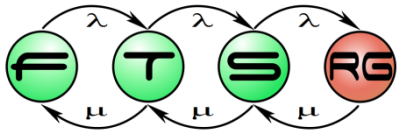
EMF-IncQuery

Incremental evaluation of model queries over EMF models

Gábor Bergmann, Ákos Horváth,
István Ráth, Dániel Varró
András Balogh, Zoltán Balogh,
András Ökrös, Zoltán Ujhelyi

Budapest University of Technology and Economics
OptXware Research and Development LLC



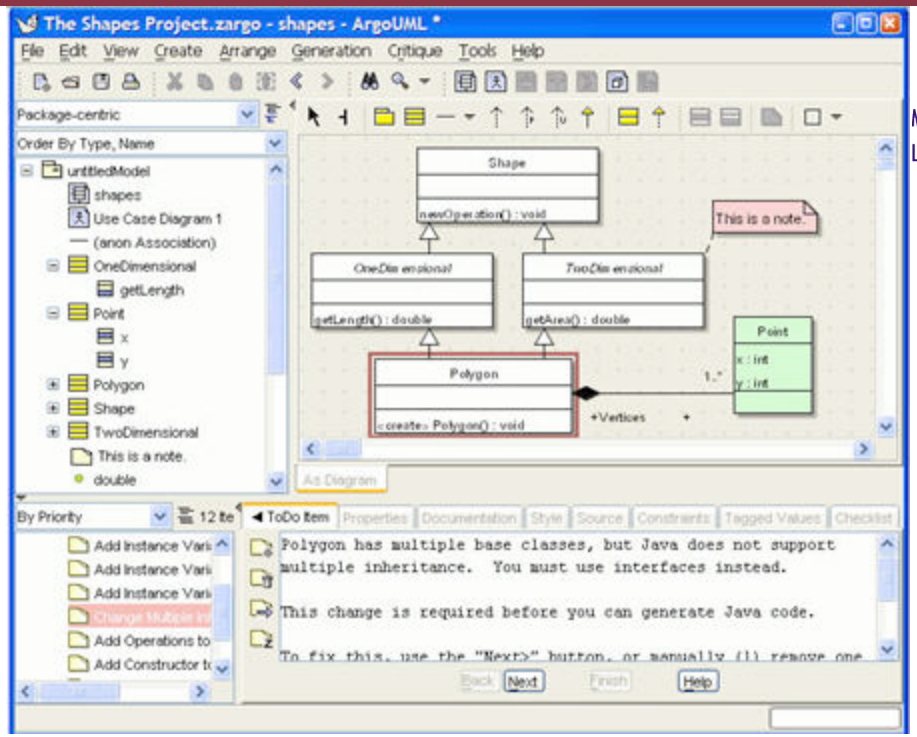
MOTIVATION

Hi Jane, what do you do at work?

Jane

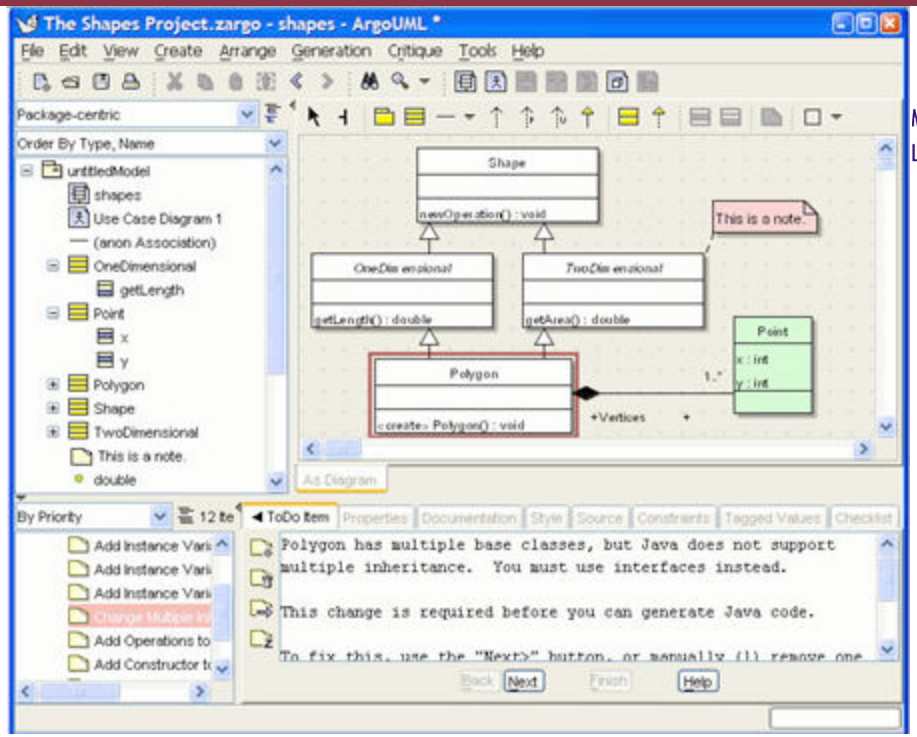
Hi Jane, what do you do at work?

Jane



Hi Jane, what do you do at work?

Jane

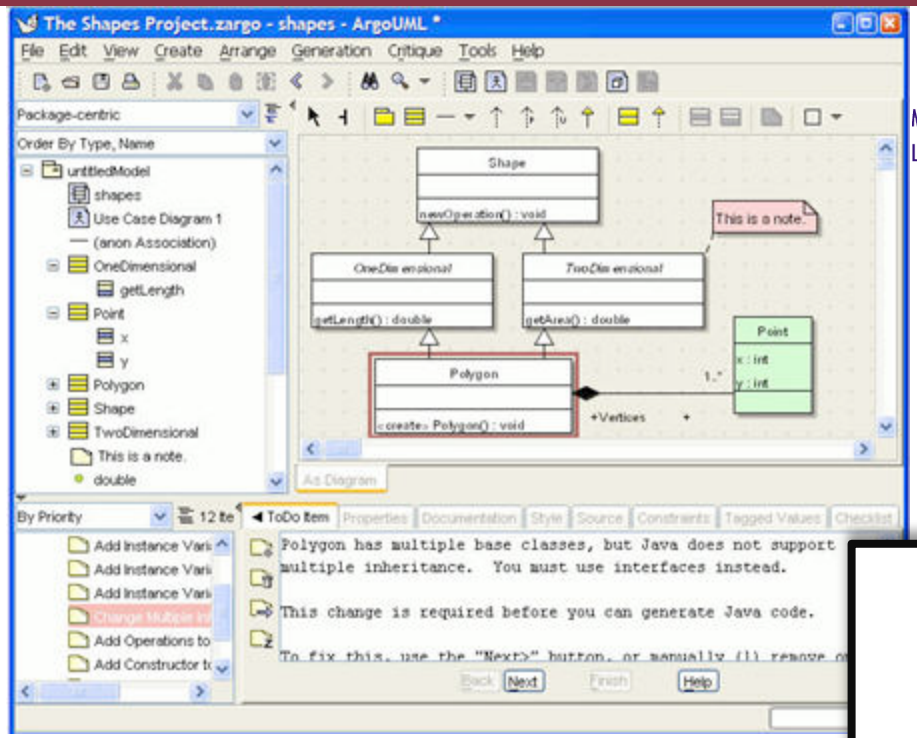


UNIFIED MODELING LANGUAGE™

Boss

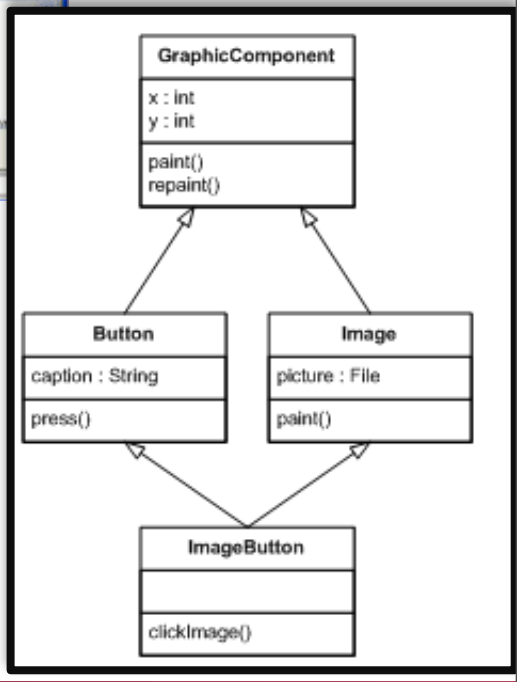
Hi Jane, what do you do at work?

Jane



UNIFIED MODELING LANGUAGE™

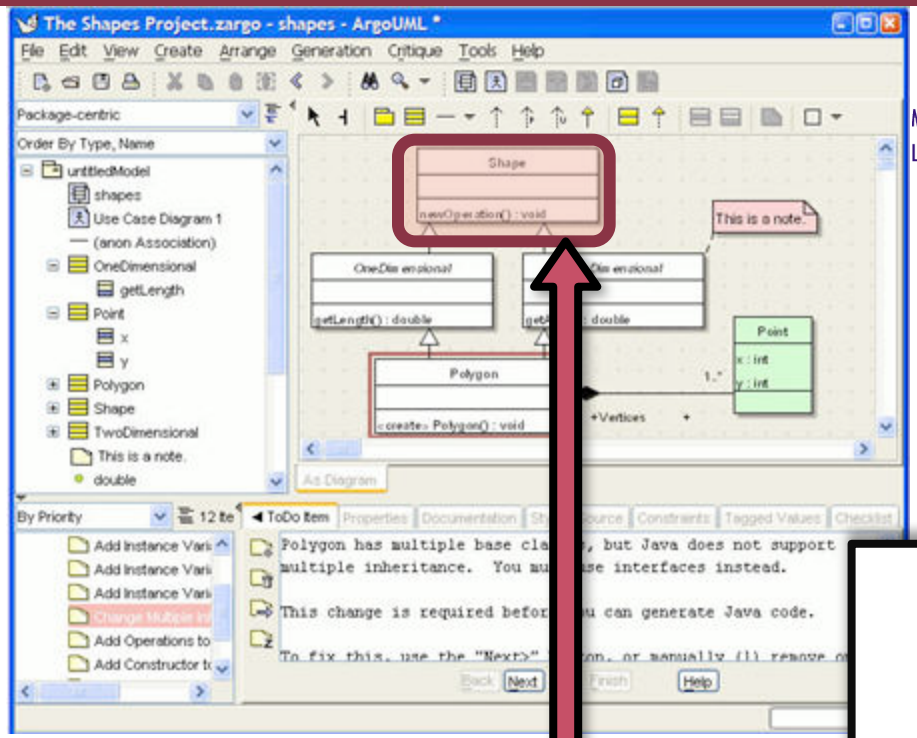
Detect!



