Automated Model Merge by Design Space Exploration

Csaba Debreceni[1(✉)], István Ráth[1], Dániel Varró[1], Xabier De Carlos[2], Xabier Mendialdua[2], and Salvador Trujillo[2]

[1] Department of Measurement and Information Systems, Budapest University of Technology and Economics, Magyar tudósok krt. 2, Budapest 1117, Hungary
{debreceni,rath,varro}@mit.bme.hu
[2] IK4-IKERLAN Research Center, P.J.M. Arizmendiarrieta, 2, 20500 Arrasate, Spain
{xdecarlos,xmendialdua,strujillo}@ikerlan.es

Abstract. Industrial applications of model-driven engineering to develop large and complex systems resulted in an increasing demand for collaboration features. However, use cases such as model differencing and merging have turned out to be a difficult challenge, due to (i) the graph-like nature of models, and (ii) the complexity of certain operations (e.g. hierarchy refactoring) that are common today. In the paper, we present a novel search-based automated model merge approach where rule-based design space exploration is used to search the space of solution candidates that represent conflict-free merged models. Our method also allows engineers 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. Furthermore, we propose to adopt a generic synthetic benchmark to carry out an initial scalability assessment for model merge with large models and large change sets.

1 Introduction

Scalable collaborative model-driven engineering (MDE) for complex projects with multiple stakeholders and development groups working in a distributed way (both geographically and in time) is a major research challenge [21]. In traditional software engineering, version control systems (VCS) such as SVN or Git assist to work with textual documents in *off-line collaboration scenarios* having long transactions and complex modifications between commits. Since multiple collaborators may try to commit changes to the same document, a *comparison* or difference is calculated prior to local commit, which may cause *conflicts* between remote changes (already published to the server) and local changes (aimed to be

This paper is partially supported by the EU Commission with project MONDO (FP7-ICT-2013-10, #611125) and the MTA-BME Lendület 2015 Research Group on Cyber-Physical Systems.

P. Stevens and A. Wąsowski (Eds.): FASE 2016, LNCS 9633, pp. 104–121, 2016.
DOI: 10.1007/978-3-662-49665-7_7

committed now). Such conflicts need to be resolved by *merging* the remote and local changes in a consistent way before a commit succeeds.

Unfortunately, the direct use of VCS in MDE is hindered by numerous factors implied by the differences between graph-based documents (e.g. models) and textual documents (e.g. source code). A major challenge is related to model comparison, which is also computationally more expensive over graphs, and it gave birth to advanced industrial strength frameworks like EMF Compare [1] or Diff/Merge [2] built into model-level version control systems (like in Papyrus UML or AMOR [5]). In order to achieve scalability for large models, these frameworks frequently assume that unique identifiers are available for model elements. That assumption results in more efficient model comparison algorithms.

While model comparison is computationally more challenging, resolving conflicting model changes is still a cumbersome task in practice, which is frequently performed manually by the engineers. EMF Compare and Diff/Merge enable automated conflict resolution in a programmatic way — but writing code for an automated merge is hardly a task for a domain expert. Furthermore, domain-specific conflict resolution strategies are rarely taken into consideration in industrial frameworks, hence the well-formedness of merge results is questionable.

In this paper, we propose a novel automated search-based model merge technique [20] which builds on off-the-shelf tools for the model comparison step, but uses guided rule-based design space exploration (DSE) [18] for merging models. In general, rule-based DSE aims to search and identify various design candidates to fulfill certain structural and numeric constraints. The exploration starts from an initial model and systematically traverses paths by applying operators. In our context, the results of model comparison will be the initial model, while target design candidates will represent the conflict-free merged models.

While many existing model merge approaches detect conflicts statically in a preprocessing phase, our DSE technique carries out *conflict detection dynamically* during exploration time as conflicting rule activations and constraint violations. Then *multiple consistent resolutions of conflicts* are presented to the domain experts. Our technique allows to incorporate domain-specific knowledge into the merge process by additional constraints, goals and operations to provide better solutions. Finally, we propose to adapt a generic scalability benchmark for assessing model merge performance for large models and large change sets, which is also an innovative aspect of the paper.

The rest of the paper is structured as follows: A motivating case study of modeling wind turbine control systems is presented in Sect. 2 together with the basics of model comparison and merge. A high-level overview of our approach is provided in Sect. 3. A detailed explanation of executing a merge process is discussed in Sect. 4. The case study will also serve as an initial assessment of the usefulness of a domain-specific merge technique while scalability evaluation will be carried out by adapting the Train Benchmark [29] in Sect. 5. Related work is summarized in Sect. 6 while Sect. 7 concludes our paper.

2 Preliminaries

2.1 From Model Comparison to Model Merge

Model comparison refers to identifying the differences between models. It requires reliability, precision and completeness as the merge process frequently relies on the output of this phase to detect conflicts and to resolve the detected conflicts. Altmanninger et al. [6] classifies model comparison methods based on the kind of information available. Only models are provided as input for *state-based* techniques, while *change-based* comparison relies on a list of the performed changes, e.g. op_1, op_2, ... op_n.

Based on the results of model comparison, *model merge* synthesizes a combined model which reconciles the identified differences. This is not always possible due to conflicts between model changes carried out by different collaborators. A merged model is called *syntactically correct* if it corresponds to its metamodel, and *consistent* when additional constraints of the domain are satisfied.

We use a simplified difference model derived from the EMF Compare tool [1] to store the changes in EMF models. This allows us to accept different types of comparison model (e.g. EMF Compare or Diff/Merge [2]) as an input of model merge. It contains the following default change types: (1) *create* or *delete* an object; (2) set, add or remove a value or an object to/from an *attribute* or a *reference*, respectively. Furthermore, we annotate the *priority* of changes as *may* or *must* which will be decided by users. Changes with *must* priority are mandatory to be involved in the solutions while the others with *may* priority can be omitted.

