VIATRA Solver: A Framework for the Automated Generation of Consistent Domain-Specific Models

Oszkár Semeráth[1,2], Aren A. Babikian[3], Sebastian Pilarski[3], and Dániel Varró[1,2,3]
[1]*MTA-BME Lendület Cyber-Physical Systems Research Group, Budapest, Hungary*
[2]*Budapest University of Technolgy and Economics, Dept. of Measurement and Information Systems, Budapest, Hungary*
[3]*McGill University, Dept. of Electrical and Computer Engineering, Montréal, QC, Canada*
Email: {semerath,varro}@mit.bme.hu, {aren.babikian,sebastian.pilarski}@mail.mcgill.ca

***Abstract*—VIATRA Solver [1] is a novel open source software tool to automatically synthesize consistent and diverse domain-specific graph models to be used as a test suite for the systematic testing of CPS modelling tools. Taking a metamodel, and a set of well-formedness constraints of a domain as input, the solver derives a diverse set of consistent graph models where each graph is compliant with the metamodel, satisfies consistency constraints, and structurally different from each other. The tool is integrated into the Eclipse IDE or it is executable from the command line.**
Video demonstration: https://youtu.be/fUopeDFIUKA

I. INTRODUCTION

a) Motivation and Challenge: Model-based systems engineering (MBSE) is widely used in the design of complex cyber-physical systems (CPS) [2] by using various modeling tools. MBSE tools help catch design flaws early, thus saving significant costs. Moreover, they also enable the automated synthesis and consistent co-evolution of different design artifacts, making the development process more productive.

As any piece of software, modeling tools are not free from defects, and defects of a tool may propagate into the system it is used for designing. Thus safety standards of critical CPSs (like DO-178C in avionics) prescribe that only the output of a qualified tool can be trusted. Unfortunately, software tool qualification is extremely costly due to the lack of well-founded and scalable cross-domain systematic testing techniques for modeling tools. The reason for this is that, unlike in case of testing imperative or object-oriented programs, test cases of CPS modeling tools are highly data-driven. They take the form of complex graph models that need to satisfy various structural constraints (e.g. over 500 constraints in case of modeling tools compliant with the AUTOSAR automotive standard). Test models constructed manually by engineers are ineffective in finding bugs in those tools.

Thus, the systematic automated testing of modeling tools necessitates use of graph model generators to produce well-formed (or intentionally malformed) models as test inputs. A consistent (or well-formed) model is compliant with the metamodel of the domain or the modeling language, connected, and it also satisfies the extra well-formedness constraints. Automated generation of consistent models is a challenging task, which necessitates the use of complex logic solvers, graph algorithms, and tight tool integration with the underlying modeling environment. Finally, such auto-generated graphs can be beneficial for database testing, system-level assurance for autonomous CPS or in various benchmarking scenarios.

b) Objectives and Scope: VIATRA Solver [1] is a novel open source software framework to automatically synthesize consistent and diverse models to be used as a test suite for the systematic testing of modeling tools. As input (see Figure 1), it takes the specification of the target tool given in the form of (1) a *metamodel* using the popular Eclipse Modeling Framework (EMF), (2) a set of *consistency constraints captured by graph queries* using the industrial VIATRA framework [3], and optionally, (3) an *initial model fragment*. As output, it generates *a diverse set of consistent graph models*.

Each output graph is compliant with the metamodel, satisfies all consistency constraints, connected, contains the initial model fragment. The output graphs are structurally different from each other to ensure the diversity of the test suite. Moreover, the solver is complete in the sense that it can enumerate all structurally different graphs over a selected equivalence relation.

