Softw Syst Model (2016) 15:609–629
DOI 10.1007/s10270-016-0530-4

EXPERT VOICE

Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework

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

Received: 22 March 2016 / Revised: 21 April 2016 / Accepted: 23 April 2016 / Published online: 12 May 2016

Abstract The current release of VIATRA provides open-source tool support for an event-driven, reactive model transformation engine built on top of highly scalable incremental graph queries for models with millions of elements and advanced features such as rule-based design space exploration complex event processing or model obfuscation. However, the history of the VIATRA model transformation framework dates back to over 16 years. Starting as an early academic research prototype as part of the M.Sc project of the the first author it first evolved into a Prolog-based engine followed by a family of open-source projects which by now matured into a component integrated into various industrial and open-source tools and deployed over multiple technologies. This invited paper briefly overviews the evolution of the VIATRA/IncQuery family by highlighting key features and illustrating main transformation concepts along an open case study influenced by an industrial project.

Keywords Model transformations · Incremental evaluation · Reactive transformations · Graph queries

Communicated by Prof. Rumpe, Bernhard.

✉ Dániel Varró
varro@mit.bme.hu

[1] Department of Measurement and Information Systems, Budapest University of Technology and Economics, Magyar tudósok krt. 2, Budapest 1117, Hungary

[2] MTA-BME Lendület Research Group on Cyber-Physical Systems, Nádor utca 5, Budapest 1051, Hungary

[3] IncQuery Labs Ltd., Bocskai út 77-79, Budapest 1117, Hungary

1 Software tools in systems engineering

Model-driven engineering (MDE) plays an important role in the design of critical embedded and cyber-physical systems in various application domains such as automotive, avionics or telecommunication. MDE tools aim to *simultaneously improve quality and decrease costs by early validation* by highlighting conceptual design flaws well before traditional testing phases in accordance with the correct-by-construction principle. Furthermore, they improve productivity of engineers by automatically synthesizing different design artifacts (source code, configuration tables, test cases, fault trees, etc.) necessitated by certification standards (like DO-178C [117], DO-330 [116] or ISO 26262[78]).

Certain shares in the software tool market of systems engineering are dominated by very few industrial tools (e.g., MATLAB Simulink, Dymola, DOORS, MagicDraw) each of which typically provides advanced support for certain development stages (requirements engineering, simulation, allocation, test generation, etc). To protect their intellectual property rights, these tools are of closed nature, which implies huge tool integration costs for system integrators (such as airframers or car manufacturers). On the other hand, recent initiatives (such as PolarSys, OpenModelica) have started to promote open language standards and the systematic use of open-source software components in tools for critical systems to reduce licensing costs and risks of vendor lock-in.

Certification standards of critical cyber-physical systems require that software tools used for developing such critical system are validated with the same scrutiny as the system under design by *software tool qualification* [87,116], especially, when no further human checking is carried out on the outputs of such tools. Software tool qualification distinguishes between *design tools* which, by definition, may

introduce new errors to the system and *verification tools* which may fail to reveal existing errors of the system [87] or falsely reduce or simplify the verification process itself by automation [116].

Unsurprisingly, software tool qualification is extremely costly due to high algorithmic complexity [97], tightly coupled architecture and unexpected feature interaction of such tools [86]. In fact, many companies rather opt for using tools just as design aids to highlight errors quickly, and then, they carry out the traditional verification and validation process by thorough simulation and testing [32,156]. Anyhow, systematic software engineering techniques to simultaneously improve quality and reduce the costs of software tool qualification would be highly beneficial. Existing software engineering practices may guarantee the quality of the system itself, but they frequently *fail to ensure the quality of the software tool* used in systems engineering [141]. Furthermore, the rapid increase in the size and complexity of systems models introduces significant *scalability challenges* for these tools [84].

Software language engineering aims to provide foundations, techniques and tools for domain-specific modeling languages to capture the models. *Model transformation engineering* aims to systematically develop queries and transformations used in automated code generators, simulators or debuggers to process these models. Of course, seamless *integration* of these techniques is needed when developing industrial tools.

The VIATRA open-source software project provides advanced support for incremental and reactive model transformations [20,141] built on top of incremental graph queries to assist the systematic development of novel tools for systems engineering. This invited paper extends previous papers [20,39,137,141,142,148] to provide an overview on the history and evolution of the VIATRA model transformation framework. It first started as part of the MSc research project of the first author in 1999 [146,147], then evolved into a Prolog-based model transformation engine [39,148] (Sect. 3) followed by an open-source Eclipse project founded in 2005 [142] and used in various European projects (Sect. 4). Since 2010, the VIATRA family includes IncQuery [22,137] which supports the scalable incremental evaluation of graph queries over large models of heterogeneous technological spaces (Sect. 5.2). The current industrial release of VIATRA [20] (Sects. 5.3, 5.4) is a reactive and incremental transformation engine built on top of graph queries following several principles of reactive programming [13] and active databases [106]. We overview the main features of each major release along an open case study influenced by an industrial project (Sect. 2) which uses reactive transformations for model-based deployment with run-time models. We present selected academic and industrial applications of the VIATRA family and summarize related work in a historical context.

2 Motivating example

As a motivating example, we investigate model deployment transformations for dynamic and self-adaptive systems frequently considered in the context of smart cyber-physical systems (CPS) [92]. The *source domain* describes a high-level generic infrastructure where applications (services) are dynamically allocated to connected hosts. The *target domain* represents low-level system deployment configuration with stateful applications deployed on hosts. *Traceability links* between the source and target models are also persisted as models to comply with traceability requirements of CPSs.

The full case study presents a complex challenge including (1) *continuous validation of well-formedness constraints*, (2) a *model synchronization scenario*, i.e., a model-to-model (M2M) transformation from the CPS model to a deployment model and (3) a *code generation scenario* from the deployment model to Java code.

In a real design tool, some of these steps can be addressed by *batch (on-demand) transformations* which are initiated explicitly by the engineer, while others are defined as *live (reactive) transformations* which are triggered automatically by certain changes in the underlying model. Some transformation steps can be *target incremental* which only update or move target model elements instead of regenerating them from scratch. Furthermore, *source incremental* transformation steps traverse or query exactly those source elements which are relevant for change detection and propagation [68,123].

Due to data and control dependencies, the different transformation phases heavily depend upon each other. For instance, continuous validation of constraints has to be suspended, while a transformation is running; otherwise, constraint violations may be unintentionally detected in an incomplete state. In a traditional MDE toolchain, separate tool features (e.g., plugins) 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. Needless to say that such orchestrators are extremely hard to develop and debug.

Metamodels We present simplified fragments of the metamodels in Fig. 1 to provide better focus for our paper. The source (CPS) domain (Fig. 1a) contains classes (nodes) *HostInstance*s and AppInstances, respectively, typed by HostTypes and AppTypes as denoted by the corresponding instances references (edges). AppInstances are allocated to a HostInstance captured by allocatedTo references. In the target (Deployment) domain (Fig. 1b), DeploymentHosts and DeploymentApplications are derived from their counterparts in the CPS model, but hosted applications are deployed directly under their hosts (see reference apps). The

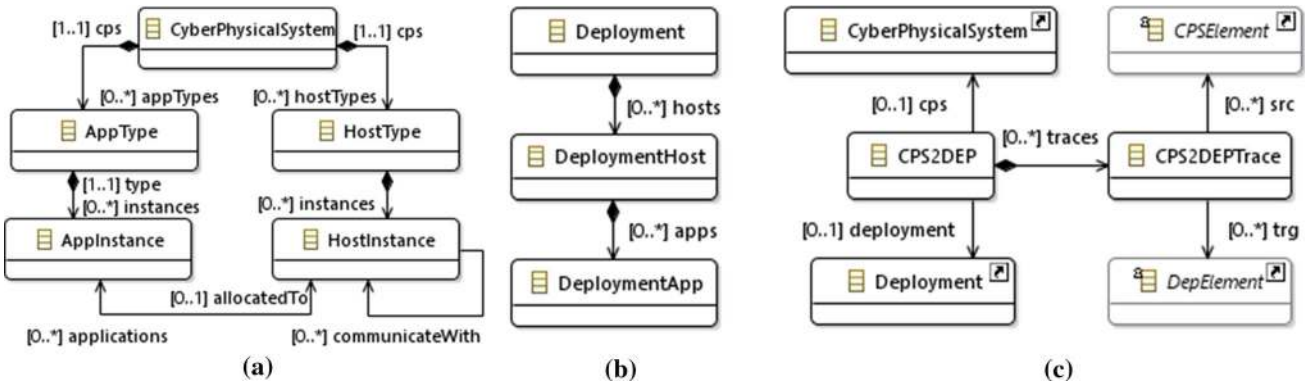

Fig. 1 Metamodels extracts of source, target and traceability domains

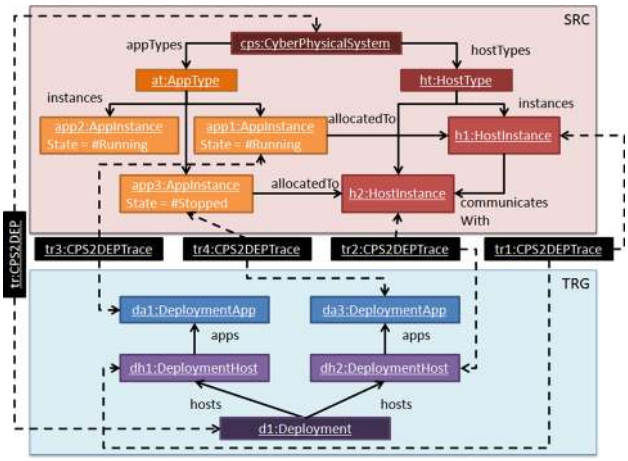

Fig. 2 Sample source and target models

containment hierarchy is defined along references marked by black diamonds (at the container end).

Finally, the mappings between the two domains are persisted in a simple traceability model using CPS2DEPTrace elements and src and trg references.

Instance models A sample source model is depicted in the top part of Fig. 2 with one AppType and HostType, three AppInstances (app1, app2 and app3) and two HostInstances (host1 and host2). Application instances app1 and app2 are running, while app3 is stopped. Moreover, app1 is allocated to host1, while app3 is allocated to host2. Note that the containment hierarchy of elements is not explicitly depicted in the instance models.

Its corresponding target deployment model is illustrated in the bottom part of Fig. 2 with two DeploymentHosts (dh1 and dh2) each containing a DeploymentApp (da1 and da3). Traceability is represented in a separated model with four elements of type CPS2DEPTrace linking the corresponding elements of the source and target domains and a top-level trace CPS2DEP linking the models themselves.

A transformation problem Below, we only overview the M2M scenario, which aims to illustrate the specification of transformations with different levels of incrementality, such as *batch transformations*, *reactive/live transformations* and *traceability-driven transformations*. Initially, we derive a deployment model from the CPS model, and then, incremental model transformations propagate changes observed in the CPS model to the deployment model as well as to the traceability model.

The informal transformation rules are the following:

1. The root element of a CPS model root is mapped to the root of the Deployment model (after clearing all existing traceability elements).
2. Each HostInstance in the source model is mapped to a DeploymentHost in the target model and connected to the root of the Deployment model.
3. Each allocated AppInstance in the source model is mapped to a DeploymentApp in the target model and deployed to its DeploymentHost.

This transformation problem will be used in the sequel to exemplify and compare the different versions of the VIATRA framework. However, the reader is encouraged to check the complete source code, documentation and performance evaluation results available from https://github.com/IncQueryLabs/incquery-examples-cps.

3 VIATRA1: a Prolog-based transformation framework

3.1 Motivation

The motivation for a well-founded model transformation framework came from the HIDE project [28] which aimed to carry out model-based evaluation of functional and dependability attributes of the system under design by formal methods. Assuming that system models are captured in high-level languages (e.g., in UML as in HIDE [28,39] or using languages such as SysML, AADL, BPMN), the key idea was to carry out early systematic *formal analysis of design models*

by generating appropriate mathematical models by *automated model transformations (MT)*. Precise formal analysis retrieves a list of problems, which can be *back-annotated* to the high-level engineering models to allow system designer to make corrections prior to investing in manual coding for implementation. This way, formal methods are hidden by automated model transformations which project system models into various mathematical domains [28,134]. Furthermore, the source code of the target system can be derived by *automatic code generation*.

3.2 Key innovations and features

The initial concepts of [146,147] continued as the Ph.D. project of the first author to develop the first version of the VIATRA model transformation framework published in two key papers [39,148]. The key innovative features of VIATRA1 included the following:

(a) *XMI-based model export/import* VIATRA1 imported and exported models serialized in accordance with the XMI 1.0 standard for *arbitrary MOF-based metamodels*—5 years before EMF achieving the same for Ecore metamodels [45].

