Refinery: Graph Solver as a Service

Refinement-based Generation and Analysis of Consistent Models

Kristóf Marussy∗ Attila Ficsor∗ Oszkár Semeráth∗
marussy@mit.bme.hu ficsor@mit.bme.hu semerath@mit.bme.hu
Budapest University of Technology and Economics, Department of Measurement and Information Systems, Budapest, Hungary

Dániel Varró
daniel.varro@liu.se
Linköping University, Department of Computer and Information Science, Linköping, Sweden

McGill University, Department of Electrical and Computer Engineering, Montreal, Canada

ABSTRACT

Various software and systems engineering scenarios rely on the systematic construction of consistent graph models. However, automatically generating a diverse set of consistent graph models for complex domain specifications is challenging. First, the graph generation problem must be specified with mathematical precision. Moreover, graph generation is a computationally complex task, which necessitates specialized logic solvers. Refinery is a novel open-source software framework to automatically synthesize a diverse set of consistent domain-specific graph models. The framework offers an expressive high-level specification language using partial models to succinctly formulate a wide range of graph generation challenges. Moreover, it provides a modern cloud-based architecture for a scalable graph solver as a service, which uses logic reasoning rules to efficiently synthesize a diverse set of solutions to graph generation problems by partial model refinement. Applications include system-level architecture synthesis, test generation for modeling tools or traffic scenario synthesis for autonomous vehicles.

Video demonstration: https://youtu.be/Qy_3udNsW5M
Continuously deployed at: https://refinery.services/

KEYWORDS

Model generation, Partial modeling, Logic solver, Cloud service

1 INTRODUCTION

Motivation and challenge. Model-based systems engineering is a popular approach in the industry for the design of critical software-intensive cyber-physical systems [31] supported by a variety of modeling tools (like Artop, Capella, Yakindu, OpenModelica and many closed-source alternatives). These tools help reveal design flaws early, thus reducing costs and improving product quality. In this context, there is an increasing need for a diverse set of synthetic graph models to represent test cases and benchmarks for modeling tools or candidate designs in systems engineering. However, such synthetic graph models need to be consistent to comply with the underlying domain-specific standards (e.g. AUTOSAR, SysML) captured in the form of metamodels and well-formedness constraints.

However, the automated synthesis of a diverse set of consistent domain-specific graph models is very challenging. (A) Random model generators can derive large and diverse graphs, but the derived graphs are not consistent. (B) Search-based model generators [17, 24] can derive large and consistent graphs using evolutionary algorithms but without any guarantees for completeness or diversity. (C) Solver-based model generators [6–8, 32] map the graph generation problem to a SAT- or SMT-problem in the background to detect inconsistencies of specifications, but they fail to derive a diverse set of graphs with more than a few hundred of graph nodes. Conceptually, a graph solver can also be regarded as an SMT-solver for the domain of complex graph (or relational) data, but the main focus is to derive a diverse set of consistent models (if they exist).

Objective and scope. The Refinery framework supports the efficient generation of consistent and diverse domain-specific graph models. It offers (1) a high-level specification language to capture the domain and control the range of graphs requested by end users, (2) a semantically well-founded graph generation approach based on refinement of partial models using 4-valued logic, and (3) a modern cloud-based architecture that provides a partial modeling editor, a partial model reasoner and a graph solver for engineers made available in a web browser or programmatically as a Java library.

First, the end user needs to provide a domain specification, which consists of a metamodel, an (optional) initial partial model, a set of predicates and constraints and a scope definition to restrict the size of the generated models. Then server-side automated graph generation back-end can be initiated by a push of a button (or
programmatically). The diverse set of auto-generated consistent graph models can be visualized or serialized in a textual format.

A main use case of Refinery is to synthesize graphs as test cases in applications such as the testing of modeling tools or system-level testing of autonomous vehicles [2]. Refinery also helps experts to semi-automatically provide a variety of consistent design candidates with complex graph structures as part of design space exploration [14], partial modeling [5], or feature modeling [12].