Boss

Hi Jane, what do you do at work?

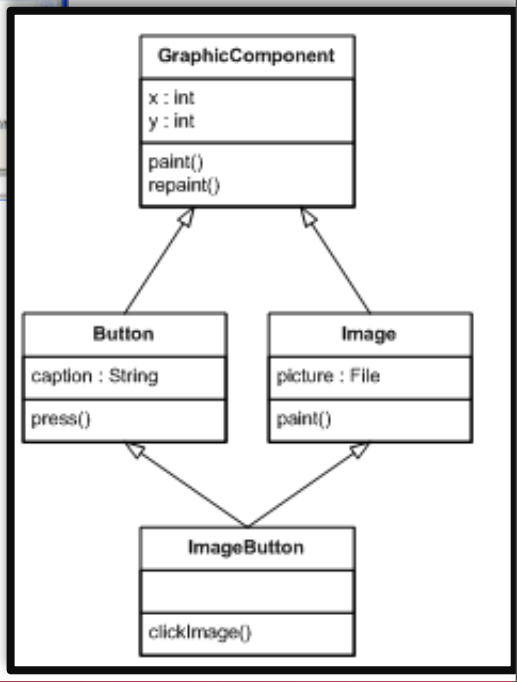
Jane



UNIFIED MODELING LANGUAGE™

Detect!

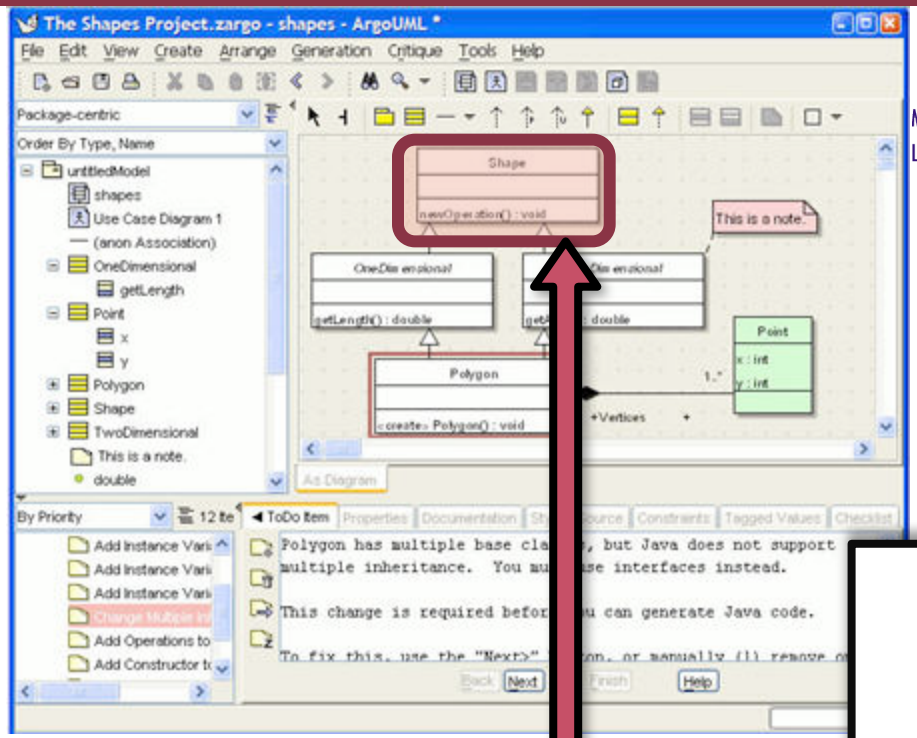
Report!



Boss

Hi Jane, what do you do at work?

Jane



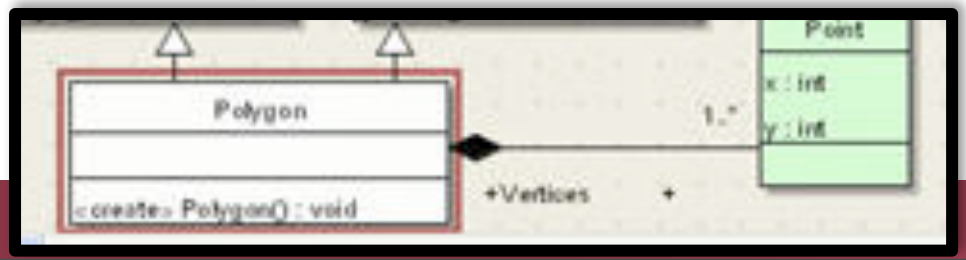
UNIFIED MODELING LANGUAGE™

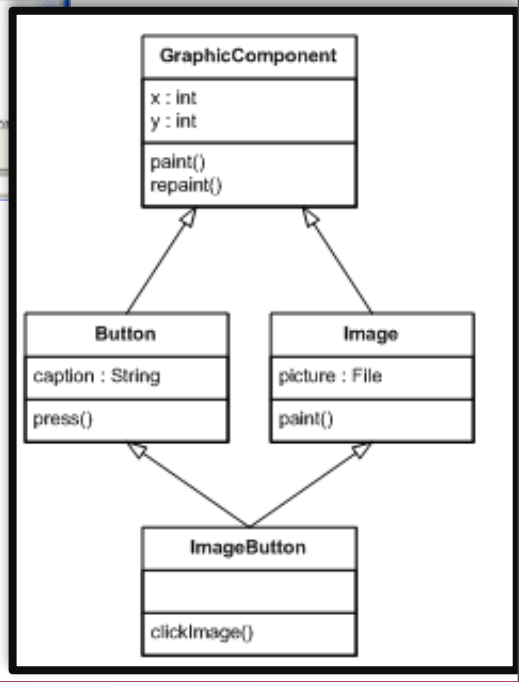
Detect!

Report!

Query / View

Boss

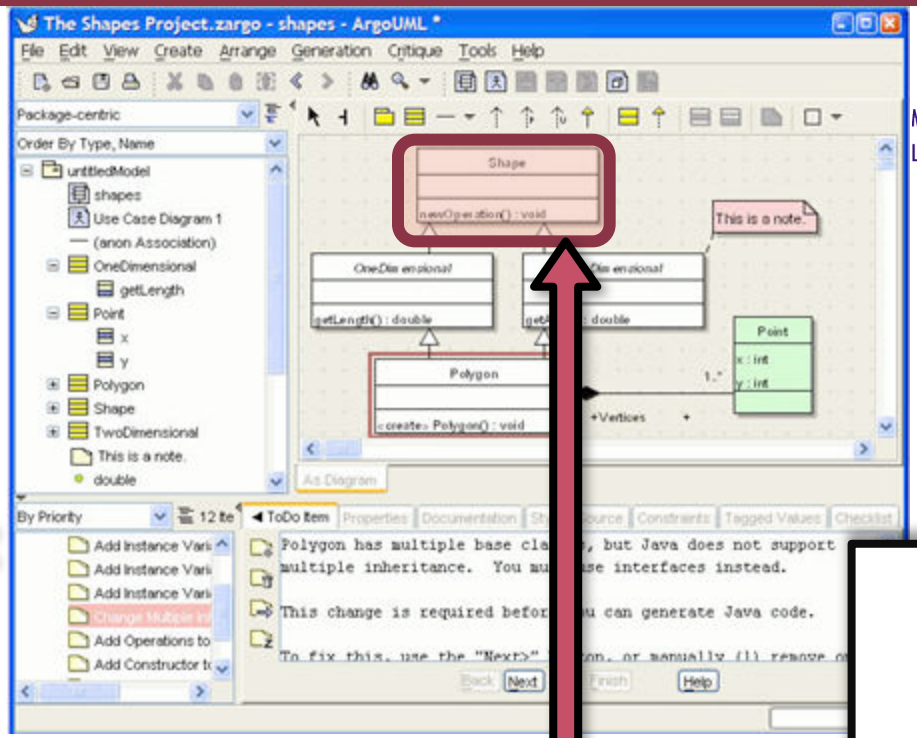


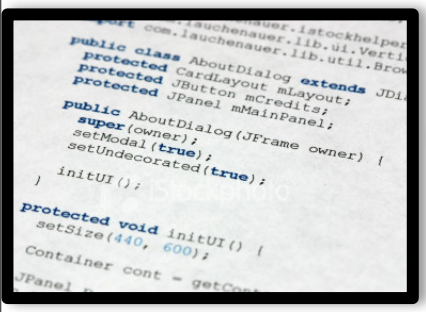


Constraint

Hi Jane, what do you do at work?

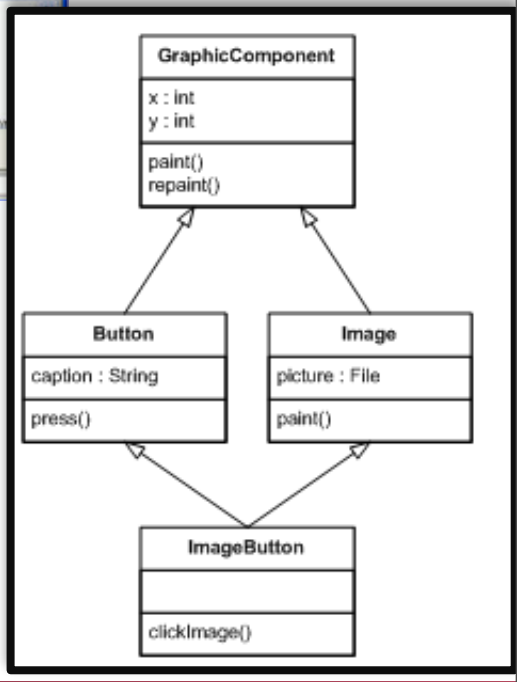
Jane





Gen

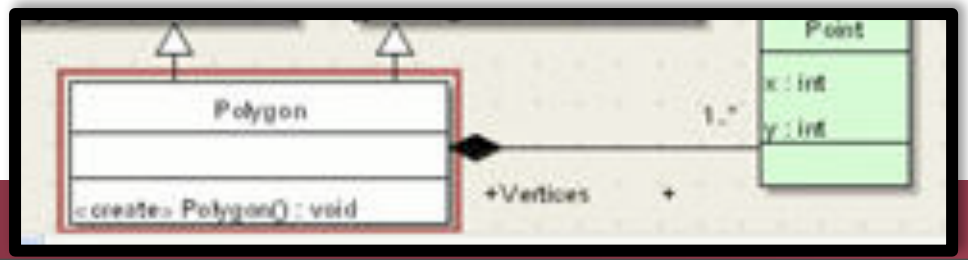
Detect!



