VIATRA 3: A Reactive Model Transformation Platform[⋆]

Gábor Bergmann[1], István Dávid[3], Ábel Hegedüs[2], Ákos Horváth[1,2], István Ráth[1,2], Zoltán Ujhelyi[2] and Dániel Varró[1]

[1] Budapest University of Technology and Economics,
Department of Measurement and Information Systems,
1117 Budapest, Magyar tudósok krt. 2.
{bergmann,varro}@mit.bme.hu
[2] IncQuery Labs Ltd.
{hegedus, horvath, rath, ujhelyi}@incquerylabs.com
[3] University of Antwerp (Modelling, Simulation and Design Lab)
Middelheimlaan 1, 2020 Antwerp, Belgium
istvan.david@uantwerpen.be

Abstract. Model-driven tools frequently rely on advanced technologies to support model queries, view maintenance, design rule validation, model transformations or design space exploration. Some of these features are initiated explicitly by domain engineers (batch execution) while others are executed automatically when certain trigger events are detected (live execution). Unfortunately, their integration into a complex industrial modeling environment is difficult due to hidden interference and unspecified interaction between different features. In this paper, we present a reactive, event-driven model transformation platform over EMF models, which captures tool features as model queries and transformations, and provides a systematic, well-founded integration between a variety of such tool features. VIATRA 3 offers a family of internal DSLs (i.e. dedicated libraries) to specify advanced tool features built on top of existing languages like EMF-INCQUERY and Xtend. Its main innovation is a source incremental execution scheme built on the reactive programming paradigm ssupported by an event-driven virtual machine.

Keywords: event-driven transformation, virtual machine, reactive programming, source incremental transformations

1 Introduction

With the increasing adoption of model-driven engineering in critical systems development, the increasing complexity of development processes and modeling artefacts poses new challenges for tool developers, especially in collaboration and scalability. Nowadays, such challenges are typically addressed with dedicated problem-specific solutions such as on-the-fly constraint evaluation engines [1,2]

⋆ This work was partially supported by the MONDO (EU ICT-611125) project.

(to improve the scalability of model validation), incremental model transformation tools [3] for scalable model synchronization, or design space exploration tools [4] (to synthesize optimal models wrt some objectives). Some of these scenarios are initiated explicitly by domain engineers (*batch execution*) while others are executed automatically upon certain trigger events (*live execution*).

Unfortunately, integrating different technologies into a complex industrial modeling environment is often difficult and costly. This is due to hidden interference and unspecified interaction between different tool features. For instance, a notification originating from a model change may easily trigger conflicting actions in different plugins. As a consequence, complex tool platforms such as the Eclipse Modeling Framework (EMF) [5] are known to suffer from severe performance and quality issues caused e.g. by the concurrent asynchronous execution of various model indexing and validation mechanisms.

In this paper, we present a *source incremental event-driven model transformation platform based on the reactive programming paradigm* [6] to drive the systematic, well-founded integration of tool features in various scenarios over EMF models. The VIATRA 3 *Event-driven Virtual Machine* (EVM) provides basic executional building blocks and primitives with clearly defined event-based semantics. EVM also enables to combine various advanced tool features so that complex interactions can be constructed easily and executed consistently.

VIATRA 3 offers a family of internal DSLs (i.e. dedicated libraries and APIs) built on top of existing languages to specify advanced tool features as model queries and transformations. The EMF-INCQUERY language is seamlessly integrated to capture any conditions and constraints for a transformation. Furthermore, Java and Xtend-based internal DSLs (APIs) are used to specify transformations rules as well as complex interactions between different tool features.

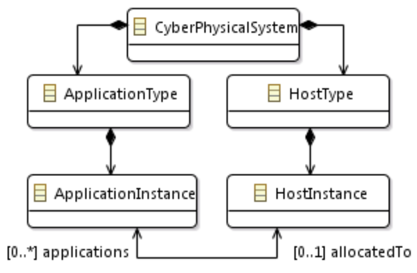
While VIATRA 3 is designed to support a wide spectrum of tooling scenarios, our case study focuses on a typical scenario including incremental deployment to present challenges that arise in the interaction between *batch and live model transformations*. The aim of the example is to illustrate to what an extent the integration complexity is reduced by capturing and handling all tool features and their interactions based on a uniform event-driven virtual machine.