(b) *Model transformations by graph transformation* Transformations were formally captured by graph transformation (GT) rules [47,115], which offer a rule and pattern-based approach for manipulating graph-based models. In GT, the preconditions of applying a rule are captured by the left-hand side (LHS) graph, while the right-hand side (RHS) graph declaratively captures how to rewrite the graph as a result of the rule application.
While GT had already been a well-established formal conceptual framework [115] with several applications and tool support [47], using GT for capturing language semantics or model-to-model transformations was a new idea in 2000 reported in [48,50,147,148]. Triple graph grammars were proposed in [119] as a concept for bidirectional transformations, although most of tool support was still ongoing work at that time.

(c) *UML as visual syntax* The concrete syntax of GT rules was defined by UML Class diagrams. Transformation programs were assembled from rules using UML Activity Diagrams and core control primitives (e.g., as-long-as-possible mode). Rational Rose was used as a UML editor for modeling transformation rules and exporting them also in an XMI format.
Embedding graph transformation rules into the UML language was investigated by FUJABA [50] or GReAT [14] with significantly higher maturity compared to VIATRA1. UML was used as a transformation language later, e.g., in [5,82].

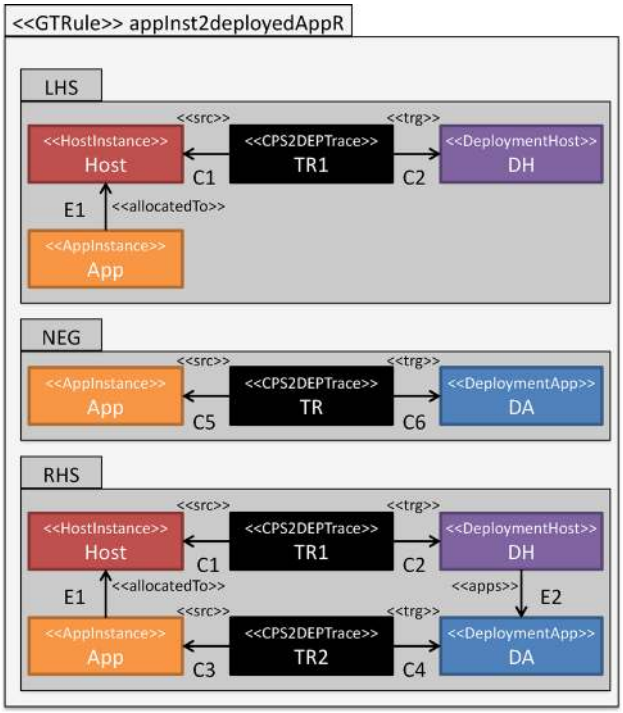

Fig. 3 A graph transformation rule (in UML)

(d) *Auto-generated transformation code* VIATRA1 took a compiled approach by automatically generating Prolog code from GT rules captured as UML models. To be more precise, the target language of the compilation was a internal domain-specific language (DSL) over Prolog. The compilation process consisted of a sequence of model transformation steps [139], and it used only metamodel-level information to derive a local-search-based traversal for edges of graph patterns. Some manual modification of the auto-generated Prolog code was needed in case of complex, recursive transformation rules.
By that time, a similar compiler-based approach was already taken by several graph transformation tools including PROGRES [120], FUJABA [50,102] or OPTIMIX [10].

(e) *Prolog as transformation engine* Model transformations were executed using SWI-Prolog as the underlying engine. Debugging of transformations was carried out on the Prolog-engine level. Logging of transformations was limited to reporting the number of rule applications during a transformation run.
Prolog has remained a popular platform for model transformations as demonstrated by [6,118].

Example A graph transformation rule for mapping- allocated applications to the deployment platform is depicted in Fig. 3. When a AppInstance is found which is allocated

```
% Generated Prolog code
appInst2deployedAppR :-
  % Left hand side
  node(cps:hostInstance(Host)),
  edge(cps:allocatedTo(E1,App,Host)),
  node1(cps:appInstance(App)),
  edge(ref:src(C1,TR1,Host)),
  node1(ref:cPS2DEPTrace(TR1)),
  edge(ref:trg(C2,TR1,DH)),
  node1(dep:deploymentHost(DH)),
  % Negative application condition
  (
     edge(ref:src(C5,TR,App)),
     node1(ref:cPS2DEPTrace(TR)),
     edge(ref:trg(C6,TR,DA)),
     node1(dep:deploymentApp(DA)) -> fail
   ;
     true
  ),
  % Actions for right hand side
  add(node(dep:deploymentApp(DA))),
  add(node(ref:cPS2DEPTrace(TR2))),
  add(edge(dep:apps(E2,DH,DA))),
  add(edge(ref:src(C3,TR2,App))),
  add(edge(ref:trg(C4,TR2,DA))).
```

Fig. 4 Generated Prolog code for rule of Fig. 3

to a HostInstance but not yet mapped to a DeploymentApp (identified by the lack of traceability structures), then the rule creates a new DeploymentApp and linked with a CPS2DEPTrace traceability node and src and trg traceability edges.

In its UML notation, a VIATRA1 rule is specified by three UML packages (for the LHS, the RHS and a negative condition NEG) containing classes and associations between them where (1) each model element (node or edge) is identified by a unique name (defining a corresponding variable) and (2) the types of model elements are defined by appropriate stereotypes. If several classes (or associations) appear in different packages with identical names, it refers to the same model element.

For example, «DeploymentHost»DH appears both in the LHS and the RHS packages; thus, it defines a node which needs to be matched in the LHS and preserved when applying the rule. However, edge «apps» E2 only appears in the RHS package; thus, a new edge is created by the rule with a unique identifier resolved to E2.

VIATRA1 automatically generated the following Prolog code from rule appInst2deployedAppR (listed in Fig. 4). The code generator used a simple depth first search for ordering the Prolog clauses of the fully declarative LHS always starting the search at a designated node of the LHS.

Prolog meta-programming and the cut (!) construct were intensively used for checking attributes and nodes already visited (e.g., as part of node1). Control structures of transformation programs were restricted to a few constructs (see Fig. 5).

```
% Generics for matching
node(Term) :- call(Term).
node1(Term) :- call(Term), !.
edge(Term) :- call(Term).
% Generic control structures
try(Rule) :- call(Rule), !.
loop(Rule) :- try(Rule), loop(Rule).
loop(Rule).
forall(Rule) :- call(Rule), fail.
forall(Rule).
```

Fig. 5 Generic VIATRA1 library in Prolog (extract)

3.3 Application in projects

The first complex model transformations implemented in VIATRA1 were focused around challenges of the HIDE project [28], and numerous follow-up national projects. Key transformations included the following:

- *SC2Promela* mapping UML statecharts to Promela for functional behavior analysis by model checking [94];
- *SCComplete* completeness analysis of UML statecharts using VIATRA1 and Prolog [105]
- *UML2DFN* mapping UML component diagrams to dataflow networks for fault modeling and fault propagation analysis [29];
- *SC2SPN* mapping UML statecharts to stochastic Petri nets for reliability analysis [41].

3.4 Software engineering aspects

The VIATRA1 transformation framework was developed by the first author as part of his Ph.D. project without having a real software engineering process behind the project. Taking the semiformal high-level specification of transformations, he was the developer for most of the model transformations as well since (1) developing transformation rules by UML profiles within Rational Rose had severe usability issues, and (2) Prolog as an internal transformation language and execution engine turned out to be an obstacle for many colleagues.

4 VIATRA2: a model transformation framework in Eclipse

4.1 Motivation

The development of VIATRA2 was started in early 2004 to serve as a general-purpose model transformation engineering framework that aims at supporting the entire lifecycle, i.e., the specification, design, execution, validation and maintenance of transformations within and between various modeling languages and domains [142,145]. A specific goal was to provide foundations for precise model transformations aim-

ing to bridge engineering and formal mathematical domains. The birth of VIATRA2 was also triggered by the open call for proposals for the QVT standard [104], although it took a significantly different approach.

4.2 Key innovations and features

(a) *Model management* VIATRA2 was built on the VPM metamodeling approach [143,144] which facilitated a unified treatment of classes and objects similarly to clabjects to support multilevel metamodeling [11,12] along arbitrary (fluid) metalevels. VIATRA2 introduced the concepts of a model space which provided uniform storage and handling of metamodels, models and transformation models. Unlike in contemporary EMF-based tools which stored metamodels in a separate registry and generated Java code from metamodel-level information, VPM provided a single namespace (registry) for models, metamodels and transformations with fully qualified names and offered a generic model manipulation library without additional code generation.

(b) *Transformation language* The VIATRA2 framework offered a rule- and pattern-based transformation language for manipulating graph models by combining graph transformation and abstract state machines (ASMs) into a single specification paradigm [142]. This textual language provided advanced constructs for querying (e.g., recursive graph patterns [152]), unidirectional graph transformation rules for elementary model manipulations, and complex transformation programs captured by ASMs. As further key innovation, generic and meta-transformations were introduced in [142,145] which are known nowadays as higher-order transformations [8,133].

Popular graph transformation tools developed in parallel with VIATRA2 included AGG [49], ATOM3 [93], FUJABA [102], GReAT [14], GrGen.Net [55], MOFLON [7], VMTS [99]. Popular model transformation languages within the Eclipse framework were ATL [80], Epsilon [83], GEMS [127], or Tefkat [95]. Several of these projects later evolved to components of the top-level Eclipse Modeling project.

(c) *Graph query and transformation engine* Graph queries could initially be evaluated on instance models using *local-search-based pattern matching* [33]. First search plans were statically generated using metamodel-level information (and magic sets for recursive patterns [152]), and later, we experimented with *model-level adaptive search plans* [153]. As a key innovation, we developed *incremental graph pattern matching* by using a custom [154] and Rete-based [52] caching mechanism in [109]. Moreover, a *hybrid approach* for combining incremental and local search techniques was proposed in [21]. VIATRA2 also had a *live model transformation engine* [109] where rules are permanently loaded and they immediately react to changes in the underlying models in order to support change-driven transformations [24,112].

Several efficient search plan-based techniques have been proposed since then for GrGEN [55], EMF models [150], adaptive search plans [55] as well as for the current VIATRA [34]. Moreover, incremental model transformation techniques were proposed, e.g., for Tefkat [62], ATL [81] and in triple graph grammar tools [56,69].

(d) *Transformation plugins* While most transformations were executed by the VIATRA2 transformation engine, extensive research has been dedicated to generate stand-alone transformation plugins from high-level specifications using the VIATRA 2 transformation language. These plugins could be embedded and executed in industrial platforms without the need for the VIATRA 2 engine. Target platforms included relational databases [23,151] or Enterprise Java Beans [16,149].

(e) *Add-ons* The VIATRA2 transformation started to serve as a core for high-level features and add-on. Most notable examples include the *VIATRA-DSM framework* [111] which aimed to support the development of custom domain-specific languages and simulators. However, unlike the Graphical Modeling Framework [128], it avoided the use of code generators by providing a customizable generic core framework with real-time reflection to changes in language specifications. Interestingly, similar concepts are used in the Sirius framework [131] nowadays.

VIATRA2 also served as core engine for solving constraint satisfaction [74,75] and design space exploration [65,66] problems with complex structural constraints directly over models. A key challenge here is to find consistent sequences of rule applications leading to a designated target state fulfilling goal constraints. Incremental pattern matching was beneficial to quickly identify constraint violations during traversal. VIATRA2 also served as the conceptual basis for a stochastic simulator for GT systems [135].

State space exploration over graph models has also been investigated in model checkers for GT systems (e.g., Groove [113,114], Augur [85]). Nowadays, rule-based design space exploration approaches following related ideas include [4,44,51,54].

Example As a sample transformation rule of VIATRA2 , we present in Fig. 6 how to derive DeploymentApp elements for each AppInstance allocated to a HostInstance (which was discussed for VIATRA1 transformations in Fig. 3 and Fig. 4).