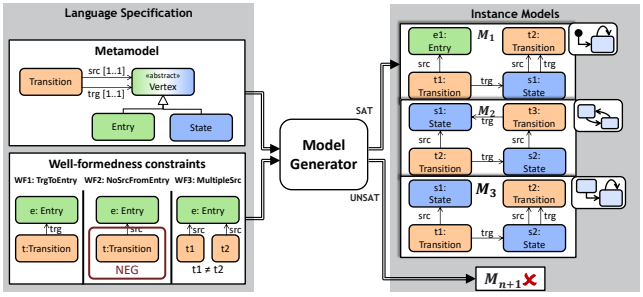

Fig. 1: Conceptual overview of the approach

c) Envisioned Users: Our model generator framework offers benefits to the developers of industrial MBSE tools like Capella, Yakindu, Artop, or Papyrus (and many closed source software products), as it produces systematically assembled test suite used as part of development or regression testing. Moreover, researchers can also use the tool for testing code generators or model transformation tools. In fact, the VIATRA Solver has already been used in *several research projects* at McGill University and at Budapest University Technology and Economics, and it receives regular feedback from software engineers at IncQuery Labs Ltd.

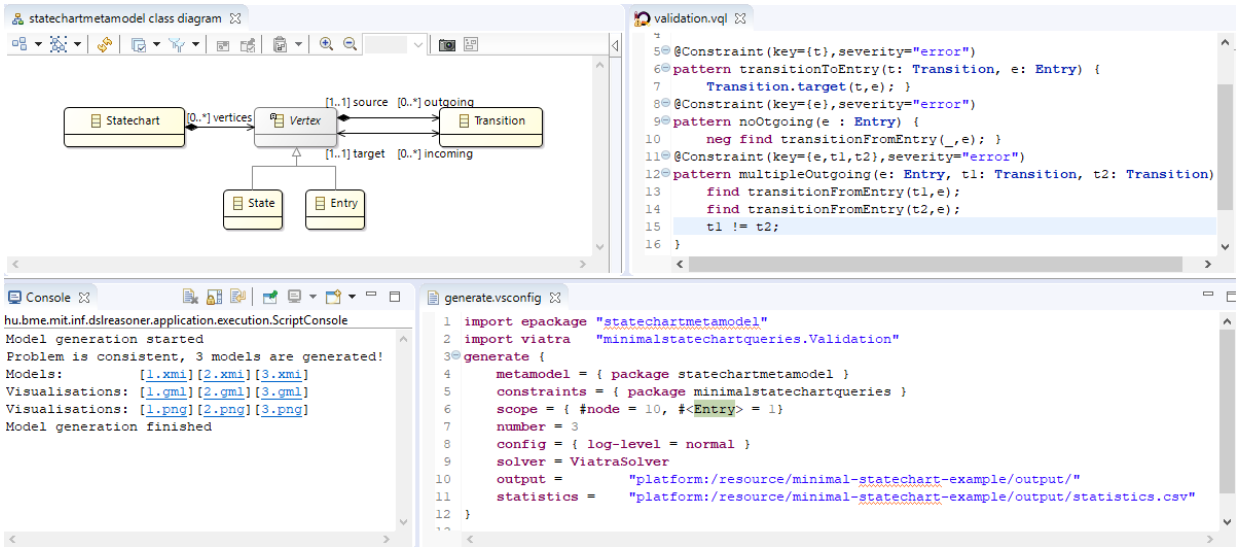

Fig. 2: A screenshot of the VIATRA Solver used for Yakindu Statecharts

II. FRAMEWORK USAGE

The VIATRA Solver [1] is available as a plugin integrated into the Eclipse IDE or as a standalone command line tool. In both cases, the input and output specifications are identical.

a) Domain specification: The core features of the framework are illustrated in the context of Yakindu Statecharts [4] which is an industrial DSL tool (by Itemis AG) for developing reactive, event-driven systems with support for validation and code generation. An extract of the EMF metamodel of Yakindu Statecharts is illustrated in Figure 2 (top left) which defines that statecharts consist of state vertices which are either regular states or entry states. These state vertices can be connected by (outgoing and incoming) transitions.

