Querying Automotive System Models and Safety Artifacts with MMINT and Viatra

Alessio Di Sandro, Sahar Kokaly, Rick Salay, Marsha Chechik
{adisandro, skokaly, rsalay, chechik}@cs.toronto.edu

University of Toronto

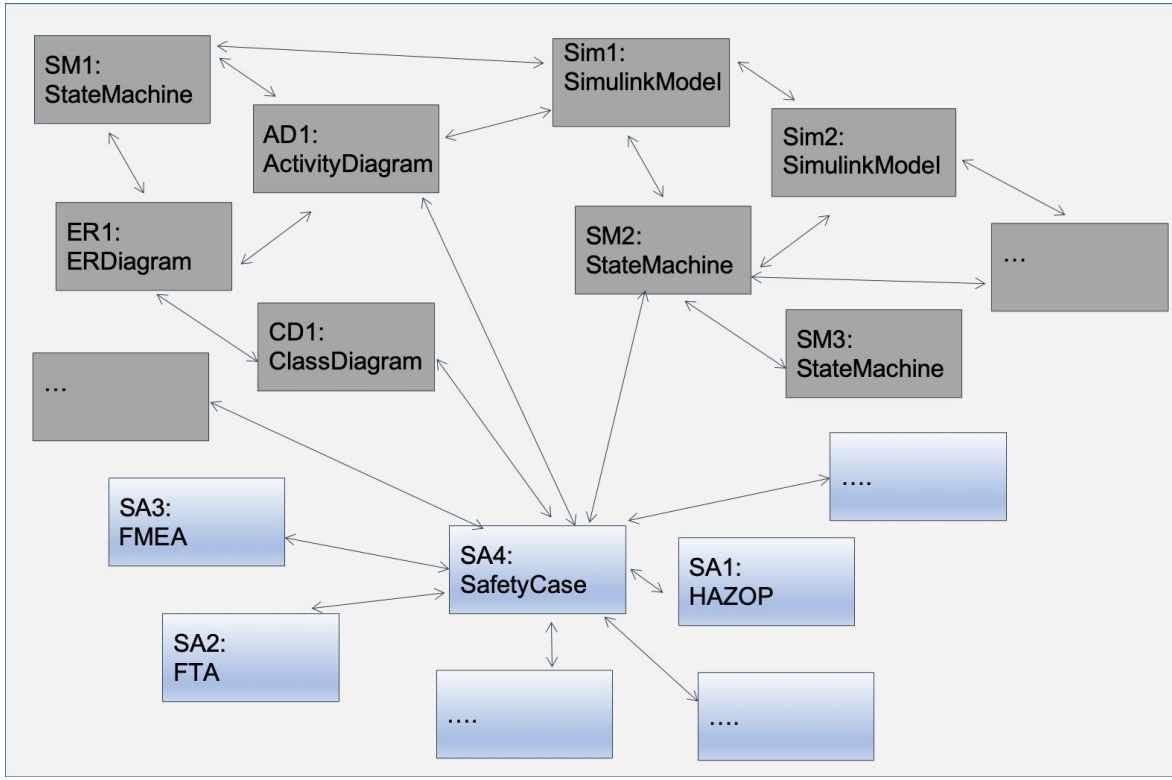
MASE, Sep 15 2019, Munich, Germany

Automotive domain complexity

- Increasing number of interconnected electronic and software components
- ISO 26262 functional safety standard: analyze hazards and provide evidence that the system being designed is **safe**

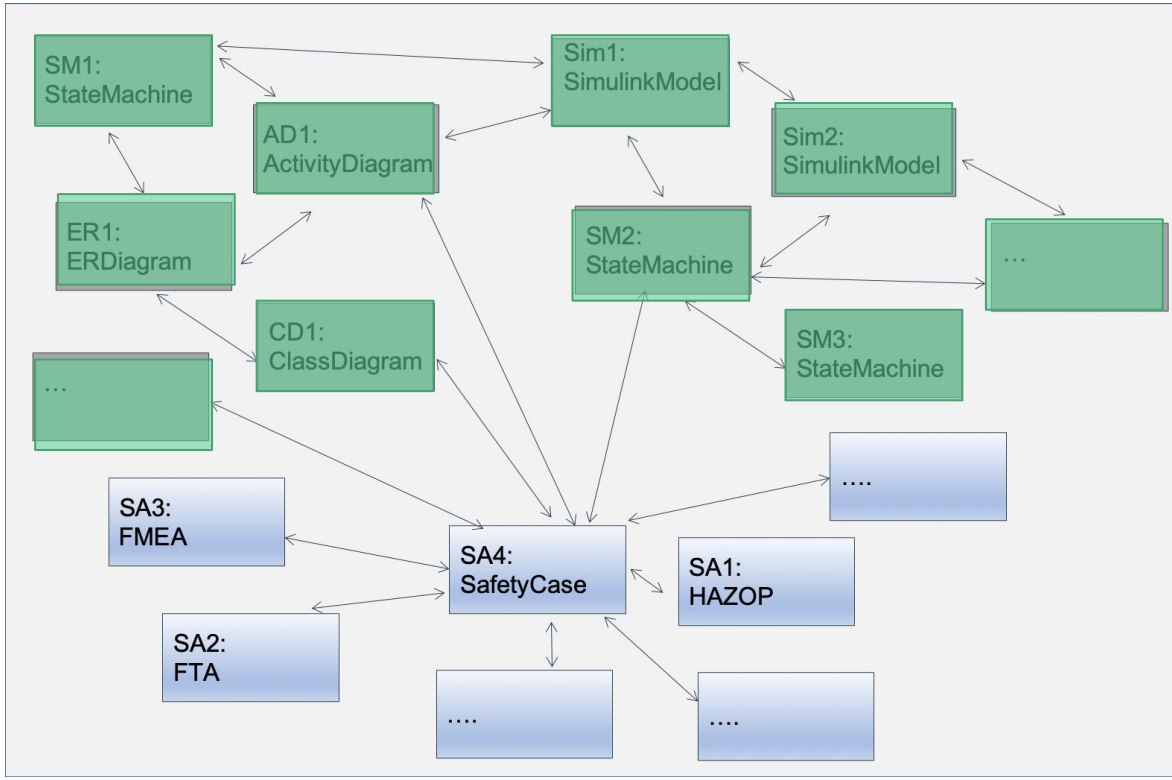
Automotive models

- Taming the domain complexity with models
  - heterogeneous
  - large
  - interconnected



