TECHNISCHE UNIVERSITÄT DRESDEN

LUND UNIVERSITY

Johannes Mey[1], René Schöne[1], Görel Hedin[2], Emma Söderberg[2], Thomas Kühn[1], Niklas Fors[2], Jesper Öqvist[2], and Uwe Aßmann[1]

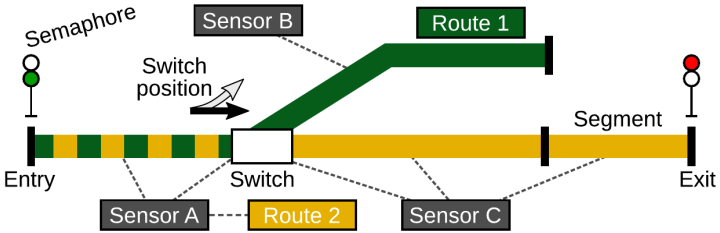
[1]Technische Universität Dresden
[2]Lund University

Continuous Model Validation Using Reference Attribute Grammars

SLE'18, Boston, November 5, 2018

Working with Runtime Models

Running example: Modeling train tracks and routes.



Example model, from [Szárnyas et al., 2017]

Use Case:

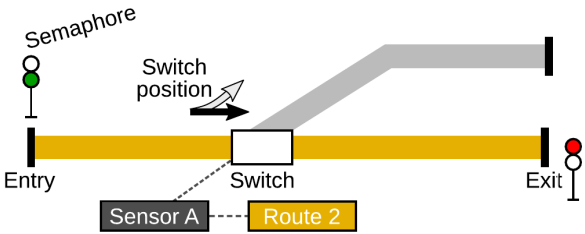
— Modeling editor for rail networks

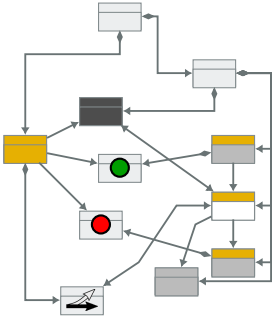
— Continuously *find* and *repair* faults

[Szárnyas et al., 2017] Szárnyas, G., Izsó, B., Ráth, I., and Varró, D. (2017).
The Train Benchmark: cross-technology performance evaluation of continuous model queries. Software & Systems Modeling, pages 1--29.

Working with Runtime Models

Running example: Modeling train tracks and routes.





Example model, from [Szárnyas et al., 2017]

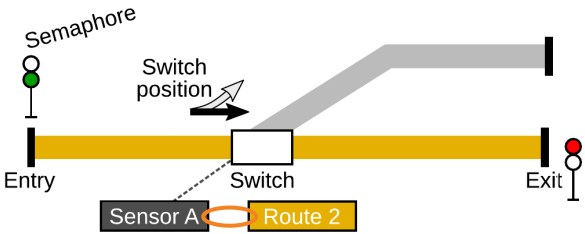
Use Case:

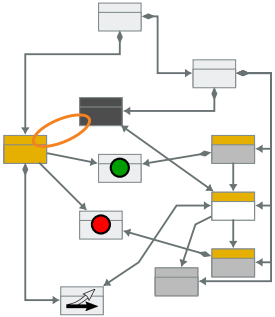
- Modeling editor for rail networks
- Continuously *find* and *repair* faults

[Szárnyas et al., 2017] Szárnyas, G., Izsó, B., Ráth, I., and Varró, D. (2017).
The Train Benchmark: cross-technology performance evaluation of continuous model queries. Software & Systems Modeling, pages 1--29.

Working with Runtime Models

Running example: Modeling train tracks and routes.





Example model, from [Szárnyas et al., 2017]

Use Case:

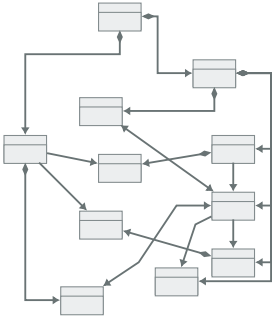
- Modeling editor for rail networks
- Continuously *find* and *repair* faults

[Szárnyas et al., 2017] Szárnyas, G., Izsó, B., Ráth, I., and Varró, D. (2017).
The Train Benchmark: cross-technology performance evaluation of continuous model queries. Software & Systems Modeling, pages 1--29.