This GT rule appInst2deployedAppR also uses explicit traceability models to identify which elements are already

```
namespace cps2dep.xforms;
// Importing types from metamodels
// (to disambiguate type names below)
import cps.metamodel;
import cps2dep.metamodel;
import dep.metamodel;

// Helper to identify mapped source and target elements
pattern mappedElement(CPS, TR, DEP) =
{
  CPSElement(CPS);
  CPS2DEPTrace.src(_C1,TR,CPS);
  CPS2DEPTrace(TR);
  CPS2DEPTrace.trg(_C2,TR,DEP);
  DepElement(DEP);
}

// Graph transformation rule: pattern + action
gtrule appInst2deployedAppR(in App, out TR2, out DA) =
{
  precondition pattern lhs(App, DH) = {
    HostInstance(Host);
    AppInstance.allocatedTo(E1,App,Host);
    AppInstance(App);
    find mappedElement(Host, TR1, DH);
    DeploymentHost(DH);
    neg find mappedElement(App, _TR2, _DH);
  }
  action {
    new( DeploymentApp(DA) in DepModel);
    new( DeploymentHost.apps(_E1, DH, DA));
    new( CPS2DEPTrace(TR2) in TraceModel));
    new( CPS2DEPTrace.src(_C3, TR2, App));
    new( CPS2DEPTrace.trg(_C4, TR2, DA));
  }
}
```

Fig. 6 A graph transformation rule in VIATRA2

mapped during transformation by introducing graph pattern mappedElement. The LHS of the rule is also a graph pattern which consists of local structural constraints (e.g., AppInstance(App) and pattern composition using the predefined mappedElement pattern in both positive and negative way: We check that Host is already mapped to a DeploymentHost, while App is not yet mapped as requested by a negative application condition. The complete graph pattern is composed and flattened at compile time by the VIATRA2 engine (also arranging the actual predicates in an efficient order using search plans). This time the action of the transformation rule is defined in an imperative way by using model manipulation operations with abstract state machine constructs.

An extract from a sample transformation program assembled by using abstract state machine constructs is listed in Fig. 7. The transformation first handles the root CPS element of the source model (identified by variable CPSModel) by applying rule cps2deploymentR. Then, it applies rule hostInst2deployedHostR for all HostInstances within CPSModel. Finally, rule appInst2deployedAppR (detailed in Fig. 6) is initiated to handle (allocated) AppInstances.

4.3 Selected applications

The VIATRA2 framework served as a key underlying model transformation technology of several European projects for dependable embedded systems and service-oriented applications including DECOS, DIANA, MOGENTES and SEN-

```
namespace cps2dep.xforms;

rule main() =
let CPSModel = cps.models.name1,
    TraceModel = trace.models.name2,
    DepModel = dep.models.name3,
    ... // few more variable declarations
in seq {
  choose CPS below CPSModel
    apply cps2deploymentR(CPS,TR,Dep) do
      print("Rule cps2deploymentR:" + name(CPS));
  forall HO below CPSModel
    apply hostInstS2deployedHostR(HO,TR1,DH) do
      print("Rule hostInst2deployedHostR:" + name(HO));
  forall App below CPSModel
    apply appInst2deployedAppR(App,TR2,DA) do
      print("Rule appInst2deployedAppR:"+ name(App));
  log(info,"Transformation terminated successfully.");
 }
```

Fig. 7 A transformation program in VIATRA2

SORIA and the INDEXYS project within the industry-driven ARTEMIS platform. Academic and industrial partners in these projects became the first end users of the VIATRA2 framework. Regular usage of the framework has been reported at ARCS and TU Vienna (Austria), University of Leicester (UK), LMU Munich (Germany), TU Kaiserslautern (Germany), University of Pisa (Italy), University of Waterloo (Canada), Georgia University of Technology and NASA (USA).

4.3.1 Transformations for service-oriented computing

The SENSORIA European project developed a comprehensive, model-driven approach for service engineering including (1) novel languages for service modeling, (2) qualitative and quantitative techniques for service analysis, (3) automated mechanisms for model-driven service deployment and (4) transformations for legacy service re-engineering. Model transformation served as a key technology for model-driven service engineering by bridging different languages and tools in the context of service-oriented applications. Various model transformations were developed in the scope of the project:

- *Automated formal analysis of BPEL processes* The consistency of business processes captured using the standard BPEL notation [103] was formally analyzed by the SAL model checker [19], which exhaustively investigates all potential execution paths of a dynamic behavioral model to decide if a designated property (requirement) holds or not. SAL models were automatically derived by a complex model transformation [88,89].
- *Back-annotation of model checking results to BPEL processes* As a reverse problem, back-annotation of the results retrieved by the SAL model checker to the BPEL model of service engineers was carried out [63] by a mapping between traces captured by change-driven transformations [112].

- *Model-driven performability analysis.* Performability is a nonfunctional system-level parameter, which aims to assess the cost of using fault-tolerant techniques in terms of performance. We developed a model-driven performability analysis approach [59] by mapping UML-based service models to formal process model for the PEPA framework [58]. The system-level performability model was assembled from a library of core performability components driven by system-level UML component diagrams.
- *Model-driven service deployment* The derivation of configuration descriptors required for service deployment was automated in [57,60,90] by a chain of generic model transformations [145]. This approach was successfully adapted to a number of standard service platforms including the application interface specification of the service availability forum, the web service description language (WSDL), WS-reliable messaging, the IBM RAMP platform, and Rampart and Sandesha configuration files for the Apache Axis2 framework.

4.3.2 Transformations for critical embedded systems

The VIATRA2 model transformation framework has been intensively used for providing tool support for developing and verifying critical embedded systems in numerous European research projects such as DECOS, DIANA, MOGENTES, INDEXYS and SecureChange.

- *Model-driven tool integration* Model transformations served a key role in tool integration scenarios to bridge a variety of off-the-shelf industrial tools where tool integration scenarios were driven by the underlying development process [15,67].
- *Model-driven development tools* A user-guided interactive model transformation chain [15] served as the foundation for model-driven tools in the automotive and avionics domain aiming to support the development of configuration tables for hardware-software allocation [67,76].

4.4 Software engineering aspects

To facilitate the increased involvement of graduate students, VIATRA2 was implemented in Java as a plugin of the new open source Eclipse framework. In September 2005, VIATRA2 became an open-source component of Generative Modeling Tools (GMT) project, which was a joint incubator of several model transformation technologies including ATL [80] or Epsilon [83].

The VIATRA2 project has been initially led by the first author, while the first contribution of VIATRA2 was developed by one Ph.D. student and 3 MSc students. Since then, VIATRA2 has continuously been developed until mid of 2013 with 6 major releases altogether involving a total of 8 Ph.D. students, including all co-authors of the paper, András Balogh, Gergely Varró and Dániel Tóth and well over 20 MSc student throughout the years.

5 VIATRA3: a reactive transformation platform built on incremental queries

5.1 Motivation

By 2010, it was clear to us that the major strength of VIATRA2 is its query language and incremental evaluation engine, while its most severe practical limitation is the lack of seamless support for models captured using the industrial Eclipse Modeling Framework (EMF) [45]. An industrial tool development project carried out in collaboration with OptXware Ltd. further revealed that writing queries and validation rules for EMF models was especially cumbersome due to, e.g., problematic testability of embedded loops, complex navigation expressions and export–import functionality.

This led us to focus most of our research and development efforts to incremental model queries over EMF models [22] by giving birth to the EMF- INCQUERY framework in 2010—and decelerating the development of VIATRA until late 2012. Later VIATRA has become a reactive and live model transformation platform exploiting the incremental query evaluation provided by INCQUERY. In 2015, EMF- INCQUERY successfully made transition out of the incubation phase at the Eclipse Foundation. From April 2016, the two projects were unified (where EMF- INCQUERY became "VIATRA Query"), and the full VIATRA release leaves the incubation phase.

5.2 Incremental model queries: key features

INCQUERY started as an open-source Eclipse project to define declarative graph queries over EMF models [45] without manual coding and execute them efficiently using incremental graph pattern matching techniques over an imperative programming language such as Java. The main features of INCQUERY include:

1. *High-level declarative graph query language* The query language of INCQUERY [25,137] and thus VIATRA3 conceptually builds upon the query language of VIATRA2, but its type system is tightly integrated with EMF models and offers many powerful language shortcuts. A main conceptual extension is the introduction of transitive closure over edges defined as arbitrary binary relations [23].
2. *Incremental query engine* INCQUERY offers a highly efficient engine to evaluate queries over models with millions of elements [22,137,138] by adapting Rete networks [52]

to change notifications sent by EMF models. To decrease memory consumption, a query can also be evaluated using local-search-based techniques [34] which exploit model-specific search plans [150].

3. *Integrated development environment.* The advanced IDE of IncQuery [137] enables to construct and validate model queries supported by state-of-the-art Xtext tooling with syntax highlighting, auto-completion, type checking, traceability between models and queries, debugger or incremental compilation.
4. *Integration with EMF tools* The modular architecture of IncQuery enables easy integration with existing EMF-based modeling tools and applications [137] such as Papyrus UML [130], Capella [108], Sirius [131] or Artop [1]. The primary use case for model queries is to support the live validation of well-formedness constraints of a domain in order to highlight and report inconsistencies as soon as they are introduced by engineers. Additional main use cases include advanced support for incremental maintenance of base model indexers [137], derived features [110], soft traceability links [64] or incremental view maintenance [43]. In all these cases, language-level annotations eliminate manual coding for integration. Note that while we illustrate IncQuery and Viatra in the context of EMF models, core queries and transformations are regular Java programs which have been successfully adapted to other technological spaces (outside Eclipse).

Detailed scalability assessment of IncQuery is carried out in numerous papers for validation of well-formedness constraints [22,137], detection of source code anti-patterns [138] or maintenance of soft traceability links [64] over models with 10 million elements.

Related work Other query technologies in the context of EMF include EMF Model Query 2 [129], which provides simple query primitives for selecting model elements that satisfy a set of conditions. The OCL development environment of the Eclipse OCL project [46] provides different ways to edit OCL constraints: an Xtext-based editor for file-based editing, an embedded editor inside Ecore model editors. The Epsilon Validation Language is dedicated to support the construction of validation rules within the Epsilon family [83], while the Acceleo Query Language (AQL) is heavily used within the Sirius project[131] to populate views from underlying models. In addition, some academic approaches support incremental constraint evaluation over models [27,35,61,136].

Example The definition of a sample well-formedness constraint [141] for checking valid allocations of application instances to host instances is listed in Fig. 8. The query

```
1  //EMF-IncQuery pattern in the query definition file
2  @Constraint(
3    key = {"app"}, severity = "error"
4    message = "$app.id$ is not allocated but running",
5  )
6  pattern notAllocatedButRunning(app: AppInstance) {
7    AppInstance.state(app, ::Running);
8    neg find allocatedApp(app);
9  }
10
11 private pattern allocatedApp(app: AppInstance) {
12   AppInstance.allocatedTo(app, _host);
13 }
```

Fig. 8 Sample queries for well-formedness constraints (adapted from [77])

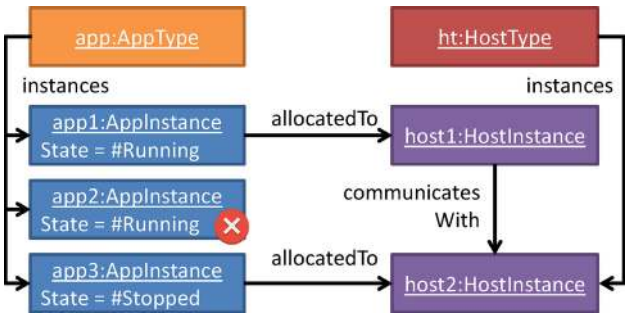

Fig. 9 Erroneous situation

notAllocatedButRunning captures an erroneous situation for allocation when an application *app* is running, but not allocated to a host instance (using another graph pattern allocatedApplication by negative composition). When checking this constraint on the source instance model depicted in Fig. 9, app2 is the only AppInstance which matches the pattern (thus violates the constraint) since app1 is allocated to a host instance ht1, while app3 is stopped.

By using a @Constraint annotation, a query will be automatically integrated into an EMF-based model editor. As a result, an error marker will immediately be placed on the model whenever this consistency constraint is violated, which is removed automatically once the source of the problem is corrected (e.g., the application instance is stopped or allocated to a host instance).

5.3 A reactive transformation platform: key features

Viatra is a reactive, event-driven model transformation platform [20] built on top of incremental graph queries where transformations are executed continuously as *reactions* to changes in the underlying model. The main features of the Viatra project[1] are as follows:

1. *Reactive transformation framework* Viatra adopts the principles of *reactive programming* [13] and active database systems [106]. The core concept of reactive programming is *event-driven behavior*: Components are

[1] http://www.eclipse.org/viatra/.

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 when continuous interaction with the environment has to be maintained by the application based on external events without a priori knowledge on their sequence. VIATRA has proven to be an efficient execution platform for incremental transformations in [77,107].