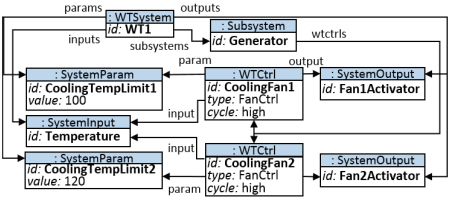
In the paper, we focus on *three-way merge*, which also uses the common ancestor O of local copy L and remote copy R to derive the merged model M. To determine the changes executed on O, a comparison is conducted between $O \leftrightarrow L$ and $O \leftrightarrow R$. The solution of merge M is obtained by applying a combination of changes performed either on L or R to the original model O.

2.2 A Motivating Model Merge Scenario

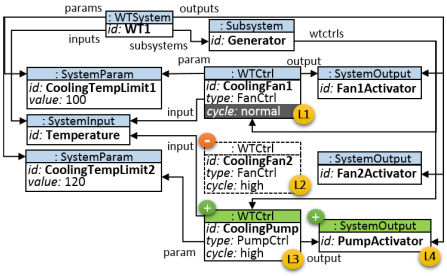
The domain of our motivating example describes *Wind Turbine Control Systems (WTCS)* developed by *IK*4-*Ikerlan* where different artefacts and algorithms for controlling a wind turbine are specified and connected to sensors and actuators. Models are specified by several collaborators, and consequently modifications could result in merge conflicts.

We introduce a simplified example of a wind turbine (WT1) in Fig. 1. Real models are obviously larger, sample models of this paper contain only artifacts related to the cooling of the Generator Subsystem:

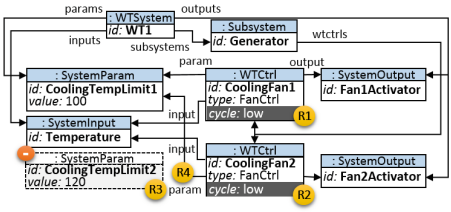
- *Inputs*: Wind turbine WT1 gets data from a temperature sensor specified by the SystemInput identified as Temperature.
- *Outputs*: WT1 acts on two fans for cooling the wind turbine generator specified by the SystemOutputs: Fan1Activator and Fan2Activator.



(a) Original model



(b) Local instance (modified by User1)



(c) Remote instance (modified by User2)

Fig. 1. Local and remote changes for 3-way merge

– *Params*: temperature limits for starting generator cooling can be specified by SystemParams: CoolingTempLimit1 and CoolingTempLimit2.

Subsystem Generator contains all the control units for cooling the Generator:

– CoolingFan1: this control unit (of type FanCtrl) specifies the control algorithm for fan #1 with High priority cycle with Temperature as SystemInput, Fan1Activator as SystemOutput, CoolingTempLimit1 as SystemParam.
– CoolingFan2: this control unit (of type FanCtrl) specifies the control algorithm for fan #2 with High priority cycle with Temperature as SystemInput, Fan2Activator as SystemOutput and CoolingTempLimit2 as SystemParam.

As a running example, we investigate the following scenario:

Local Changes. The first expert creates a Local version of the model with the following changes: (L1) the cycle attribute of CoolingFan1 is changed to Normal, (L2) CoolingFan2 instance is deleted. (L3) A new control unit (WTCtrl) is created with CoolingPump id. The new control unit is of type PumpCtrl with High cycle. Its input references the existing Temperature and its param references the existing CoolingTempLimit2. In contrast, (L4) its output references a new SystemOutput instance identified as PumpActivator.

Remote Changes. Another expert also remotely modified and already committed the model (before the first expert working on the local version managed to commit the model) to introduce the following remote changes: (R1) the cycle

attribute of CoolingFan1 is changed to Low, (R2) the cycle attribute of CoolingFan2 is changed to Low, (R3) deletes SystemParam instance identified as CoolingTempLimit2 and (R4) changes param reference of control unit identified as CoolingFan2 to SystemParam instance identified as CoolingTempLimit1.

Model Comparison. Table 1 shows the result of model comparison between the different versions of the model calculated by using existing tools (using e.g. EMF Compare or Diff/Merge [2]). The differences between the local and the original model is denoted with $\Delta(L, O)$ (or shortly ΔL), while $\Delta(R, O)$ (or ΔR) represents the differences between the remote and the original model.

Table 1. Elements of $\Delta(L)$ and $\Delta(R)$

Δ(L,O)comparison model	Δ(R,O)comparison model
MAY attribute{CoolingFan1,cycle,Normal}	MAY attribute{CoolingFan1,cycle,Low}
MUST delete{CoolingFan2}	MUST attribute{CoolingFan2,cycle,Low}
MUST create{CoolingPump,WTCtrl,WT1,ctrls}	MAY delete{CoolingTempLimit2}
MAY attribute{CoolingPump,type,PumpCtrl}	MAY reference{CoolingFan2,param, CoolingTempLimit1}
MAY attribute{CoolingPump,cycle,High}	
MAY reference{CoolingPump,param,CoolingTempLimit2}	
MUST create{PumpActivator,SystemOutput,WT1,outputs}	
MAY reference{CoolingPump,output,PumpActivator}	

Change Annotation. After the comparison, the local collaborator annotates local changes $L2$, $L3$ and $L4$ and remote change $R2$ as *must* which prescribes that all such changes have to be present in the merged model unless some of them are in a conflict. In such a case, the merged model should contain as many (non-conflicting) *must* changes as possible, while some (conflicting) *must* changes might be omitted from the merged model. All other changes are marked as *may* to denote that the corresponding change may be included in the merged model.