Consistency constraints are captured as graph patterns using the existing query language and editor of the industrial VIATRA framework [3]. Three constraints are illustrated in Figure 2 (top right): (1) an entry state is disallowed to have an incoming transition; (2+3) an entry state is required to have exactly one outgoing transition. Graph patterns capture the erroneous case, thus a match of a graph pattern in an instance model reports a constraint violation.

Furthermore, an EMF instance model can also be created using an existing model editor (e.g. Yakindu) which can serve as a seed model fragment for the solver that has to be included in any auto-generated graph model. The domain specification taken by the VIATRA Solver intentionally reuses (and seamlessly integrates with) existing Eclipse-based tools and technologies which are widely used in various (CPS) modeling tools to increase the chance of industrial adoption.

b) Solver configuration: The configuration of the solver can be a complex task as several parameters originating from different (off-the-shelf) tools are required to be set correctly. Therefore, our tool provides a lightweight configuration language and textual editor (see Figure 2; bottom right and its grammar in Figure 3) to support the consistent editing of configurations and eliminate syntactic configuration errors. The textual editor developed using Xtext supports rich editor functionality such as syntax highlighting, cross referencing (e.g. to navigate to metamodel or query specifications) and content assist (e.g. to select the target metamodel, constraint specification or underlying solver). The development of this configuration DSL was a major software engineering challenge due to various integration challenges in Eclipse projects.

```
(import epackage ⟨path⟩ | import viatra ⟨path⟩)*
(generate {
  metamodel = ⟨selected packages and classes⟩,
  partial-model = ⟨path to a model to be extended⟩,
  constraints = ⟨selected model queries⟩,
  scope = { ( #⟨type⟩ = ( ⟨int⟩ | ⟨int⟩ .. ⟨int⟩))* },
  number = ⟨int⟩ , runs = ⟨int⟩,
  solver = (SMT | Alloy | Viatra),
  config = { (⟨key⟩=⟨value⟩)* },
  log=⟨path⟩,stats=⟨path⟩,output=⟨path⟩
})*
```

Fig. 3: Simplified grammar for configuration language

The configuration file imports *the metamodel and the query specifications* and then it allows to prune them prior to model generation. For example, one can restrict model generation to a fragment of the metamodel by importing only certain packages or excluding certain classes and references (**metamodel** block). Similarly, consistency constraints can be included or excluded separately (**constraint** block).

The configuration file also needs to contain *parameters for the solver* itself. The most important parameters include (1) the **scope** of the search (i.e. how many nodes are allowed and of which type), (2) the **number** of models to be derived consecutively, and (3) the number of independent **runs** of the solver. While the VIATRA Solver implements a novel scalable graph solver technique [5] as the core model generation approach, it can also generate models use existing logic solvers

(e.g. `Alloy` or `Z3`) by exploiting mappings defined in [6]. The designated `solver` can be selected using content assist.

Finally, the configuration file may also contain an `output` folder and a CSV file for storing various statistics (`stats`) of the model generation process (see next section).

c) Overview of the model generation approach: As a key innovation reported in [5], our approach operates natively over graph models by innovatively combining the concepts of partial models [7], neighborhood shapes [8], incremental graph pattern matching [9] with standard SAT-solving procedures.

Model generation starts from an abstract partial model or a seed model fragment specified by the user. Partial models are grown by decision and unit propagation rules adapted from core SAT-solving techniques. Each step during model generation refines and extends a previous partial model while continuously ensuring that consistency constraints are not surely violated. These checks necessitate 3-valued graph pattern matching over partial models (i.e. to detect if a pattern *may*, *must* or *cannot* match on an refinement), which is approximated by rewriting constraints into regular 2-valued incremental constraint checks supported by VIATRA [9], [3].