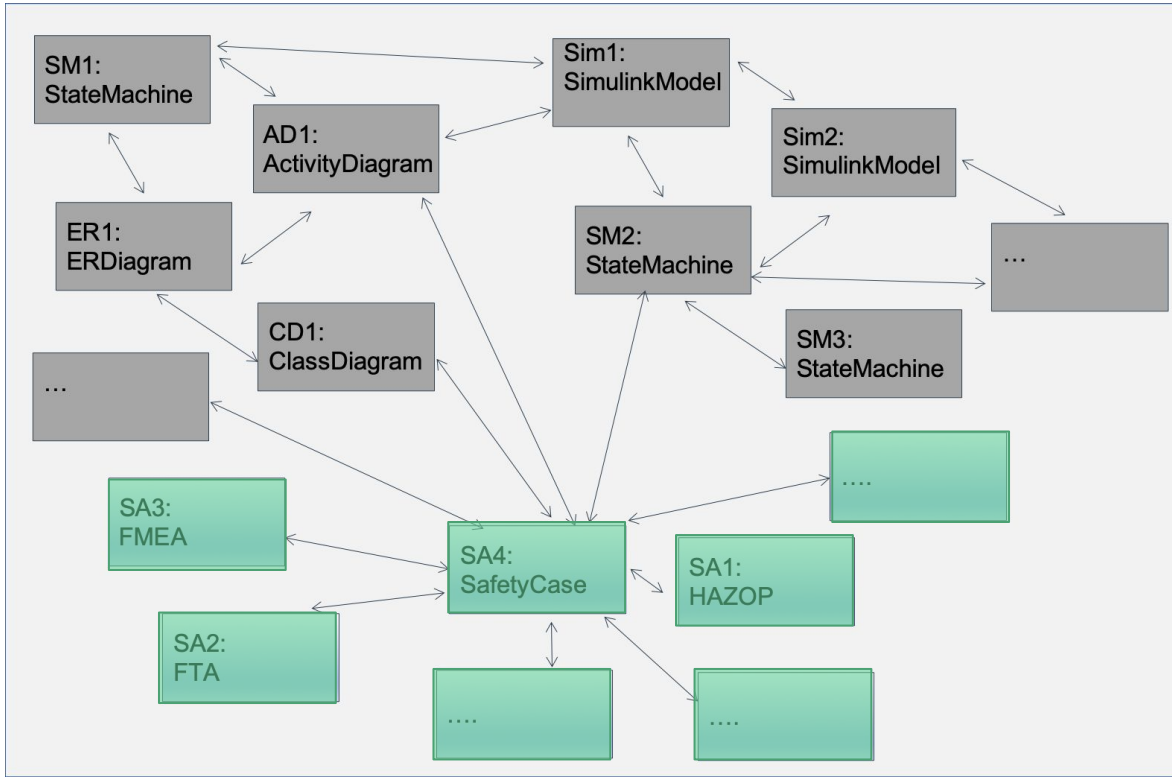
Automotive models

- Taming the domain complexity with models
  - heterogeneous
  - large
  - interconnected
- System models
  - SM, AD, ER, CD, Simulink



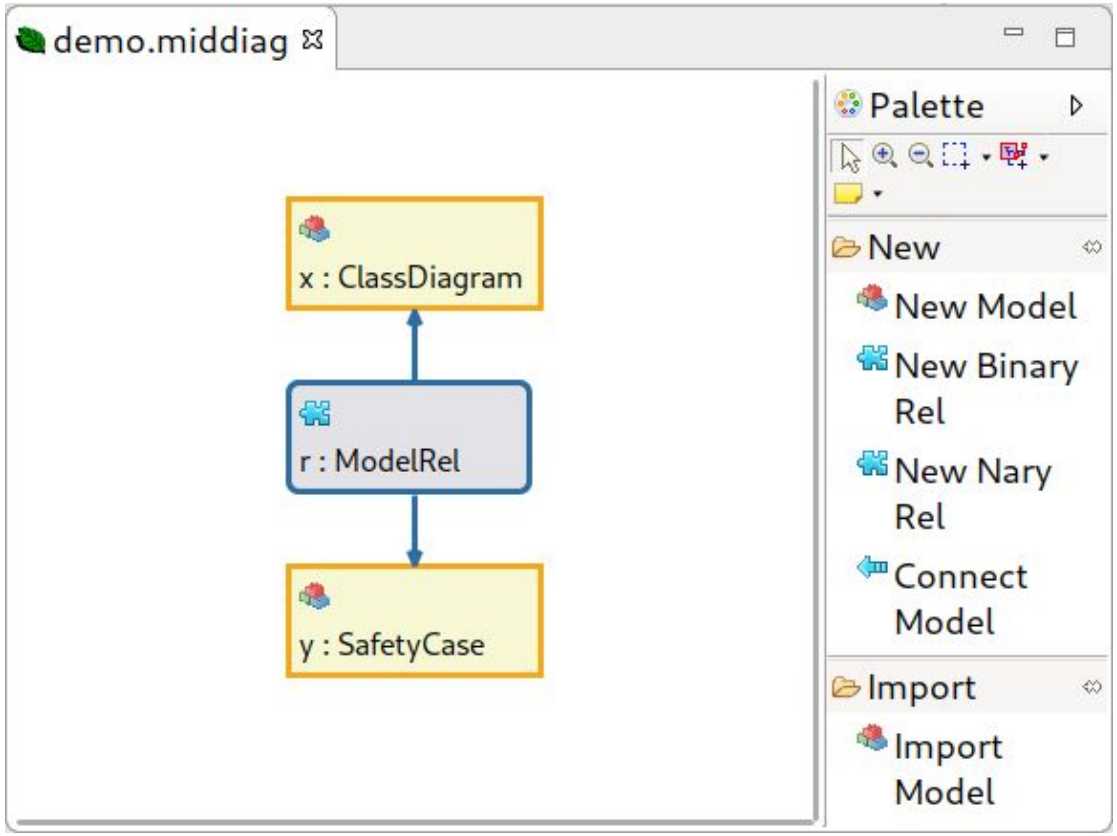
Automotive models

- Taming the domain complexity with models
  - heterogeneous
  - large
  - interconnected
- System models
  - UML models, Simulink models, etc.
- ISO 26262 safety artifacts
  - FMEA, FTA, HAZOP, Safety Case, etc.



MMINT

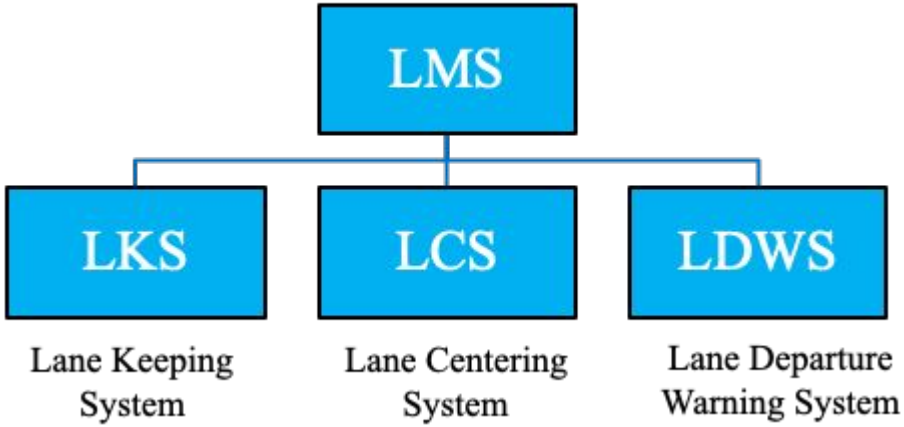
- Interactive framework for model management using Eclipse EMF
- Megamodels: collection of models connected by relationships
- Megamodel editor
  - create/import models and relationships
  - invoke operations