2. *Internal DSL as transformation language* VIATRA uses an internal DSL for specifying both *batch and event-driven, reactive transformations*, which is an advanced API over Java and Xtend [132]. The *specification* of a VIATRA transformation program contains (1) *rule specifications* consisting of *model queries*, which serve as a precondition to the transformation, and *actions*, which typically prescribe model manipulations. Then, (2) *execution schemas* are defined to orchestrate the reactive behavior.
3. *Complex event processing* VIATRA supports complex event processing (CEP) [42] over EMF models to detect complex sequences of (hierarchical) events and specify reactions upon them. Event hierarchies are constituted from external events (that appear on the event stream), notifications of elementary model changes and aggregated model changes identified by changes in the result set of queries.
4. *Rule-based design space exploration* The current VIATRA framework natively supports rule-based design space exploration over EMF models [66] to explore design candidates as graph model which satisfy multiple criteria over states and trajectories and evolve along certain operations. As a key innovation, it extends previous rule-based DSE concepts with multi-objective optimization [4].
5. *Model obfuscator and secure model access* In order to remove sensitive information from a confidential model provided by an industrial partner (e.g., to create a bug report), VIATRA supports model obfuscation which changes sensitive data stored in models to randomly generated names. Ongoing work carried out partly in the context of the MONDO European project[2] aims to support collaborative model-driven engineering by providing (1) secure model access to model fragments compliant with high-level policies, (2) secure bidirectional model transformations using lenses and (3) property-based locks [37].

Related work Various *EMF-based model transformation tools* provide support for specifying, executing and evaluation of transformations including frameworks such as

[2] http://www.mondo-project.org/.

```
1  // Located in query definition file .eiq
2  // Finds an appl instance allocated to a host instance
3  pattern applicationInstance(appType, appInstance, host)
4  {
5    AppType.instances(appType, appInstance);
6    AppInstance.allocatedTo(appInstance, host);
7  }
8  ----------------------------------------------------------
9  CPS2DEP mapping // reference to the mapping model
10 // Located in query definition file .xtend
11 val appInstanceRule = createRule()
12   .name("appInstanceRule")
13   //a graph pattern as precondition
14   .precondition(applicationInstance)
15   //a lifecycle for event-driven behavior
16   .lifeCycle(ActivationLifecycles.default)
17   //action to be executed when a new match appears
18   .action(CRUDActivationStates.CREATED) [
19     //lookup host in the target model
20     val dh = getTrgTrace(host) as DeploymentHost
21     //create new DeploymentApp in the target model
22     val da = createChild(dh, DeploymentHost.apps,
23                          DeploymentApp)
24     createTrace(app,da)
25   ]
26   //action to be executed when a match gets lost
27   .action(CRUDActivationStates.DELETED) [
28     //lookup app in the target model
29     val da = getTrgTrace(appInstance) as DeploymentApp
30     //remove trace and app
31     removeTrgTrace(da)
32     remove(da)
33   ].build // create rule
```

Fig. 10 A reactive transformation rule

ATL [80], Henshin [9] or eMoflon [2]. Many industrial applications rely on Xtend[132] as a code generation and transformation language based on Java. Epsilon [83] provides the Epsilon Transformation Language and the low-level Epsilon Object Language with an advanced execution platform.

However, relatively few tools support event-driven or reactive transformations: New features of ATL include target incremental computation [81] combined into the ReactiveATL transformation engine. Generic low-level transformation engines include EMFTVM [155] or T-Core [124]. SIGMA [91] provides Scala-embedded DSLs that map Epsilon model transformation functionality directly into Scala as a specification language, and the JVM as the execution platform. A detailed overview of the state-of-the-art for rule-based design space exploration frameworks is provided in [66]

Example A reactive and event-driven version of our sample transformation rule of Fig. 3 is illustrated in Fig. 10. The application of the rule has double effects: (1) When an AppInstance gets allocated to a HostInstance in the source model, then it creates the corresponding DeploymentApp in the target model, and (2) when a mapped AppInstance is no longer allocated to a HostInstance in the source model, then it removes the corresponding DeploymentApp from the target model.

The execution of (1) is triggered by the appearance of a new match of its precondition pattern applicationInstance, while the execution of (2) is initiated when an existing match of the same pattern disappears.

Let us now assume that the source CPS model changes by removing the allocatedTo edge between app3 and

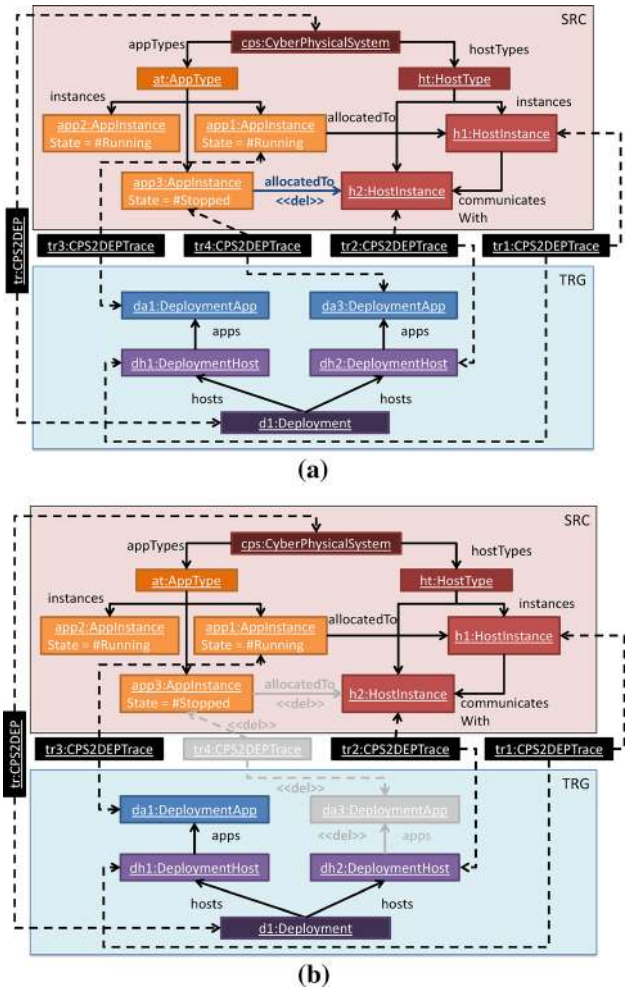

Fig. 11 Effect of reactive transformations upon change

h2 (see Fig. 11a). Due to this change, an existing match of pattern applicationInstance disappears, which correspondingly triggers the application of the reactive rule appInstanceRule. As a result, the corresponding DeploymentApp is removed from the target model together with the traceability structures to yield the transformation result depicted in Fig. 11b.

5.4 Reactive execution architecture of VIATRA

A reactive transformation program in VIATRA consists of two parts. First, the *rule specifications* are defined by a *precondition* and some *actions*. A precondition is most frequently captured as a *query* over a given *model*(s) or alternatively by detections of complex events, while actions include *model manipulations* and *control structures*. As a key innovation, the activation of rules is clearly separated from the execution of their actions by defining *execution schemas* are defined in order to orchestrate the reactive behavior, e.g., to resolve conflicts and schedule rule activations. Now, we

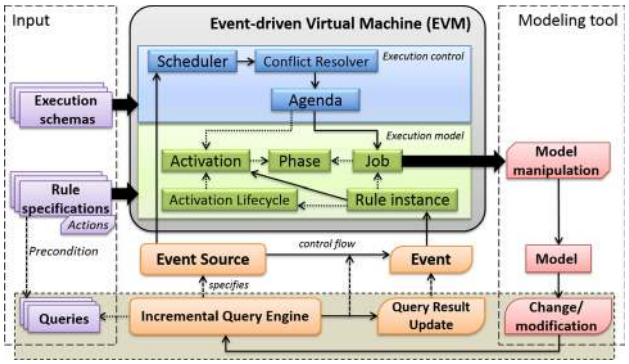

Fig. 12 The VIATRA architecture for reactive model transformations

briefly describe the behavior of core components of the *event-driven virtual machine* (EVM) of VIATRA in Fig. 12.

5.4.1 Events and activation lifecycles

Events Activations (i.e., matches) and executions of reactive transformation rules are triggered by events, which are either (1) *controlled events* which are initiated explicitly by a transformation program, a transaction or the user, (2) *observed events* which are caused by external behavior, and the exact time of their occurrence may not be determined by the transformation program. Such observed events include notifications upon elementary model changes, event sequences or updated results of model queries. In both category of events, detection of events is followed by the *firing* of a corresponding rule along specific parameter bindings.

Activation lifecycles Reactive transformation rules react to events in accordance with their activation lifecycles, which is a state transition system reflecting the current state of a rule activation. An activation lifecycle consists of different (1) *phases* and (2) event-triggered transitions between such phases. Optionally, (3) a transition may be associated with a *job*, which represents the executable *actions* of a input rule specification. Figure 13 presents two typical activation lifecycles for event-driven and batch transformations.

To unify the behavior of model transformations over the reactive VIATRA platform, activations of both event-driven and batch transformations are executed as reactive programs. For instance, the *enabled* phase of an activation lifecycle represents reactions to *observed* events, while the firing of the actual reactive jobs is tied to *controlled* events. A library of most common lifecycles is readily available in VIATRA which can be customized in Java by transformation developers.

5.4.2 Agenda, scheduler, conflict resolver

Scheduler External observed events influence activation phases according to the lifecycle, and the job to be executed

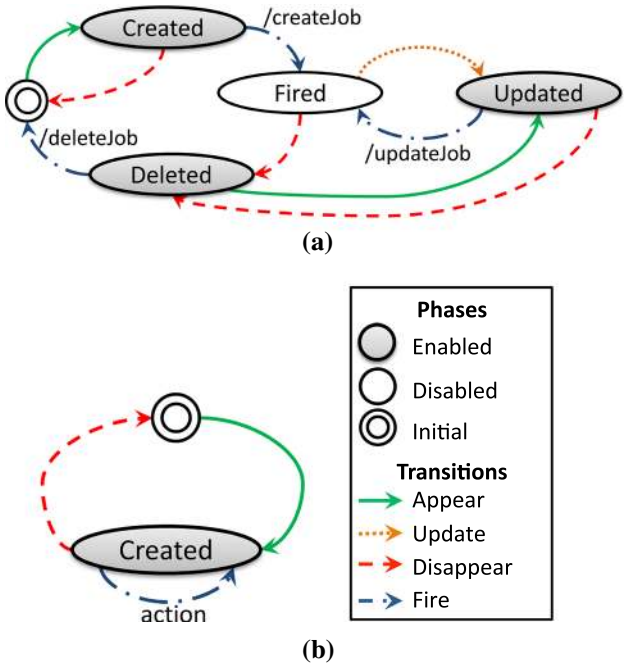

Fig. 13 Typical rule lifecycles [20]. **a** Event-driven transformation. **b** Batch execution

(if any) is determined by the active phase. However, it is the *scheduler* component that determines when the EVM can fire these controlled scheduling events to actually execute the jobs.

Practical examples of scheduling events include (1) the signal of the query engine indicating that the update of query results has concluded after an elementary model manipulation; (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:

- By default, most rules are executed as soon as possible; thus, their effects are observable by the user or available for further transformation steps.
- However, certain rules may require that the models is consistent (or inconsistent in case of quick fix rules), while others may be inefficient to execute while a large-scale transaction is incomplete.

Scheduling events offer better control over consistently executing different features compared to traditional plugins and still provide a high level of flexibility.

Event-driven rules may 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.

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 behavior 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 it is labeled with the event type. Afterward, 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 phase.

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

5.4.3 Execution primitives

Model manipulation primitives The createChild method in Fig. 10 is one of the model manipulation primitives provided by the VIATRA framework to enable more concise transformation definitions by hiding EMF-related details, e.g., transaction handling. The common operations supported by the framework are the following.[3]

(A) (*createModel, Class*) Creates an object of the corresponding Class, and places it as the root of the selected Model (called Resource in EMF).

[3] A complete guide on model manipulation primitives and batch transformation operations is available from https://wiki.eclipse.org/VIATRA/Transformation_API.

(B) *createChild(Object, Reference, Class)* Creates an object of type Class to be contained by its parent Object along the specified Reference.
(C) *addTo(SObject, StructuralFeature, TObject)* Adds a noncontainment reference or attribute of type StructuralFeature from the source SObject to the target TObject (list semantics).
(D) *set(SObject, StructuralFeature, TObject)* Sets the value of a single-valued StructuralFeature (reference or attribute) of SObject to the TObject.
(E) *remove(Object)* Removes the Object from the model including an implicit removal of all dangling references pointing to the Object.
(F) *move(FObject, TObject, StructuralFeature)* Moves element FObject to a new container TObject along containment StructuralFeature and removes it from the old one.