Model generation is carried out as a design space exploration process where (the hash code of) each candidate partial model is stored. Moreover, neighborhood shapes and predicate abstractions help enforce the synthesis of structurally diverse models. If a constraint violation is no longer repairable, or a partial model is already visited before, the model generation process backtracks. For debugging purposes, the states of the model generation process can also be visualized as a graph.

d) Outputs: Model generation can be initiated from an Eclipse menu or from the command line. Our framework provides various types of output as result:

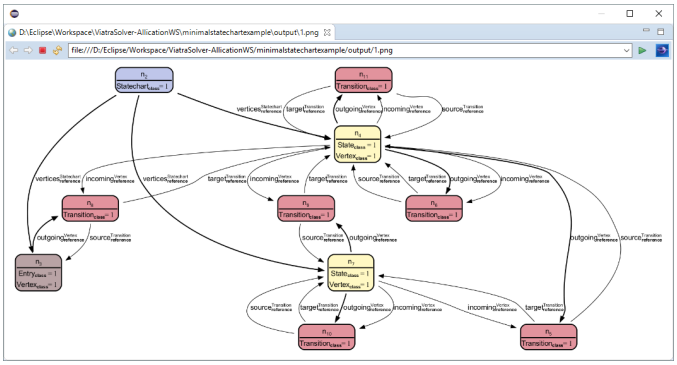

Fig. 4: Sample visualized output

- **EMF instance models**: First, EMF-compliant instance models are derived, which are then serialized in a standard XMI format. These models can be loaded with off-the-shelf modeling environments, automatically processed by the underlying tooling, or used as test inputs.
- **Graph visualizations**: For manual inspection, a graph visualization of the derived models is available in two formats: as a PNG file as well as in a standard GML formal, which is widely accepted by popular and powerful graph drawing (and layouting) tools to improve its readability. A sample visualisation illustrated in Figure 4.
- **Statistics**: The solver can record various statistics about the model generation process, most importantly, the execution times for a campaign. This feature is very useful when using our solver as part of an experimental evaluation for scientific papers - already used by the authors themselves when preparing [10], [11], [5].

III. Results of Validation Studies

The VIATRA Solver have been successfully applied in multiple research projects with industrial collaborators.

- In the Trans-IMA project [12], the VIATRA Solver helped carry out a wide range of validation tasks on an avionic modeling environment by automated theorem proving (with Z3 as a back-end SMT solver, and proving that there is no counterexample for expected language properties). We showed consistency, unambiguity and completeness of the fragment of the DSL [6] by detecting design flaws in the specification.
- In the R3-COP ARTEMIS project[1], the VIATRA Solver supported the testing of autonomous and cooperative robot systems with iterative test context generation.
- The VIATRA Solver was used as background solver for backward change propagation technique in the toolchain of a remote health care case study in the Concerto project[2], which developed an environment for pulse and blood pressure measurement controlled by a smart phone.

In [5] we evaluated the scalability of our approach in the context of 6 test sets of four different domains with 5 minute timeout. (1) A small File System (*FS*) example was taken from the Alloy documentation. *Ecore*, the meta-metamodeling language of EMF, has been used as a case study by different approaches [13], [14], [15] using Alloy as a background solver for model generation purposes. Our measurements also covered two DSLs that were developed in industrial projects, namely, (3) *Yakindu* [4] and (4) Functional Architecture Model (*FAM*) developed for avionics [12]. In addition to their direct practical relevance, these DSLs have already been used in the context of model generation in numerous papers [16], [6], [11] in the past. According to our scalability experiments (illustrated in Table I), the graph solver of the VIATRA Solver (**GS**) is capable of generating consistent graph models of 1-2 orders of magnitude larger (with 1000-6000 nodes) compared to models derived by Alloy (regardless of the underlying background solver **Sat4J** or **MiniSat**).