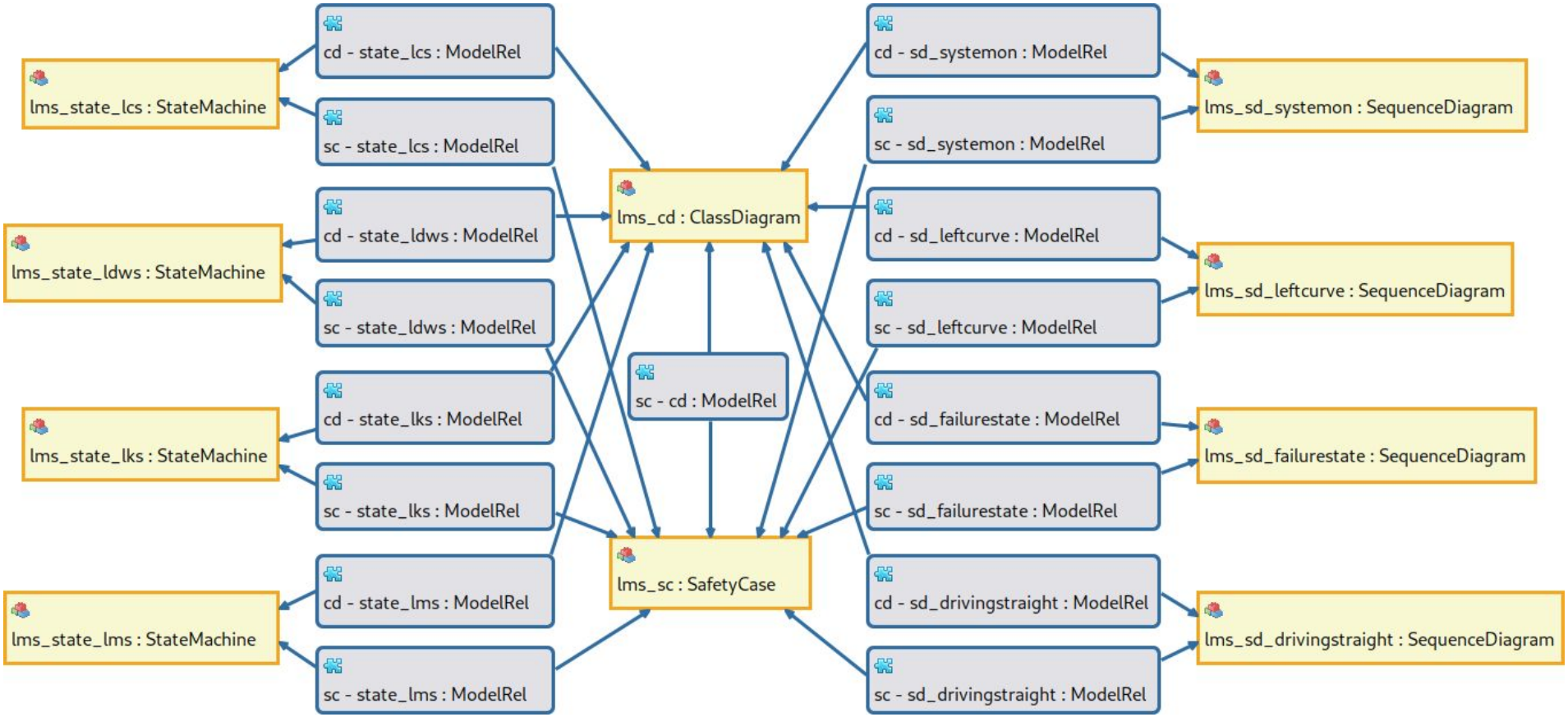
https://github.com/adisandro/MMINT

Lane Management System (LMS)

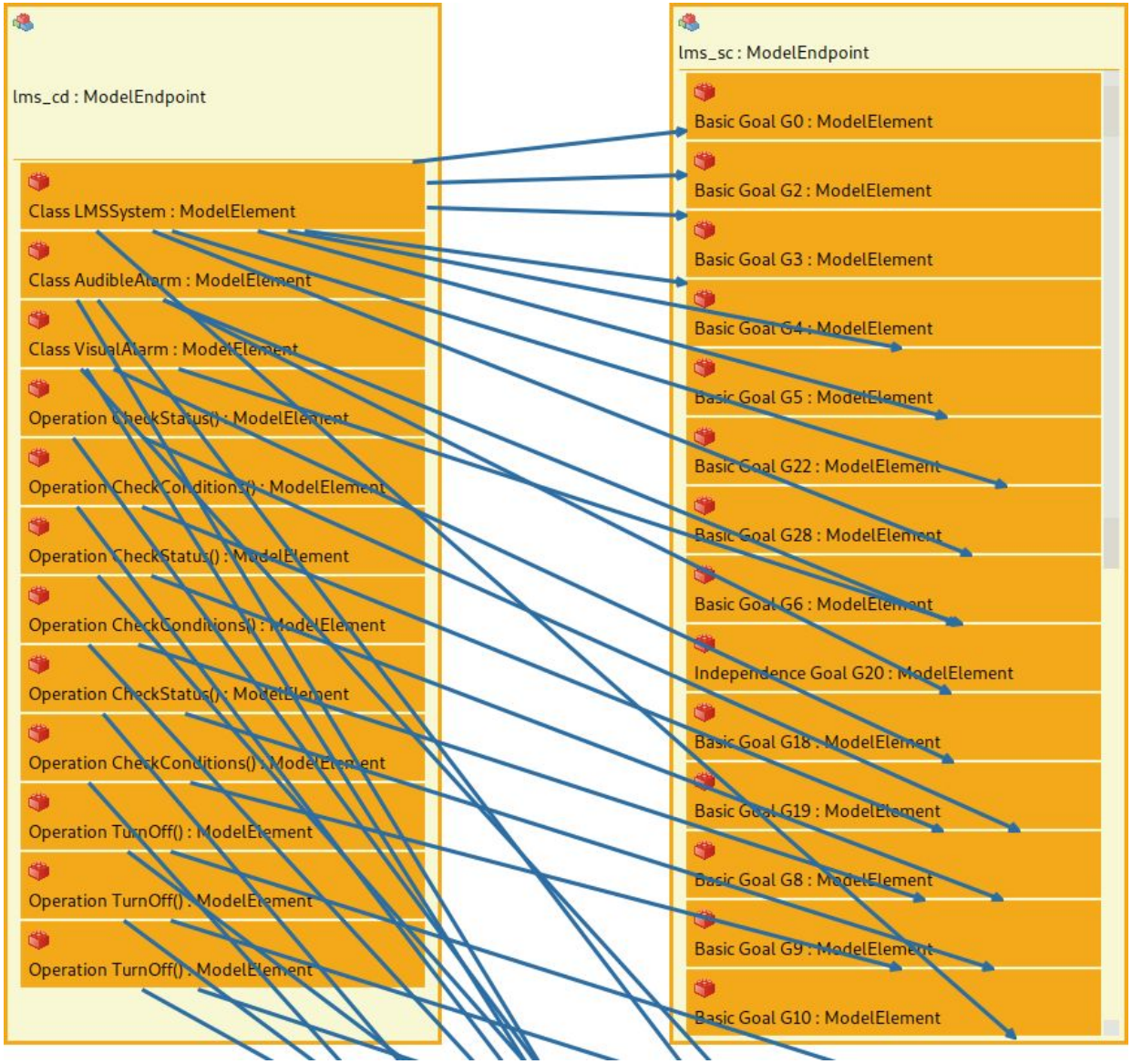
- Driver assistance system to keep the vehicle within a lane
- Takes control of braking and steering
- Safety critical, subject to the ISO 26262 standard