The rest of the paper is structured as follows: first, we overview modeling scenarios that motivate the development of the generalized virtual machine architecture and introduce the case study in Sec. 2. Then we present our virtual machine for reactive event-driven transformations (Sec. 3). Related work is discussed in Sec. 4 and Sec. 5 concludes the paper.

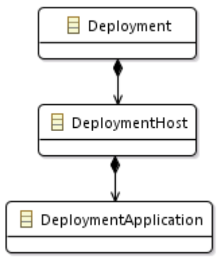
2 Motivating Example

In our motivating example[4], we investigate batch and incremental model-to-model transformations. The source domain describes a generic infrastructure for cyber-physical systems (CPS) where applications (services) are dynamically

[4] The complete source code, documentation and performance evaluation results are available from `https://github.com/IncQueryLabs/incquery-examples-cps`



(a) Hosts and Applications of the CPS.



(b) Deployed Hosts and Applications.

Fig. 1: Source and target models

allocated to connected hosts. The target domain represents the system deployment configuration with stateful applications deployed on hosts. Initially, we aim to derive a deployment model from the CPS model, and then incremental model transformations are used to propagate changes in the CPS model to the deployment model and the traceability model.

Metamodel Due to space considerations, we present a limited fragment of the metamodel in Figure 1, the description of the domain is adopted from [4]. The simplified CPS source model (Figure 1a) contains *HostInstance*s and *ApplicationInstance*s, typed by *HostType*s and *ApplicationType*s, respectively. *ApplicationInstance*s are allocated to a *HostInstance*. In the Deployment model (Figure 1b), *DeploymentHost*s and *DeploymentApplication*s are derived from their instance counterparts in the CPS model, respectively; and the hosts are associated with the hosted applications. Finally, the mappings between the two domains are persisted in a traceability model.

Scenarios In the original case study, we had to provide integrated tooling to cover the following use cases:

1. **Batch transformations** are used to map *HostInstance*s of a given *HostType* to a *DeploymentHost* (the mapping is stored in an explicit trace model);
2. **Live transformations** are used to automatically map *ApplicationInstances* to *DeploymentApplication*s in an event-driven way (i.e. fired upon changes to the source model to keep the target and trace models consistent).
3. **On-the-fly validation** is continuously performed (i.e. before and after model synchronization) to ensure the correctness of the mapping.

Due to (data and control) dependencies, model synchronization phases should only be initialized once the batch transformations have completely terminated and when the (source) model is free of errors as indicated by validation results. In

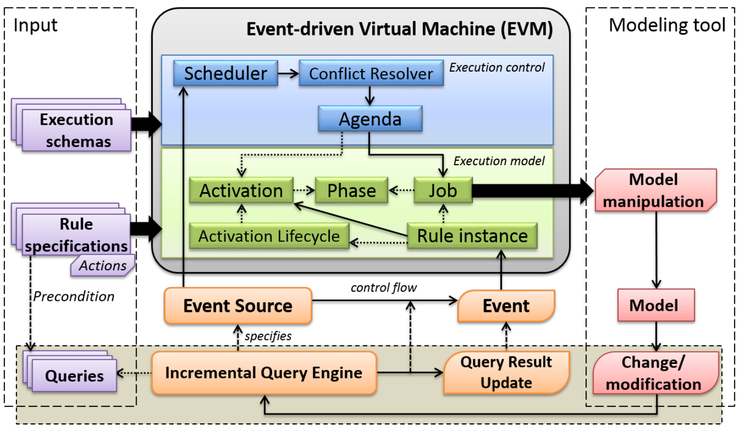


Fig. 2: Architecture of the EVM for model transformations

a traditional MDE toolchain, separate tool features would be used to describe the various phases, requiring an external orchestrator to facilitate the coordination. Complex features in real MDE tools (like model indexing or file operations) add further complexity to the integration of tool features. The current paper presents how an event-driven virtual machine can reduce such complexity.