Control structures for batch execution In many practical cases of batch transformations, it is worth controlling the behavior of rule execution after firing a rule on a specific activation by explicit control structures and semantic modifiers such as the following:

(a) *fireOne* Selects an enabled activation of the given rule nondeterministically and fires the rule on it.
(b) *fireAllCurrent* Fires all of the currently enabled activations of the given rules (in the order specified by the conflict resolver). Newly enabled activations are ignored. If one of the selected activations gets disabled during the execution, it will not be fired.
(c) *fireAsLongAsPossible* Repeatedly performs *fireOne* as long as there are enabled activations to fire.
(d) *fireUntil(exitCondition)* Repeatedly performs *fireOne* as long as there are any enabled activations to fire, and the exit condition evaluates to false.

The role of these control structures is similar to the old VIATRA versions (see Figs. 5 and 7).

5.5 Selected applications

The INCQUERY and VIATRA frameworks have actively been used in different industrial and academic projects carried out by various researchers and practitioners. Below we provide a short overview of selected applications of these frameworks within our own projects.

(a) *Dedicated systems engineering toolchains* A recent project industrial aimed to define a model-driven approach and tool chain for the synthesis of complex, integrated MATLAB Simulink models capable of simulating the software and hardware architecture of an aircraft [64,70]. We actively developed industrial toolchains for other application domains (e.g., automotive, telecommunication) on contractual basis, which heavily built on incremental model queries and transformations.
(b) *Integration of MATLAB Simulink and EMF models* The Massif (MATLAB Simulink Integration Framework for Eclipse)[4] framework [70,73] provides a bidirectional bridge between MATLAB Simulink models and their EMF model counterpart via the MATLAB API. Massif was initiated within an industrial project with Embraer and then continued as part of the CONCERTO European ARTEMIS-JU project.
(c) *Formal validation of DSLs* Designing advanced tooling for a new DSL is an error-prone task as it is surprisingly easy to introduce contradicting or incomplete well-formedness constraints for a language. DSL-level validation of language specifications is carried out in [121,122] by using back-end logic solvers where derived features and well-formedness constraints are captured by queries. This technique derives small instance models as proofs of consistency by composing required model fragments.
(d) *Incremental code generators* Incremental code generators [72,140] aim to avoid complete regeneration in case of small changes by exploiting incremental queries and transformations. VIATRA allows incorporating incremental transformations on *different levels of granularity* (resource, model fragment, model element) and information stored in traceability models can also be customized. Source incremental computation provided by VIATRA turns out to be an efficient technique also for code generation.
(e) *Custom views by queries* Incremental recomputation of graphical views [43,71] can be also be driven by reactive transformations. Each node and edge in a graphical view is defined as a query (or a transformation in a more complex case), and the view is automatically (and incrementally) recalculated upon each change in the source model. VIATRA is flexible enough to aggregate multiple model elements in the underlying model and represent them as a single node in the view.
(f) *Live movement detection* Live detection of human gestures and movements is carried out in [42] by using streaming transformations and complex event processing (CEP) [101]. A live model forms the basis of calculations which is updated rapidly (25 times each second), and respective model changes are turned into events. Finally, relevant situations are detected as event sequences by CEP techniques.

[4] https://github.com/FTSRG/massif/wiki.

5.6 Software engineering aspects

The significant growth in the complexity of the projects necessitated to better distribute responsibilities. Since its foundation in 2010, István Ráth has been the project lead for the IncQuery project (now together with Ábel Hegedüs), Zoltán Ujhelyi has become the project lead of Viatra in 2012, while Massif is led by Ákos Horváth. Viatra is expected to join the yearly release cycles imposed by the Eclipse Foundation in 2017. Consequently, committers are now dominated by professional software engineers (working for industrial companies), and novel features of the next release are highly influenced by customers and tool vendors. Moreover, the development team is international as Viatra regularly receives contributors from several countries.

However, novel innovative or high-risk components (e.g., the local search engine, complex event processing, design space exploration) are still dominantly prototyped by researchers or students (working for academia). This synergy enables to accelerate the transfer of academic results into industry while also providing highly challenging usage scenarios for researchers.

6 Summary and future work

6.1 Summary

In the past 16 years, the three generations of the Viatra model transformation framework have continuously served as a general means to design complex mappings within and between DSLs. As a summary, Fig. 14 presents the main features, specificities and differences between the various major Viatra versions. More extensive survey of model transformation tools are available in [40,69,79].

In our practice, particularly deep and complex transformations are those that need to bridge large conceptual gaps between different formalisms, such as mapping high-level engineering models to low-level mathematical models in order to carry out formal verification and validation. Despite the wide range of existing languages and formalisms, our experience shows that many underlying concepts of transformation design have a common conceptual basis.

Moreover, an increasing number of scenarios with industrial relevance necessitated to use transformations (1) for developing new domain-specific (or general-purpose) modeling tools within the open- source Eclipse framework or (2) for integrating existing tools into complex, end-to-end toolchains. Checker tools for early validations or code generators for automated synthesis of various engineering artifacts have also been major application scenarios in our practice. The continuous advances of different model transformation techniques and tools have turned the science of transformation design into an engineering tasks.

However, there is still much to do to decrease the extreme costs of tool qualification, especially in the context of critical embedded and cyber-physical systems. While some transformations are now precisely specified on the semantic level (using techniques such as semantic anchoring [38] or formalism transformation graphs [100]), it is still not possible to guarantee that the actual implementation for the integration of different tools or interaction of tool features is free of flaws. Scalability of engineering tools is also a major challenge [84]. Testing of customized DSL tools is still in its infancy due to the lack of automatically synthesized consistent and large model instances with given diversity and coverage criteria. Industry-driven tool integration initiatives (like OSLC [3]) cover a broad spectrum of processes, but fail to address many of the underlying technical challenges. Altogether, developing software tools in an open component architecture still lacks scientific foundations.

6.2 Future of modeling tools?

The lack of such foundations may become severe in the context of smart cyber-physical systems (CPS) [36,96,126], which are open, interconnected and highly distributed complex systems expected to consist of 50 billion smart objects by 2020 [31]. They will integrate simple sensors and actuators to the Internet-of-Things (IoT) to exploit low latency cloudlets as in edge/ fog computing [30]. High-level web services may exploit the immense computation power of cloud computing. But a CPS also interconnects critical infrastructures and systems (such as cars, medical devices, aircrafts) with tightly integrated real-time computing platforms and physical systems [36] where a system failure may result in major financial loss, severe damage or even casualties.

For smart CPS, the distinction between design-time and run-time models [26] is more and more blurred [18,96] which gives birth to run-time modeling frameworks (like Kevoree [53]). Thus, incremental query and transformation techniques will likely be used as part of the underlying middleware, which triggers further open research challenges such as *how to support the development of scalable future tools for smart and trusted CPS where run-time models are directly connected to the system and interact with it in close synergy*. While still providing various consistency guarantees for tool qualification purposes, we foresee that future modeling frameworks will support the following tasks:

1. Interaction with the underlying system and its context;
2. Identification of critical situations and triggering reactions at run-time;
3. Deployment of run-time models as services over heterogeneous platforms;

	VIATRA	eclipse.org/viatra	
	V1	V2	V3 (with IncQuery)
History			
Period	2000-2004	2004-2014	2010-
Eclipse Project	No	Incubation	Release (1.2)
Key Research Projects	HIDE, National	DECOS, DIANA, MOGENTES, SENSORIA	SecureChange, CERTIMOT, TRANS-IMA, MONDO, CONCERTO,
Key Integrations		UML, BPMN, SysML	EMF, EMF-UML, Xtend, Sirius, MPS, ARTOP, Capella, Matlab,
Models			
Standard export/import	XMI	XML	XMI
Metamodeling	MOF	VPM	EMF
UML Support	via XMI	via XMI	Yes
UML Profiles	For MT rules	Limited	Yes
DSM Support	None	ViatraDSM	Graphiti, Sirius, Zest, jFace
Language			
Language Syntax	Graphical	Textual	Textual
Own/Hosted	Host: UML	Own: GT + ASM	IncQuery: Own, VIATRA: Hosted
Queries	UML + Prolog	Graph patterns	IQPL + any
Manipulations	UML + Prolog	GT + ASM	VIATRA API
MT Programs	UML + Prolog	ASM	Xtend
Rule Iterations	ALAP, ForEach	ALAP, ForEach	Xtend + Execution mode library
Recursion	Tail recursion	Rich (Magic Sets)	Xtend + Transitive closure
Derived Features	No	No	Yes
Compositionality	No	Queries	Queries + Rules
Rule Inheritance	No	No	No
MT Features			
Pattern Matching Strategy	Local Search (LS)	LS+INC+Hybrid	LS+INC
Incrementality	No	Queries	Several modes
Parallel / Distributed	No	Parallel	Parallel
Generic/Meta Xform	Partly in Prolog	Yes	as in Xtend
Traceability	Explicit	Explicit	Explicit + Implicit
Bidirectional	No	No	No
Batch execution mode	Yes	Yes	Yes
Live execution mode	No	Yes	Yes
Reactive execution mode	No	No	Yes
View Models	No	No	Unidirectional + Incremental
MT Workflow	No	Yes	MWE
Design Space Exploration	No	Yes	Yes
Complex Event Processing	No	Proof-of-concept	Yes
Obfuscation	No	No	Yes
Tooling			
Editor	Rational Rose	Eclipse (Custom)	Xtext
Syntax highlight	No	Yes	Yes
Type checking	No	Yes	Yes
Content Assist	No	No	Yes
Interpreter	Prolog	GT + ASM	Java
Compilers	UML2Prolog	Relational DB, EJB	Xtend + IncQuery
Debugger	Native Prolog	Limited	Yes
Testing	Ad hoc	Simple test suite	Complex test suite
Builds	No	Release + Developer	Continuous

Fig. 14 Feature overview of the VIATRA family (partly adapted from [79])

4. Hierarchical abstractions over time and structure.

As ongoing research in the MONDO project, we have started the development of IncQuery-D to support the incremental evaluation of graph queries in a distributed environment by distributing the nodes of Rete networks [125]. Models can be captured by different graph representations and stored in existing graph storages (e.g., RDF triple stores,

distributed storage frameworks like Hadoop or Spark). The IncQuery- D framework is an independent query layer on top of such storages to efficiently support incremental reevaluation over large and evolving graph data. This frameworks is intended to support run-time checks of cyber-physical systems and detect relevant situations in IoT applications.

Related ongoing work [17] aims to address the run-time verification [98] of rich and high-level specifications (such as graph queries or complex event processing languages) which is being extended toward over heterogeneous and distributed platforms which include smart IoT devices with limited resources, mobile phones as well as cloud-based computations. This technique was used as part of a complex demonstrator for the Eclipse IoT Challenge 2016.[5]

Acknowledgments The authors would like to acknowledge key additional contributors for the Viatra and IncQuery frameworks and its applications (listed alphabetically):
V1 György Csertán, Gábor Huszerl, Zsigmond Pap, András Pataricza, István Majzik, Gergely Varró;
V2 Andás Balogh, Zoltán Balogh, Tibor Bende, Zsolt Déri, László Gönczy, Máté Kovács, András Kövi, István Majzik, Gergely Nyilas, András Ökrös, András Pataricza, Balázs Polgár, Zsolt Sándor, András Schmidt, Tibor Somodi, Dániel Tóth, Dávid Vágó, Gergely Varró, Szilvia Varró-Gyapay.
V3 Márton Búr, István Dávid, Csaba Debreceni, Miklós Földényi, Dénes Harmath, Benedek Izsó, Péter Lunk, András Szabolcs Nagy, István Papp, Oszkár Semeráth, Gábor Szárnyas, Tamás Szabó (includes developers of IncQuery).
The preparation of this paper was partially supported by the MONDO Project (EU ICT-611125), the FP7 ARTEMIS CONCERTO (ART-2012-333053) project and the MTA-BME Lendület Research Group on Cyber-Physical Systems. Furthermore, several other projects supported the whole line of underlying research in the past 16 years, most notably: European projects (HIDE, DECOS, DIANA, SecureChange, MOGENTES, INDEXYS), national projects (CERTIMOT ERC-HU), and numerous industrial contracts and grants (with CEA, Embraer, Ericsson, IBM, Thales, ThyssenKruppPresta, TeqBall).