LMS megamodel



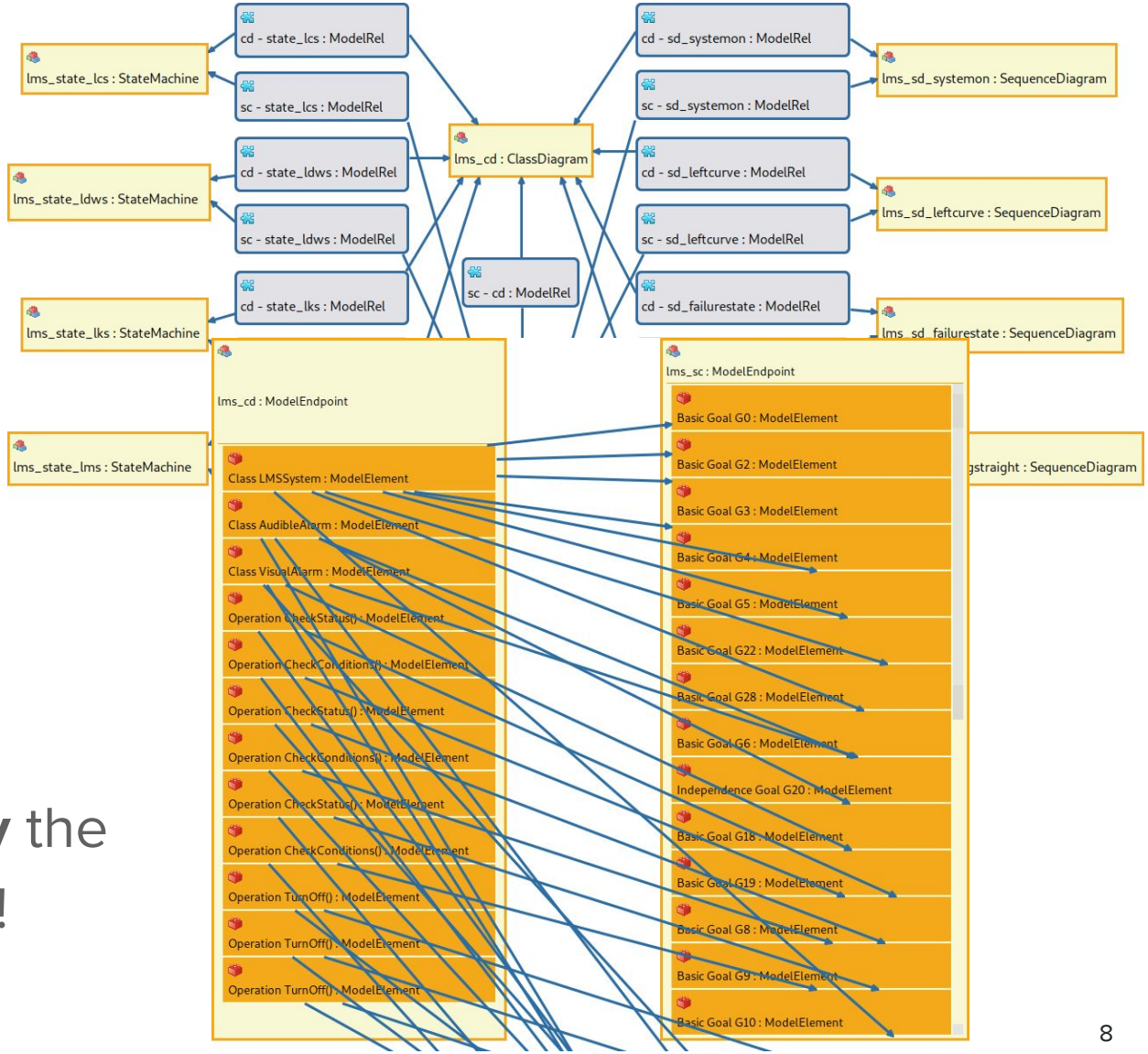
LMS megamodel



Extracting info from megamodels

- Megamodels can easily grow in size
- Like databases, they contain organized data (models and relationships)

Need a way to **query** the information required!



Query engine requirements

Generic

1. Navigation inter-model and intra-model
2. Handle heterogeneous models in the same query
3. Get a particular result or all results from a query
4. Select query inputs and display results in a megamodel
5. Scale with big models

Implementation-specific

1. Integration with Eclipse EMF
2. APIs to programmatically load and invoke queries

OCL

- OMG standard
- Default query and constraint language in Eclipse EMF
- Declarative syntax, functions with inputs and outputs, explicit collection of results

```
import 'http://se.cs.toronto.edu/mmint/MID'
import 'http://se.cs.toronto.edu/modelepedia/SafetyCase'

context mid::ModelElement

def: connectedModelElems : OrderedSet(ModelElement) =
  let mid = self.oclContainer().oclContainer().oclAsType(MID) in
  let rels = mid.models->select(rel |
    rel.oclIsTypeOf(ModelRel))->collect(oclAsType(ModelRel)) in
  let modelElems = rels->collect(mappings)
                       ->collect(modelElemEndpoints)
                       ->collect(target.oclAsType(ModelElement))
                       ->asOrderedSet() in
  modelElems->select(modelElemTgt |
    self <> modelElemTgt and
    rels->exists(rel |
      rel.mappings->exists(mapping |
        mapping.modelElemEndpoints->collect(target)
                                   ->includesAll(Set{self, modelElemTgt}))))

context mid::MID

def: connectedModelElems1 : Set(Tuple(src : ModelElement,
                                      tgt : ModelElement)) =
  let modelElems = ModelElement.allInstances() in
  modelElems->collect(e | Set{e}->product(connectedModelElems2(e)))->asSet()

def: connectedModelElems2(modelElemSrc : ModelElement) : Set(ModelElement) =
  modelElemSrc.connectedModelElems->asSet()

def: connectedModelElems3(modelElemSrc : ModelElement,
                          modelElemTgt : ModelElement) : Boolean =
  connectedModelElems2(modelElemSrc)->includes(modelElemTgt)

def: allConnectedModelElems1 : Set(Tuple(src : ModelElement,
                                         tgt : ModelElement)) =
  let modelElems = ModelElement.allInstances() in
  modelElems->collect(e | Set{e}->product(allConnectedModelElems2(e)))->asSet()

def: allConnectedModelElems2(modelElemSrc : ModelElement) : Set(ModelElement) =
  modelElemSrc->closure(connectedModelElems)->excluding(modelElemSrc)

def: allConnectedModelElems3(modelElemSrc : ModelElement,
                             modelElemTgt : ModelElement) : Boolean =
  allConnectedModelElems2(modelElemSrc)->includes(modelElemTgt)
```

https://www.eclipse.org/ocl

Viatra

- Incremental query engine based on the Rete algorithm
- Graph pattern based language (VQL)

```
1  package library
2
3  import "http://se.cs.toronto.edu/mmint/MID"
4  import "http://se.cs.toronto.edu/mmint/MID/Relationship"
5
6  pattern connectedModelElems(modelElemSrc: ModelElement,
7                              modelElemTgt: ModelElement) {
8    modelElemSrc != modelElemTgt;
9    Model.modelElems(modelSrc, modelElemSrc);
10   Model.modelElems(modelTgt, modelElemTgt);
11   modelSrc != modelTgt;
12   Mapping.modelElemEndpoints.target(mapping, modelElemSrc);
13   Mapping.modelElemEndpoints.target(mapping, modelElemTgt);
14 }
15
16 pattern allConnectedModelElems(modelElemSrc: ModelElement,
17                                modelElemTgt: ModelElement) {
18   modelElemSrc != modelElemTgt;
19   Model.modelElems(modelSrc, modelElemSrc);
20   Model.modelElems(modelTgt, modelElemTgt);
21   modelSrc != modelTgt;
22   find connectedModelElems+(modelElemSrc, modelElemTgt);
23 }
```

- Prolog-like, pattern arguments can be used as inputs or outputs, implicit collection of results

https://www.eclipse.org/viatra

Comparison between OCL and VQL

OCL	VQL	Generic
✔	✔	1. Navigation inter-model and intra-model

Comparison between OCL and VQL

OCL	VQL	Generic
✔	✔	1. Navigation inter-model and intra-model
✔	✔	2. Handle heterogeneous models in the same query

Comparison between OCL and VQL

OCL	VQL	Generic
✔	✔	1. Navigation inter-model and intra-model
✔	✔	2. Handle heterogeneous models in the same query
✔ (sep)	✔	3. Get a particular result or all results from a query

Comparison between OCL and VQL

```
import 'http://se.cs.toronto.edu/mmint/MID'
import 'http://se.cs.toronto.edu/modelepedia/SafetyCase'

context mid::ModelElement

def: connectedModelElems : OrderedSet(ModelElement) =
  let mid = self.oclContainer().oclContainer().oclAsType(MID) in
  let rels = mid.models->select(rel |
    rel.oclIsTypeOf(ModelRel))->collect(oclAsType(ModelRel)) in
  let modelElems = rels->collect(mappings)
                       ->collect(modelElemEndpoints)
                       ->collect(target.oclAsType(ModelElement))
                       ->asOrderedSet() in
  modelElems->select(modelElemTgt |
    self <> modelElemTgt and
    rels->exists(rel |
      rel.mappings->exists(mapping |
        mapping.modelElemEndpoints->collect(target)
                                   ->includesAll(Set{self, modelElemTgt}))))

context mid::MID

def: connectedModelElems1 : Set(Tuple(src : ModelElement,
                                      tgt : ModelElement)) =
  let modelElems = ModelElement.allInstances() in
  modelElems->collect(e | Set{e}->product(connectedModelElems2(e)))->asSet()

def: connectedModelElems2(modelElemSrc : ModelElement) : Set(ModelElement) =
  modelElemSrc.connectedModelElems->asSet()

def: connectedModelElems3(modelElemSrc : ModelElement,
                          modelElemTgt : ModelElement) : Boolean =
  connectedModelElems2(modelElemSrc)->includes(modelElemTgt)

def: allConnectedModelElems1 : Set(Tuple(src : ModelElement,
                                         tgt : ModelElement)) =
  let modelElems = ModelElement.allInstances() in
  modelElems->collect(e | Set{e}->product(allConnectedModelElems2(e)))->asSet()

def: allConnectedModelElems2(modelElemSrc : ModelElement) : Set(ModelElement) =
  modelElemSrc->closure(connectedModelElems)->excluding(modelElemSrc)

def: allConnectedModelElems3(modelElemSrc : ModelElement,
                             modelElemTgt : ModelElement) : Boolean =
  allConnectedModelElems2(modelElemSrc)->includes(modelElemTgt)
```

