfactory results, then the data-parallel approach has to be studied for reactive transformations. (RO2.5.) Exploit declarative languages and static analysis to derive efficient imperative transformation code.
Multi-tenant, reactive model transformation benchmark (RL3)	*Characterize reactive model transformations on multi-tenant low-code platforms to select the most suitable transformation engine in the given context.* Design and implement a novel model transformation benchmark for reactive transformations on multi-tenant platforms with increasingly more complex transformation rules and models.	(RO3.1.) Empirically identify the Key Performance Indicators of reactive model transformations on multi-tenant platforms. (RO3.2.) Derive evaluation criteria that can be used to assess the model transformation engines. (RO3.3.) Design transformation rules and models that are similar in complexity to real-world scenarios. (RO3.4.) Implement a benchmark workflow that runs the transformation rules on the engines using the source models along the reactive scenario descriptions.

4.4 Summary of research lines and challenges

Table 2 summarizes the research lines together with the challenges they address and the open questions within each of them.

5 CONCLUSION

In this paper, we outlined three research lines to address *scalability* and *productivity* challenges in Low-Code Development Platforms. We used a model validation and verification workflow to highlight the challenges in a real-world scenario. Besides, we provided a mapping of the challenges to research lines:

- adoption of multi-tenant architecture patterns for model transformations on low-code platforms, to improve the scalability in terms of number of users,
- parallel reactive model transformations, to improve the scalability of model transformations in terms of model size,
- multi-tenant, reactive model transformation benchmark, to increase productivity by providing different selection criteria that can be used to assess the performance of reactive model transformation engines on multi-tenant platforms.

As future work, the practical solutions of the research lines are going to be integrated to IncQuery Server, a scalable query evaluation middleware in the cloud [23]. The goal is to enhance the middleware into a collaborative, multi-tenant engineering platform with advanced reactive model transformation capabilities over cloud-based model repositories.

ACKNOWLEDGMENTS

This work is funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 813884. The authors are grateful for the valuable feedback of Dániel Varró and István Ráth about the paper.

REFERENCES

[1] Engineer Bainomugisha, Andoni Lombide Carreton, Tom Van Cutsem, Stijn Mostinckx, and Wolfgang De Meuter. 2013. A survey on reactive programming. *Comput. Surveys* 45, 4 (2013), 52:1–52:34.
[2] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. 2004. A Tutorial on Uppaal. In *Formal Methods for the Design of Real-Time Systems (LNCS, Vol. 3185)*. Springer, 200–236.
[3] Amine Benelallam, Abel Gómez, Massimo Tisi, and Jordi Cabot. 2015. Distributed Model-to-Model Transformation with ATL on MapReduce. In *Proc. of the International Conference on Software Language Engineering (SLE 2015)*. Association for Computing Machinery, 37–48.
[4] Amine Benelallam, Massimo Tisi, István Ráth, Benedek Izsó, and Dimitris S. Kolovos. 2014. Towards an Open Set of Real-World Benchmarks for Model

Queries and Transformations. In *Proc. of the 2nd Workshop on Scalability in Model Driven Engineering (CEUR-WS Proceedings, Vol. 1206)*. CEUR-WS.org, 14–22.

[5] Gábor Bergmann, István Dávid, Ábel Hegedüs, Ákos Horváth, István Ráth, Zoltán Ujhelyi, and Dániel Varró. 2015. Viatra 3: A Reactive Model Transformation Platform. In *Proc of the 8th International Conference on the Theory and Practice of Model Transformations (LNCS, Vol. 9152)*. Springer, 101–110.

[6] Gábor Bergmann, Ákos Horváth, István Ráth, Dániel Varró, András Balogh, Zoltán Balogh, and András Ökrös. 2010. Incremental Evaluation of Model Queries over EMF Models. In *MODELS*, Vol. 6394. Springer, 76–90.