References

1. Artop: The AUTOSAR Tool Platform (2015). https://www.artop.org/
2. eMoflon (2015). http://www.moflon.org/
3. OSLC: Open services for lifecycle collaboration (2015). http://open-services.net/
4. Abdeen, H., Varró, D., Sahraoui, H.A., Nagy, A.S., Debreceni, C., Hegedüs, Á., Horváth, Á.: Multi-objective optimization in rule-based design space exploration. In: ACM/IEEE International Conference on Automated Software Engineering, ASE '14, Vasteras, Sweden - September 15–19, 2014, pp. 289–300 (2014). doi:10.1145/2642937.2643005
5. Acretoaie, V., Störrle, H., Strüber, D.: Transparent model transformation: turning your favourite model editor into a transformation tool. In: Theory and Practice of Model Transformations—8th International Conference, ICMT 2015, L'Aquila, Italy, pp. 121–130 (2015)
6. Almendros-Jiménez, J.M., Iribarne, L.: A model transformation language based on logic programming. In: SOFSEM 2013: Theory and Practice of Computer Science, 39th International Conference on Current Trends in Theory and Practice of Computer Science, Špindlerův Mlýn, Czech Republic, pp. 382–394 (2013)
7. Amelunxen, C., Königs, A., Rötschke, T., Schürr, A.: MOFLON: A standard-compliant metamodeling framework with graph transformations. In: Model Driven Architecture—Foundations and Applications, Second European Conference, ECMDA-FA 2006, *LNCS*, vol. 4066, pp. 361–375. Springer (2006)
8. Amelunxen, C., Legros, E., Schürr, A.: Generic and reflective graph transformations for the checking and enforcement of modeling guidelines. In: IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008, Herrsching am Ammersee, Germany, pp. 211–218. IEEE Computer Society (2008)
9. Arendt, T., Biermann, E., Jurack, S., Krause, C., Taentzer, G.: Henshin: Advanced concepts and tools for in-place EMF model transformations. In: Model Driven Engineering Languages and Systems—13th International Conference (MODELS 2010), Proceedings, Part I, *LNCS*, vol. 6394, pp. 121–135. Springer (2010)
10. Assmann, U.: In [47], chap. OPTIMIX: A Tool for Rewriting and Optimizing Programs, pp. 307–318. World Scientific (1999)
11. Atkinson, C., Kühne, T.: The essence of multilevel metamodelling. In: Proceedings of the UML 2001—The Unified Modeling Language. Modeling Languages, Concepts and Tools, *LNCS*, vol. 2185, pp. 19–33. Springer (2001)
12. Atkinson, C., Kühne, T.: Reducing accidental complexity in domain models. Softw. Syst. Model. **7**(3), 345–359 (2007)
13. Bainomugisha, E., Carreton, A.L., Cutsem, T.V., Mostinckx, S., Meuter, W.D.: A Survey on Reactive Programming. ACM Computing Surveys, NewYork (2012)
14. Balasubramanian, D., Narayanan, A., van Buskirk, C.P., Karsai, G.: The graph rewriting and transformation language: great. ECEASST **1** (2006)
15. Balogh, A., Bergmann, G., Csertán, G., Gönczy, L., Horváth, Á., Majzik, I., Pataricza, A., Polgár, B., Ráth, I., Varró, D., Varró, G.: Workflow-driven tool integration using model transformations. In: Graph Transformations and Model-Driven Engineering—Essays Dedicated to Manfred Nagl on the Occasion of his 65th Birthday, *LNCS*, vol. 5765, pp. 224–248. Springer (2010)
16. Balogh, A., Varró, G., Varró, D., Pataricza, A.: Compiling model transformations to EJB3-specific transformer plugins. In: ACM Symposium on Applied Computing—Model Transformation Track (SAC 2006), pp. 1288–1295. ACM Press, Dijon, France (2006)
17. Balogh, L., István Dávid István Ráth, D.V., Vörös, A.: Distributed and heterogeneous event-based monitoring in smart cyber-physical systems. In: 1st Workshop on Monitoring and Testing of Cyber-Physical Systems, Vienna, Austria (2016)
18. Baresi, L., Ghezzi, C.: The disappearing boundary between development-time and run-time. In: FoSER '10 Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research, pp. 17–22 (2010)
19. Bensalem, S., Ganesh, V., Lakhnech, Y., Munoz, C., Owre, S., Rueß, H., Rushby, J., Rusu, V., Saïdi, H., Shankar, N., Singerman, E., Tiwari, A.: An overview of SAL. In: LFM 2000: Fifth NASA Langley Formal Methods Workshop, pp. 187–196 (2000)

[5] http://modes3.tumblr.com/.

20. Bergmann, G., Dávid, I., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., Varró, D.: Viatra 3: A reactive model transformation platform. In: Theory and Practice of Model Transformations—8th International Conference, ICMT 2015, *LNCS*, vol. 9152, pp. 101–110 (2015). doi:10.1007/978-3-319-21155-8_8
21. Bergmann, G., Horváth, Á., Ráth, I., Varró, D.: Efficient model transformations by combining pattern matching strategies. In: Theory and Practice of Model Transformations, Second International Conference, ICMT 2009, Zurich, Switzerland, June 29-30, 2009. Proceedings, *LNCS*, vol. 5563, pp. 20–34. Springer (2009). doi:10.1007/978-3-642-02408-5_3
22. Bergmann, G., Horváth, Á., Ráth, I., Varró, D., Balogh, A., Balogh, Z., Ökrös, A.: Incremental evaluation of model queries over EMF models. In: Model Driven Engineering Languages and Systems—13th International Conference, MODELS 2010, Oslo, Norway, October 3–8, 2010, Proceedings, Part I, *LNCS*, vol. 6394, pp. 76–90. Springer (2010). doi:10.1007/978-3-642-16145-2_6
23. Bergmann, G., Ráth, I., Szabó, T., Torrini, P., Varró, D.: Incremental pattern matching for the efficient computation of transitive closure. In: Sixth International Conference on Graph Transformation, LNCS, pp. 386–400. Springer, Bremen, (2012)
24. 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). doi:10.1007/s10270-011-0197-9
25. Bergmann, G., Ujhelyi, Z., Ráth, I., Varró, D.: A graph query language for EMF models. In: Proceedings of the International Conference on Model Transformation, *LNCS*, vol. 6707, pp. 167–182. Springer (2011)
26. Blair, G., Bencomo, N., France, R.B.: Models@ run.time. Computer **42**(10), 22–27 (2009). doi:10.1109/MC.2009.326
27. Blanc, X., Mougenot, A., Mounier, I., Mens, T.: Incremental detection of model inconsistencies based on model operations. In: Proceedings of the 21st International Conference on Advanced Information Systems Engineering, CAiSE '09, pp. 32–46. Springer, Berlin, Heidelberg (2009)
28. Bondavalli, A., Dal Cin, M., Latella, D., Majzik, I., Pataricza, A., Savoia, G.: Dependability analysis in the early phases of UML based system design. Int. J. Comput. Syst. Sci. Eng. **16**(5), 265–275 (2001)
29. Bondavalli, A., Majzik, I., Mura, I.: Automatic dependability analyses for supporting design decisions in UML. In: HASE'99: The 4th IEEE International Symposium on High Assurance Systems Engineering (1999)
30. Bonomi, F., Milito, R., Zhu, J., Addepalli, S.: Fog computing and its role in the internet of things. In: Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing—MCC '12, p. 13. ACM Press, New York (2012)
31. Brech, B., Jamison, J., Shao, L., Wightwick, G.: The Interconnecting of Everything. An IBM Redbooks Point-of-View publication (2010), 6 (2013)
32. Broy, M., Kirstan, S., Krcmar, H., Schätz, B.: What is the benefit of a model-based design of embedded software systems in the Car industry? In: Emerging Technologies for the Evolution and Maintenance of Software Models, pp. 343–369 (2012)
33. Bruni, R., Varró, D. (eds.): Fifth International Workshop on Graph Transformation and Visual Modelling Techniques, ENTCS. Elsevier (2006)
34. Búr, M., Ujhelyi, Z., Horváth, Á., Varró, D.: Local search-based pattern matching features in emf-incquery. In: Graph Transformation—8th International Conference, ICGT 2015, L'Aquila, Italy, *LNCS*, vol. 9151, pp. 275–282. Springer (2015). doi:10.1007/978-3-319-21145-9_18
35. Cabot, J., Teniente, E.: Incremental integrity checking of UML/OCL conceptual schemas. J. Syst. Softw. **82**(9), 1459–1478 (2009)
36. Cengarle, M.V., Bensalem, S., McDermid, J., Passerone, R., Sangiovanni-Vincentelli, A., Törngren, M.: Technical Report 611430. http://www.cyphers.eu
37. Chechik, M., Dalpiaz, F., Debreceni, C., Horkoff, J., Ráth, I., Salay, R., Varró, D.: Property-based methods for collaborative model development. In: Joint Proceedings of the 3rd International Workshop on the Globalization Of Modeling Languages and the 9th International Workshop on Multi-Paradigm Modeling, GEMOC+MPM@MoDELS 2015, Ottawa, Canada, *CEUR Workshop Proceedings*, vol. 1511, pp. 1–7. CEUR-WS.org (2015)
38. Chen, K., Sztipanovits, J., Abdelwahed, S., Jackson, E.K.: Semantic anchoring with model transformations. In: ECMDA-FA, pp. 115–129 (2005)
39. Csertán, G., Huszerl, G., Majzik, I., Pap, Z., Pataricza, A., Varró, D.: VIATRA: Visual automated transformations for formal verification and validation of UML models. In: Proceedings of the ASE 2002: 17th IEEE International Conference on Automated Software Engineering, pp. 267–270. IEEE Press, Edinburgh (2002)
40. Czarnecki, K., Helsen, S.: Feature-based survey of model transformation approaches. IBM Syst. J. **45**(3), 621–646 (2006)
41. Dal Cin, M., Huszerl, G., Kosmidis, K.: Evalution of safety-critical system based on guarded statecharts. In: HASE'99 The 4th IEEE International Symposium on High Assurance Systems Engineering (1999)
42. Dávid, I., Ráth, I., Varró, D.: Streaming model transformations by complex event processing. In: Model-Driven Engineering Languages and Systems—17th International Conference, MODELS 2014, Valencia, Spain, pp. 68–83 (2014). doi:10.1007/978-3-319-11653-2_5
43. Debreceni, C., Horváth, A., Hegedüs, A., Ujhelyi, Z., Ráth, I., Varró, D.: Query-driven incremental synchronization of view models. In: Proceedings of the 2nd Workshop on View-Based, Aspect-Oriented and Orthographic Software Modelling, VAO '14, pp. 31:31–31:38. ACM, New York, NY, (2014). doi:10.1145/2631675.2631677
44. Denil, J., Jukšs, M., Verbrugge, C., Vangheluwe, H.: Search-based model optimization using model transformations. Technical Report, McGill University, Canada (2014)
45. Eclipse Foundation: Eclipse Modeling Framework (EMF). http://eclipse.org/modeling/emf/
46. The Eclipse Project: MDT OCL. http://www.eclipse.org/modeling/mdt/?project=ocl
47. Ehrig, H., Engels, G., Kreowski, H.J., Rozenberg, G. (eds.): Handbook on Graph Grammars and Computing by Graph Transformation. In: Applications, Languages and Tools, vol. 2, World Scientific (1999)
48. Engels, G., Hausmann, J.H., Heckel, R., Sauer, S.: Dynamic meta modeling: a graphical approach to the operational semantics of behavioral diagrams in UML. In: UML 2000—The Unified Modeling Language, Advancing the Standard, Third International Conference, York, UK, October 2–6, 2000, Proceedings, pp. 323–337 (2000). doi:10.1007/3-540-40011-7_23
49. Ermel, C., Rudolf, M., Taentzer, G.: In [47], chap. The AGG-Approach: Language and Tool Environment, pp. 551–603. World Scientific (1999)
50. Fischer, T., Niere, J., Torunski, L., Zündorf, A.: Story diagrams: a new graph transformation language based on UML and Java. In: Proceedings of the Theory and Application to Graph Transformations (TAGT'98), *LNCS*, vol. 1764. Springer (2000)
51. Fleck, M., Troya, J., Wimmer, M.: Marrying search-based optimization and model transformation technology (2015)
52. Forgy, C.L.: Rete: a fast algorithm for the many pattern/many object pattern match problem. Artif Intell **19**(1), 17–37 (1982)
53. Fouquet, F., Nain, G., Morin, B., Daubert, E., Barais, O., Plouzeau, N., Jézéquel, J.M.: Kevoree Modeling Framework (KMF): Effi-