OCL

```
package library

import "http://se.cs.toronto.edu/mmint/MID"
import "http://se.cs.toronto.edu/mmint/MID/Relationship"

pattern connectedModelElems(modelElemSrc: ModelElement,
                            modelElemTgt: ModelElement) {
  modelElemSrc != modelElemTgt;
  Model.modelElems(modelSrc, modelElemSrc);
  Model.modelElems(modelTgt, modelElemTgt);
  modelSrc != modelTgt;
  Mapping.modelElemEndpoints.target(mapping, modelElemSrc);
  Mapping.modelElemEndpoints.target(mapping, modelElemTgt);
}

pattern allConnectedModelElems(modelElemSrc: ModelElement,
                               modelElemTgt: ModelElement) {
  modelElemSrc != modelElemTgt;
  Model.modelElems(modelSrc, modelElemSrc);
  Model.modelElems(modelTgt, modelElemTgt);
  modelSrc != modelTgt;
  find connectedModelElems+(modelElemSrc, modelElemTgt);
}
```

VQL

Comparison between OCL and VQL

```
import 'http://se.cs.toronto.edu/mmint/MID'
import 'http://se.cs.toronto.edu/modelepedia/SafetyCase'

context mid::ModelElement

def: connectedModelElems : OrderedSet(ModelElement) =
  let mid = self.oclContainer().oclContainer().oclAsType(MID) in
  let rels = mid.models->select(rel |
    rel.oclIsTypeOf(ModelRel))->collect(oclAsType(ModelRel)) in
  let modelElems = rels->collect(mappings)
                       ->collect(modelElemEndpoints)
                       ->collect(target.oclAsType(ModelElement))
                       ->asOrderedSet() in
  modelElems->select(modelElemTgt |
    self <> modelElemTgt and
    rels->exists(rel |
      rel.mappings->exists(mapping |
        mapping.modelElemEndpoints->collect(target)
                                   ->includesAll(Set{self, modelElemTgt}))))

context mid::MID

def: connectedModelElems1 : Set(Tuple(src : ModelElement,
                                      tgt : ModelElement)) =
  let modelElems = ModelElement.allInstances() in
  modelElems->collect(e | Set{e}->product(connectedModelElems2(e)))->asSet()

def: connectedModelElems2(modelElemSrc : ModelElement) : Set(ModelElement) =
  modelElemSrc.connectedModelElems->asSet()

def: connectedModelElems3(modelElemSrc : ModelElement,
                          modelElemTgt : ModelElement) : Boolean =
  connectedModelElems2(modelElemSrc)->includes(modelElemTgt)

def: allConnectedModelElems1 : Set(Tuple(src : ModelElement,
                                         tgt : ModelElement)) =
  let modelElems = ModelElement.allInstances() in
  modelElems->collect(e | Set{e}->product(allConnectedModelElems2(e)))->asSet()

def: allConnectedModelElems2(modelElemSrc : ModelElement) : Set(ModelElement) =
  modelElemSrc->closure(connectedModelElems)->excluding(modelElemSrc)

def: allConnectedModelElems3(modelElemSrc : ModelElement,
                             modelElemTgt : ModelElement) : Boolean =
  allConnectedModelElems2(modelElemSrc)->includes(modelElemTgt)
```

```
package library

import "http://se.cs.toronto.edu/mmint/MID"
import "http://se.cs.toronto.edu/mmint/MID/Relationship"

pattern connectedModelElems(modelElemSrc: ModelElement,
                            modelElemTgt: ModelElement) {
  modelElemSrc != modelElemTgt;
  Model.modelElems(modelSrc, modelElemSrc);
  Model.modelElems(modelTgt, modelElemTgt);
  modelSrc != modelTgt;
  Mapping.modelElemEndpoints.target(mapping, modelElemSrc);
  Mapping.modelElemEndpoints.target(mapping, modelElemTgt);
}

pattern allConnectedModelElems(modelElemSrc: ModelElement,
                               modelElemTgt: ModelElement) {
  modelElemSrc != modelElemTgt;
  Model.modelElems(modelSrc, modelElemSrc);
  Model.modelElems(modelTgt, modelElemTgt);
  modelSrc != modelTgt;
  find connectedModelElems+(modelElemSrc, modelElemTgt);
}
```

- OCL requires multiple queries to achieve the same flexibility of a single VQL query

Comparison between OCL and VQL

OCL	VQL	Generic
✔	✔	1. Navigation inter-model and intra-model
✔	✔	2. Handle heterogeneous models in the same query
✔ (sep)	✔	3. Get a particular result or all results from a query
✔	✔	4. Select query inputs and display results in a megamodel

Comparison between OCL and VQL

OCL	VQL	Generic
✔	✔	1. Navigation inter-model and intra-model
✔	✔	2. Handle heterogeneous models in the same query
✔ (sep)	✔	3. Get a particular result or all results from a query
✔	✔	4. Select query inputs and display results in a megamodel
✘ [1,2]	✔	5. Scale with big models

[1] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös, “Incremental evaluation of model queries over EMF models”, MODELS 2010, Oslo, Norway, October 3-8, 2010
[2] Z. Ujhelyi, G. Szoke, Á. Horváth, N. I. Csiszár, L. Vidács, D. Varró, and R. Ferenc, “Performance comparison of query-based techniques for anti-pattern detection”, Information & Software Technology, vol. 65, pp. 147–165, 2015

Comparison between OCL and VQL

OCL	VQL	Generic
✔	✔	1. Navigation inter-model and intra-model
✔	✔	2. Handle heterogeneous models in the same query
✔ (sep)	✔	3. Get a particular result or all results from a query
✔	✔	4. Select query inputs and display results in a megamodel
✘ [1,2]	✔	5. Scale with big models
		Implementation-specific
✔	✔	1. Integration with Eclipse EMF
✔	✔	2. APIs to programmatically load and invoke queries

[1] G. Bergmann, Á. Horváth, I. Ráth, D. Varró, A. Balogh, Z. Balogh, and A. Ökrös, “Incremental evaluation of model queries over EMF models”, MODELS 2010, Oslo, Norway, October 3-8, 2010
[2] Z. Ujhelyi, G. Szoke, Á. Horváth, N. I. Csiszár, L. Vidács, D. Varró, and R. Ferenc, “Performance comparison of query-based techniques for anti-pattern detection”, Information & Software Technology, vol. 65, pp. 147–165, 2015