Query / View

Report!

Boss



Constraint

Hi Jane, you look so sad. What is wrong?

Jane

Boss

Hi Jane, you look so sad. What is wrong?

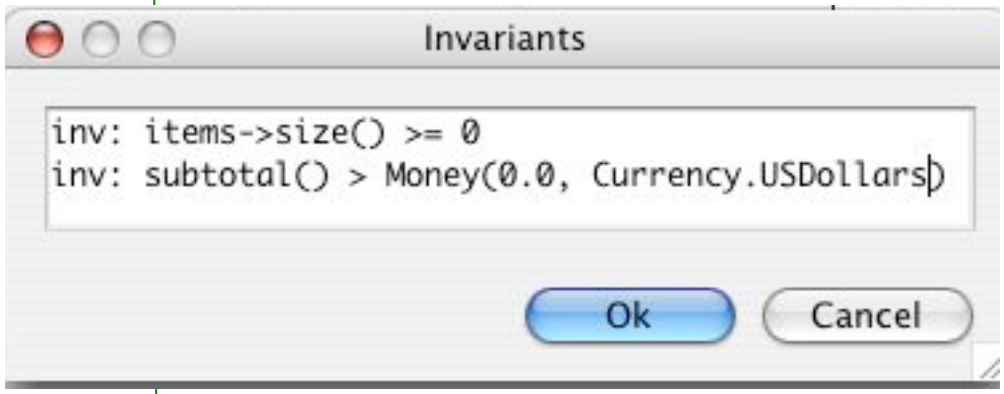
- It takes me hours to write a query of an EMF model

Jane Boss

Hi Jane, you look so sad. What is wrong?

- It takes me hours to write a query of an EMF model

- Why don't you use a declarative query language (like OCL)?



Jane Boss

Hi Jane, you look so sad. What is wrong?

Jane

Boss

Hi Jane, you look so sad. What is wrong?

- It takes me hours to write a query of an EMF model

- I need to re-run the checks from scratch every time

Jane Boss

Hi Jane, you look so sad. What is wrong?

- It takes me hours to write a query of an EMF model

- I need to re-run the checks from scratch every time

- Why don't you try something incremental?

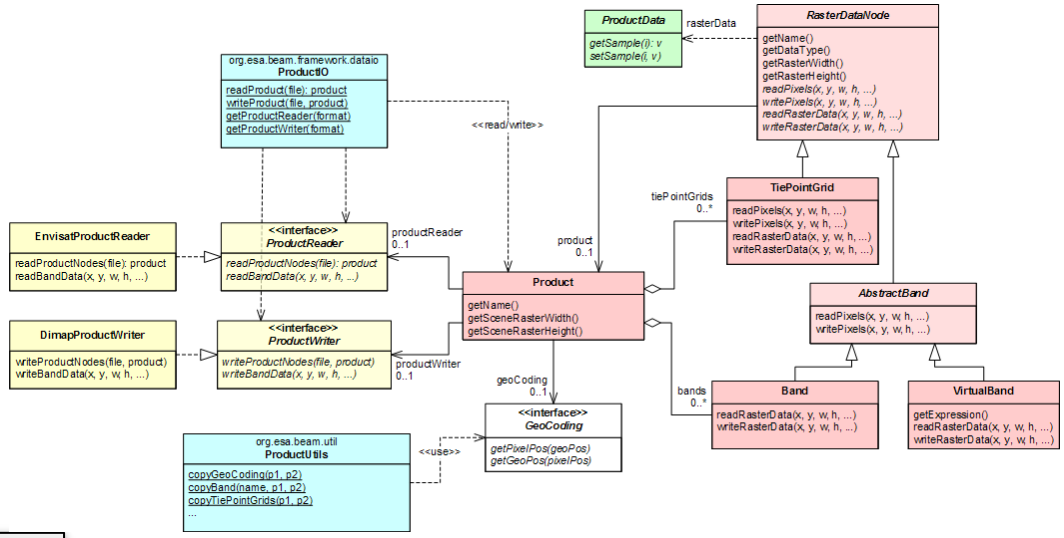
Jane Boss

Hi Jane, you look so sad. What is wrong?

Jane Boss

Hi Jane, you look so sad. What is wrong?

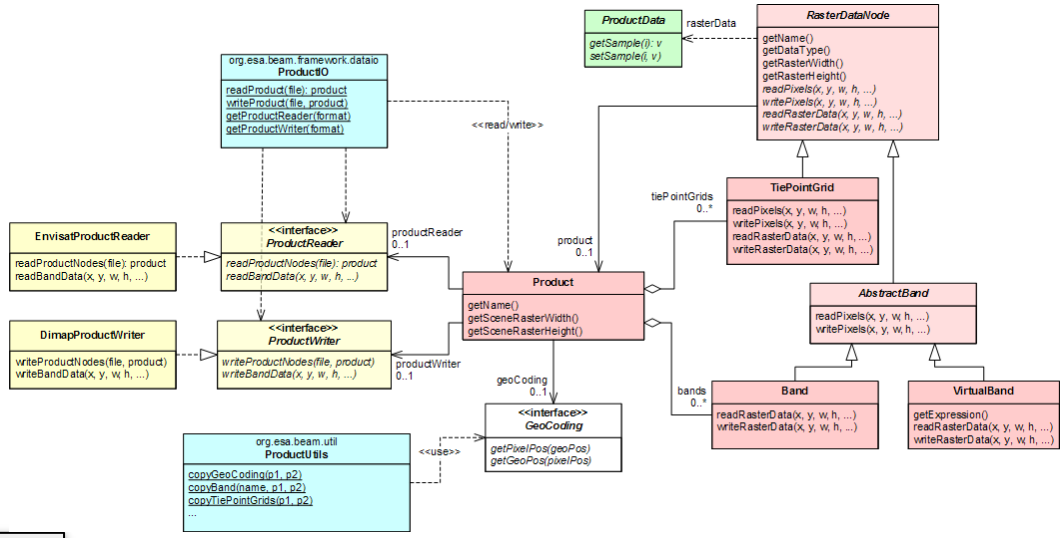
- It takes me hours to write a query of an EMF model
- I need to re-run the checks from scratch every time
- I go to have lunch while running well-formedness checks on large UML models



Jane Boss

Hi Jane, you look so sad. What is wrong?

- It takes me hours to write a query of an EMF model
- I need to re-run the checks from scratch every time
- I go to have lunch while running well-formedness checks on large UML models



Oh, Jane, you must eat a lot!

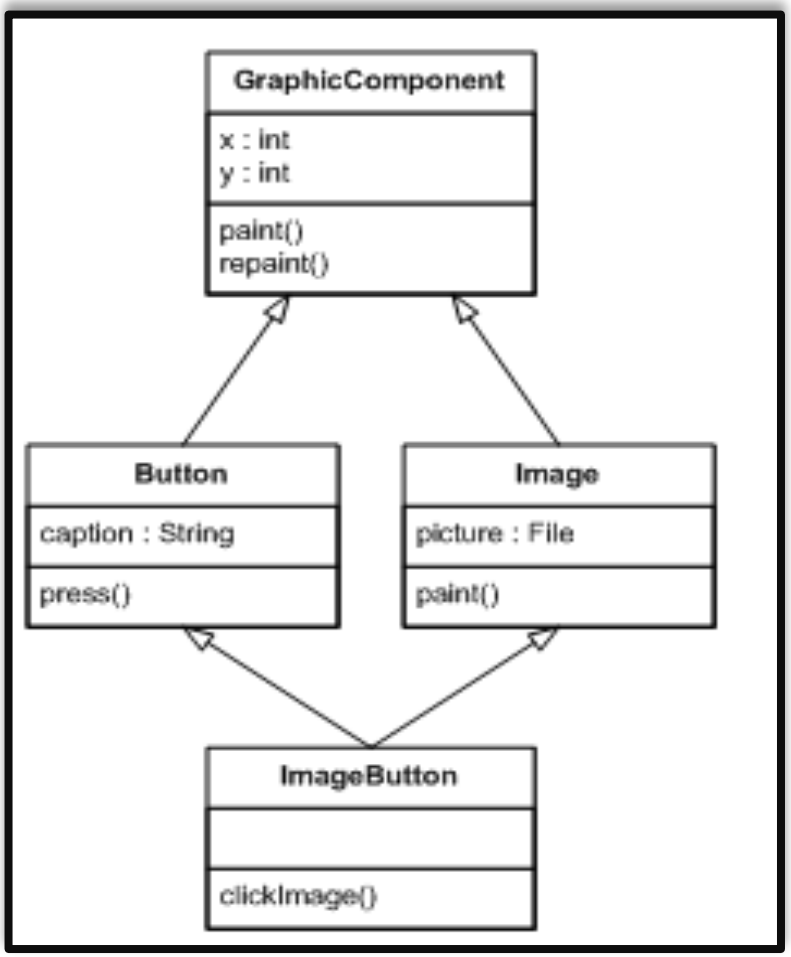
Jane Boss

STATE OF THE ART

Problem 1: Expressiveness

- EMF Query (declarative)
  - Low expressiveness
  - Limited navigability
    - no „cycles”
- OCL (declarative)
  - Verbose
  - Lack of reusability support
  - Local constraints of a model element
  - Poor handling of recursion