Validating Changing Models

Models:

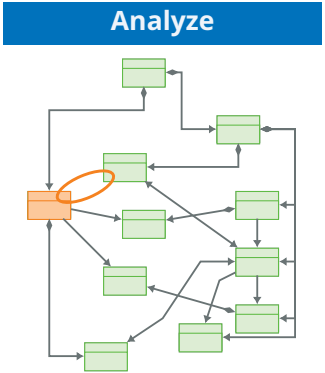
- **Analyze**
  Here: Search for errors/inconsistencies
- **Modify**
  Here: Fix errors/inconsistencies



Validating Changing Models

Models:

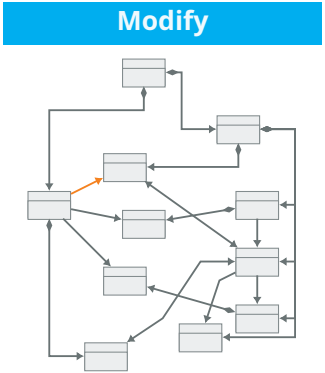
- **Analyze**
  Here: Search for errors/inconsistencies
- **Modify**
  Here: Fix errors/inconsistencies



Validating Changing Models

Models:

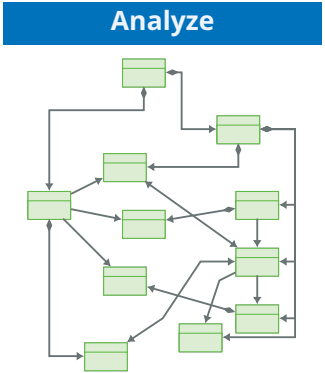
- **Analyze**
  Here: Search for errors/inconsistencies
- **Modify**
  Here: Fix errors/inconsistencies



Validating Changing Models

Models:

- **Analyze**
  Here: Search for errors/inconsistencies
- **Modify**
  Here: Fix errors/inconsistencies



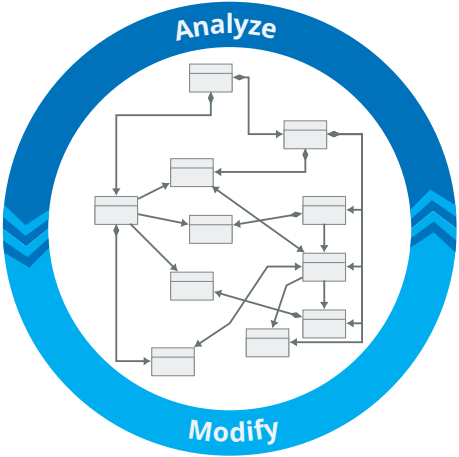
Validating Changing Models

Models:

- **Analyze**
  Here: Search for errors/inconsistencies
- **Modify**
  Here: Fix errors/inconsistencies

Models at runtime:

- Analyze **incrementally**
- Modify **continuously**

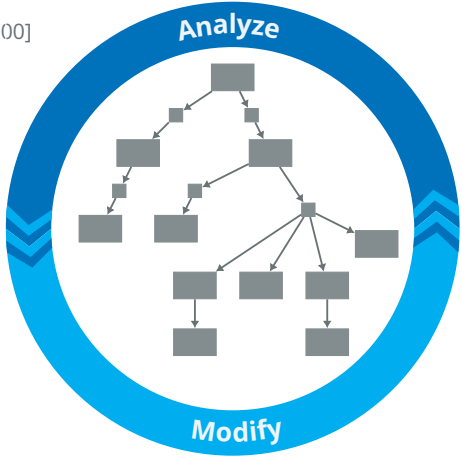


Reference Attribute Grammars as Models

Our approach:

Reference Attribute Grammars (RAGs) [Hedin, 2000]

- Context-free grammar
- Attributes:
  - *Intrinsic*
  - *Computed*: synthesized, inherited, . . .
  - *Reference*: Intrinsic *or* computed
- We use: **JastAdd** [Hedin and Magnusson, 2003]



RAGs for modeling offer:

- Shorthands for navigation and computation on trees
- Efficiency through memoization
- Incremental evaluation

[Hedin, 2000] Hedin, G. (2000). Reference attributed grammars. Informatica (Slovenia), 24(3):301–317.