Viatra integration in MMINT

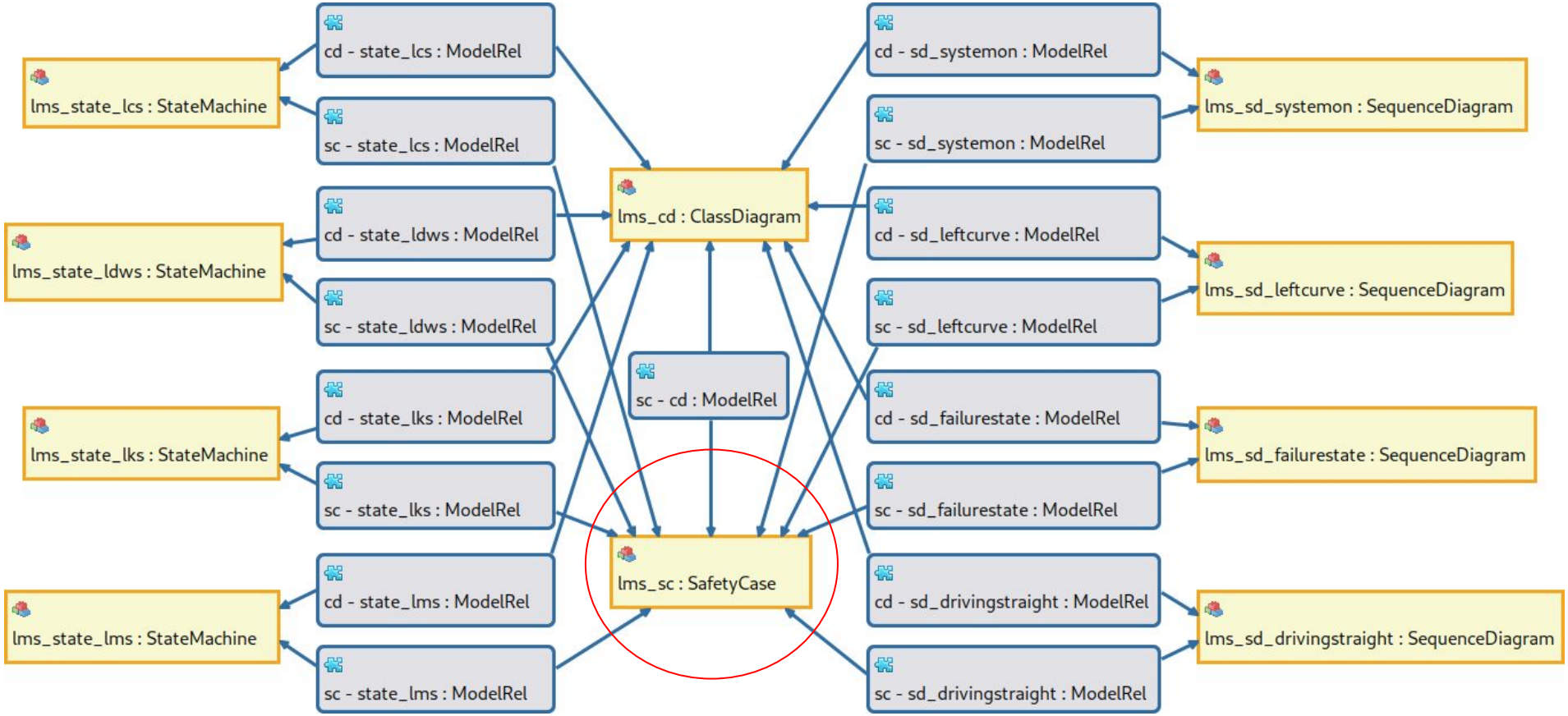
- Query Abstraction Layer (QAL) programming interface
  a. select query inputs graphically
  b. select query
  c. dispatch query+inputs to specific engine
  d. return query results as EMF objects

```java
/**
 * Evaluates a query to find elements within a megamodel.
 *
 * @param queryFilePath
 *          The path to the query file.
 * @param queryName
 *          The name of the query to be evaluated
 *          (a query file can contain multiple queries).
 * @param context
 *          The context where the query is executed,
 *          i.e. a megamodel, or one of its contained elements.
 * @param queryArgs
 *          The actual arguments to the query.
 * @return A list of megamodel elements that match the query.
 */
public default List<Object> evaluateQuery(String queryFilePath,
                                          String queryName,
                                          EObject context,
                                          List<? extends EObject> queryArgs) {
  return List.of();
}
```

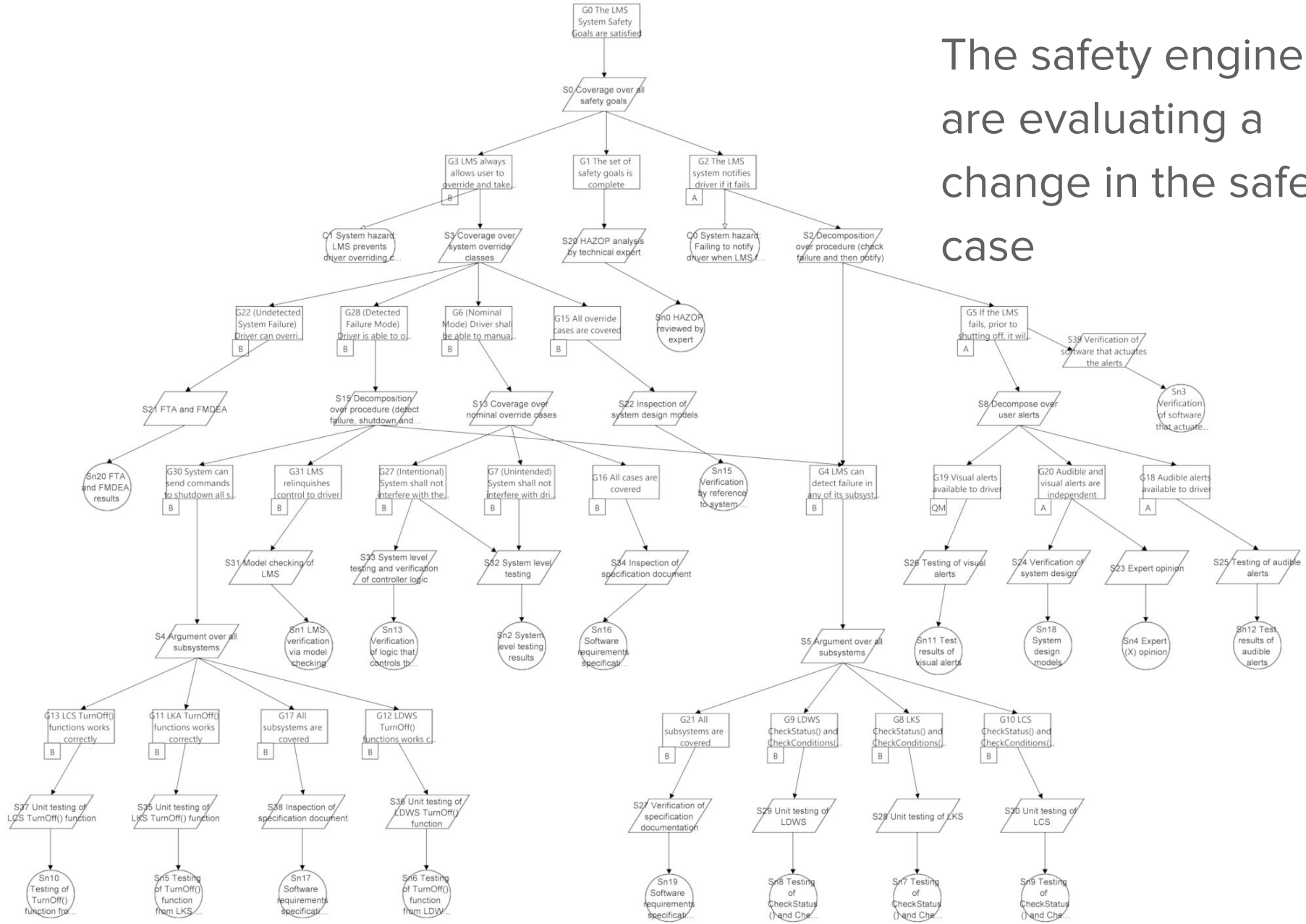
- Viatra QAL implementation
- VQL library
  - extract megamodel navigation
  - users can focus on the automotive questions