Challenges. The following challenges need to be addressed for our example:

- Calculate *merged models* automatically as a maximal subset of non-conflicting changes from the local and remote change set. When there is a large number of possible combination of changes where some of them are selected from the local and the others from the remote branch, a merged model may be restricted to solutions compliant with *must* and *may* change annotations.
- Use *domain-specific goals and constraints* to restrict merged models to consistent ones (to ensure that all inputs and parameters are referenced by at least one control unit and each output is referenced by different control unit).
- Specify *domain-specific composite operations* to guide the merge process into a consistent solution (e.g. to remove inputs, parameters and outputs not referenced by any control unit).

3 Model Merge by Design Space Exploration: Concepts

3.1 Conceptual Overview

We propose to exploit guided rule-based *design space exploration (DSE)* [18] for automated model merge with an architecture depicted in Fig. 2. Rule-based DSE aims at finding optimal *solutions* from the several *design candidates* which satisfy several structural and numeric constraints, and they are reachable from an initial model along a trajectory by applying a sequence of exploration rules. The input of a rule-based DSE includes (1) the *initial model* used as the start of the exploration; (2) *goals* which need to be satisfied by solutions; (3) the set of *exploration rules*; (4) *constraints* that need to be respected in each exploration state and (5) further *guidance* for the exploration process.

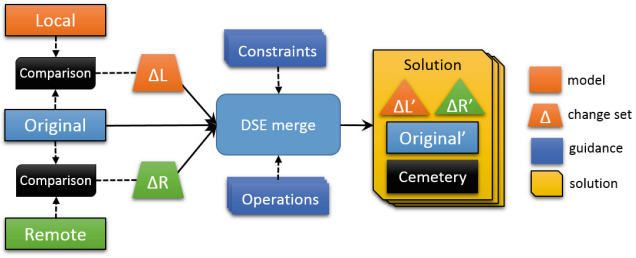


Fig. 2. Architecture of DSE Merge

We applied three-way model merge to a DSE problem as follows:

(1) the **initial model** contains the *original* model O and two *difference models* (ΔL and ΔR)
(2) the main **goal** is that there are no executable changes left in ΔL and ΔR along a specific exploration path.
(3) the **operations** are defined by change driven transformation rules to process generic change objects (*create*, *delete*, *set*, *add*, *remove*) of the difference models, and potentially composite (domain-specific) operators;
(4) **constraints** may identify inconsistencies and conflicts to eliminate certain trajectories;
(5) as main **exploration strategy**, any changes annotated as *must* are tried to be merged before resolving *may* changes.

Input. Our model merge approach takes three models as input: the original model O and the *difference models* between local and original models ΔL as well as the remote and original models ΔR. These together constitute the initial model for DSE. The calculation of the difference models ΔL and ΔR is carried out by an external comparison tool such as EMF Compare or Diff/Merge. Furthermore, in order to derive efficient state encoding for the exploration process, we assume that each element in the original model has some unique identifier.

Output. The output of the merge process automatically derived by DSE is a *set of solutions* where each solution consists of (i) the merged model M derived by applying a (non-extensible and non-conflicting) subset of local and remote changes on the original model O; (ii) the set of non-executed changes $\Delta L'$,$\Delta R'$; and (iii) the collection of the deleted objects stored in $Cemetery$.

3.2 Key Aspects of Exploration Process

Each solution is derived along a trajectory from the initial state to a solution state by *applying generic and domain-specific operations*. Along this trajectory, we transform the original model O into the merged model M, and the change models ΔL and ΔR are gradually reduced to $\Delta L'$ and $\Delta R'$. In each exploration step, *conflicts are detected and resolved* by incrementally tracking the matches (activations) of operations and constraints. Finally, a solution state is identified if all *goals* are satisfied without violating a *constraint* along the trajectory.

Operations. We incorporate two kinds of operations in the exploration based model merge: *generic merge operations* [30] and (domain-specific) *composite operations* [14,23] (such as refactorings, or repair rules). Each operation is captured by (graph) transformation rules [16], which consist of a *precondition* described as a graph pattern (using the EMF-IncQuery language [10] in our case) and an *action* part which captures model manipulations.

Generic merge operations are *change-driven transformations* [9], which consume or produce change models as additional input or output. The precondition selects an applicable change c from the deltas $\Delta L \cup \Delta R$ and may require the existence of certain model elements in the origin model O. The action part of a generic merge operation (1) modifies the original model O to apply a change, (2) moves the change c from the difference set $\Delta L \cup \Delta R$ into a completed set $Comp$ to prevent the application of the change multiple times. Thus such change-driven rules transform state-based merging into operation-based merging [12].

By default, domain-specific *composite operations* only manipulate the model O without consuming the deltas. Therefore, they need to be complemented with generic change-driven rules which identify the model-level changes carried out by them and record them as difference models in the completed set. In most cases, domain experts are responsible for capturing complex (domain-specific) operations only at the preparation of the merge tool for the specific domain. Collaborating engineers only use them as part of the merge process.

Conflict Detection and Resolution. A local change $l \in \Delta L$ and a remote change $r \in \Delta R$ may be conflicting, i.e. it is impossible to obtain a consistent merged model M by applying both l and r. Alternatively, in an operation-based interpretation, a conflict denotes a pair of operations o_1 and o_2, whereas one operation masks the effect of the other (i.e., they do not commute) or one operation disables the applicability of the other [23].

Instead of static (a priori) detection of conflicts as proposed in [17,24,27], we detect conflicts *on- the-fly* during the exploration process by relying upon the *incremental book-keeping of rule activations and constraints*. In each state of the

DSE, we investigate one by one all (enabled) activations of transformation rules, and try to find a solution by firing them. In case of a conflict, (1) firing one rule may prevent the application of another activation, or (2) both rules are fireable, but the result state violates a constraint. When two operations are confluent (i.e. they can be applied in arbitrary order), state encoding of DSE [19] helps identify that an already traversed state is reached. Hence applying operations in a different order has no impact on the results.