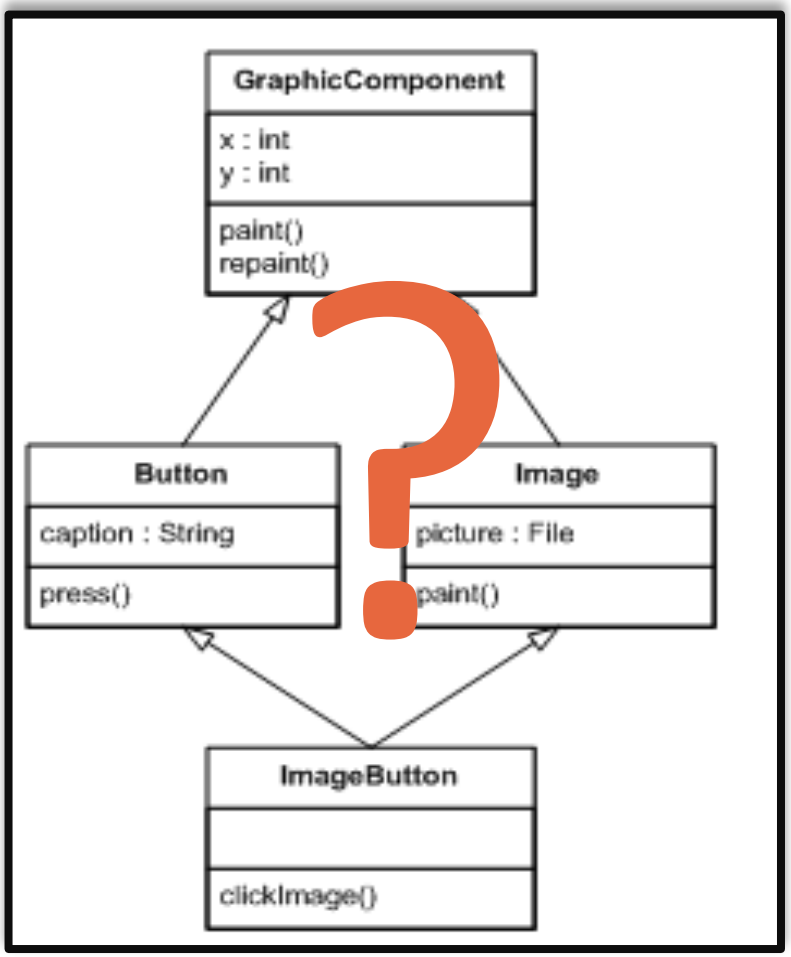
→Challenging to use



Problem 1: Expressiveness

- EMF Query (declarative)
  - Low expressiveness
  - Limited navigability
    - no „cycles”
- OCL (declarative)
  - Verbose
  - Lack of reusability support
  - Local constraints of a model element
  - Poor handling of recursion

→Challenging to use



Problem 2: Incrementality

- Goal: Incremental evalation of model queries
  - Incremental maintenance of result set
  - Avoid unnecessary re-computation
- Related work:
  - Constraint evaluation (by A. Egyed)
    - Arbitrary constraint description
      - Can be a bottleneck for complex constraints
      - Always local to a model element
    - Listen to model notifications
    - Calculate which constraints need to be reevaluated
  - No other related technology directly over EMF
  - Research MT tools: with varying degrees of support

Problem 3: Performance

- Native EMF queries (Java program code): Lack of
  - Reverse navigation along references
  - Enumeration of all instances by type
  - Smart Caching

- Scalability of (academic) MT tools
  - Queries over >100K model elements (several proofs): FUJABA, VIATRA2 (Java), GrGEN, VMTS (.Net), Egyed's tools

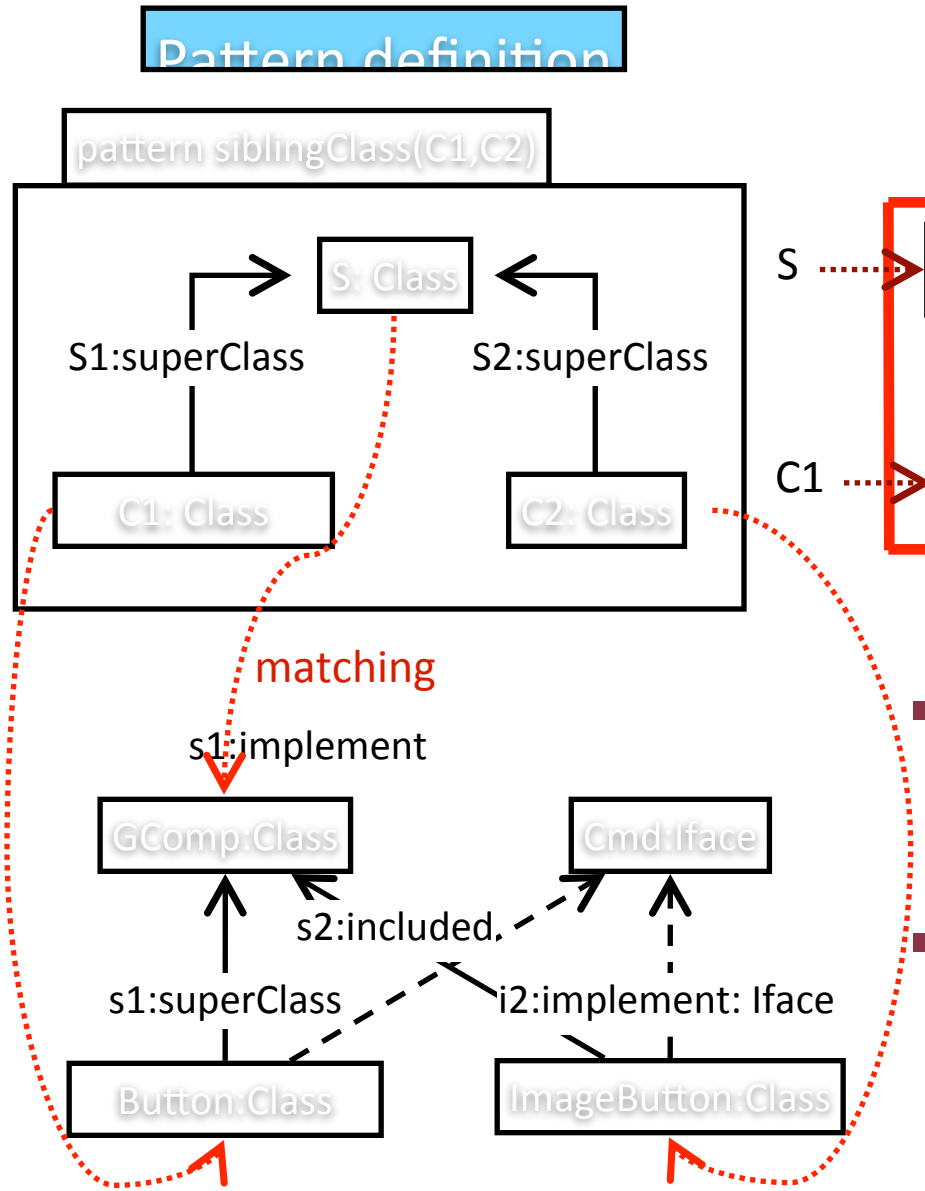
Contributions

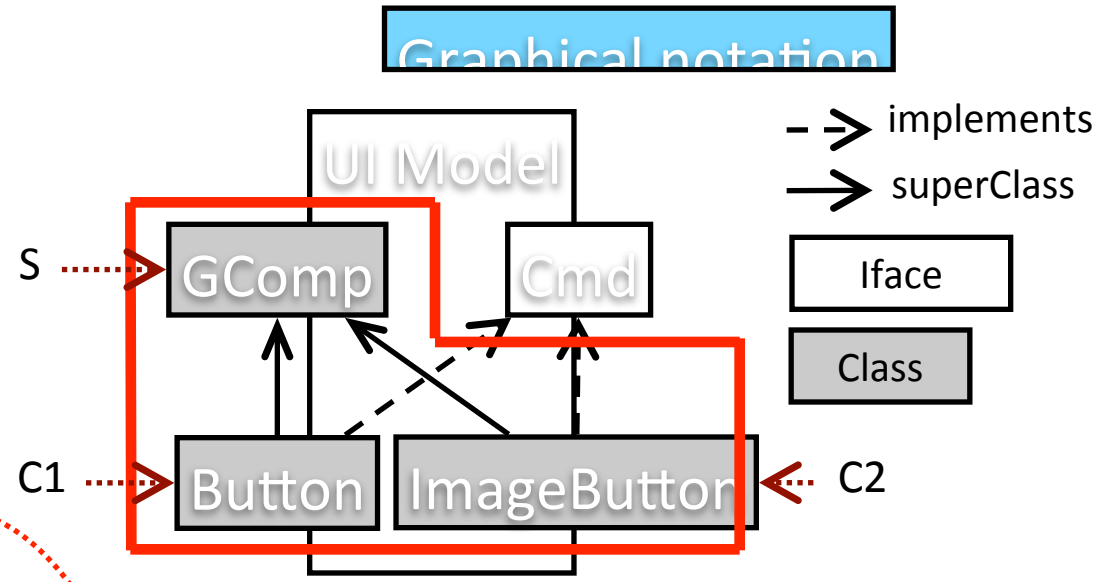
- **Expressive** declarative query language by graph patterns
- **Incremental** cache of matches (materialized view)
- **High performance** for large models

QUERY LANGUAGE

Contributions

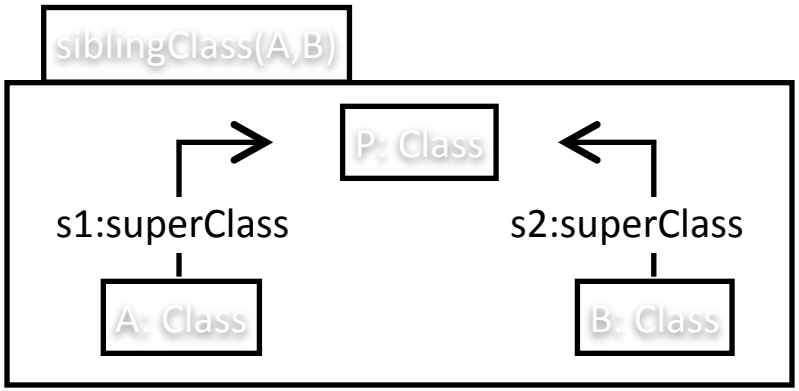
- **Expressive** declarative query language by graph patterns
  - Capture local + global queries
  - Compositionality + Reusabilility
  - „Arbitrary” Recursion, Negation
- **Incremental** cache of matches (materialized view)
- **High performance** for large models