Example: querying the LMS megamodel

The safety engineers are evaluating a change in the safety case



Example: querying the LMS megamodel

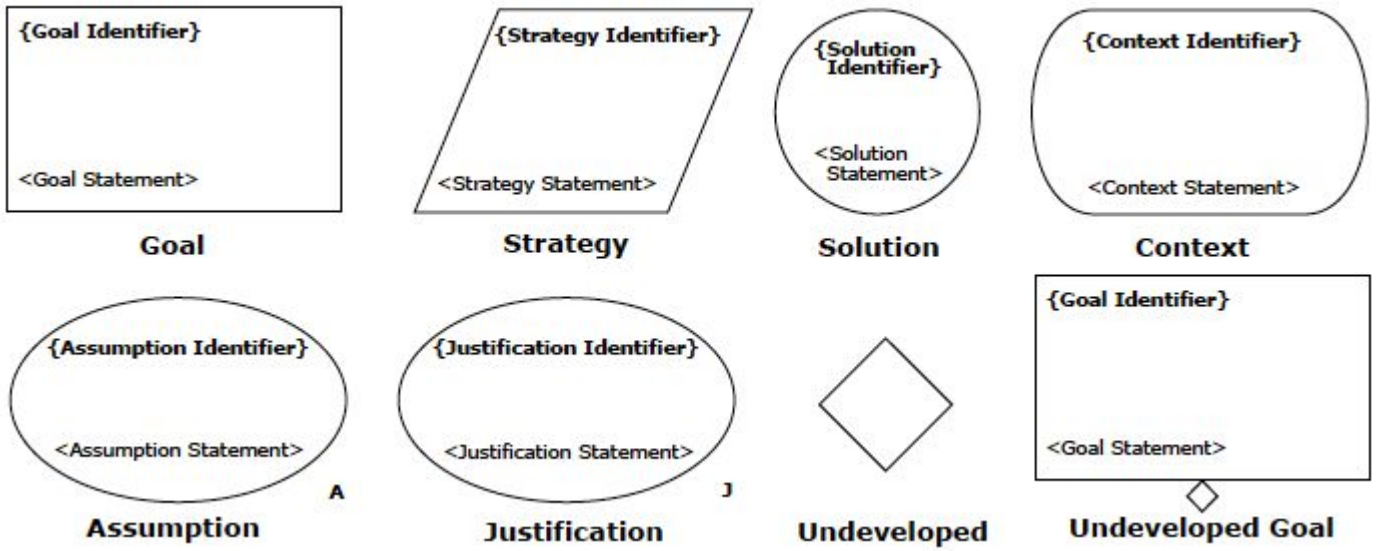


The safety engineers are evaluating a change in the safety case

Example: querying the LMS megamodel

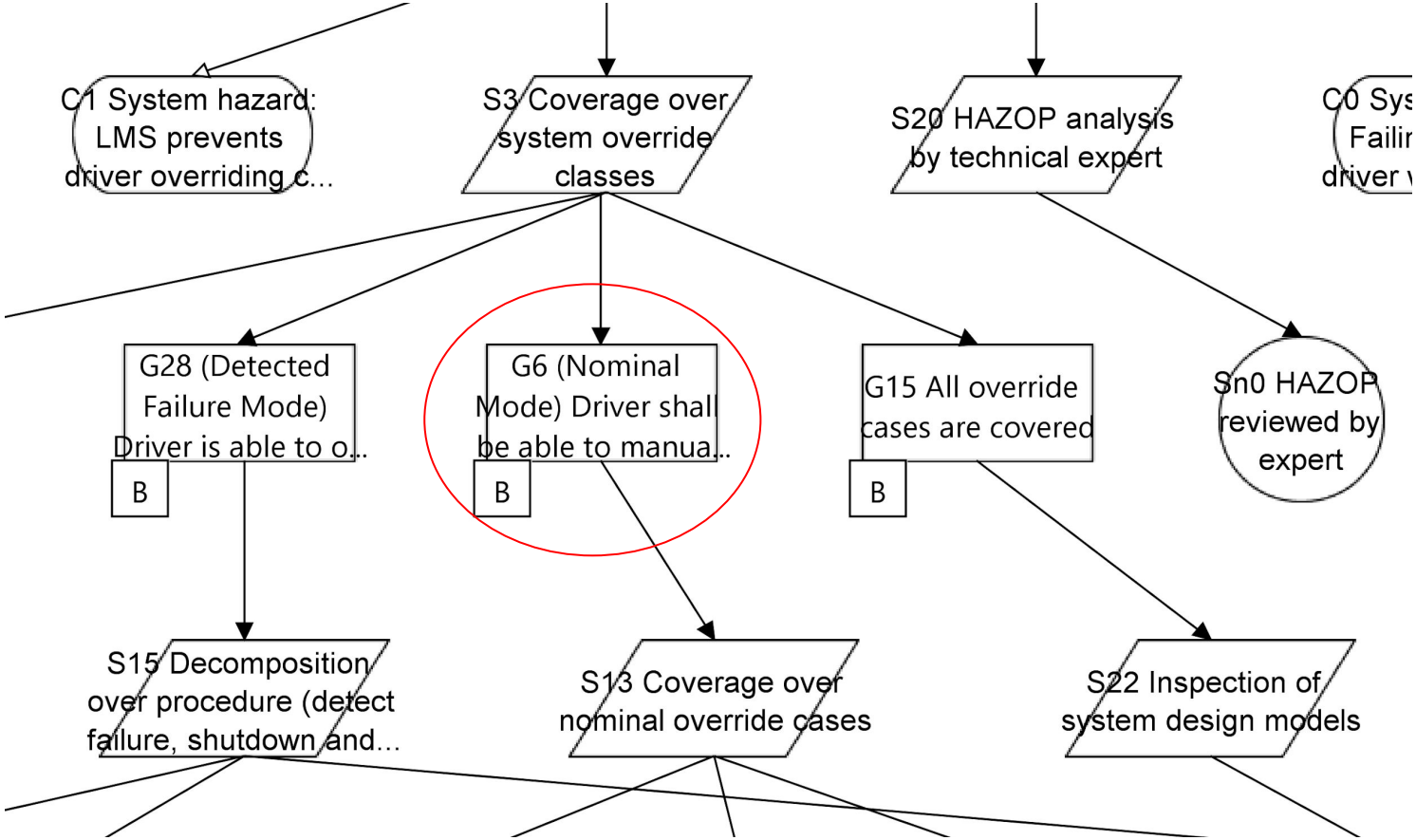
Safety case for LMS:

- Uses Goal Structured Notation (GSN)
- Structured argument that the LMS is safe to operate, supported by evidence
- Top level goal gets decomposed into solution leaves



Example: querying the LMS megamodel

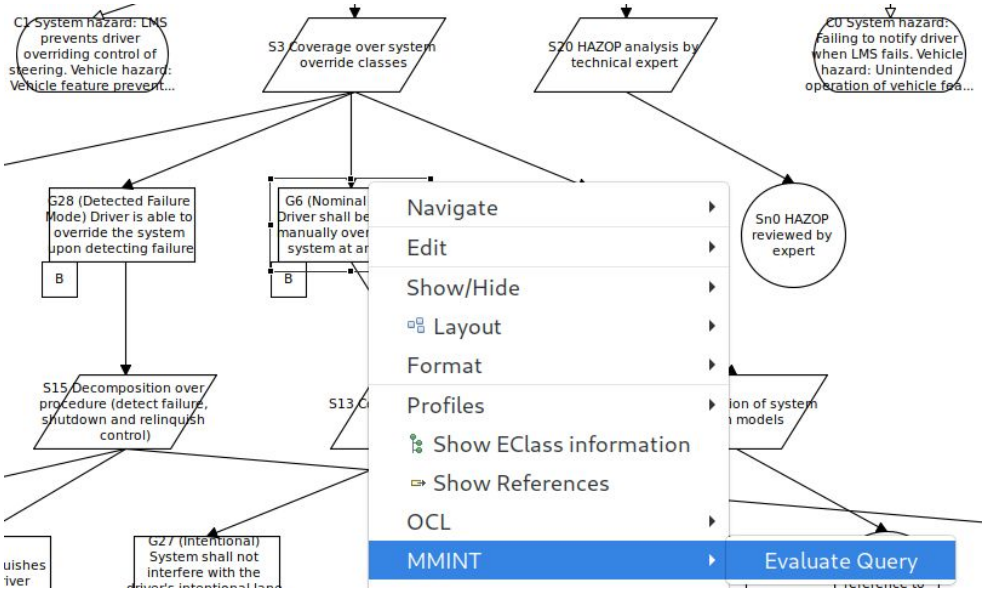
The safety engineers are evaluating a change to the Goal G6 in the safety case



Querying the LMS megamodel

connectedModelElems

- Which system elements are directly connected to G6?




```
1  package library
2
3  import "http://se.cs.toronto.edu/mmint/MID"
4  import "http://se.cs.toronto.edu/mmint/MID/Relationship"
5
6  pattern connectedModelElems(modelElemSrc: ModelElement,
7                              modelElemTgt: ModelElement) {
8    modelElemSrc != modelElemTgt;
9    Model.modelElems(modelSrc, modelElemSrc);
10   Model.modelElems(modelTgt, modelElemTgt);
11   modelSrc != modelTgt;
12   Mapping.modelElemEndpoints.target(mapping, modelElemSrc);
13   Mapping.modelElemEndpoints.target(mapping, modelElemTgt);
14 }
15
16 pattern allConnectedModelElems(modelElemSrc: ModelElement,
17                                modelElemTgt: ModelElement) {
18   modelElemSrc != modelElemTgt;
19   Model.modelElems(modelSrc, modelElemSrc);
20   Model.modelElems(modelTgt, modelElemTgt);
21   modelSrc != modelTgt;
22   find connectedModelElems+(modelElemSrc, modelElemTgt);
23 }
```