cient modeling techniques for runtime use. Technical Report, University of Luxembourg, TR-SnT-2014-11 (2014)
54. Galvao Lourenco da Silva, I., Zambon, E., Rensink, A., Wevers, L., Akşit, M.: Knowledge-based graph exploration analysis. In: AGTIVE, LNCS 7233, pp. 105–120 (2011)
55. Geiß, R., Batz, V., Grund, D., Hack, S., Szalkowski, A.M.: GrGen: A fast SPO-based graph rewriting tool. In: Proceedings of the 3rd International Conference on Graph Transformation, *LNCS*, vol. 4178, pp. 383–397. Springer, Natal, Brazil (2006)
56. Giese, H., Wagner, R.: From model transformation to incremental bidirectional model synchronization. Soft Syst Model (SoSyM) **8**(1), 21–43 (2009)
57. Gilmore, S., Gönczy, L., Koch, N., Mayer, P., Tribastone, M., Varró, D.: Non-functional properties in the model-driven development of service-oriented systems. Syst. Model. Softw. (2011). doi:10.1007/s10270-010-0155-y
58. Gilmore, S., Hillston, J.: The PEPA Workbench: A tool to support a process algebra-based approach to performance modelling. In: Computer Performance Evaluation, Modeling Techniques and Tools, 7th International Conference, Vienna, Austria, *LNCS*, vol. 794, pp. 353–368. Springer (1994)
59. Gönczy, L., Déri, Z., Varró, D.: Model transformations for performability analysis of service configurations. In: Models in Software Engineering, Workshops and Symposia at MODELS 2008, Toulouse, France, *LNCS*, vol. 5421, pp. 153–166. Springer (2008)
60. Gönczy, L., Hegedüs, Á., Varró, D.: Methodologies for model-driven development and deployment: An overview. In: Wirsing, M., Hölzl, M. (eds.) Rigorous Software Engineering for Service-Oriented Systems, *LNCS*, vol. 6582. Springer (2011)
61. Groher, I., Reder, A., Egyed, A.: Incremental consistency checking of dynamic constraints. In: Fundamental Approaches to Software Engineering (FASE 2009), *LNCS*, vol. 6013, pp. 203–217. Springer (2010)
62. Hearnden, D., Lawley, M., Raymond, K.: Incremental model transformation for the evolution of model-driven systems. In: Proceedings of the 9th International Conference on Model Driven Engineering Languages and Systems, *LNCS*, vol. 4199, pp. 321–335. Genova (2006)
63. Hegedüs, Á., Bergmann, G., Ráth, I., Varró, D.: Back-annotation of simulation traces with change-driven model transformations. In: Proceedings of the Eighth International Conference on Software Engineering and Formal Methods, pp. 145–155. IEEE Computer Society, Pisa (2010). doi:10.1109/SEFM.2010.28
64. Hegedüs, Á., Horváth, Á., Ráth, I., Starr, R.R., Varró, D.: Query-driven soft traceability links for models. Softw Syst Model (2014). doi:10.1007/s10270-014-0436-y
65. Hegedüs, Á., Horváth, Á., Ráth, I., Varró, D.: A model-driven framework for guided design space exploration. In: 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), November 6–10, pp. 173–182. IEEE, Lawrence, KS (2011)
66. Hegedüs, Á., Horváth, Á., Varró, D.: A model-driven framework for guided design space exploration. Autom. Softw. Eng. **22**(3), 399–436 (2015). doi:10.1007/s10515-014-0163-1
67. Herzner, W., Csertán, B.H.G., Balogh, A.: The DECOS toolchain: Model-based development of distributed embedded safety-critical real-time systems. In: Proceedings of the DECOS/ERCIM Workshop at SAFECOMP 2006, pp. 22–24. ERCIM (2006)
68. Hidaka, S., Tisi, M., Cabot, J., Hu, Z.: Feature-based classification of bidirectional transformation approaches. Softw. Syst. Model. 1–22 (2015). doi:10.1007/s10270-014-0450-0
69. Hildebrandt, S., Lambers, L., Giese, H., Rieke, J., Greenyer, J., Schäfer, W., Lauder, M., Anjorin, A., Schürr, A.: A survey of triple graph grammar tools. ECEASST **57** (2013). doi:10.14279/tuj.eceasst.57.865
70. Horváth, Á., Hegedüs, Á., Búr, M., Varró, D., Starr, R.R., Mirachi, S.: Hardware-software allocation specification of ima systems for early simulation. In: Digital Avionics Systems Conference (DASC). IEEE, Colorado Spings, Colorado (2014)
71. Horváth, A., Ráth, I.: IncQuery gets Sirius: faster and better diagrams. In: EclipseCon Europe (2015)
72. Horváth, A., Ráth, I., Hegedüs, A., Balogh, A.: Decreasing your coffee consumption with incremental code regeneration. In: EclipseCon France (2015)
73. Horváth, A., Ráth, I., Starr, R.R.: Massif—the love child of Matlab Simulink and Eclipse. In: EclipseCon NA (2015)
74. Horváth, Á., Varró, D.: CSP(M): Constraint satisfaction problem over models. In: Schürr, A., Selic, B. (eds.) Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4–9, 2009. Proceedings, *LNCS*, vol. 5795, pp. 107–121. Springer (2009)
75. Horváth, A., Varró, D.: Dynamic constraint satisfaction problems over models. Softw. Syst. Model. **11**(3), 385–408 (2012). doi:10.1007/s10270-010-0185-5
76. Horváth, Á., Varró, D., Schoofs, T.: Model-driven development of ARINC 653 configuration tables. In: 29th IEEE & AIAA Digital Avionics System Conference (DASC), pp. 5.A.5–1–5.A.5–115. IEEE, IEEE, Salt Lake City (2010). doi:10.1109/DASC.2010.5655322
77. IncQuery Labs Ltd.: CPS Demonstrator: A model transformation benchmark (2015). https://github.com/IncQueryLabs/incquery-examples-cps/wiki/
78. ISO: Road vehicles Functional safety (ISO 26262) (2011)
79. Jakumeit, E., Buchwald, S., Wagelaar, D., Dan, L., Hegedüs, Á., Herrmannsdörfer, M., Horn, T., Kalnina, E., Krause, C., Lano, K., Lepper, M., Rensink, A., Rose, L.M., Wätzoldt, S., Mazanek, S.: A survey and comparison of transformation tools based on the transformation tool contest. Sci. Comput. Program. **85**, 41–99 (2014)
80. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I.: ATL: a model transformation tool. Sci. Comput. Program. **72**(1–2), 31–39 (2008)
81. Jouault, F., Tisi, M.: Towards incremental execution of ATL transformations. Theory and Practice of Model Transformations. LNCS, vol. 6142, pp. 123–137. Springer, Berlin (2010)
82. Kalnins, A., Barzdins, J., Celms, E.: Model transformation language MOLA. Model Driven Architecture. European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, LNCS, vol. 3599, pp. 62–76. Springer, Linköping, Sweden (2005)
83. Kolovos, D., Rose, L., Garcia-Domínguez, A., Paige, R.: The Epsilon Book (2015). http://www.eclipse.org/epsilon/doc/book/
84. 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., Cabot, J.: A research roadmap towards achieving scalability in model driven engineering. In: Proceedings of the Workshop on Scalability in Model Driven Engineering, BigMDE '13, pp. 2:1–2:10. ACM (2013)
85. König, B., Kozioura, V.: Augur 2—a new version of a tool for the analysis of graph transformation systems. ENTCS **211**, 201–210 (2008)
86. Kornecki, A.J., Zalewski, J.: The qualification of software development tools from the DO-178B certification perspective. J. Def. Softw. Eng. **19**(4), 19–22 (2006)
87. Kornecki, A.J., Zalewski, J.: Certification of software for real-time safety-critical systems: state of the art. Innov. Syst. Softw. Eng. **5**(2), 149–161 (2009)
88. Kovács, M., Gönczy, L., Varró, D.: Formal modeling of BPEL workflows including fault and compensation handling. In: EFTS '07: Proceedings of the 2007 Workshop on Engineering Fault Tolerant Systems, p. 1. ACM, New York, NY (2007). doi:10.1145/1316550.1316551

89. Kovács, M., Gönczy, L., Varró, D.: Formal analysis of BPEL workflows with compensation by model checking. Int. J. Comput. Syst. Eng. **23**(5), 349 (2008)
90. Kövi, A., Varró, D.: An Eclipse-based framework for AIS service configurations. In: Malek, M., Reitenspieß, M., van Moorsel, A.P.A. (eds.) Proceedings of the 4th International Service Availability Symposium, ISAS 2007, Durham, NH, USA, May 21–22, 2007, *LNCS*, vol. 4526, pp. 110–126. Springer (2007)
91. Krikava, F., Collet, P., France, R.B.: SIGMA: Scala internal domain-specific languages for model manipulations. In: Model-Driven Engineering Languages and Systems, *LNCS*, vol. 8767, pp. 569–585. Springer (2014). doi:10.1007/978-3-319-11653-2_35
92. Krupitzer, C., Roth, F.M., VanSyckel, S., Schiele, G., Becker, C.: A survey on engineering approaches for self-adaptive systems. Perv. Mob. Comput. **17**, 184–206 (2015)
93. de Lara, J., Vangheluwe, H.: AToM3: A tool for multi-formalism and meta-modelling. In: 5th International Conference, FASE 2002: Fundamental Approaches to Software Engineering, Grenoble, France, April 8–12, 2002, Proceedings, *LNCS*, vol. 2306, pp. 174–188. Springer (2002)
94. Latella, D., Majzik, I., Massink, M.: Automatic verification of a behavioural subset of UML statechart diagrams using the SPIN model-checker. Form. Asp. Comput. **11**(6), 637–664 (1999). doi:10.1007/s001659970003
95. Lawley, M., Steel, J.: Practical declarative model transformation with tefkat. In: Satellite Events at the MoDELS 2005 Conference, MoDELS 2005 International Workshops, Revised Selected Papers, *LNCS*, vol. 3844, pp. 139–150. Springer (2006)
96. Lee, E.A., Hartmann, B., Kubiatowicz, J., Rosing, T.S., Wawrzynek, J., Wessel, D., Rabaey, J.M., Pister, K., Sangiovanni-Vincentelli, A.L., Seshia, S.A., Blaauw, D., Dutta, P., Fu, K., Guestrin, C., Taskar, B., Jafari, R., Jones, D.L., Kumar, V., Mangharam, R., Pappas, G.J., Murray, R.M., Rowe, A.: The swarm at the edge of the cloud. IEEE Des. Test **31**(3), 8–20 (2014)
97. Leroy, X.: Formal verification of a realistic compiler. Commun. ACM **52**(7), 107–115 (2009)
98. Leucker, M., Schallhart, C.: A brief account of runtime verification. J. Log. Algebraic Program. **78**(5), 293–303 (2009)
99. Levendovszky, T., Lengyel, L., Mezei, G., Charaf, H.: A systematic approach to metamodeling environments and model transformation systems in VMTS. ENTCS **127**(1), 65–75 (2005)
100. Lucio, L., Mustafiz, S., Denil, J., Vangheluwe, H., Jukss, M.: FTG+PM: an integrated framework for investigating model transformation chains. In: SDL 2013: Model-Driven Dependability Engineering, Montreal, Canada, *LNCS*, vol. 7916, pp. 182–202. Springer (2013)
101. Luckham, D.C.: The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems. Addison-Wesley Longman Publishing Co., Inc, Boston (2001)
102. Nickel, U., Niere, J., Zündorf, A.: Tool demonstration: The FUJABA environment. In: The 22nd International Conference on Software Engineering (ICSE). ACM Press, Limerick (2000)
103. OASIS: Web Services Business Process Execution Language Version 2.0 (OASIS Standard) (2007). http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.html
104. Object Management Group: QVT: MOF 2.0 Query/View Transformation (2008). http://www.omg.org/spec/QVT/1.0/
105. Pap, Z., Majzik, I., Pataricza, A.: Checking general safety criteria on UML statecharts. In: Computer Safety, Reliability and Security, 20th Int. Conf., SAFECOMP 2001, Budapest, Hungary, September 26–28, 2001, Proceedings, pp. 46–55 (2001). doi:10.1007/3-540-45416-0_5
106. Paton, N., Diaz, O.: Active database systems. ACM Comput. Surv. **31**(1), 63–103 (1999)
107. van Pinxten, J., Basten, T.: Motrusca: Interactive model transformation use case repository. In: 7th York Doctoral Symposium on Computer Science and Electronics, p. 57 (2014)
108. Polarsys: Capella (2015). https://www.polarsys.org/capella/
109. Ráth, I., Bergmann, G., Ökrös, A., Varró, D.: Live model transformations driven by incremental pattern matching. In: Vallecillo, A., Gray, J., Pierantonio, A. (eds.) Proceedings of the First International Conference on the Theory and Practice of Model Transformations (ICMT 2008), *LNCS*, vol. 5063, pp. 107–121. Springer (2008). doi:10.1007/978-3-540-69927-9_8
110. Ráth, I., Hegedüs, Á., Varró, D.: Derived features for EMF by integrating advanced model queries. In: Modelling Foundations and Applications—8th European Conference, ECMFA 2012, Kgs. Lyngby, Denmark, July 2-5, 2012. Proceedings, *LNCS*, vol. 7349, pp. 102–117. Springer (2012)
111. Ráth, I., Vágó, D., Varró, D.: Design-time simulation of domain-specific models by incremental pattern matching. In: IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008, Herrsching am Ammersee, Germany, 15–19 September 2008, Proceedings, pp. 219–222. IEEE (2008)
112. Ráth, I., Varró, G., Varró, D.: Change-driven model transformations. In: A. Schürr, B. Selic (eds.) Model Driven Engineering Languages and Systems, 12th International Conference, MODELS 2009, Denver, CO, USA, October 4–9, 2009. Proceedings, *LNCS*, vol. 5795, pp. 342–356. Springer (2009)
113. Rensink, A.: Canonical graph shapes. In: Schmidt, D.A. (ed.) Programming Languages and Systems—European Symposium on Programming (ESOP), *LNCS*, vol. 2986, pp. 401–415. Springer (2004)
114. Rensink, A.: Time and space issues in the generation of graph transition systems. GraBaTs **127**, 127–139 (2004)
115. Rozenberg, G. (ed.): Handbook of Graph Grammars and Computing by Graph Transformations: Foundations. World Scientific, Singapore (1997)
116. RTCA: Software Tool Qualification Considerations (DO-330) (2011)
117. RTCA: Software Considerations in Airborne Systems and Equipment Certification (DO-178C) (2012)
118. Schätz, B.: Formalization and rule-based transformation of EMF ecore-based models. In: Software Language Engineering, First International Conference, SLE 2008, Toulouse, France, September 29–30, 2008. Revised Selected Papers, pp. 227–244 (2008). doi:10.1007/978-3-642-00434-6_15
119. Schürr, A.: Specification of graph translators with triple graph grammars. In: Proc. WG94: International Workshop on Graph-Theoretic Concepts in Computer Science, no. 903 in LNCS, pp. 151–163. Springer (1994)
120. Schürr, A., Winter, A.J., Zündorf, A.: In [47], chap. The PROGRES Approach: Language and Environment, pp. 487–550. World Scientific (1999)
121. Semeráth, O., Barta, A., Horváth, A., Szatmári, Z., Varró, D.: Formal validation of domain-specific languages with derived features and well-formedness constraints. Softw. Syst. Model. (2015). doi:10.1007/s10270-015-0485-x
122. Semeráth, O., Horváth, Á., Varró, D.: Validation of derived features and well-formedness constraints in DSLs—by mapping graph queries to an SMT-solver. In: Proceedings of MODELS 2013: Model-Driven Engineering Languages and Systems—16th International Conference, *LNCS*, vol. 8107, pp. 538–554 (2013)
123. Stevens, P.: Bidirectional model transformations in QVT: semantic issues and open questions. Softw. Syst. Model. **9**(1), 7–20 (2008)
124. Syriani, E., Vangheluwe, H., LaShomb, B.: T-Core: a framework for custom-built model transformation engines. Softw. Syst. Model. **14**(3), 1215–1243 (2015)