Furthermore, we proposed distance and diversity metrics in [11] to characterize models sequences generated by graph generators. We executed diversity measurements in the *Yakindu* statechart domain (see a brief extract in Figure 5). According to these diversity metrics, models generated by our solver

[1]R3-COP (Resilient Reasoning Robotic Co-operative Systems). ARTEMIS project n 100233, www.r3-cop.eu/

[2]CONCERTO ARTEMIS project. www.concerto-project.org/

	Problem size			Largest model (#Objects)		
	#Class	#Ref	#WF.	GS	Sat4J	MiniSat
FS	4	4	7	4750	87	89
Ecore	19	33	24	2000	38	41
FAM	9	15	23	6250	58	61
Yakindu	10	6	25	1000	–	–

TABLE I: Maximal model size comparison

(**GS**) produced more diverse models than Alloy (**A**) compared with multiple configuration affecting diversity (**s=0..20**). In this measurement, we also included 1250 manually created statechart models (**Human**), which performed between (**GS**) and (**A**). As a summary, our model generation technique significantly outperformed Alloy and manually created models with respect to diversity.

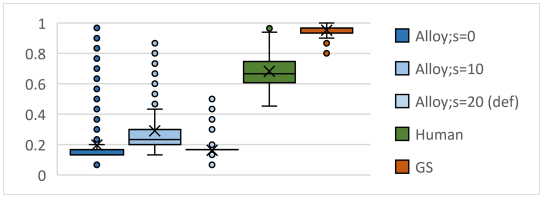

Fig. 5: Internal diversity [11] of Graph Solver and Alloy

IV. Related Work

Existing generators of consistent models (such as EMFtoCSP [16], USE [14] or Formula [17], Clafer [18]) usually take the high-level specification of the modeling language and translate it to a logic representation and then derive consistent (graph-)models using back-end logic solvers (like KodKod [19], Korat [20] or the Z3 SMT-solver [21]). Unfortunately, despite their conceptual elegance, existing techniques only scale for tree-like models [22] but they do not scale for complex graph structures. Random model generators (like RandomEMF) scale much better, but they cannot provide any guarantees for consistent models. Only very recent research results such as [23], [5] attempted to generate consistent models natively, i.e. without mapping them into backend solvers. A detailed comparison of model generator approaches and expected graph properties is provided in [10] that covers graph generators developed in other disciplines (like graph databases or network science).

While all these tools are able to generate consistent models, favorable scalability was only reported in [23], [5] to synthesize graph models with at least 1,000 nodes. Thanks to the innovative foundational techniques and algorithms reported in [5], [10], [11], the VIATRA Solver provides *unique incrementality, completeness and diversity guarantees*: (1) it can enumerate all different models with respect to a customizable equivalence class (based on neighborhood shapes), and (2) if no models are derived up to a certain size, then search can continue from these partial solutions. Finally, (3) the derived set of models can be more diverse than other approaches using logic solvers in the backend, thus our solver is more appropriate to be used e.g. for mutation testing scenarios.

References

[1] Viatra Solver Project, http://github.com/viatra/VIATRA-Generator.

[2] J. Whittle, J. Hutchinson, and M. Rouncefield, "The state of practice in model-driven engineering," *IEEE Software*, vol. 31, no. 3, pp. 79–95, 2014.

[3] D. Varró, G. Bergmann, Á. Hegedüs, Á. Horváth, I. Ráth, and Z. Ujhelyi, "Road to a reactive and incremental model transformation platform: three generations of the VIATRA framework," *Software and Systems Modeling*, vol. 15, no. 3, pp. 609–629, 2016.

[4] Yakindu Statechart Tools, *Yakindu*, 2017, http://statecharts.org/.

[5] O. Semeráth, A. S. Nagy, and D. Varró, "A graph solver for the automated generation of consistent domain-specific models," in *40th Int. Conf. on Software Engineering*. Gothenburg, Sweden: ACM, 2018.

[6] O. Semeráth, A. Barta, A. Horváth, Z. Szatmári, and D. Varró, "Formal validation of domain-specific languages with derived features and well-formedness constraints," *Software and Systems Modeling*, vol. 16, no. 2, pp. 357–392, 2017.

[7] M. Famelis, R. Salay, and M. Chechik, "Partial models: Towards modeling and reasoning with uncertainty," in *34th Int. Conf. on Software Engineering*. Piscataway, NJ, USA: IEEE Press, 2012, pp. 573–583.