3 An Event-driven Virtual Machine (EVM)

Event-driven model transformations are executed continuously as *reactions* to changes of the underlying model. To facilitate this sort of execution, we adopted reactive programming principles. The core concept of reactive programming is the *event-driven behavior*: components are connected to event sources and their behavior is determined by the event instances observed on *event streams*. Compared to sequential programming, the benefits of reactive programming are remarkable especially in cases when continuous interaction with the environment has to be maintained by the application based on external events without a priori knowledge on their sequence [6].

Figure 2 presents the architecture of the Event-driven Virtual Machine (EVM), the novel execution engine of the VIATRA 3 platform[5]. Although this paper demonstrates its use in model transformation scenarios, EVM is an engine for executing reactive programs in general.

The *specification* of an EVM program consists of two parts. First, the *Rule specifications* are defined as *Queries* over a given *Model*(s) and serve as a precondition to the transformation. Second, the *Actions* to be executed are specified, which in this case are *Model manipulations* over the input models. Furthermore, *Execution schemas* are defined in order to orchestrate the reactive behavior.

[5] `http://wiki.eclipse.org/EMFIncQuery/DeveloperDocumentation/EventDrivenVM` contains the complete technical documentation.

Now we briefly describe the behavior of other core components of Figure 2 in the sequel.

3.1 Events

In batch transformation scenarios[6], the sequence of executing actions associated with a batch transformation is usually determined solely by the activations initiated from the transformation program. However, the core features of EVM enable reactions to events. We distinguish between two kinds of events.

- *Controlled* events are initiated explicitly by the transformation program, and involve the *firing* of a selected rule with concrete values substituted for its parameters. Thus a controlled event is characterized by a rule activation.
- *Observed* events are caused by external behavior, and the time of their occurrence may not be determined by the transformation program. Such observed events include elementary model notifications and updated results of model queries. However, more complex ways of detecting changes in the model (see change patterns [7]) or aggregating temporal behavior of changes in the past (see complex event processing [8]) are also possible over the EVM platform.

3.2 Activation Lifecycles

Listing 1 presents an event-driven transformation to keep already mapped *ApplicationInstances* of the CPS model in sync with their *DeploymentApplication* counterpart in the Deployment model.

The actions of event-driven transformations (in Lines 10-12, 14-24 and 26-28) are associated with a specific *events* reflecting the current state of the activation. As opposed to simple batch transformations, these events suggest that in addition to executing an action on the *appearance* of an activation, *updates* and the *disappearance* of the same activation might be also relevant from transformation point of view and can also serve as triggers for executing actions.

Events reflecting the current state of the activation constitute a transition system called the *Activation Lifecycle* (Line 8), serving as the centerpiece of the reactive paradigms supported by EVM. An Activation Lifecycle consists of of different (1) *Phase*s (see Figure 2) an *Activation* can be associated with during its existence; and (2) event-triggered transitions between the Phases. Optionally, (3) a transition may be associated with a *Job*, which represents the executable *Actions* of an input rule specification. Figure 3 presents two typical Activation Lifecycles.