Envisioned users. The Refinery framework primarily aims to target systems and software engineers to derive complex test suites for industrial modeling tools (like Artop, Capella, Yakindu, OpenModelica and many closed source alternatives). An ongoing initiative is to provide test models for the new SysML standard [13]. However, the modern web interface enables the use of the framework by other domain experts, e.g. safety experts developing test scenarios in autonomous vehicles, or Blockchain experts (see section 4). During the development of the framework, we have received regular feedback from researchers at Budapest University Technology and Economics (Hungary), McGill University (Canada) and Linköping University (Sweden) and engineers at IncQuery Labs.

2 CONCEPTUAL OVERVIEW AND USAGE
Our framework offers a high-level specification language for model generation using refinement of 4-valued partial models.

2.1 Specification Language
The framework provides a concise yet precise specification language for generating graphs based on the syntax for partial models proposed in [10]. The language allows to control the range of generated models using four kinds of language elements:

A domain specification in Refinery captures the key concepts and relations of the domain using an essential subset of XCore [27], a popular textual metamodeling language integrated with Eclipse Modeling Framework [26]. In a domain specification, the user can declare classes and associations as relational symbols (denoted by \( s \) in the grammar below), while a large set of logic constraints imposed by the structure of the metamodel is automatically translated to assertions and predicates (including the type hierarchy, multiplicities, and containment hierarchy, as illustrated in [10]).

\[
\text{domain} := (\text{abstract}\?\text{class}(s)\text{extends}(s)(\text{..}\text{)})*\{ \text{contains}(s)\text{[min]}\text{..}[max]?(s)\text{opposite}(s))\}\text{.}
\]

For example, graphs representing file structures may provide classes such as FileSystem, File, Directory, and Symbolic Links (SymLink). A FileSystem contains a File as a root; each Directory contains multiple Files, and a SymLink can refer to other Files.

```
class FileSystem { contains File[1] root }  
class File. 
class Dir extends File {contains File[0..*]element}  
class SymLink extends File {File[1] target }
```

Assertions define facts in an instance model (similarly to Prolog or Datalog). Each assertion assigns a truth-value to a relational term (which represents a graph node or edge).

\[
\text{assertion} := (s)\text{([term]}\text{,}([\text{term}])\text{)}*:\text{truth-value}.
\]

A non-annotated (truth-value) denotes true value assignment, and the ! symbol denotes false value assignment. For example, one can prescribe that a directory called resources exists in a graph, which contains an img file and a symbolic link pointing to the image, while one can state that the image file cannot be a Dir:

\[
\text{Dir} \text{(resources)}\text{. element(resources, img).} \\
\text{element(resources, link). target(link, img). !Dir(img).}
\]

Logic predicates provide custom model views while constraints (error patterns) allow to further restrict the range of valid graphs. A logic predicate in Refinery declares a new n-ary relational symbol (with header variables \((v)\)), and defines a constraint formulated as a disjunction of multiple bodies (separated by the ";" character), which are composed as a conjunction of literals (separated by the "," character). A literal refers to the truth-value of a symbol with variables: by default, the literal refers to "s" denotes transitive closure. The keyword error denotes error patterns in a valid model, such predicates must be false for each node [4].

\[
\begin{align*}
\text{error} \text{pred selfLoop}(s) & \text{<=> target}(s,s). \\
\text{error} \text{pred emptyDir}(d) & \text{<=> Dir}(d), \text{!element(d,_)}. \\
\text{pred important}(f) & \text{<=> target}(l1,f), \text{target}(l2,f), l1=l2.
\end{align*}
\]

The scope controls the size of the generated models by defining the minimum and maximum number of instances (or predicate occurrences) of the scoped symbol with a true truth-value.

\[
\text{scope} := \text{scope}(s)\text{=(min)}\text{..}[max]\,(s)\text{=(min)}\text{..}[max])\text{.}
\]

One may generate models with 25 to 30 nodes, two file systems, and at least one match for the predicate detecting important files:

\[
\begin{align*}
\text{scope node} & = 0..30, \text{FileSystem}=2, \text{important}=1..*
\end{align*}
\]

2.2 Generation with 4-Valued Partial Models
Model generation can be initiated by the user (Generate button) to derive a consistent model of the specification, if such a model exists. Simple inconsistencies can be highlighted in Refinery by error markers on the derived graph. To obtain a diverse set of graphs, each newly generated graph is structurally different from previous ones ensured by shape-based graph diversity metrics [22].