[7] Gábor Bergmann, István Ráth, and Dániel Varró. 2009. Parallelization of Graph Transformation Based on Incremental Pattern Matching. *ECEASST* 18 (2009).

[8] Antonio Bucchiarone, Jordi Cabot, Richard F. Paige, and Alfonso Pierantonio. 2020. Grand challenges in model-driven engineering: an analysis of the state of the research. *SoSyM* 19, 1 (2020), 5–13.

[9] Loli Burgueño, Eugene Syriani, Manuel Wimmer, Jeffrey G. Gray, and Antonio Vallecillo. 2014. LinTraP: Primitive Operators for the Execution of Model Transformations with LinTra. In *Proc. of the 2nd Workshop on Scalability in Model Driven Engineering (CEUR-WS Proceedings, Vol. 1206)*. CEUR-WS.org, 23–30.

[10] Loli Burgueño, Javier Troya, Manuel Wimmer, and Antonio Vallecillo. 2015. Parallel In-place Model Transformations with LinTra. In *Proc. of the 3rd Workshop on Scalable Model Driven Engineering part of the Software Technologies: Applications and Foundations (CEUR-WS Proceedings, Vol. 1406)*. CEUR-WS.org, 52–62.

[11] Loli Burgueño, Manuel Wimmer, and Antonio Vallecillo. 2016. A Linda-based platform for the parallel execution of out-place model transformations. *Information and Software Technology* 79 (2016), 17–35.

[12] Hong Cai, Ning Wang, and Ming Jun Zhou. 2010. A Transparent Approach of Enabling SaaS Multi-tenancy in the Cloud. In *Proc of the 6th World Congress on Services*. IEEE Computer Society, 40–47.

[13] Théo Le Calvar, Frédéric Jouault, Fabien Chhel, and Mickael Clavreul. 2019. Efficient ATL Incremental Transformations. *JOT* 18, 3 (2019), 2:1–17.

[14] Edmund M Clarke, Thomas A Henzinger, Helmut Veith, and Roderick P Bloem. 2018. *Handbook of model checking*. Springer.

[15] Pierre David, Vincent Idasiak, and Frédéric Kratz. 2010. Reliability study of complex physical systems using SysML. *Reliability Engineering & System Safety* 95, 4 (2010), 431–450.

[16] Juan de Lara and Dániel Varró. 2010. Preface of the 4th International Workshop on Graph-Based Tools. *ECEASST* 32 (2010).

[17] Csaba Debreceni, Gábor Bergmann, István Ráth, and Dániel Varró. 2017. Property-Based Locking in Collaborative Modeling. In *MODELS*. IEEE Computer Society, 199–209.

[18] Hartmut Ehrig. 1978. Introduction to the Algebraic Theory of Graph Grammars (A Survey). In *Graph-Grammars and Their Application to Computer Science and Biology (LNCS, Vol. 73)*. Springer, 1–69.

[19] Juergen Etzlstorfer, Angelika Kusel, Elisabeth Kapsammer, Philip Langer, Werner Retschitzegger, Johannes Schoenboeck, Wieland Schwinger, and Manuel Wimmer. 2013. A Survey on Incremental Model Transformation Approaches. In *Proc. of the Workshop on Models and Evolution (CEUR-WS Proceedings, Vol. 1090)*. CEUR-WS.org, 4–13.

[20] Christoph Fehling, Frank Leymann, Ralph Retter, Walter Schupeck, and Peter Arbitter. 2014. *Cloud Computing Patterns - Fundamentals to Design, Build, and Manage Cloud Applications*. Springer.

[21] Charles Forgy. 1982. Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artificial Intelligence* 19, 1 (1982), 17–37.

[22] Kevin Forsberg, Hal Mooz, and Howard Cotterman. 2005. *Visualizing project management: models and frameworks for mastering complex systems*. John Wiley & Sons.