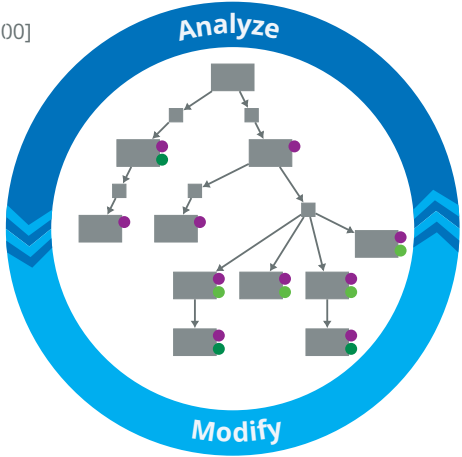
[Hedin and Magnusson, 2003] Hedin, G. and Magnusson, E. (2003). JastAdd: an aspect-oriented compiler construction system. Science of Computer Programming, 47(1):37--58.

Reference Attribute Grammars as Models

Our approach:

Reference Attribute Grammars (RAGs) [Hedin, 2000]

- Context-free grammar
- Attributes:
  - *Intrinsic*
  - *Computed*: synthesized, inherited, . . .
  - *Reference*: Intrinsic *or* computed
- We use: **JastAdd** [Hedin and Magnusson, 2003]



RAGs for modeling offer:

- Shorthands for navigation and computation on trees
- Efficiency through memoization
- Incremental evaluation

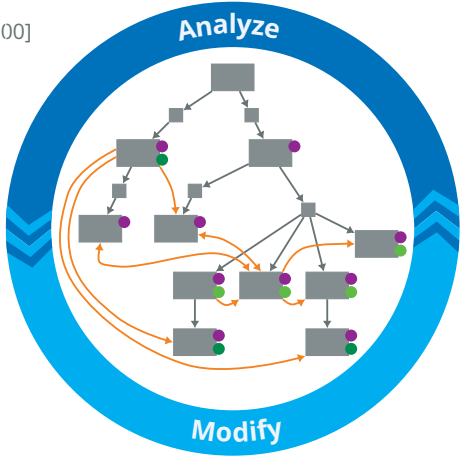
[Hedin, 2000] Hedin, G. (2000). Reference attributed grammars. Informatica (Slovenia), 24(3):301–317.
[Hedin and Magnusson, 2003] Hedin, G. and Magnusson, E. (2003). JastAdd: an aspect-oriented compiler construction system. Science of Computer Programming, 47(1):37--58.

Reference Attribute Grammars as Models

Our approach:

Reference Attribute Grammars (RAGs) [Hedin, 2000]

- Context-free grammar
- Attributes:
  - *Intrinsic*
  - *Computed*: synthesized, inherited, . . .
  - *Reference*: Intrinsic *or* computed
- We use: **JastAdd** [Hedin and Magnusson, 2003]



RAGs for modeling offer:

- Shorthands for navigation and computation on trees
- Efficiency through memoization
- Incremental evaluation

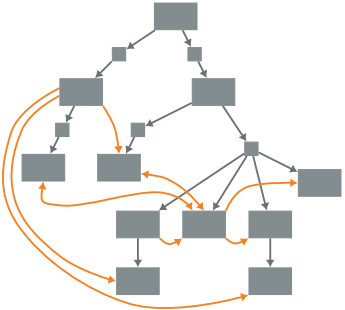
[Hedin, 2000] Hedin, G. (2000). Reference attributed grammars. Informatica (Slovenia), 24(3):301–317.

[Hedin and Magnusson, 2003] Hedin, G. and Magnusson, E. (2003). JastAdd: an aspect-oriented compiler construction system. Science of Computer Programming, 47(1):37--58.