Activations of rules and constraints are continuously and efficiently maintained when firing an operation (either generic or composite), thus disabled rules and violated constraints are immediately identified. For that purpose, we rely upon the reactive VIATRA framework [8] and incremental model queries. The technicalities of conflict detection will be illustrated in Sect. 4.

Conflict Resolution by Exploration Strategy. In case of a conflict between two operations, DSE will investigate both trajectories as possible resolutions and derive two separate solutions correspondingly. Thus a merged model M derived automatically as a solution contains no conflicts by definition.

In case of many conflicts, the result set can too large to be presented to experts. Therefore, in order to reduce the number of solutions retrieved by DSE and guide the exploration in case of conflicts, model changes can be prioritized by the collaborators as *may* and *must* (see Table 1) prior to executing merge.

- If a change c_1 with *must* priority is in conflict with another change c_2 of *may* priority, then the merge will always select the former (c_1).
- If two conflicting changes c_1 and c_2 are both annotated with *may* than the merge will randomly select one.
- However, if two changes c_1 and c_2 of *must* priority are in conflict, then the merge process will enumerate both of them separately (in different solutions).

Goals. In generic, we aim to apply as many changes in ΔL and ΔR as possible to derive the merged model M. When extending a trajectory by any of the remaining changes in $\Delta L'$ or $\Delta R'$ would cause a conflict with some already applied change, a solution state of the DSE is reached. Technically, it is detected by the termination of the rule system, i.e. no operations are activated. Additionally, domain experts can provide domain-specific goals that act as heuristics for the exploration and provide consistent solutions.

Altogether, we define a *fully automated model merge approach* where all possible resolutions of conflicts are calculated, and all consistent merged models are prompted to experts, which was claimed to be beneficial in [31]. Representation of solutions contains several layouts (e.g. tree, graph) and metrics (e.g. number of executed changes) which help experts select the best solution for their purpose.

4 Elaboration of Model Merge on an Example

4.1 Operations and Goals

Change-Driven Rules for Generic Operations. We defined the following generic operations in the merge process for *creating/deleting object*,

setting/adding/removing attribute and *setting/adding/removing reference*. For space considerations, we only discuss operations for setting an attribute (*setAttribute*) and deleting an object (*deleteObject*) in details (depicted in Fig. 3).

- *setAttribute(ac,o)*: The precondition prescribes that an attribute change *ac* is available in change set $\Delta L' \cup \Delta R'$ and its object *o* exists in the current model. Its action sets (i) attribute *ac.attribute* of object *o* to the given value *ac.value*, and (ii) moves the change *ac* to the completed set *Comp*.
- *deleteObject(dc,o)*: The precondition states that a delete change *dc* is available in the current change set $\Delta L' \cup \Delta R'$ and its referred object *o* exists in the current state of the model where *o* is a leaf in the containment hierarchy. The action part (i) deletes the object *o* from current state, (ii) puts it into *Cemetary* and (iii) moves the change *dc* to the completed set *Comp*.

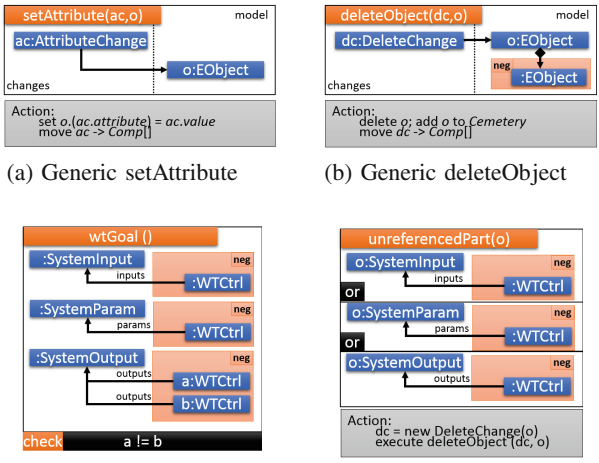
Domain-Specific Goals and Operations. Our example introduced in Sect. 2 requires to extend model merge with domain-specific knowledge to guarantee the consistency of solutions. In the *Wind Turbine Control System (WTCS)* domain, it is mandatory that all SystemInput and SystemParam instances should be referenced by at least one control unit and each SystemOutput has to be referenced by a unique control unit. Model merge needs to respect such domain specific knowledge, which can be captured by additional goals specified as constraints and depicted in a graphical representation in Fig. 3c.

A domain-specific operation called *unreferencedPart* can be defined to eliminate unreferenced SystemInput, SystemOutput and SystemParam instances (see Fig. 3d). Here the precondition selects the unreferenced object *o* while the action part (i) initiates a new *delete change* independently from the current change set and (ii) executes the action part of the generic *delete* operation.

4.2 Conflict Detection in a Sample Exploration Step

Conflict detection and resolution is carried out during exploration by incrementally tracking rule activations and special constraints. We illustrate this step in the context of our running example (see Fig. 4, which is an extract of *iteration* 3 *and* 9 of merge session from Sect. 4.3). It demostrates a delete/use conflict: simultaneously setting the cycle attribute of CoolingFan2 and deleting CoolingFan2. Any solution of model merge may only contain one of the two changes.

1. In the beginning, both operations have an activation (left in Fig. 4) in the context of object CoolingFan2. Initially, all changes are located in ΔL or ΔR, cemetery and completed changes are empty. In this state, all constraints are satisfied, but goals are violated which means this state is not a solution.
2. Our merge process first selects and executes the deleteObject operation (top branch of Fig. 4) which removes CoolingFan2 from the model, moves CoolingFan2 to the *cemetery*, and the corresponding change is moved from ΔL to the completed set *Comp*. As a side effect, operation setAttribute loses its activation in the context of CoolingFan2 since its precondition is no longer be satisfied in



(a) Generic setAttribute (b) Generic deleteObject

(c) Domain-specific goal (d) Domain-specific operation

Fig. 3. Operations and goal

the new state. This fact is immediately identified by the underlying reactive transformation engine [8]. In the new state, the exploration incrementally checks that all constraints are satisfied and goals are violated, and then selects another enabled (activated) operation for execution.