Figure 3a illustrates the lifecycle of an event-driven transformation rule. Apart from the initial phase, we distinguish between enabled and disabled phases depending on the presence or absence of a *Fire* transition. Event-driven transformations define executable actions for enabled states of the lifecycle. If an

[6] `https://github.com/IncQueryLabs/incquery-examples-cps/wiki/Alternative-transformation-methods#Batch`

Listing 1 Event-driven transformation rule for maintaining *ApplicationInstances*

```
//finds every every transformed and allocated deploymentApp
pattern mappedApplicationInstance(
  appInstance, deploymentApp, hostInstance, deploymentHost) {...}
---------------------------------------------------------------------------
CPSToDeployment mapping //reference to the mapping model
val applicationUpdateRule = createRule().name("application update")
  .precondition(mappedApplicationInstance) //a graph pattern as precondition
  .lifeCycle(ActivationLifecycles.default)
  //action to be executed when a pattern match appears
  .action(ActivationStates.APPEARED) [
    debug("Starting monitoring mapped application with ID: " + appInstance.id)
  ]
  //action to be executed when a pattern match gets updated
  .action(ActivationStates.UPDATED) [
    debug("Updating application with ID: " + appInstance.id)
    //case 1: ID changed
    if (appInstance.id != deploymentApp.id) {
      deploymentApp.set(id, appInstance.id)
    }
    //case 2: host changed
    if (!deploymentHost.applications.contains(deploymentApp)) {
      deploymentHost.set(deploymentHost_Applications, deploymentApp)
    }
  ]
  //action to be executed when a pattern match disappears
  .action(ActivationStates.DISAPPEARED) [
    debug("Stopped monitoring mapped application with ID: " + appInstance.id)
  ].build
```

activation enters that specific phase, it may fire and upon the transition, the associated job (defined by the action in the transformation rule) gets executed.

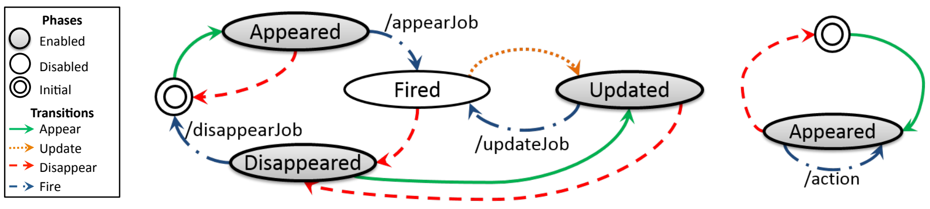
For example, the first time a match of the *MappedApplicationInstance* model query is found, an activation of the rule will occur in the *APPEARED* state. If the EVM fires that activation, the *appearJob* will be executed.

To unify the behavior of model transformations over the EVM platform, both event-driven and batch transformations are executed as reactive programs (using the the activation lifecycle of Figure 3b for the batch case). The enabled phases of an activation lifecycle represent outstanding reactions to *observed* events, but the firing of the actual reactive jobs is tied to *controlled* events.

3.3 Scheduler

External observed events influence activation phases according to the lifecycle, and the active phase selects the job to be executed (if any). However, it is the chosen *Scheduler* component that determines when EVM can fire these controlled events (i.e. execute the jobs).

Practical examples for the scheduling event include (1) the signal of the query engine indicating that the updating of query results after an elementary model manipulation has concluded; (2) the successful commit of a model editing transaction; or (3) some combination of the former events with a timer. The choice of scheduling event has to take into account the following factors:



(a) Event-driven Transformation

(b) Batch transformation

Fig. 3: Typical rule lifecycles

– The rules may require a certain level of consistency in the model, e.g. some rules are inefficient to execute while a large-scale transaction is incomplete;
– Otherwise, the rules should be executed as soon as possible thus their effects are observed by the user or by further transformation steps.

Event driven transformation rules may also explicitly invoke other rules, which is a direct rule dependency. However, indirect rule dependency may also exist when model manipulation in a job causes observed changes which, in turn, enable activations and trigger the scheduler.

3.4 Agenda

The *Agenda* stores the current phases (states) of all activations of each rule. Its role is dual: it helps maintain the phase of activations in reaction to events, and it supplies the set of rule activations being in an enabled phase, i.e. activations that can be fired. The core operation of EVM is intrinsically tied to the agenda: in case of an observed or controlled event, the rule activation corresponding to the specific event will change phase according to the transition in the lifecycle model defined for the rule that starts at the current phase and is labeled with the event type token. Afterwards, if there is a job associated with the transition, it is invoked with the activation providing the input parameters.