[23] Ábel Hegedüs, Gábor Bergmann, Csaba Debreceni, Ákos Horváth, Péter Lunk, Ákos Menyhért, István Papp, Dániel Varró, Tomas Vileiniskis, and István Ráth. 2018. Incquery server for teamwork cloud: scalable query evaluation over collaborative model repositories. In *MODELS*. ACM, 27–31.

[24] Kai Hu, Lei Lei, and Wei-Tek Tsai. 2016. Multi-tenant Verification-as-a-Service (VaaS) in a cloud. *Simulation Modelling Practice and Theory* 60 (2016), 122–143.

[25] Benedek Izsó, Gábor Szárnyas, István Ráth, and Dániel Varró. 2014. MONDO-SAM: A Framework to Systematically Assess MDE Scalability. In *Proc. of the 2nd Workshop on Scalability in Model Driven Engineering (CEUR-WS Proceedings, Vol. 1206)*. CEUR-WS.org, 40–43.

[26] Nafiseh Kahani, Mojtaba Bagherzadeh, James R. Cordy, Juergen Dingel, and Dániel Varró. 2019. Survey and classification of model transformation tools. *SoSyM* 18, 4 (2019), 2361–2397.

[27] Dimitrios S. Kolovos, Louis M. Rose, Nicholas Drivalos Matragkas, Richard F. Paige, Esther Guerra, Jesús Sánchez Cuadrado, Juan de Lara, István Ráth, Dániel Varró, Massimo Tisi, and Jordi Cabot. 2013. A research roadmap towards achieving scalability in model driven engineering. In *Proc. of the Workshop on Scalability in Model Driven Engineering*. ACM, 2.

[28] Philip Langer and Manuel Wimmer. 2013. A Benchmark for Conflict Detection Components of Model Versioning Systems. *Softwaretechnik-Trends* 33, 2 (2013).

[29] Tihamer Levendovszky, László Lengyel, Gergely Mezei, and Hassan Charaf. 2005. A Systematic Approach to Metamodeling Environments and Model Transformation Systems in VMTS. *Electronic Notes in Theoretical Computer Science* 127, 1 (2005), 65–75.

[30] Sina Madani, Dimitris S. Kolovos, and Richard F. Paige. 2019. Towards Optimisation of Model Queries: A Parallel Execution Approach. *Journal Object Technology* 18, 2 (2019), 3:1–21.

[31] Gergely Mezei, Tihamer Levendovszky, Tamas Meszaros, and Istvan Madari. 2009. Towards truly parallel model transformations: A distributed pattern matching approach. In *IEEE EUROCON 2009*. IEEE, 403–410.

[32] Ralph Mietzner, Frank Leymann, and Tobias Unger. 2011. Horizontal and vertical combination of multi-tenancy patterns in service-oriented applications. *Enterprise Information Systems* 5, 1 (2011), 59–77.

[33] Ralph Mietzner, Tobias Unger, Robert Titze, and Frank Leymann. 2009. Combining Different Multi-tenancy Patterns in Service-Oriented Applications. In *Proc. of the 13th International Enterprise Distributed Object Computing Conference*. IEEE, 131–140.

[34] OMG. 2019. *OMG System Modeling Language (SysML)*. formal/19-11-01.

[35] Norman W. Paton and Oscar Díaz. 1999. Active Database Systems. *ACM Comput. Surv.* 31, 1 (1999), 63–103.

[36] Salvador Martínez Perez, Massimo Tisi, and Rémi Douence. 2017. Reactive model transformation with ATL. *Science of Computer Programming* 136 (2017), 1–16.

[37] Mehrdad Sabetzadeh, Shiva Nejati, Lionel C. Briand, and Anne-Heidi Evensen Mills. 2011. Using SysML for Modeling of Safety-Critical Software-Hardware Interfaces: Guidelines and Industry Experience. In *Proc of the 13th International Symposium on High-Assurance Systems Engineering*. IEEE, 193–201.