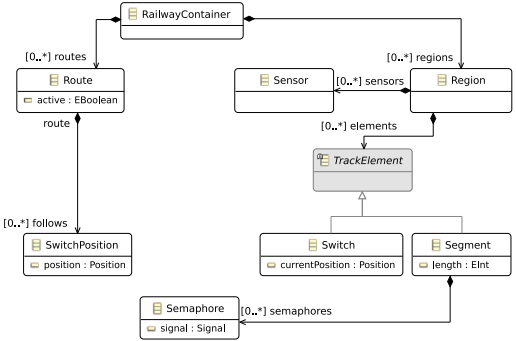
Model-Grammar Mismatch

Relations are different:

- In **models**:
  - Containment relations form *overlay tree*
  - Non-containment relations
  - Bidirectional relations
- In **grammars**:
  - Containment references: **tree**
  - Non-containment references } **this paper**
  - Bidirectional references } **this paper**



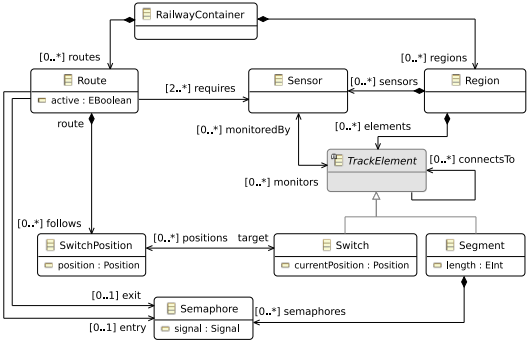
Metamodels and Grammars



Abstract grammar (JastAdd syntax)

```
RailwayContainer ::= Route* Region*;
abstract RailwayElement ::= <Id:int>;
Region : RailwayElement ::= TrackElement* Sensor*;
Semaphore : RailwayElement ::= <Signal:Signal>;
Route : RailwayElement ::= <Active:boolean> SwitchPosition*;
SwitchPosition : RailwayElement ::= <Position:Position>;
Sensor : RailwayElement;
abstract TrackElement:RailwayElement;
Segment : TrackElement ::= <Length:int> Semaphore*;
Switch : TrackElement ::= <CurrentPosition:Position>;
```

Metamodels and Grammars



Abstract grammar (JastAdd syntax)

```
RailwayContainer ::= Route* Region*;
abstract RailwayElement ::= <Id:int>;
Region : RailwayElement ::= TrackElement* Sensor*;
Semaphore : RailwayElement ::= <Signal:Signal>;
Route : RailwayElement ::= <Active:boolean> SwitchPosition*;
SwitchPosition : RailwayElement ::= <Position:Position>;
Sensor : RailwayElement;
abstract TrackElement:RailwayElement;
Segment : TrackElement ::= <Length:int> Semaphore*;
Switch : TrackElement ::= <CurrentPosition:Position>;
```

How to capture non-containment relations?

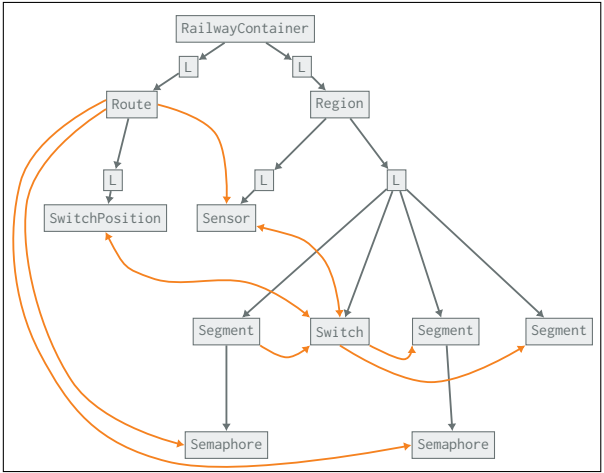
Non-Containment Relations in Detail

Non-Containment References

Cardinality

- 1 : 1
- 1 : {0..1}
- 1 : N

Bidirectional References



Non-Containment Relations in Detail

Non-Containment References:

— **In RAGs:** typed reference nodes: R

Cardinality:

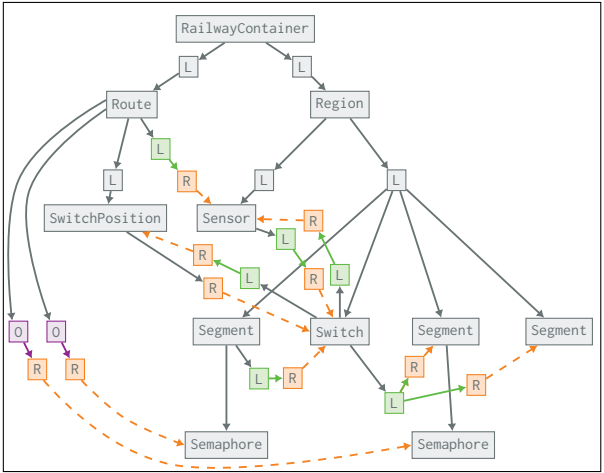
— `1 : 1`

— `1 : {0..1}`

— `1 : N`

— **In RAGs:** *optional* O and *list nodes*: L

Bidirectional References



Non-Containment Relations in Detail

Non-Containment References:

— **In RAGs:** typed reference nodes: R

Cardinality:

— `1 : 1`

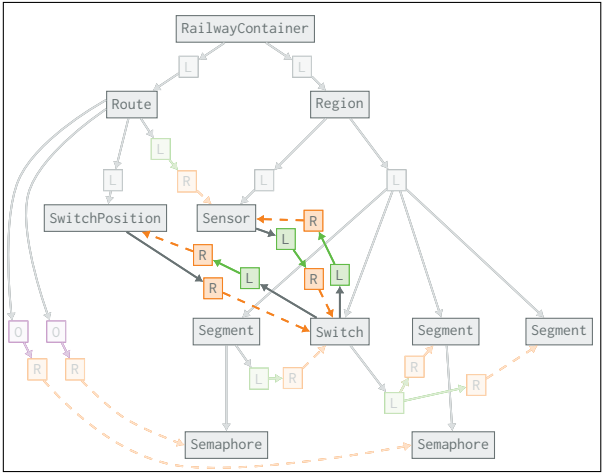
— `1 : {0..1}`

— `1 : N`

— **In RAGs:** *optional* O and *list nodes*: L

Bidirectional References:

— **In RAGs:**
One direction in grammar,
the other in grammar or attribute



Handling Non-containment Relations in RAGs

Approach 1: Name analysis

- Unique identifier `Id` for each object
- Non-containment relations as `Id` uses
- Resolve with name analysis attributes

Approach 2: Explicit intrinsic reference attributes

- Store references as (Java) object references
- Resolve during model loading

Problem: Bidirectional relations:

- Either use *collection attributes* to reverse references (**slow!**)
- Or two unidirectional relations (**risk of inconsistency!**)

Approach 1: Name Analysis

Structure:

— A reference is an **Id**

```
Ref ::= <Value:int>
SensorRef : Ref;
```

Method:

— Collect all elements of a type in *map*

— Resolve reference by *map lookup*

Drawbacks:

— Entire map must be recomputed on model change

— Bidirectional references slow or inconsistent

Map creation:

```
syn Map<Integer, Sensor> RailwayContainer.sensorMap() {
  Map<Integer, Sensor> sensors = new HashMap<>();
  for (Region region : getRegionList()) {
    for (Sensor sensor : region.getSensorList())
      sensors.put(sensor.getId(), sensor);
  }
  return sensors;
}
```

Reference resolving:

```
syn Sensor SensorRef.getSensor() {
  return getRoot().sensorMap().get(this.getValue());
}
```

Approach 2: Intrinsic Reference Attributes

Structure:

— A reference is an **`Id`** *and* an intrinsic reference attribute

```
abstract Ref ::= <Value:int> ;
SensorRef : Ref ::= <Sensor:Sensor>;
```

Method:

— Name analysis during load

— Resolve reference by accessing intrinsic reference attribute

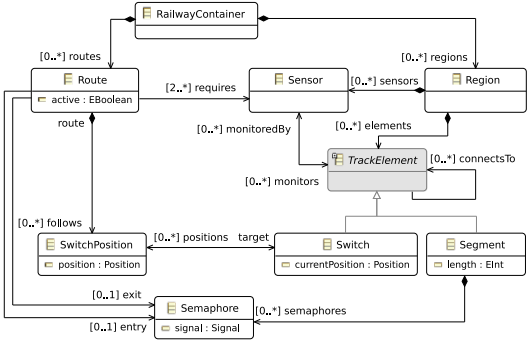
Generated API:

```
public class Sensor {
  // ...
  public Sensor getSensor() {
    // ...
  }
  // ...
}
```

Drawback:

— Bidirectional references slow or inconsistent

Solution: Extending the Grammar



Abstract grammar

```
RailwayContainer ::= Route* Region*;
abstract RailwayElement ::= <Id:int>;
Region : RailwayElement ::= TrackElement* Sensor*;
Semaphore : RailwayElement ::= <Signal:Signal>;
Route : RailwayElement ::= <Active:boolean>
            SwitchPosition*;
SwitchPosition : RailwayElement ::= <Position:Position>;
Sensor : RailwayElement;
abstract TrackElement:RailwayElement;
Segment : TrackElement ::= <Length:int> Semaphore*;
Switch : TrackElement ::= <CurrentPosition:Position>;
```

Extending RAGs with relations

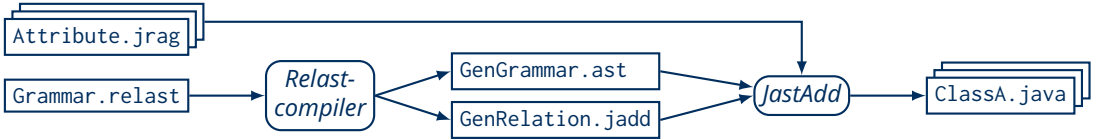
```
rel Route.requires* -> Sensor;
rel Route.entry? -> Semaphore;
rel Route.exit? -> Semaphore;
rel SwitchPosition.target <-> Switch.positions*;
rel Sensor.monitors* <-> TrackElement.monitoredBy*;
rel TrackElement.connectsTo* -> TrackElement;
```

Solution: The RelAST Preprocessor

A preprocessor for JastAdd

Automatically generates

- Grammar with non-containment relations
- Accessor attributes
- Setter attributes
  - Ensuring consistency for bidirectional relations



Evaluation

We investigated:

1. **Usability and Conciseness**
   - Measure complexity reduction
2. **Performance**
   - Compare the three approaches with model- and graph-based solutions

Use Case:

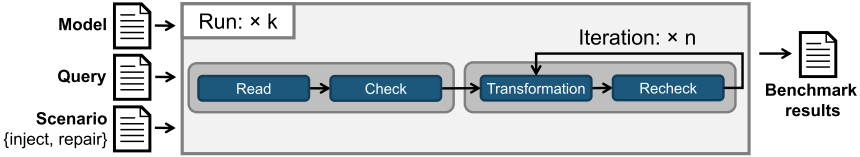
— Iterative model analysis and transformation with *Train Benchmark* [Szárnyas et al., 2017]

— Six model queries

— Fault injection and repair transformations for each

[Szárnyas et al., 2017] Szárnyas, G., Izsó, B., Ráth, I., and Varró, D. (2017).
The Train Benchmark: cross-technology performance evaluation of continuous model queries. Software & Systems Modeling, pages 1--29.

The Train Benchmark



Benchmark process, from [Szárnyas et al., 2017]

Read Load a model from file

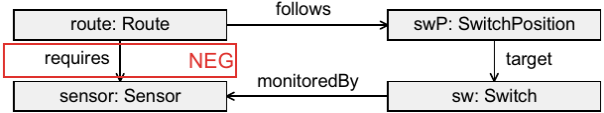
Check Perform a consistency check

Transformation Fix some faults found during the preceding *check*

Recheck Repeat the check on the transformed model

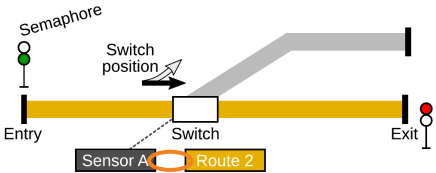
Example Query: RouteSensor

RouteSensor:



RouteSensor pattern, from [Szárnyas et al., 2017]

Example:



— If a switch is *monitored by* a sensor, the route that uses the switch must *require* the sensor.

— If the above pattern is found, there is an error in the model.