As the set of all possible activations is practically infinite (as each rule parameter may point to any memory address), the implementation considers only those activations that are currently not in their initial phase. This makes the agenda finitely representable, since a finite number of events may have moved only a finite number of activations out of their initial phases.

3.5 Conflict resolution

At any point in time, the rules (or a selected subset of rules in a common case of batch transformations) might have multiple activations in an enabled state, which is called a *conflict*. If the transformation is to invoke a rule firing, a single enabled activation has to be selected from the conflicting ones (mainly due to

Algorithm 1 The execution algorithm of the case study

```
PROCEDURE Agenda.processEvent(Event $e$)                          ▷ processes a single event
1: for all RuleInstance $ri$ if triggered by event $e$ do
2:     $act$ := $ri$.activationForEvent($e$)   ▷ creates activation if unstored (i.e. in the initial phase)
3:     agenda.updatePhase($act$, $e$)                      ▷ updates activation based on event $e$
4: end for

PROCEDURE Scheduler.main()                                       ▷ drives the reaction to events
5: Agenda.executeActivations()                    ▷ execute enabled activation based on the CR

PROCEDURE Agenda.executeActivations()                            ▷ executes all activations
6: while (ConflictResolver.hasNextActivation()) do
7:     $act$ := ConflictResolver.nextActivation()                   ▷ gets next activation
8:     Executor.fire($act$)                                          ▷ fires the activation
9: end while
```

the single threaded manipulation of EMF models). This selection can be done manually by the transformation code, but EVM also provides an automated mechanism that delegates this decision to a user-specified *Conflict Resolver* (see Figure 2). Built-in conflict resolvers include FIFO, LIFO, fair random choice, rule priority (with a secondary conflict resolver within priority levels), interactive choice (e.g. on the user interface), but it is possible to implement arbitrary conflict resolution strategies.

3.6 Execution

The execution of event-driven transformations is handled by the EVM. We present the process by walking through the execution Algorithm 1 using an example scenario. The scenario presents an update of ApplicationInstance *a1* where its associated HostInstance is replaced.

Step 1 The HostInstance reference of ApplicationInstance *a1* is removed. The matched precondition of the rule generates an *updates* event.

Step 2 In the EVM, the *Agenda* processes the event. The activation of the transformation rule is updated (Line 3).

Step 3 When the notifications are processed, the *Scheduler* initiates the rule execution by notifying the *Agenda* (Line 5). The *Agenda* attempts to acquire the next transformation activation from the ConflictResolver (Line 7) and fire it (Line 8).

4 Related work

Virtual machines for model queries and transformations. The ATL virtual machine was the first to provide execution primitives on which different transformation languages (like, ATL and QVT) can be executed. Recently introduced new features include lazy evaluation [9], incremental [10] combined into the ReactiveATL transformation engine. As a main conceptual difference, this approach is target incremental, i.e. a transformation is executed only when its result is needed — unlike our source incremental virtual machine.

EMFTVM [11] is an execution engine for EMF models that provides both a very low-level language and execution primitives to have a more simple compiler architecture. Similarly, T-Core [12] is based on the same concept for providing an execution engine for graph transformation rules. Their main advantage is that they provide better performance for the low-level primitives and optimization capabilities in case of translation from high-level languages. Model transformation chains [13] aim at composing different transformations proposing a loosely coupled integration between existing transformation steps.

Our virtual machine is unique in combining different best practices: (1) it provides tight integration between many heterogeneous tool features (queries, transformations, validation, exploration, etc.) built upon (2) a source-incremental reactive event-driven paradigm to provide well-founded integration.

Event-driven techniques and model transformations. Event-driven techniques have been used in many fields. In relational database management systems, even the concept of triggers can be considered as simple operations whose execution is initiated by events, which have been utilized for event-driven model transformation purposes previously [14]. These approaches provided the basics for event-driven model transformation techniques.