125. Szárnyas, G., Izsó, B., Ráth, I., Harmath, D., Bergmann, G., Varró, D.: IncQuery-D: A distributed incremental model query framework in the cloud. In: Model-Driven Engineering Languages and Systems—17th International Conference, MODELS 2014, Valencia, Spain, pp. 653–669 (2014). doi:10.1007/978-3-319-11653-2_40
126. Sztipanovits, J., Koutsoukos, X., Karsai, G., Kottenstette, N., Antsaklis, P., Gupta, V., Goodwine, B., Baras, J.: Toward a science of cyber-physical system integration. Proc. IEEE **100**(1), 29–44 (2012)
127. The Eclipse Foundation: Generic Eclipse Modeling System (GEMS) (2008). http://www.eclipse.org/gmt/gems/
128. The Eclipse Foundation: Graphical Modeling Framework (GMF) (2010). http://www.eclipse.org/modeling/gmp/
129. The Eclipse Foundation: EMF Model Query 2 (2012). http://wiki.eclipse.org/EMF/Query2
130. The Eclipse Foundation: Papyrus (2015). https://eclipse.org/papyrus/
131. The Eclipse Foundation: Sirius (2015). http://www.eclipse.com/sirius/
132. The Eclipse Foundation: Xtend (2015). http://www.eclipse.org/xtend
133. Tisi, M., Jouault, F., Fraternali, P., Ceri, S., Bézivin, J.: On the use of higher-order model transformations. In: Model Driven Architecture—Foundations and Applications, 5th European Conference, ECMDA-FA 2009, Enschede, The Netherlands, June 23–26, 2009. Proceedings, *LNCS*, vol. 5562, pp. 18–33. Springer (2009)
134. Tiwari, A., Shankar, N., Rushby, J.M.: Invisible formal methods for embedded control systems. Proc. IEEE **91**(1), 29–39 (2003)
135. Torrini, P., Heckel, R., Ráth, I.: Stochastic simulation of graph transformation systems. In: Fundamental Approaches to Software Engineering, 13th International Conference, FASE 2010, Paphos, Cyprus, *LNCS*, vol. 6013, pp. 154–157. Springer (2010)
136. Uhl, A., Goldschmidt, T., Holzleitner, M.: Using an ocl impact analysis algorithm for view-based textual modelling. ECEASST **44** (2011)
137. 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). doi:10.1016/j.scico.2014.01.004
138. Ujhelyi, Z., Szőke, G., Horváth, A., Csiszár, N.I., Vidács, L., Varró, D., Ferenc, R.: Performance comparison of query-based techniques for anti-pattern detection. Inf. Softw. Technol. **65**, 147–165 (2015). http://dx.doi.org/10.1016j.infsof.2015.01.003
139. Varró, D.: Automatic program generation for and by model transformation systems. In: Proceedings of the AGT 2002: Workshop on Applied Graph Transformation, pp. 161–173. Grenoble, France (2002)
140. Varró, D.: Patterns and styles for incremental model transformations. In: First International Workshop on Patterns in Model Engineering (PAME 2015), L'Aquila, Italy, July 20, 2015, CEUR Workshop Proceedings. CEUR-WS.org (2015). Invited talk
141. Varró, D.: Incremental queries and transformations: From concepts to industrial applications. In: 42nd International Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2016), pp. 51–59. Springer, Harrachov, Czech Republic (2016)
142. Varró, D., Balogh, A.: The model transformation language of the VIATRA2 framework. Sci. Comput. Program. **68**(3), 214–234 (2007). doi:10.1016/j.scico.2007.05.004
143. Varró, D., Pataricza, A.: Metamodeling mathematics: a precise and visual framework for describing semantics domains of UML models. In: Proceedings of the Fifth International Conference on the Unified Modeling Language, *LNCS*, vol. 2460, pp. 18–33. Springer (2002)
144. Varró, D., Pataricza, A.: VPM: a visual, precise and multilevel metamodeling framework for describing mathematical domains and UML. Softw. Syst. Model. **2**(3), 187–210 (2003)
145. Varró, D., Pataricza, A.: Generic and meta-transformations for model transformation engineering. In: Proc. UML 2004: 7th International Conference on the Unified Modeling Language, *LNCS*, vol. 3273, pp. 290–304. Springer, Lisbon, Portugal (2004)
146. Varró, D., Varró, G.: Designing the automatic transformation of visual languages. Technical University of Budapest, Student Report (TDK) (1999)
147. Varró, D., Varró, G., Pataricza, A.: Designing the automatic transformation of visual languages. In: Ehrig, H., Taentzer, G. (eds.) GRATRA 2000 Joint APPLIGRAPH and GETGRATS Workshop on Graph Transformation Systems, pp. 14–21. Berlin (2000)
148. Varró, D., Varró, G., Pataricza, A.: Designing the automatic transformation of visual languages. Sci. Comput. Program. **44**(2), 205–227 (2002)
149. Varró, G.: Implementing an EJB3-specific graph transformation plugin by using database independent queries. Electron. Notes Theor. Comput. Sci. **211**, 121–132 (2008)
150. Varró, G., Deckwerth, F., Wieber, M., Schürr, A.: An algorithm for generating model-sensitive search plans for pattern matching on EMF models. Softw. Syst. Model. **14**(2), 597–621 (2015)
151. Varró, G., Friedl, K., Varró, D.: Implementing a graph transformation engine in relational databases. Softw. Syst. Model. **5**(3), 313–341 (2006)
152. Varró, G., Horváth, Á., Varró, D.: Recursive graph pattern matching: With magic sets and global search plans. In: Proceedings of the Third International Workshop and Symposium on Applications of Graph Transformation with Industrial Relevance (AGTIVE 2007), *LNCS*, vol. 5088. Springer (2008)
153. Varró, G., Varró, D., Friedl, K.: Adaptive graph pattern matching for model transformations using model-sensitive search plans. In: GraMot 2005, International Workshop on Graph and Model Transformations, *ENTCS*, vol. 152, pp. 191–205. Elsevier (2006)
154. Varró, G., Varró, D., Schürr, A.: Incremental graph pattern matching: data structures and initial experiments. In: Graph and Model Transformation (GraMoT 2006), *ECEASST*, vol. 4. EASST (2006)
155. Wagelaar, D., Tisi, M., Cabot, J., Jouault, F.: Towards a general composition semantics for rule-based model transformation. In: 14th International Conference on Model Driven Engineering Languages and Systems, MODELS'11, pp. 623–637. Springer (2011)
156. Whittle, J., Hutchinson, J., Rouncefield, M.: The state of practice in model-driven engineering. IEEE Softw. **31**(3), 79–85 (2014)

Dániel Varró is a full professor at Budapest University of Technology and Economics (BME) and the chair of the MTA-BME Lendület Research group on Cyber-Physical Systems. His main research interest is model-driven software and system engineering. He serves on the editorial board of Software and Systems Modeling and served as PC co-chair of FASE 2013, ICMT 2014 and SLE 2016 conferences. He delivered a keynote talk at CSMR 2012, SOFSEM 2016 and a plenary 10-year most influential paper presentation at MODELS 2014. He is a co-founder of the VIATRA and EMF-IncQuery open-source projects, and IncQuery Labs Ltd., as a start-up company. He

was the principal investigator at BME of the SENSORIA, DIANA, SecureChange and MONDO EU projects. Previously, he was a visiting researcher at SRI International, at the University of Paderborn and twice at TU Berlin. In 2014, he was as a visiting professor at McGill University and Université de Montréal.

Gábor Bergmann has a Ph.D. in Software Engineering from Budapest University of Technology and Economics (BME) where he is currently a research fellow. During his doctoral studies, he worked at Google for a summer internship. His main research interest is model-driven systems development with a special focus on incremental model queries. He published over 30 scientific papers and has participated in multiple academic research programs including EU FP7 projects SecureChange and MONDO.

Ábel Hegedüs is a senior developer at IncQuery Labs Ltd. He received his Ph.D. in Software Engineering from the Budapest University of Technology and Economics. He is experienced in the application of model-driven engineering in complex, reactive and event-driven applications. He is a core developer and co-lead of the VIATRA project at Eclipse.org. He has participated in multiple industrial and EU FP7 research projects. He is the co-author of over 20 scientific papers published at leading conference and journalson back-annotation, traceability and design space exploration, the latter won an ACM Distinguished Paper Award in 2011.

Ákos Horváth is a co-founder and managing director at IncQuery Labs. He received his Ph.D. in Software Engineering from the Budapest University of Technology and Economics. He has a strong background in the application of model-driven technologies for the design and optimization of complex embedded systems with more than 50 peer-reviewed papers and several best paper awards. He has been involved since 2006 in several European Union research projects such as SENSORIA, DIANA and CONCERTO, and participated in several large-scale industrial cooperations with companies from the avionics and telecommunication sectors. He is an active Eclipse committer for the EMF-IncQuery and the VIATRA projects.

István Ráth is a co-founder and the lead of innovations at IncQuery Labs. He received his Ph.D. in Software Engineering from the Budapest University of Technology and Economics (BME). His main interest is model-driven systems development with a special focus on incremental and distributed model queries. Since 2006, he is a regular participant of European Union research projects such as SENSORIA, MOGENTES, SecureChange and MONDO. He co-authored more than 50 peer-reviewed papers which received multiple Best Paper Awards. In 2010, he was a visiting scholar at the University of Waterloo. István is a long-time contributor of Eclipse open-source projects, serving as the chief technological architect of the VIATRA model transformation framework.

Zoltán Ujhelyi is a senior developer at IncQuery Labs Ltd. He has an MSc in Computer Engineering from the Budapest University of Technology and Economics where he is a Ph.D. candidate. He is experienced in the application of model-driven engineering in complex, reactive and event-driven applications and is a core developer and co-lead of the VIATRA Eclipse.org project. Zoltán is a co-author of multiple conference and journal papers on software analysis techniques for model queries and transformations, including type checking and dynamic slicing, and received an IEEE Best Paper Award in 2014.