Usability and Conciseness

Code Length:

— Halved LOC of utility attributes

Code Complexity:

— Reduced complexity by allowing consistent editing of bidirectional relations

Diff of `inject` transformation (Approaches 1/2 → Grammar extension):

```
 public void inject(Collection<Match> matches) {
-  List<SensorRef> refsToRemove = new ArrayList<>();
   for (final Match match : matches) {
-    for (SensorRef ref : match.getRoute()
-         .getRequiredSensors()) {
-      if (ref.getSensor() == match.getSensor()) {
-        refsToRemove.add(ref);
-      }
-    }
-  }
-  for(SensorRef ref : refsToRemove) {
-    ref.removeSelf();
+    match.getRoute().removeRequiredSensor(
+       match.getSensor());
   }
   driver.flushCache();
 }
```

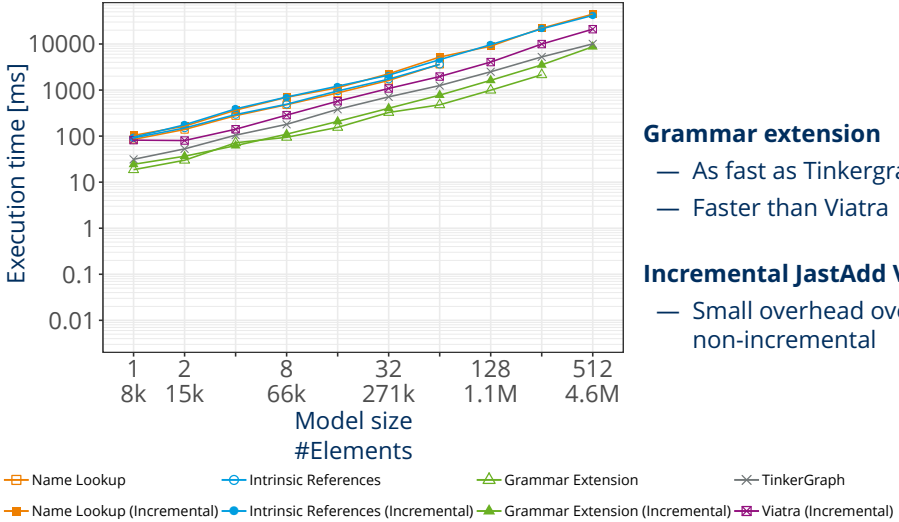
Performance

Comparing:

- Three RAG variants
  - Name lookup
  - Intrinsic Reference Attributes
  - Grammar extension
- A non-incremental graph database solution
  - Tinkergraph from [The Apache Software Foundation, 2018]
- An incremental Java modeling solution
  - Viatra from [Bergmann et al., 2015]

[Bergmann et al., 2015] Bergmann, G., Dávid, I., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., and Varró, D. (2015). Viatra 3: A reactive model transformation platform. In Kolovos, D. and Wimmer, M., editors, Theory and Practice of Model Transformations, pages 101--110, Cham. Springer International Publishing.
[The Apache Software Foundation, 2018] The Apache Software Foundation (2018). Apache tinkerpop.http://tinkerpop.apache.org/.

Read+Check: Route Sensor (Repair)



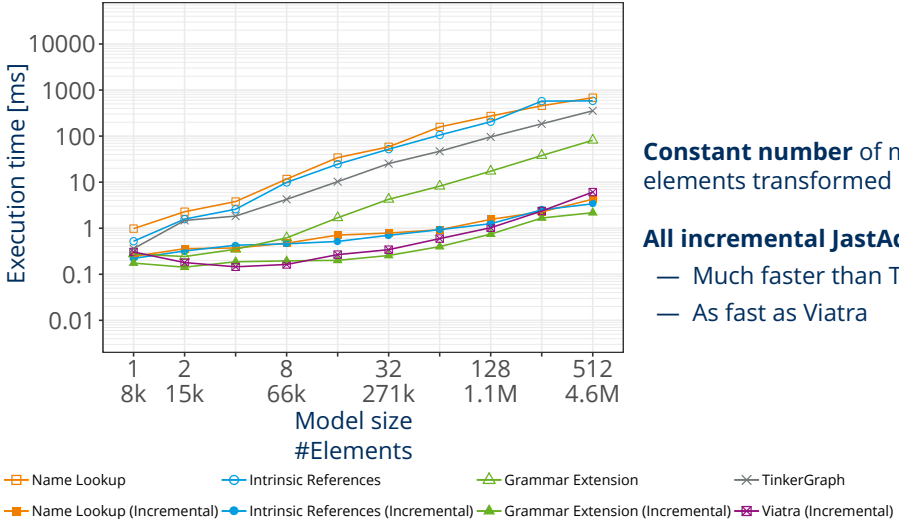
Grammar extension

- As fast as Tinkergraph
- Faster than Viatra

Incremental JastAdd Variants

- Small overhead over non-incremental

Transformation+Recheck: Route Sensor (Inject)