The Refinery framework uses 4-valued logic [3, 9] to explicitly represent incomplete, partial (paracomplete) models, and to tolerate errors and inconsistencies (paraconcistency) arising during the evaluation of computations over such models. 4-valued logic contains the usual false and true truth values, the unknown value introduced for uncertain (unspecified) properties, and the error value that signals inconsistencies. The subset \{false, true, unknown\} of logic values can express partial, but potentially consistent information (such as incomplete models). Conversely, the subset
3 ARCHITECTURE

The architecture of the Refinery graph solver as a service is illustrated in Figure 2. Refinery follows a modern multi-tier software architecture: graph generation problems are input via the frontend web application or (as a Java library), while the backend comprises auto-scalable service deployed as containers behind a load balancer.

3.1 Frontend

Web application. A Single-Page Application (SPA) was created for editing and visualizing partial models and initiating model generation. To provide editor support, we opted to perform the bulk of the parsing and semantic analysis of the partial models in the backend in order to reuse the analyses already required by the model generation. In particular, we display the logical consequences of the statements in the partial model by executing propagation operations on the backend immediately after the user edits the partial model.

The SPA establishes a WebSocket connection and sends editing operations (text deltas) from the textual partial model editor (based on the CodeMirror framework) to the backend. Features like syntax error checking, content assist, find occurrences, semantic highlighting, and automatic formatting are initiated via the WebSocket when the editor is idle or upon user request.

Additionally, when the partial model description is free of syntactic errors, the corresponding partial model semantics, including any discovered (semantic) inconsistencies are obtained back from the server. The user may apply further filtering (e.g., hide some nodes or relations) before visualizing the model as a graph using Graphviz and D3. When the user initiates graph generation, the generated solution is also visualized. Subsequent generation requests return different solutions by choosing a different random seed.

Client library. Refinery is also available as a library in the Java programming language. Model generation tasks may be programmatically submitted using our textual partial modeling language. Alternatively, more direct interaction with the partial model management library and the model generator is available for more specialized use-cases, such as iterative model generation [23].

3.2 Backend

The backend consists of three main components: the (1) partial model editor, the (2) partial model reasoner, and the (3) graph solver for generating consistent graph models.

Container-based packaging provides easy deployment to cloud providers, such as AWS. Components (1)+(2) may be deployed as a single Docker container to serve as a backend of the web-based editor, while components (1)+(2)+(3) deployed together in a container enable model generation. In both cases, Refinery does not rely on any other server-side state. Thus, it can be automatically scaled behind an load balancer that can handle WebSocket connections.

A Docker image, containing (1)+(2)+(3) as a monolith is also available for local or on-premises deployments (e.g., for use-cases where the resource limits provided by the cloud service are insufficient).

Partial model editor. The editor contains a web server to handle incoming WebSocket connections and uses Eclipse Xtext to parse partial models and provide syntactic analysis features. Then partial models are transformed into an internal semantic representation according to the Refinery language semantics.

In order to enable multiple concurrent users per server instance, we implemented an optimized, WebSocket-based protocol over the Xtext Web feature of the Xtext framework to maintain a server-side copy of the edited partial models. This allows to substantially lower resource utilization per user and latency on initial connection by serving multiple users with a single backend instance.

Partial model reasoner. The core of partial model reasoning in Refinery is an efficient model management library that enables the compact representation of multiple versions of partial models [25] as relational logic structures. This component can also be used as a standalone Java library to store and query partial models.

References

1. https://codemirror.net/
2. https://graphviz.org/
3. Graph solver

The Web-based partial model editor is connected to a Graph generation library in a host language. The frontend of the Refinery graph solver as a service is depicted in Figure 2. It consists of three main components: 1. Partial modeling editor, 2. Partial model reasoner, and 3. Graph solver.