[38] Stefano Schivo, Bugra M. Yildiz, Enno Ruijters, Christopher Gerking, Rajesh Kumar, Stefan Dziwok, Arend Rensink, and Mariëlle Stoelinga. 2017. How to Efficiently Build a Front-End Tool for UPPAAL: A Model-Driven Approach. In *Proc of the 3rd International Symposium on Dependable Software Engineering (LNCS, Vol. 10606)*. Springer, 319–336.

[39] Andy Schürr, Dániel Varró, and Gergely Varró (Eds.). 2012. *Proc of the 4th International Symposium on Applications of Graph Transformations with Industrial Relevance*. LNCS, Vol. 7233. Springer.

[40] Marc Shapiro, Nuno M. Preguiça, Carlos Baquero, and Marek Zawirski. 2011. Conflict-Free Replicated Data Types. In *Proc of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (LNCS, Vol. 6976)*. Springer, 386–400.

[41] David Sun, Chengzheng Sun, Agustina Ng, and Weiwei Cai. 2020. Real Differences between OT and CRDT in Correctness and Complexitymfor Consistency Maintenance in Co-Editors. *PACMHCI* 4, GROUP (2020), 021:1–021:30.

[42] Gábor Szárnyas, Benedek Izsó, István Ráth, Dénes Harmath, Gábor Bergmann, and Dániel Varró. 2014. IncQuery-D: A Distributed Incremental Model Query Framework in the Cloud. In *Proc. of the 17th International Conference on Model-Driven Engineering Languages and Systems (LNCS, Vol. 8767)*. Springer, 653–669.

[43] Gábor Szárnyas, Benedek Izsó, István Ráth, and Dániel Varró. 2018. The Train Benchmark: cross-technology performance evaluation of continuous model queries. *SoSyM* 17, 4 (2018), 1365–1393.

[44] Massimo Tisi, Rémi Douence, and Dennis Wagelaar. 2015. Lazy Evaluation for OCL. In *Proc. of the 15th International Workshop on OCL and Textual Modeling (CEUR-WS Proceedings, Vol. 1512)*. CEUR-WS.org, 46–61.

[45] Massimo Tisi, Jean-Marie Mottu, Dimitrios S. Kolovos, Juan de Lara, Esther Guerra, Davide Di Ruscio, Alfonso Pierantonio, and Manuel Wimmer. 2019. Lowcomote: Training the Next Generation of Experts in Scalable Low-Code Engineering Platforms. In *Co-Located Events Joint Proceedings with Software Technologies: Applications and Foundations (CEUR-WS Proceedings, Vol. 2405)*. CEUR-WS.org, 73–78.

[46] Massimo Tisi, Salvador Martínez Perez, and Hassene Choura. 2013. Parallel Execution of ATL Transformation Rules. In *Proc. of the 16th International Conference on Model-Driven Engineering Languages and Systems (LNCS, Vol. 8107)*. Springer, 656–672.

[47] Massimo Tisi, Salvador Martínez Perez, Frédéric Jouault, and Jordi Cabot. 2011. Lazy Execution of Model-to-Model Transformations. In *MODELS (LNCS, Vol. 6981)*. Springer, 32–46.

[48] Tamás Tóth, Ákos Hajdu, András Vörös, Zoltán Micskei, and István Majzik. 2017. Theta: A framework for abstraction refinement-based model checking. In *Formal Methods in Computer Aided Design*. IEEE, 176–179.

[49] Dániel Varró, Gábor Bergmann, Ábel Hegedüs, Ákos Horváth, István Ráth, and Zoltán Ujhelyi. 2016. Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework. *SoSyM* 15, 3 (2016), 609–629.

[50] Gergely Varró, Andy Schürr, and Dániel Varró. 2005. Benchmarking for Graph Transformation. In *Symposium on Visual Languages and Human-Centric Computing*. IEEE Computer Society, 79–88.