[8] A. Rensink and D. Distefano, "Abstract graph transformation," *ENTCS*, vol. 157, no. 1, pp. 39–59, 2006.

[9] Z. Ujhelyi, G. Bergmann, Á. Hegedüs, Á. Horváth, B. Izsó, I. Ráth, Z. Szatmári, and D. Varró, "EMF-IncQuery: An integrated development environment for live model queries," *Sci. Comput. Program.*, vol. 98, pp. 80–99, 2015.

[10] D. Varró, O. Semeráth, G. Szárnyas, and Á. Horváth, "Towards the automated generation of consistent, diverse, scalable and realistic graph models," in *Graph Transformation, Specifications, and Nets*. Springer, 2018, vol. 10800 LNCS, pp. 285–312.

[11] O. Semeráth and D. Varró, "Iterative generation of diverse models for testing specifications of dsl tools," in *Fundamental Approaches to Software Engineering*. Springer, 2018, pp. 227–245.

[12] Á. Horváth, Á. Hegedüs, M. Búr, D. Varró, R. R. Starr, and S. Mirachi, "Hardware-software allocation specification of IMA systems for early simulation," in *2014 IEEE/AIAA 33rd Digital Avionics Systems Conference (DASC)*, Oct 2014, pp. 4D3–1–4D3–15.

[13] F. Büttner, M. Egea, J. Cabot, and M. Gogolla, "Verification of ATL transformations using transformation models and model finders," in *14th International Conf. on Formal Engineering Methods,ICFEM'12*. LNCS 7635, Springer, 2012, pp. 198–213.

[14] M. Kuhlmann, L. Hamann, and M. Gogolla, "Extensive validation of OCL models by integrating SAT solving into use," in *TOOLS'11 - Objects, Models, Components and Patterns*, ser. LNCS, vol. 6705, 2011, pp. 290–306.

[15] M. Soeken, R. Wille, M. Kuhlmann, M. Gogolla, and R. Drechsler, "Verifying UML/OCL models using boolean satisfiability," in *Design, Automation and Test in Europe*. IEEE, 2010, pp. 1341–1344.

[16] C. A. González, F. Büttner, R. Clarisó, and J. Cabot, "EMFtoCSP: a tool for the lightweight verification of EMF models," in *Formal Methods in Software Engineering Rigorous and Agile Approaches*, 2012, pp. 44–50.

[17] E. K. Jackson, T. Levendovszky, and D. Balasubramanian, "Automatically reasoning about metamodeling," *Software & System Modeling*, vol. 14, pp. 271–285, 2015.

[18] K. Bak, Z. Diskin, M. Antkiewicz, K. Czarnecki, and A. Wasowski, "Clafer: unifying class and feature modeling," *Software & Systems Modeling*, vol. 15, pp. 811–845, 2016.

[19] E. Torlak and D. Jackson, "Kodkod: A relational model finder," in *Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2007, pp. 632–647.

[20] H. Zhong, L. Zhang, and S. Khurshid, "Combinatorial generation of structurally complex test inputs for commercial software applications," in *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE)*. ACM Press, 2016, pp. 981–986.

[21] L. de Moura and N. Bjørner, "Z3: An efficient SMT solver," in *Tools and Algorithms for the Construction and Analysis of Systems, 14th Int. Conf. (TACAS 2008)*, ser. LNCS, vol. 4963. Springer, 2008, pp. 337–340.

[22] B. Elkarablieh, Y. Zayour, and S. Khurshid, "Efficiently generating structurally complex inputs with thousands of objects," in *ECOOP 2007 Object-Oriented Programming*. Springer, 2007, pp. 248–272.

[23] G. Soltana, M. Sabetzadeh, and L. C. Briand, "Synthetic data generation for statistical testing," in *32nd IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*. IEEE, oct 2017, pp. 872–882.