### Table 1: Largest models generated in 60 seconds

<table>
<thead>
<tr>
<th>Problem size</th>
<th>Largest model</th>
</tr>
</thead>
<tbody>
<tr>
<td>#class</td>
<td>#ref</td>
</tr>
<tr>
<td>FileSys</td>
<td>4</td>
</tr>
<tr>
<td>FAM</td>
<td>9</td>
</tr>
<tr>
<td>Yakindu</td>
<td>11</td>
</tr>
<tr>
<td>Ecore</td>
<td>20</td>
</tr>
<tr>
<td>Social</td>
<td>2</td>
</tr>
</tbody>
</table>

Consistency checking and refinement of partial models requires reasoning about the type system describing the specified and unspecified parts of graphs and the graph constraints that capture consistency rules. To this end, we integrated an incremental graph query engine based on Viatra Query [4] in combination with advanced approximation techniques for partial query evaluation [21].

Propagation steps are partial model refinements that encode logical consequences of the type system and constraints. Refinery derives a refined partial model and analyzes its consistency every time the user edits the partial model. Numerical constraints are handled by external solvers, including GLOP3 for linear constraints.

Graph solver. The solver takes the initial partial model provided by the user and generates a set of consistent graphs as output [21].

At each decision step, a new partial model is derived (as a new exploration state) by decreasing the number of uncertain nodes and edges in the partial model while simultaneously increasing its size.

State space exploration needs to repeatedly detect if a partial graph has already been reached (special isomorphism detection), and if a graph constraint is surely violated, when no consecutive refinements will ever lead to a consistent model. We rely on graph shapes [16] to detect isomorphic graphs and enforce diversity [22].

### 4 EVALUATION

Scalability evaluation. An initial evaluation of the Refinery framework is provided in Table 1 across five different domains used as case studies in previous research [2, 11, 20]. The number of classes, associations, and constraints in each domain is listed in Table 1 together with the largest model successfully generated as follows. In this initial experiment, each generation run had a 60-second timeout. We incremented the model size by 250 until the generation for a given size timed out five times in a row. In the case of the Social Network domain, because of the limited size of the generated models, we increased the number of objects by 10. Finally, it is worth pointing out that model generation runs for each of these domains are available as part of the integration test suite of Refinery.

Theoretical properties. The Graph solver algorithm provides multiple formal guarantees [30], including correctness (a model is generated, then it satisfies the constraints) and completeness (if a model satisfies the constraints, it will be generated eventually).

Uses cases in research and education. The development of the Refinery framework has been supported by an Amazon Research Award, and it has been successfully used in several practical applications, which include (1) the automated synthesis of test scenes for autonomous vehicles, (2) generation of dependable blockchain architectures, and (3) automated synthesis of system architectures for early mission planning. The framework has also been used by MSc students as part of an advanced modeling lab offered at Budapest University of Technology and Economics. Moreover, a public tutorial of Refinery has been delivered at ASE 2023 by the authors.

Related work. There are other software tools that offer the automated synthesis of consistent graph models. Most notable examples include Alloy [8], Sterling [29], USE [6], Pledge [1, 24], UMLtoCSP [7], TAF [17] or VIATRA Solver [19]. However, we wish to point out that the size of models generated by Refinery compares very favorably to alternative model generation approaches (see e.g. [2, 11, 20] for detailed measurements of other tools). Compared to our previous work [19], Refinery provides a fundamentally novel, modern cloud-based architecture, a 4-valued partial modeling framework, and new decision procedures for type inference and propagation.

Running example. The example is available at [15].

Acknowledgements. This paper was partially supported by an Amazon Research Award, the European Union as part of the EDGE-Skills (101123471), the UNKP-23-4-II-BME-219 New National Excellence Program of the Ministry for Culture and Innovation from the source of the National Research, Development and Innovation Fund, and the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation.

https://developers.google.com/optimization/lp/