Continuous Model Validation Using Reference Attribute Grammars

SLE’18, Boston, November 5, 2018
Working with Runtime Models

**Running example:** Modeling train tracks and routes.

![Diagram of train tracks and routes]

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

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

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

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

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

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

---


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

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
  - Bidirectional references

---

Continuous Model Validation using Reference Attribute Grammars
SLE’18, Boston, November 5, 2018
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>;
```

Continuous Model Validation using Reference Attribute Grammars
SLE'18, Boston, November 5, 2018
Metamodels and Grammars

Abstract grammar (JastAdd syntax)

```plaintext
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?

Continuous Model Validation using Reference Attribute Grammars
SLE’18, Boston, November 5, 2018
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: $\mathcal{R}$

Cardinality:
- $1 : 1$
- $1 : \{0..1\}$
- $1 : N$

- **In RAGs**: optional $\mathcal{O}$ and list nodes: $\mathcal{L}$

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 \textit{Id} for each object
- Non-containment relations as \textit{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 \textit{collection attributes} to reverse references (\textit{slow!})
- Or two unidirectional relations (\textit{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:
```java
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:
```java
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**

\[
\text{abstract } \text{Ref} ::= <\text{Value}: \text{int}>; \\
\text{SensorRef} : \text{Ref} ::= <\text{Sensor}: \text{Sensor}>;
\]

Method:
- Name analysis during load
- Resolve reference by accessing intrinsic reference attribute

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

---

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:

— 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.

RouteSensor pattern, from [Szárnyas et al., 2017]
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):
```java
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]
**Read+Check: Route Sensor (Repair)**

![Graph](image)

**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

### Model size

<table>
<thead>
<tr>
<th>Model Size</th>
<th>#Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>8k</td>
</tr>
<tr>
<td>2</td>
<td>15k</td>
</tr>
<tr>
<td>8</td>
<td>66k</td>
</tr>
<tr>
<td>32</td>
<td>271k</td>
</tr>
<tr>
<td>128</td>
<td>1.1M</td>
</tr>
<tr>
<td>512</td>
<td>4.6M</td>
</tr>
</tbody>
</table>

### Execution Times [ms]

- **Name Lookup**
- **Intrinsic References**
- **Grammar Extension**
- **TinkerGraph**
- **Viatra (Incremental)**

**Continuous Model Validation using Reference Attribute Grammars**


SLE18, Boston, November 5, 2018
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