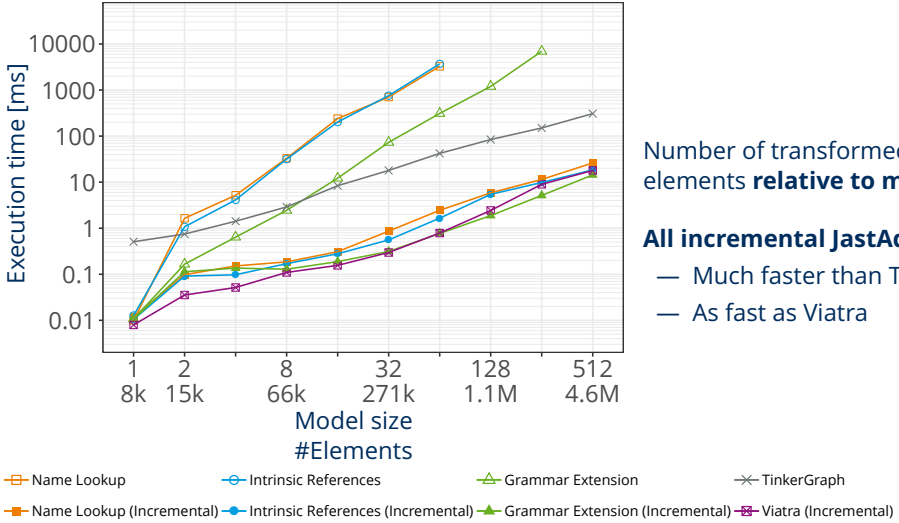


Constant number of model elements transformed in iteration

All incremental JastAdd

- Much faster than Tinkergraph
- As fast as Viatra

Transformation+Recheck: Route Sensor (Repair)



Number of transformed model elements **relative to model size**

All incremental JastAdd

- Much faster than Tinkergraph
- As fast as Viatra

Results

- RAGs can be used for analysis of models
- An extension allows easy specification of non-containment relations
- Performance on par with graph and incremental modeling approaches

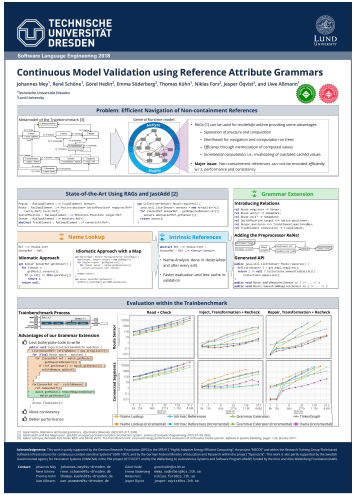
Future work:

- Directly support *ecore* models
- Built-in pattern matching

Thank you for your attention!

Please visit us at our poster!

Johannes Mey	johannes.mey@tu-dresden.de
René Schöne	rene.schoene@tu-dresden.de
Görel Hedin	gorel.hedin@cs.lth.se
Emma Söderberg	emma.soderberg@cs.lth.se
Thomas Kühn	thomas.kuehn3@tu-dresden.de
Niklas Fors	niklas.fors@cs.lth.se
Jesper Öqvist	jesper.oqvist@cs.lth.se
Uwe Aßmann	uwe.assmann@tu-dresden.de



References

Bergmann, G., Dávid, I., Hegedüs, Á., Horváth, Á., Ráth, I., Ujhelyi, Z., and Varró, D. (2015).
Viatra 3: A reactive model transformation platform.
In Kolovos, D. and Wimmer, M., editors, *Theory and Practice of Model Transformations*, pages 101–110, Cham. Springer International Publishing.

Hedin, G. (2000).
Reference attributed grammars.
Informatica (Slovenia), 24(3):301–317.

Hedin, G. and Magnusson, E. (2003).
JastAdd: an aspect-oriented compiler construction system.
Science of Computer Programming, 47(1):37–58.

Szárnyas, G., Izsó, B., Ráth, I., and Varró, D. (2017).
The Train Benchmark: cross-technology performance evaluation of continuous model queries.
Software & Systems Modeling, pages 1–29.

The Apache Software Foundation (2018).
Apache tinkerpop.
http://tinkerpop.apache.org/.