3. Later, after backtracking to the first state, operation setAttribute is scheduled for execution on object CoolingFan2 (bottom branch of Fig. 4). As a result, *Cemetery* remains empty, the change is moved to the completed set, all goals are violated, and all constraints are satisfied. As a main conceptual difference, the activation of deleteObject is not disabled on CoolingFan2 as the corresponding object still exists, hence its precondition is satisfied.
4. Next, the process selects and executes deleteObject operation. As a result, CoolingFan2 is moved to the *cemetery* and the change is moved from ΔR to the completed set *Comp*. We detect this conflict by (incrementally) checking a generic merge constraint: there are two changes in the completed-set *Comp* which modifies the same object. In this case, exploration has to backtrack and finds another executable operation.

Obviously, the first type of constraint could also be detected by using similar constraints as for the second type. However, lost activations reduce the number of states to be traversed, thus they are preferred. Furthermore, note that when two operations are applicable in both order with a confluent result, the state encoding of DSE identifies that the same model is reached as a state.

4.3 A Merge Scenario on the Motivating Example

A possible execution of the DSE Merge is depicted in Fig. 5 which displays the completed changes for two solutions. In each iteration, one change is processed.

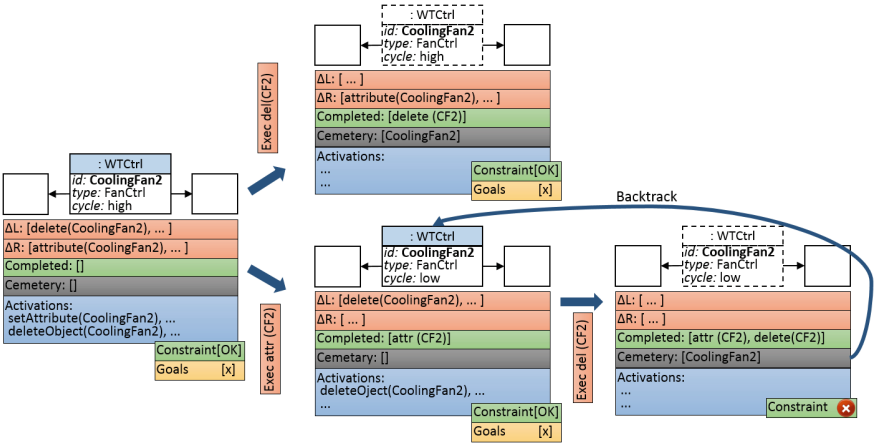


Fig. 4. Conflict resolution with incrementally tracking constraints and operations

- Itr. 1-2: all *must* changes are available and the algorithm randomly picked the createObject of CoolingPump and PumpActivator.
- Itr. 3: at this point only two conflicting transitions have activation; the algorithm picked deleteObject for CoolingFan2 non-deterministically. This leads to a state where the precondition of setAttribute operation cannot be satisfied any longer, thus it is disabled.
- Itr. 4-5: only *may* operations have activation where a setAttribute operation is selected that set the cycle attribute of CoolingFan1 to normal. Because of the generic constraint, the other setAttribute related to the same object (CoolingFan1) is disabled. The same happens when executing deleteObject for CoolingTempLimit2 that disables the setReference operation which should connect CoolingPump and CoolingTempLimit2.
- Itr. 6: this (aggregated) step is composed of all iterations that execution of operation setAttribute related to the newly created CoolingPump.
- Itr. 7: on this trajectory, deletion of CoolingFan2 leads the model into a state where the Fan2Activator output is not referenced by any control unit. Thus our domain-specific (composite) operation (unreferencedPart) has an activation that is executed on the model. After this iteration, there are no more activations and all goals are satisfied, so *Solution #1* is found.
- Itr. 8: after the solution, the strategy backtracks until it finds an activation for a *must* operation that should lead the model into a partially traversed state and forks the trajectory. Only the setAttribute operation related to *CoolingFan2* can be executed. After the execution, *deleteObject* of CoolingFan2 could have activation, but it is disabled by the generic constraint.
- Itr. 9-11: The same activations are available as for the 4th iteration except the domain-specific operation. The algorithm randomly executes these operations and finds *Solution #2*.

Resolved Conflicts. In iteration 3 and 8, two conflicting operations marked with *must* are executed which forks the exploration into two separate solutions to resolve the conflicts. At iterations of 4 and 9, two operations with *may* mark are in conflict. In each trajectory, only one of them is selected. Similar happens in iteration 5 and 10, but this time the same operation is selected in each branch.

Solution. There are two solutions in the output of the merge process. We discuss solution #1 in details where the merged model is depicted in Fig. 6. It also displays in dashed line the deleted objects stored in *Cemetery*, namely, CoolingTempLimit2, CoolingFan2 and Fan2Activator. There are four non-executed changes as shown in the bottom left corner of Fig. 6.