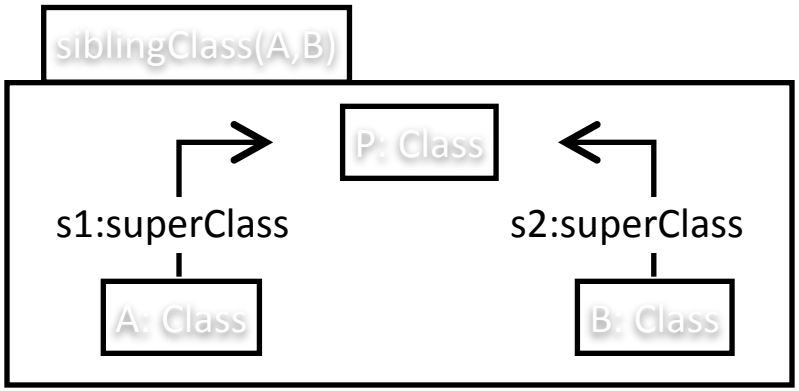
- Graph Pattern:
  - Structural condition that have to be fulfilled by a part of the model space
- Graph pattern matching:
  - A model (i.e. part of the model space) c satisfy a graph pattern,
  - if the pattern can be matched to a subgraph of the model

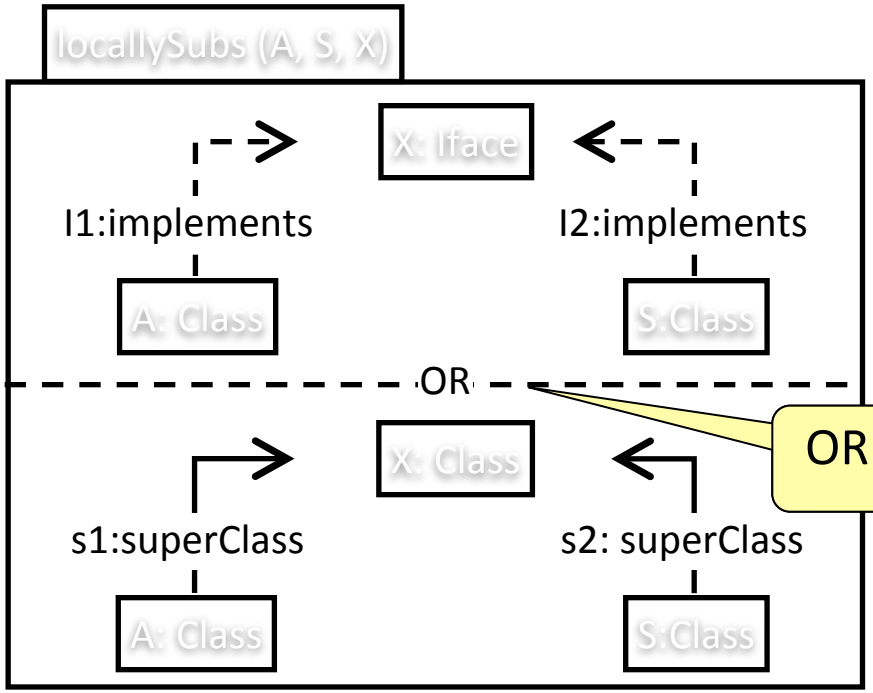
Graph patterns (VTCL)




```
// B is a sibling class of A
pattern siblingClass(A, B) =
{
    Class(A);
    Class.superClass(S1, A, P);
    Class(P);
    Class.superClass(S2, B, P);
    Class(B);
```

Graph patterns (VTCL)



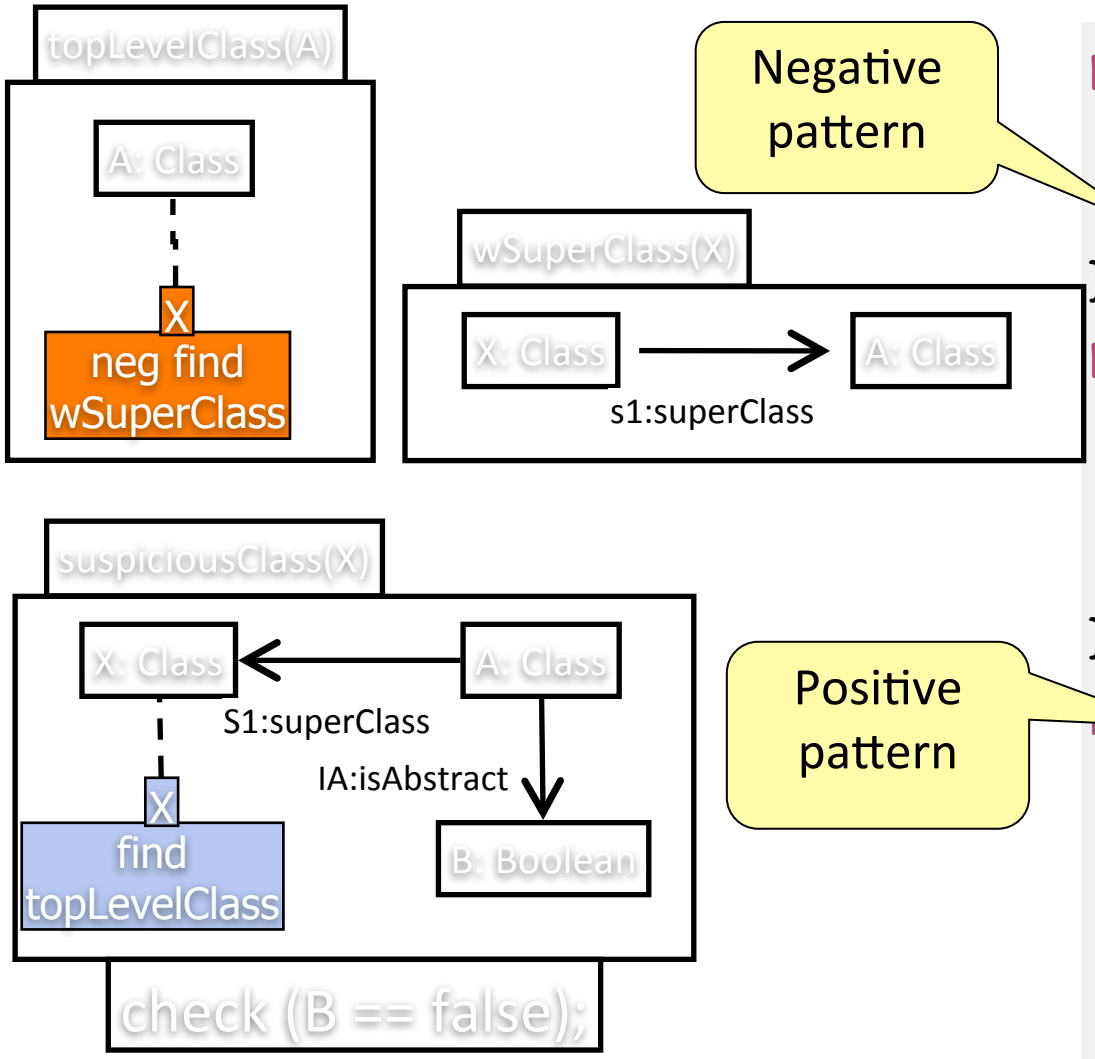


OR pattern

```
// B is a sibling class of A
pattern siblingClass(A, B) =
{
    Class(A);
    Class.superClass(S1, A, P);
    Class(P);
    Class.superClass(S2, B, P);
    Class(B);
```

```
// S is locally substitutable by A
pattern localSubs(A, S, X) = {
    Class(A);
    Iface(X);
    Class.implements(I1, A, X);
    Class(S);
    Class.implements(I2, S, X);
} or {
    Class(A);
    Class(X);
    Class.superClass(P1, X, X);
    Class(S);
```

Positive and Negative patterns (VTCL)

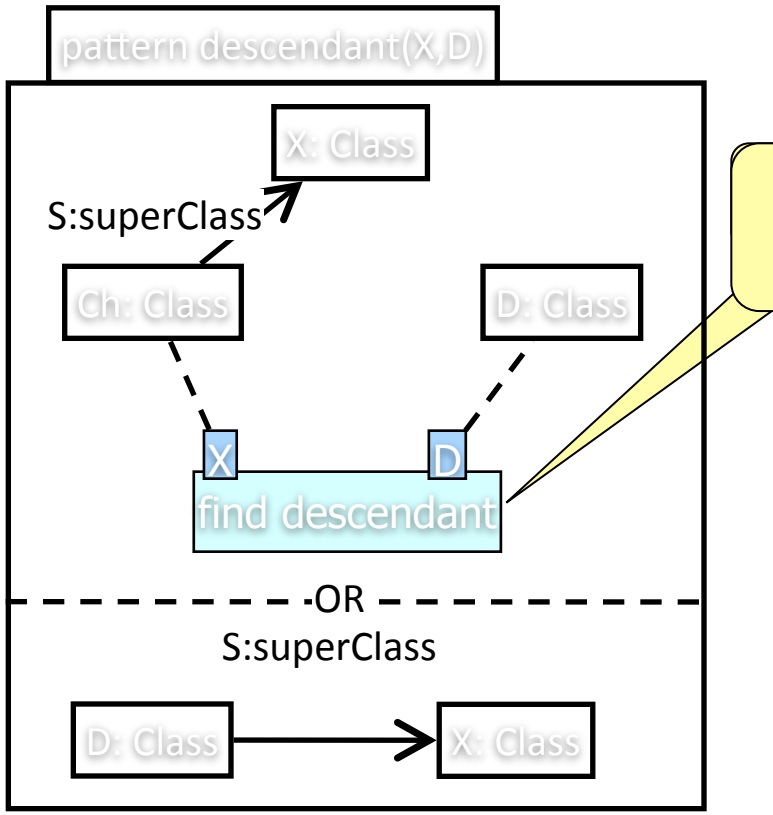


Negative pattern

Positive pattern

```
pattern topLevelClass(A) = {
    Class(A);
    neg find subClass(A);
}
pattern wSuperClass(X) = {
    Class(X);
    Class.superClass(S1, X, A);
    Class(A);
}
pattern suspiciousClass(X) = {
    Class(X);
    find topLevelClass(X);
    Class.superClass(S1, A, X);
    Class(A);
    Class.isAbstract(IA, A, B);
    Boolean(B);
```

Recursive patterns (VTCL)