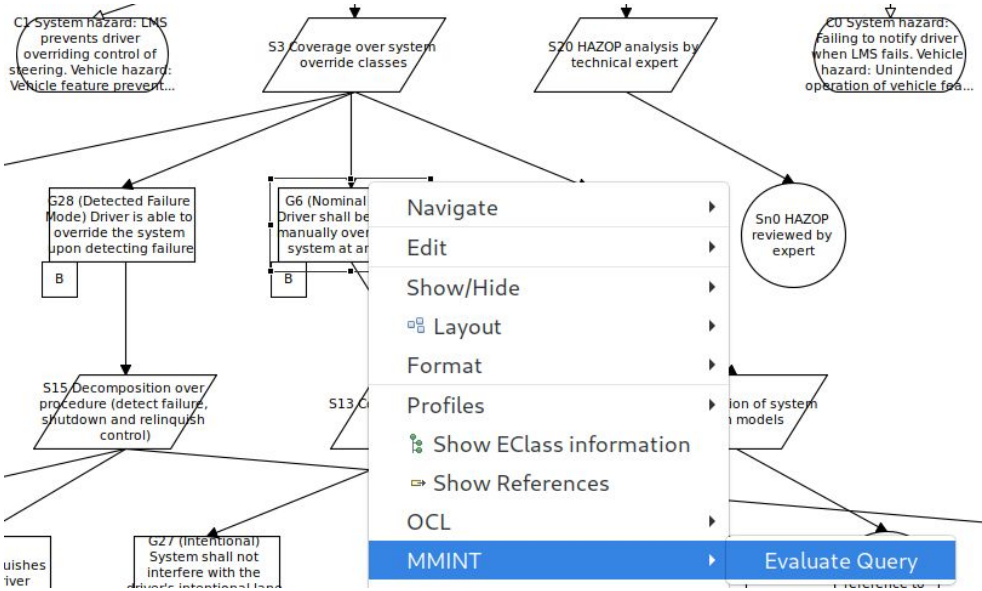
Querying the LMS megamodel

connectedModelElems

- Which system elements are directly connected to G6?

allConnectedModelElems

- Which system elements are directly and indirectly connected to G6?




```
1  package library
2
3  import "http://se.cs.toronto.edu/mmint/MID"
4  import "http://se.cs.toronto.edu/mmint/MID/Relationship"
5
6  pattern connectedModelElems(modelElemSrc: ModelElement,
7                              modelElemTgt: ModelElement) {
8    modelElemSrc != modelElemTgt;
9    Model.modelElems(modelSrc, modelElemSrc);
10   Model.modelElems(modelTgt, modelElemTgt);
11   modelSrc != modelTgt;
12   Mapping.modelElemEndpoints.target(mapping, modelElemSrc);
13   Mapping.modelElemEndpoints.target(mapping, modelElemTgt);
14 }
15
16 pattern allConnectedModelElems(modelElemSrc: ModelElement,
17                                modelElemTgt: ModelElement) {
18   modelElemSrc != modelElemTgt;
19   Model.modelElems(modelSrc, modelElemSrc);
20   Model.modelElems(modelTgt, modelElemTgt);
21   modelSrc != modelTgt;
22   find connectedModelElems+(modelElemSrc, modelElemTgt);
23 }
```

Querying the LMS megamodel

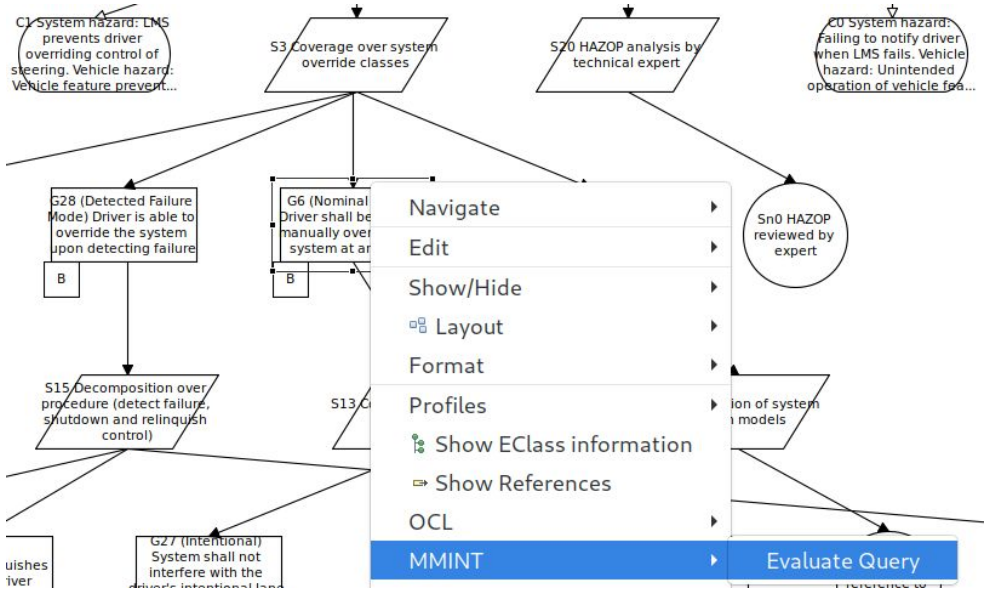
connectedModelElems

- Which system elements are directly connected to G6?

allConnectedModelElems

- Which system elements are directly and indirectly connected to G6?

(Opposite direction works too: change in a system model, which goals are affected?)




```
1  package library
2
3  import "http://se.cs.toronto.edu/mmint/MID"
4  import "http://se.cs.toronto.edu/mmint/MID/Relationship"
5
6  pattern connectedModelElems(modelElemSrc: ModelElement,
7                              modelElemTgt: ModelElement) {
8    modelElemSrc != modelElemTgt;
9    Model.modelElems(modelSrc, modelElemSrc);
10   Model.modelElems(modelTgt, modelElemTgt);
11   modelSrc != modelTgt;
12   Mapping.modelElemEndpoints.target(mapping, modelElemSrc);
13   Mapping.modelElemEndpoints.target(mapping, modelElemTgt);
14 }
15
16 pattern allConnectedModelElems(modelElemSrc: ModelElement,
17                                modelElemTgt: ModelElement) {
18   modelElemSrc != modelElemTgt;
19   Model.modelElems(modelSrc, modelElemSrc);
20   Model.modelElems(modelTgt, modelElemTgt);
21   modelSrc != modelTgt;
22   find connectedModelElems+(modelElemSrc, modelElemTgt);
23 }
```

MMINT demo

Conclusion

- Developed tool support for automotive model management with integrated querying
- Identified query engine requirements and compared between OCL and VQL
  - VQL is easier to use and faster
- Showcased three scenarios using the LMS example from industry
- Challenges:
  - creating a Query Abstraction Layer to plug in arbitrary languages
  - creating a query library for common tasks

Future work

- Expand the LMS megamodel with more safety-related artifacts (e.g., hazard analysis, FTA, test results, etc.) and write queries on top of them
- Evaluation of effectiveness and usability
- Expand library of megamodel queries
- Display results graphically
- Experiment with live queries

Thank you!

MMINT: https://github.com/adisandro/MMINT

Alessio Di Sandro, Sahar Kokaly, Rick Salay, Marsha Chechik
{adisandro, skokaly, rsalay, chechik}@cs.toronto.edu

University of Toronto

MASE, Sep 15 2019, Munich, Germany

Comparison between OCL and VQL

- Test the scalability requirement #4
- OCL QAL implementation
- 3 example scenarios
  a. safety case change
  b. identify medium risk elements:

     (hazards with Automotive Safety Integrity Level == B)

  c. identify highly interconnected elements:

     (elements with #connections > 5)

Comparison between OCL and VQL

- Execution times for 3 example scenarios:

Scenario	OCL time (s)	VQL time (s)
1	0.411	0.686
2	2.220	0.830
3	32.996	0.599

- Threats to validity:
  - limited expertise with OCL and VQL queries
  - only 3 scenarios