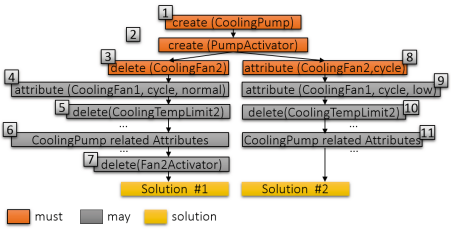


Fig. 5. Possible execution of the process

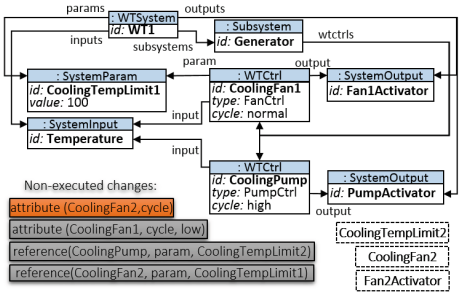


Fig. 6. Merged Model from Solution #1

5 Evaluation

As the state-of-the-art of model merge still lacks well-accepted benchmarks to measure *scalability* of model merging components (e.g. [22] measures precision and recall), we propose a new scalability benchmark for model merge by adapting of the Train Benchmark [29], which is an existing performance benchmark for model queries and well-formedness constraints (and also a case of the TTC 2015 contest [28]). The benchmark uses a domain-specific model of a railway system originating from the MOGENTES project [4]. From the existing benchmark, we reuse (1) the *model generator* to derive models of different size conforming to a railway metamodel, (2) the *fault injector* which changes the generated model (e.g. by changing structural features, and creating or deleting objects) to violate predefined well-formedness constraints, and (3) *repair actions* which pseudo-randomly resolve such violations in accordance with to a random seed value.

Based upon these components, we summarize how synthetic models are generated that contain conflicts serving as input for model comparison and model merge: (1) First, we generate a well-formed model. (2) Next, we inject several

faults into the generated model. The result of this phase acts as *original (O)* model. (3) Then, local and remote changes are simulated by repairing these violations either in the local model *(L)* or remote model *(R)* or in both of them with different random seeds. In the latter case, the framework repairs the same problems in both cases by using different values, which leads to a conflict between two models. (4) We calculate the differences between the two with an existing comparison tool (EMF Compare). (5) Finally, these two model have to be merged with *may* annotations for changes using our merge tool.

We evaluate our DSE-based automated merge approach to assess its scalability using our benchmark where we investigate the scalability of the approach by measuring execution time for model comparison (carried out by EMF Compare) and model merge with respect to (i) the size of models, (ii) the size of change set, and (iii) the number of changes in conflict. For the evaluation, we generated models where the number of model elements is from 10,000 to 350,000, the number of faults injected into the models (i.e. size of the change set) is from 10 to 2000 while the number of conflicts are set to 0 %, 50 % and 100 % of the total number of changes. Measurement results are summarized in Table 2 taking the average of 5 separate runs.

Table 2. Scalability measurement results

Size	Δ	Diff (sec)	Merge (sec) 0% conflict	50% conflict	100% conflict
11710	120	4.672	1.265	2.095	3.477
	240	7.329	2.241	3.345	4.109
	480	12.951	3.923	4.650	8.813
	960	23.323	8.853	12.008	21.842
	1920	26.368	11.352	19.766	29.948
23180	120	7.233	2.686	2.924	6.262
	240	7.569	4.355	5.106	8.596
	480	13.695	9.433	14.127	17.796
	960	23.383	18.219	22.474	40.589
	1920	41.857	34.181	57.207	96.806
46728	120	17.258	6.679	8.156	12.567
	240	18.592	10.625	12.623	20.047
	480	27.410	19.063	24.210	39.855
	960	40.915	37.961	51.924	90.295
	1920	69.344	165.203	180.534	217.343

Size	Δ	Diff (sec)	Merge (sec) 0% conflict	50% conflict	100% conflict
87396	120	28.302	10.654	13.556	22.913
	240	30.711	20.285	24.377	37.501
	480	36.378	38.154	48.655	76.703
	960	49.382	75.567	92.797	153.234
	1920	80.934	162.845	205.423	367.357
175754	120	59.236	21.332	27.699	43.492
	240	79.068	42.308	50.843	79.492
	480	93.395	80.130	95.332	162.106
	960	97.313	157.720	185.030	279.367
	1920	118.439	311.525	362.841	626.946
354762	120	176.200	47.410	57.695	89.101
	240	177.280	84.678	104.739	166.990
	480	188.028	156.568	198.307	317.629
	960	209.440	307.878	406.879	636.156
	1920	257.355	1,342.081	1,401.882	1,535.091

Analysis of Results. As expected, merge time is linear in model and change size, and also proportional to comparison time. Furthermore, fewer conflicts imply faster merge time. Our results also show that runtime of merge is lower than compare time in case of smaller change sets (120, 240), and gradually outgrows it as the change set increases. However, change sets of an average commit in real projects are even smaller than our smallest case (see also the evaluation in [23]), which means that our scalability results represent a pessimistic setup.

6 Related Work

Several approaches address the model merge as depicted in Table 3. To position them against our approach, we use several characteristics proposed in a survey on model versioning [6], which also guides the structure of this section.

Table 3. Comparison of model merge approaches

	Basis	Conflict detection	Merge automation	Merge operations	Objectives	Guidance	Evaluation
EMF Compare [1]	state	static	semi	generic	-	-	scalability
EMF Diff/Merge [2]	state	static	semi	generic	-	-	scalability
Westfachtel [30]	state	runtime	semi	generic	goals	-	preliminary
N-way Merge [25]	state	static	semi	generic	-	-	preliminary
AMOR [13]	state	static	semi	generic, composite	goals	-	precision recall
Dam H.K. et al. [14]	state	static	auto	composite	goals, constraints	repair plan	scalability (closed)
MOMM [23]	operation	runtime	auto	composite	fixed goals	global search prioritized	real data
DSE Merge	state	runtime	auto	generic, composite	goals contraints	local search may/must	scalability (open)

Comparison Basis. Based on the model comparison technique, the approaches may be classified into *state-based* and *operation-based*. [1,2,13,14,25,30] and DSE Merge are *state-based* as they execute a comparison process between model states. However, [23] uses operations as input where even more complex operations as just the simple add, update, and delete operations are considered.