Our approach presented in the this paper can be regarded as a foundation for previous work on incremental well-formedness validation [1], live and change-driven transformations [7], design space exploration [4] and streaming model transformations [8]. Despite not having been published previously, EVM has been a hidden component of EMF-INCQUERY since 2013, and has already proven to be an efficient execution platform for incremental transformations [15].

5 Conclusion

In this paper, we have proposed a novel execution infrastructure for model processing chains, based on an event-driven reactive virtual machine architecture. Its primary design principle is *flexibility*: through the customizability of rules and execution strategies, it can provide the foundations to a wide range of applications, it supports both stateless and stateful systems, and its internal DSLs (based on Xtend and Java) provide a uniform integration platform for complex model processing programs. As we have shown through the case study, VIATRA 3 is capable of unifying several previously separated advanced modeling aspects into an integrated system, which can address challenging issues such conflict management.

As a main direction for future development, we plan to externalize the DSLs into a family of extensible, yet easy-to-use languages.

Acknowledgements. The authors wish to thank all the contributors of the VIATRA 3 and EMF-INCQUERY projects, in particular Tamás Szabó.

References

1. 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. Science of Computer Programming **98** (02/2015 2015)
2. Willink, E.D.: An extensible OCL virtual machine and code generator. In: Proceedings of the 12th Workshop on OCL and Textual Modelling, ACM (2012) 13–18
3. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling (SoSyM) **8**(1) (3 2009)
4. Abdeen, H., Varró, D., Sahraoui, H., Nagy, A.S., Hegedüs, Á., Horváth, Á., Debreceni, C.: Multi-objective optimization in rule-based design space exploration. In: 29th IEEE/ACM International Conference on Automated Software Engineering (ASE 2014), Vasteras, Sweden, IEEE (2014) 289–300
5. The Eclipse Project: Eclipse Modeling Framework. Accessed: 2015-03-27.
6. Bainomugisha, E., Carreton, A.L., Cutsem, T.V., Mostinckx, S., Meuter, W.D.: A survey on reactive programming. ACM Computing Surveys (2012)
7. Bergmann, G., Ráth, I., Varró, G., Varró, D.: Change-driven model transformations. Software and Systems Modeling **11** (2012) 431–461
8. Dávid, I., Ráth, I., Varró, D.: Streaming model transformations by complex event processing. In: ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems, MODELS 2014, Valencia, Spain, Springer (2014)
9. Tisi, M., Martínez, S., Jouault, F., Cabot, J.: Lazy execution of model-to-model transformations. In Whittle, J., Clark, T., Kühne, T., eds.: Model Driven Engineering Languages and Systems. Volume 6981 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 32–46
10. Jouault, F., Tisi, M.: Towards incremental execution of atl transformations. In: Proceedings of the Third International Conference on Theory and Practice of Model Transformations. ICMT'10, Berlin, Heidelberg, Springer-Verlag (2010) 123–137
11. Wagelaar, D., Tisi, M., Cabot, J., Jouault, F.: Towards a general composition semantics for rule-based model transformation. In: 14th Int. Conf. on Model Driven Engineering Languages and Systems. MODELS'11, Springer-Verlag (2011) 623–637
12. Syriani, E., Vangheluwe, H., LaShomb, B.: T-core: a framework for custom-built model transformation engines. Software and Systems Modeling (2013) 1–29
13. Yie, A., Casallas, R., Deridder, D., Wagelaar, D.: Realizing model transformation chain interoperability. Software and System Modeling **11**(1) (2012) 55–75
14. Bergmann, G., Horváth, D., Horváth, Á.: Applying incremental graph transformation to existing models in relational databases. In: 6th Int. Conf. on Graph Transformation. Volume 7562 of LNCS., Springer (2012) 371–385
15. van Pinxten, J., Basten, T.: Motrusca: Interactive model transformation use case repository. In: 7th York Doct. Symp. on Comp. Sci. & Electronics. (2014) 57