Conflict Detection. Finding the conflicting changes in the merge process is crucial task for a correct resolution. Most approaches use an initial phase to statically analyze the changes and look for conflicting pairs such as in [1,2,13,14,25]. Westfechtel [30] defines transformation rules for searching conflicts where the satisfied preconditions selects the conflicts in each iteration. Mansoor et al. [23] uses conflict detection algorithm between operations [12]. DSE Merge identifies conflicts incrementally as violations of constraints or as deactivations of merge operations, while dependencies between rules and constraints are handled automatically by the underlying DSE engine. This extends [14] where inconsistency constraints are handled incrementally while conflict detection happens as preprocessing.

Merge Automation. Most approaches [1,2,13,25,30] are semi-automated as they use a two-phase process: (i) they apply the non-conflicting operations and then (ii) let the user prioritize and select the operation to apply in case of two conflicting changes. This always results in a single solution due to the manual resolution by the user. In comparison, [14,23] and DSE Merge resolve the conflicts automatically in different ways and offer several solutions.

Merge Operations. In this context, merge operations are responsible for applying the changes in the merged model. [1,2,25,30] use generic operations for changes. The extension [11] of [30] adaptively learns resolution patterns from user that can be applied on the models which results in composite operations. [23] applies the input operations which are composite refactorings in their case. [14] uses basic generic operators for conflicts but generates composite operations as repair plans from the description of inconsistency constraints. Our DSE Merge approach allows to combine both generic and domain-specific composite operators in the form of change-driven transformation rules.

Objectives. Quality of the merge model can be improved by objectives that have to be satisfied during (*contraints*) or at the end (*goals*) of the merge process. This is an unsupported feature in [1,2,25]. [23] uses two fixed goals which are the base of the conflict resolution. [14] provides support for incrementally detecting violations of inconsistency constraints. [13] is connected to an additional model checker component [11] which allows to check OCL constraints as goals. [30] allows to define well-formedness constraints in OCL that act as goals. DSE Merge let the users to provide additional constraints and goals using graph patterns in addition to a built-in termination condition when no operations are activated.

Guidance. The execution of the merge process can use guidance to find the solution(s) faster. The tool [26] of [30] uses a dedicated fusing algorithm for the model merge phase using a fixed priority strategy of merge operations. [23] bases their tool to a global search genetic algorithm (NSGA-II [15]) where the operations are also prioritized related to their importance. DSE Merge is built on top of the ViatraDSE framework [19] using rule-based guided local search exploration. Furthermore, annotating changes with *may*/*must* can further reduce the result set retrieved to the user, which is another key difference wrt [14,23].

Evaluation. [23] provides an empirical evaluation of the tool based on real data, but its scalability is not discussed as their largest model was the same as our smallest. [14] represents an scalability evaluation of its tool with the largest size of 33.000 model element and 1,650 changes. [25] and [26] show a preliminary evaluation which show the relevance of the approach on very small models and change set. [13] evaluated by [22], but scalability is not discussed. For comparing models, [1] has a scalability test presented in [7]. Scalability of [2] is not well covered, however, we evaluated ourselves on the proposed benchmark [3]. DSE Merge is evaluated on an open scalability benchmark [29]. As future work, we plan to create an empirical user study from the usability aspect of our tool.

Summary. To summarize the key differences with [14] and [23], we rely on state-based comparison, apply a guided local-search strategy (vs. [23]), detect conflicts at runtime and allow complex generic merge operations (vs. [14]). Internally, we uniquely use incremental and change-driven transformations to derive the merged models. Finally, we report scalability of merge process for models which are at least one order of magnitude larger compared to [14] and [23].

7 Conclusion

The current paper presented an automated technique for three-way model merge exploiting design space exploration in the background. The original model and two difference models (original model↔remote version, and original model↔local version) calculated with existing model comparison tools (e.g. EMF Compare or Diff/Merge) serve as an input of our technique. Our technique automatically derives consistent and semantically correct merged models in all possible ways and also highlights the remaining (unresolved thus conflicting) model differences. Our approach incorporates the use of change-driven model

transformations [9] to capture and execute merge operations, and relies on an incremental reactive model transformation engine [8] to detect and resolve merge conflicts. We proposed scalability benchmark for scalability aspect of merge components that demonstrates that DSE-based model merge can be executed for models around 350,000 elements and conflicting change sets with 1000 elements.

Our approach is fully implemented in a tool developed as part of a European project, which operates on well-known open source components of the Eclipse framework, such as EMF Compare [1] or Diff/Merge for [2] for model comparison and using the Viatra DSE [18,19] as underlying design space exploration framework built on reactive transformations [8].

As future work, we plan to improve our model merge technique by further search strategies to better exploit the dependencies between rules and constraints and compare it with other search-based merge techniques [23]. Currently, we are conducting an experimental user evaluation to compare the usability of the presented DSE Merge tool with EMF-Compare and Diff/Merge.

Acknowledgments. We thank to Gábor Szárnyas for improving the syntectic performance benchmark for the evaluation and András Szabolcs Nagy for his assistance on design space exploration.

References

1. EMF compare. https://www.eclipse.org/emf/compare/
2. EMF Diff/Merge. http://eclipse.org/diffmerge/
3. Evaluation of EMF Compare and Diff/Merge. https://github.com/FTSRG/publication-pages/wiki/Evaluation-of-EMF-Diff-Merge-and-EMF-Compare
4. Mogentes EU project. http://www.mogentes.eu/
5. Altmanninger, K., Kappel, G., Kusel, A., Retschitzegger, W., Seidl, M., Schwinger, W., Wimmer, M.: AMOR-towards adaptable model versioning. In: 1st International Workshop on Model Co-Evolution and Consistency Management, in conjunction with MODELS, vol. 8, pp. 4–50 (2008)
6. Altmanninger, K., Seidl, M., Wimmer, M.: A survey on model versioning approaches. IJWIS **5**(3), 271–304 (2009)
7. Barbero, M.: EMF compare 2.0: scaling to millions. In: EclipseCON 2013, Boston
8. Bergmann, G., Dávid, I., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., Varró, D.: Viatra 3: a reactive model transformation platform. In: Kolovos, D., Wimmer, M. (eds.) ICMT 2015. LNCS, vol. 9152, pp. 101–110. Springer, Heidelberg (2015)
9. Bergmann, G., Ráth, I., Varró, G., Varró, D.: Change-driven model transformations - change (in) the rule to rule the change. Softw. Syst. Model. **11**(3), 431–461 (2012)
10. Bergmann, G., Ujhelyi, Z., Ráth, I., Varró, D.: A graph query language for EMF models. In: Cabot, J., Visser, E. (eds.) ICMT 2011. LNCS, vol. 6707, pp. 167–182. Springer, Heidelberg (2011)
11. Brosch, P., Egly, U., Gabmeyer, S., Kappel, G., Seidl, M., Tompits, H., Widl, M., Wimmer, M.: Towards semantics-aware merge support in optimistic model versioning. In: Kienzle, J. (ed.) MODELS 2011 Workshops. LNCS, vol. 7167, pp. 246–256. Springer, Heidelberg (2012)

12. Brosch, P., Kappel, G., Langer, P., Seidl, M., Wieland, K., Wimmer, M.: An introduction to model versioning. In: Bernardo, M., Cortellessa, V., Pierantonio, A. (eds.) SFM 2012. LNCS, vol. 7320, pp. 336–398. Springer, Heidelberg (2012)
13. Brosch, P., Seidl, M., Wieland, K., Wimmer, M.: We can work it out: collaborative conflict resolution in model versioning. In: Wagner, I., Tellioğlu, H., Balka, E., Simone, C., Ciolfi, L. (eds.) ECSCW 2009, pp. 207–214. Springer, London (2009)
14. Dam, H.K., Reder, A., Egyed, A.: Inconsistency resolution in merging versions of architectural models. In: 2014 IEEE/IFIP Conference on Software Architecture, WICSA 2014, Sydney, Australia, pp. 153–162, 7–11 April 2014
15. Deb, K., Agrawal, S., Pratap, A., Meyarivan, T.: A fast and elitist multiobjective genetic algorithm: NSGA-II. IEEE Trans. Evol. Comput. **6**(2), 182–197 (2002)
16. Ehrig, H., Kreowski, H.J., Rozenberg, G.: Handbook of Graph Grammars and Computing by Graph Transformation, vol. 2. World Scientific, Singapore (1999)
17. Feather, M.S.: Detecting interference when merging specification evolutions. In: ACM SIGSOFT Software Engineering Notes. vol. 14, pp. 169–176. ACM (1989)
18. Hegedus, A., Horváth, A., Ráth, I., Varró, D.: A model-driven framework for guided design space exploration. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 173–182. IEEE Computer Society (2011)
19. Hegedüs, Á., Horváth, Á., Varró, D.: A model-driven framework for guided design space exploration. Autom. Softw. Eng. **22**(3), 399–436 (2015)
20. Kessentini, M., Werda, W., Langer, P., Wimmer, M.: Search-based model merging. In: Genetic and Evolutionary Computation Conference, GECCO 2013, Amsterdam, The Netherlands, pp. 1453–1460, 6–10 July 2013
21. Kolovos, D.S., Rose, L.M., Matragkas, N., Paige, R.F., Guerra, E., Cuadrado, J.S., De Lara, J., Ráth, I., Varró, D., Tisi, M., et al.: A research roadmap towards achieving scalability in model driven engineering. In: Proceedings of the Workshop on Scalability in Model Driven Engineering, p. 2. ACM (2013)
22. Langer, P., Wimmer, M.: A benchmark for conflict detection components of model versioning systems, vol. 33 (2013)
23. Mansoor, U., Kessentini, M., Langer, P., Wimmer, M., Bechikh, S., Deb, K.: MOMM: multi-objective model merging. J. Syst. Softw. **103**, 423–439 (2015)
24. Mens, T.: A state-of-the-art survey on software merging. IEEE Trans. Softw. Eng. **28**(5), 449–462 (2002)
25. Rubin, J., Chechik, M.: N-way model merging. In: Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE 2013, Saint Petersburg, Russian Federation, pp. 301–311, 18–26 August 2013
26. Schwägerl, F., Uhrig, S., Westfechtel, B.: Model-based tool support for consistent three-way merging of EMF models. In: Proceedings of the workshop on ACadeMics Tooling with Eclipse, p. 2. ACM (2013)
27. Steyaert, P., Lucas, C., Mens, K., D'Hondt, T.: Reuse contracts: managing the evolution of reusable assets. In: Proceedings of the 1996 ACM SIGPLAN Conference on Object-Oriented Programming Systems, Languages & Applications (OOPSLA 1996), San Jose, California, pp. 268–285, 6–10 October 1996
28. Szárnyas, G., Semeráth, O., Ráth, I., Varró, D.: The TTC 2015 train benchmark case for incremental model validation. In: Transformation Tool Contest, pp. 129–141 (2015)

29. Ujhelyi, Z., Bergmann, G., Hegedüs, Á., Horváth, Á., Izsó, B., Ráth, I., Szatmári, Z., Varró, D.: EMF-IncQuery: an integrated development environment for live model queries. Sci. Comput. Program. **98**, 80–99 (2015)
30. Westfechtel, B.: Merging of EMF models - formal foundations. Softw. Syst. Model. **13**(2), 757–788 (2014)
31. Wieland, K., Langer, P., Seidl, M., Wimmer, M., Kappel, G.: Turning conflicts into collaboration. Comput. Support. Coop. Work **22**(2–3), 181–240 (2013)