Recursive pattern

```
pattern descendant(X, D) = {
    Class(X);
    Class.superClass(S, Ch, X);
    Class(Ch);
    find descendant(Ch, D)
    Class(D);
} or {
    Class(X);
    Class.superClass(S, D, X);
    Class(D);
```

Origins of the idea

- Model transformations by VIATRA2
  - Transformation language
    - Declarative patterns for model queries
    - Graph transformation rules for elementary mapping specifications
    - ASM rules for control structure
  - Matching strategy
    - Local search-based (optimized search plans)
    - **Incremental pattern matching (based on RETE)**
    - Hybrid pattern matching (adaptive combination of INC and LS)
- Development team
  - Developed by BUTE and OptXware Ltd.
  - 9 developers
- More info
  - http://eclipse.org/gmt/VIATRA2
  - http://wiki.eclipse.org/VIATRA2

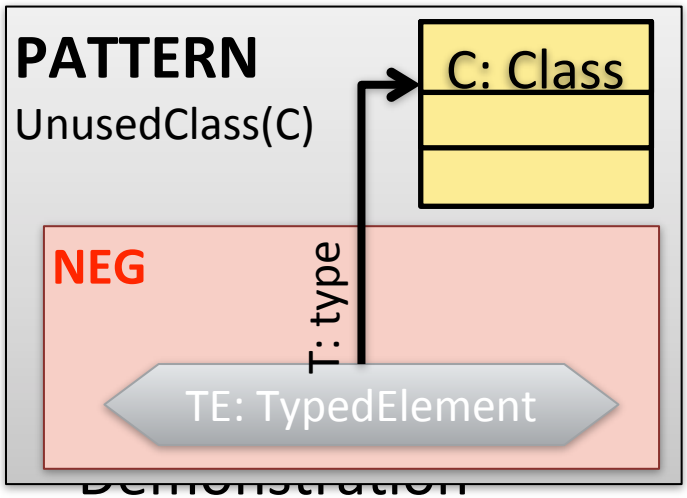
INCREMENTAL PATTERN MATCHING

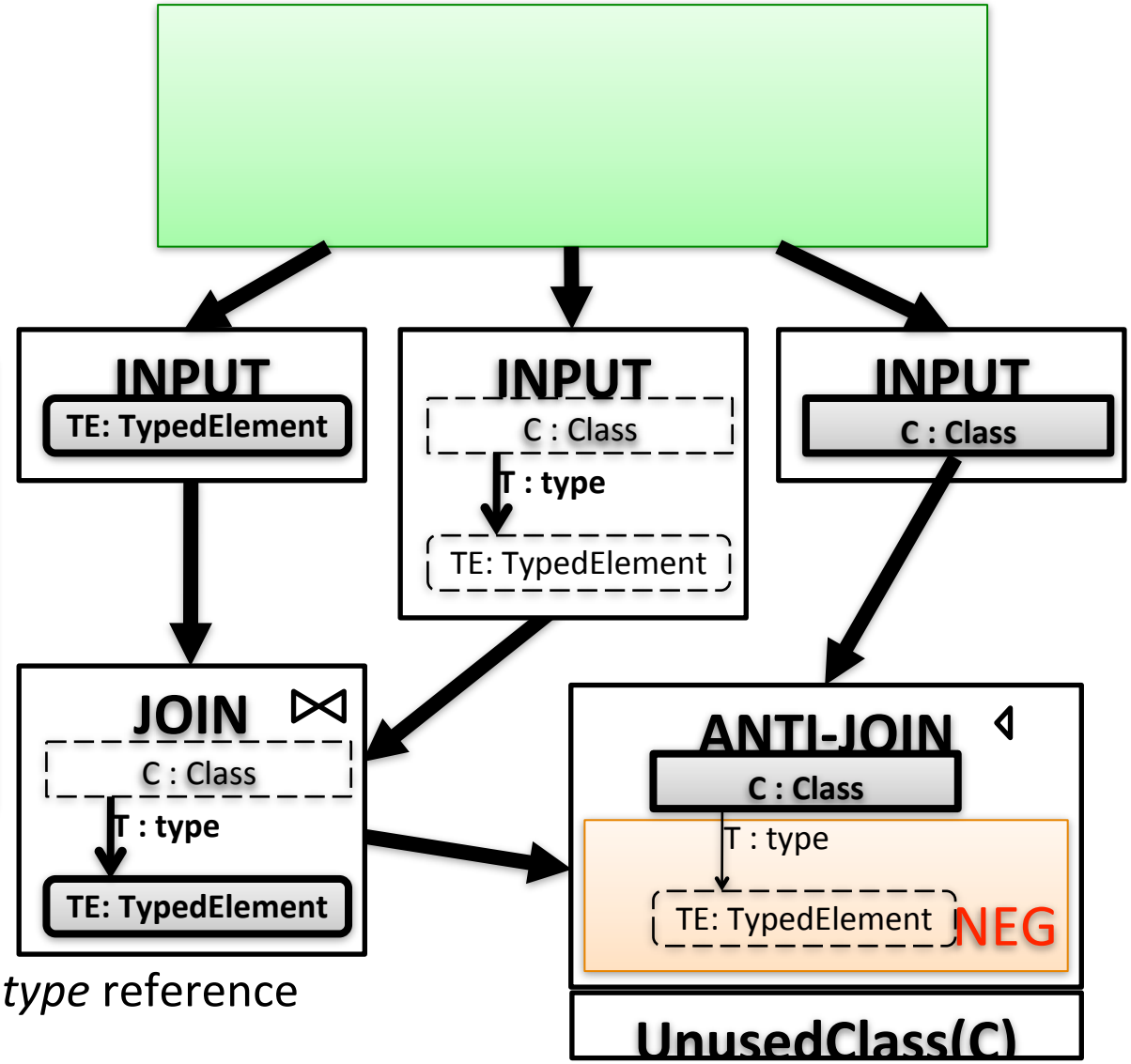
Contributions

- **Expressive** declarative query language by graph patterns
  - Capture local + global queries
  - Compositionality + Reusabilility
  - „Arbitrary” Recursion, Negation
- **Incremental** cache of matches (materialized view)
  - Cheap maintenance of cache (only memory overhead)
  - Notify about relevant changes (new match – lost match)
  - Enable reactions to complex structural events
- **High performance** for large models

RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation

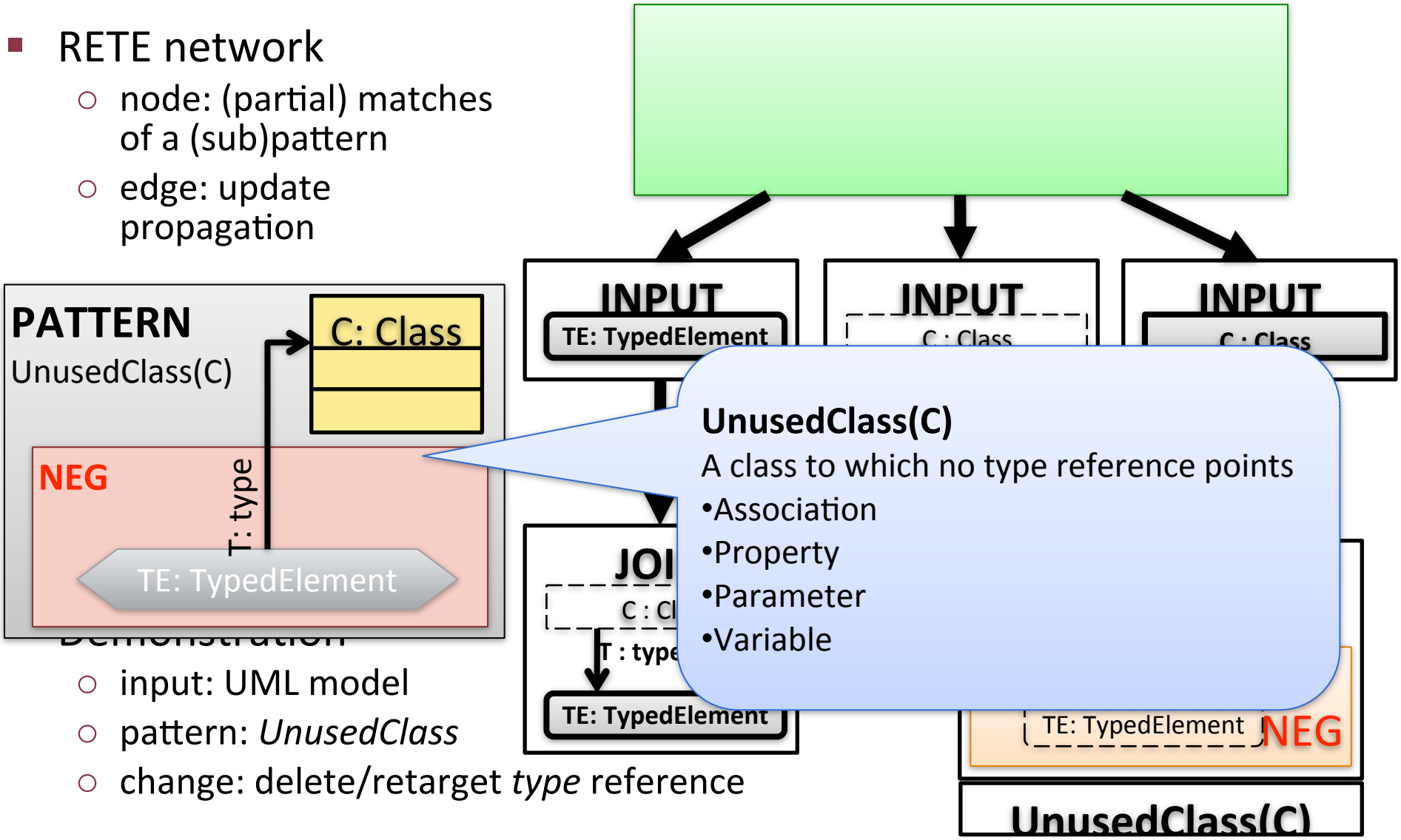




- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

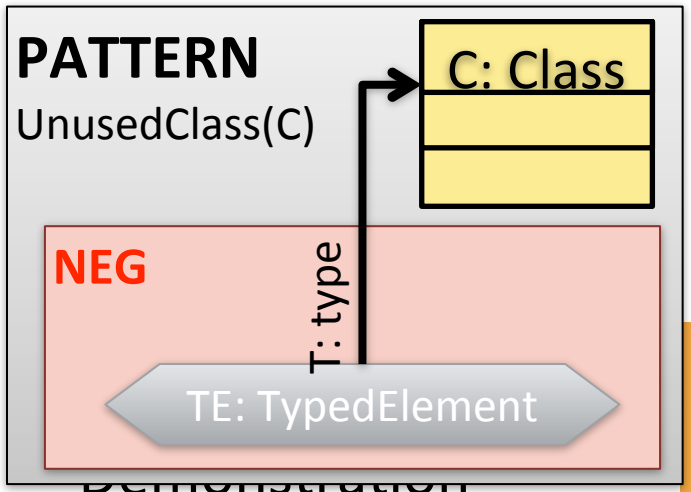
- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation

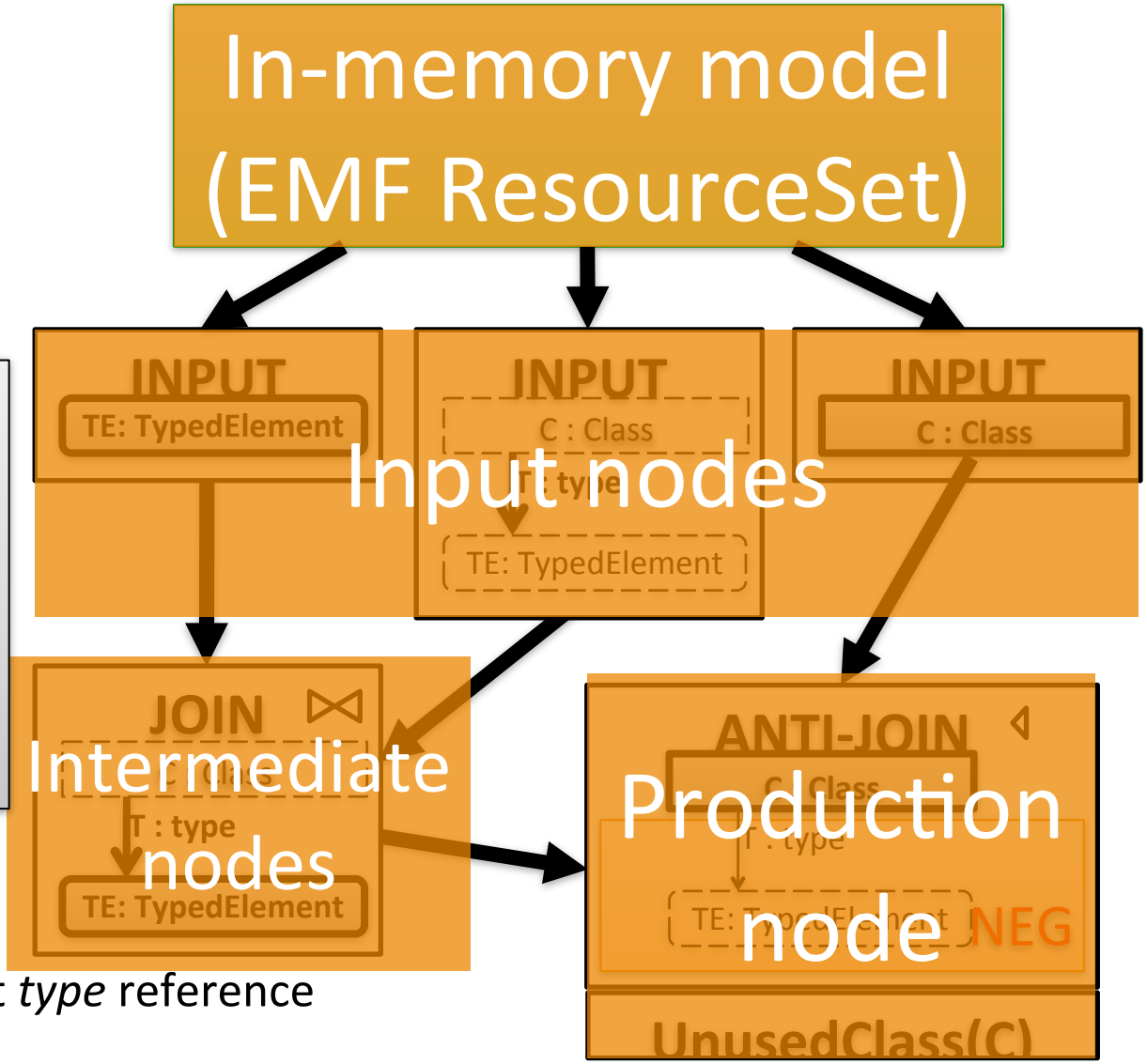


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation



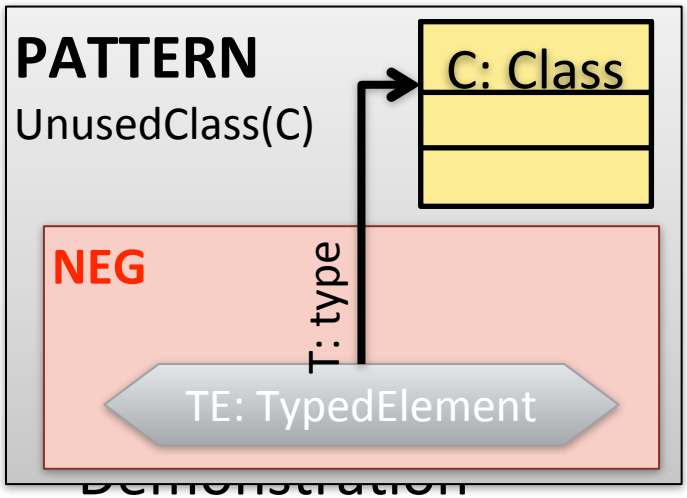


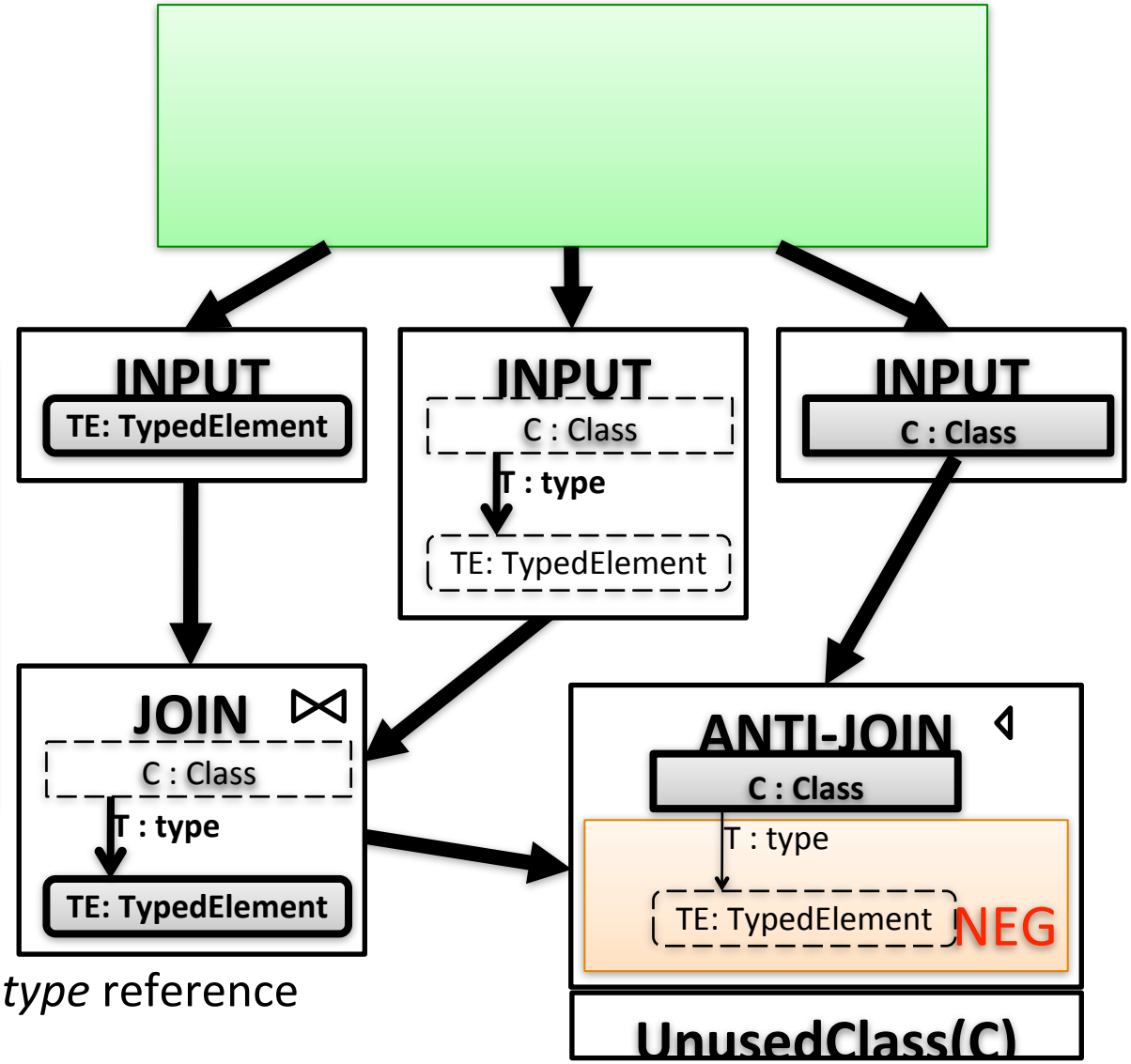
Demonstration

- input: UML model
- pattern: *UnusedClass*
- change: delete/retarget *type* reference

RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation

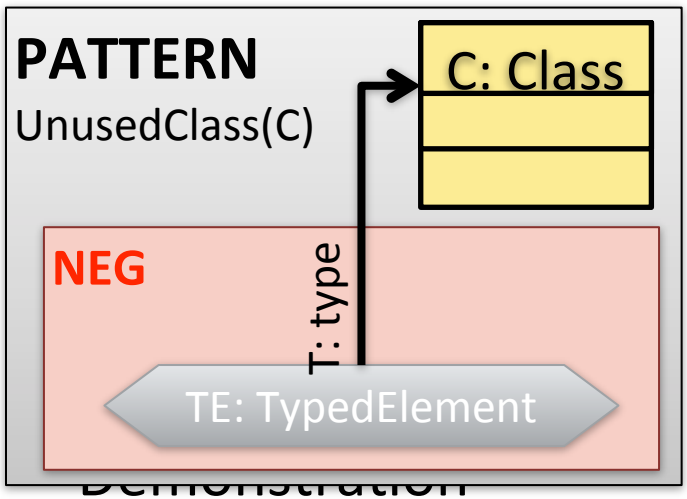


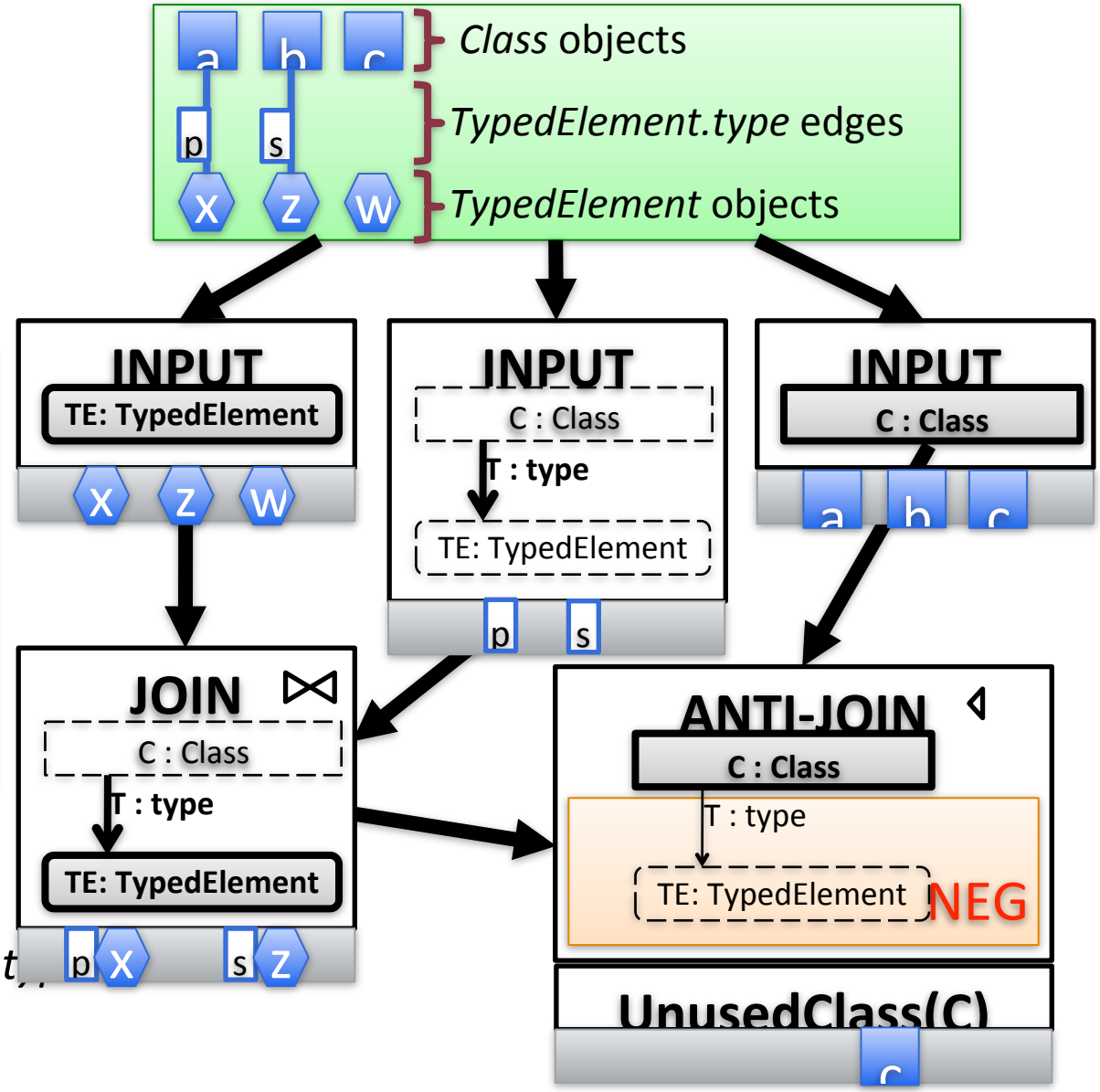


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation

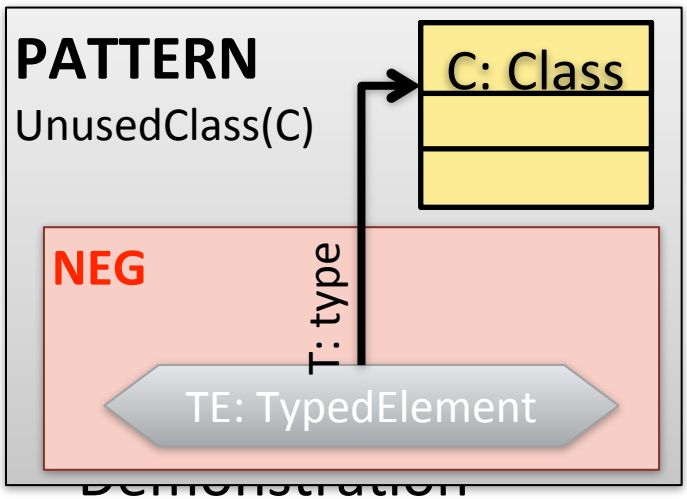


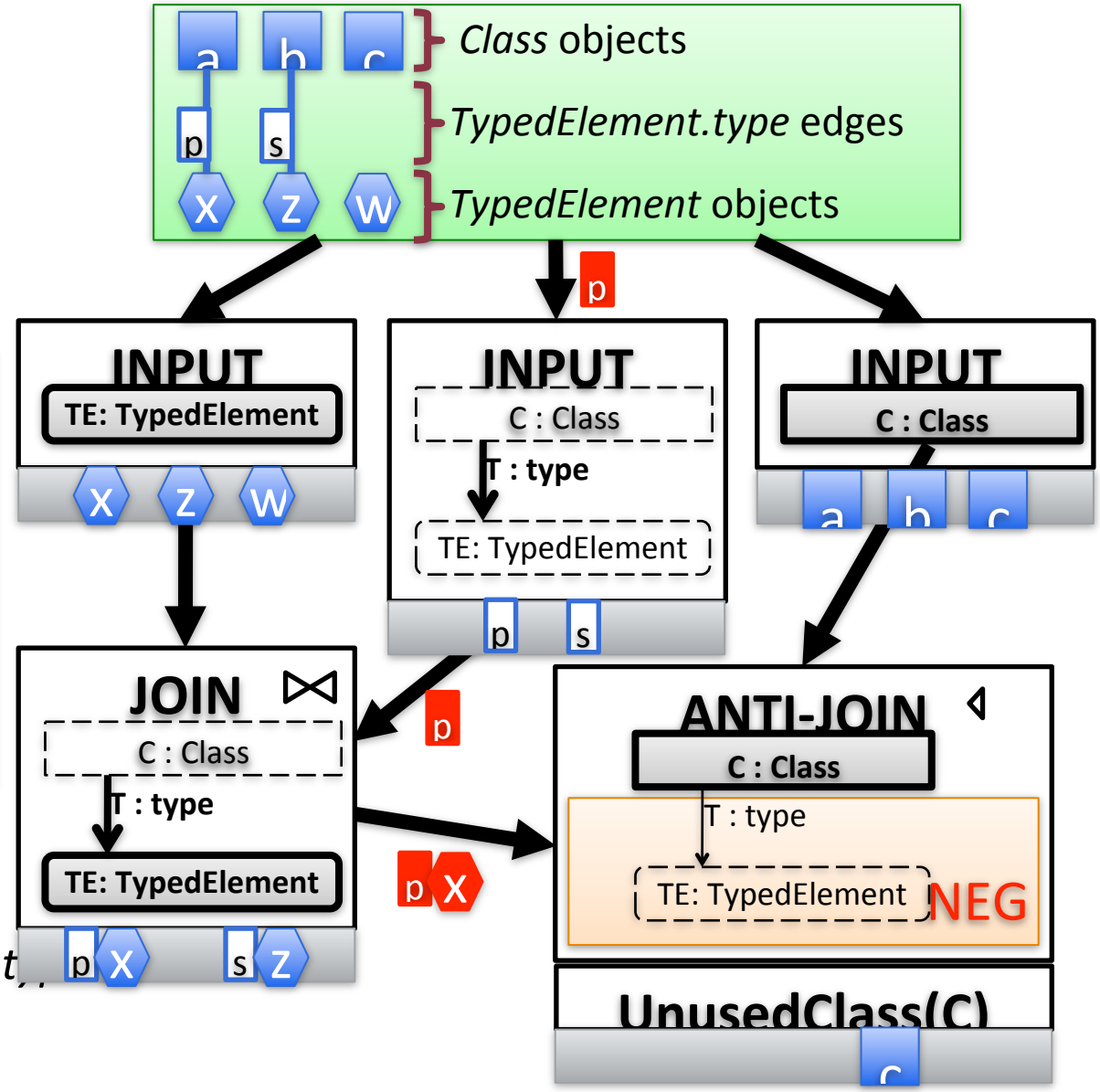


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *t*

RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation

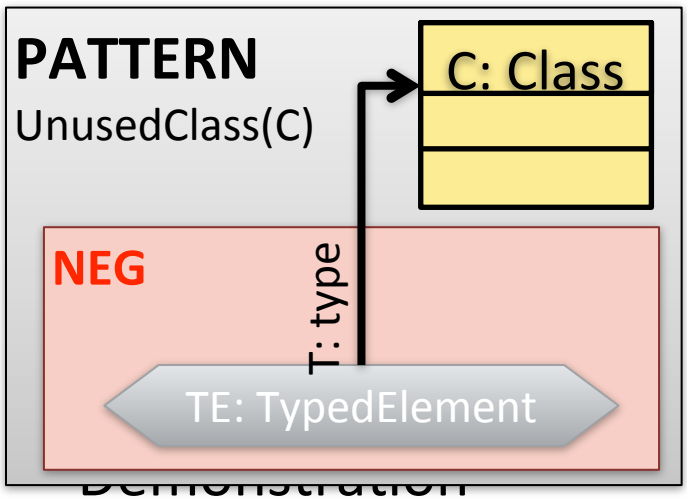


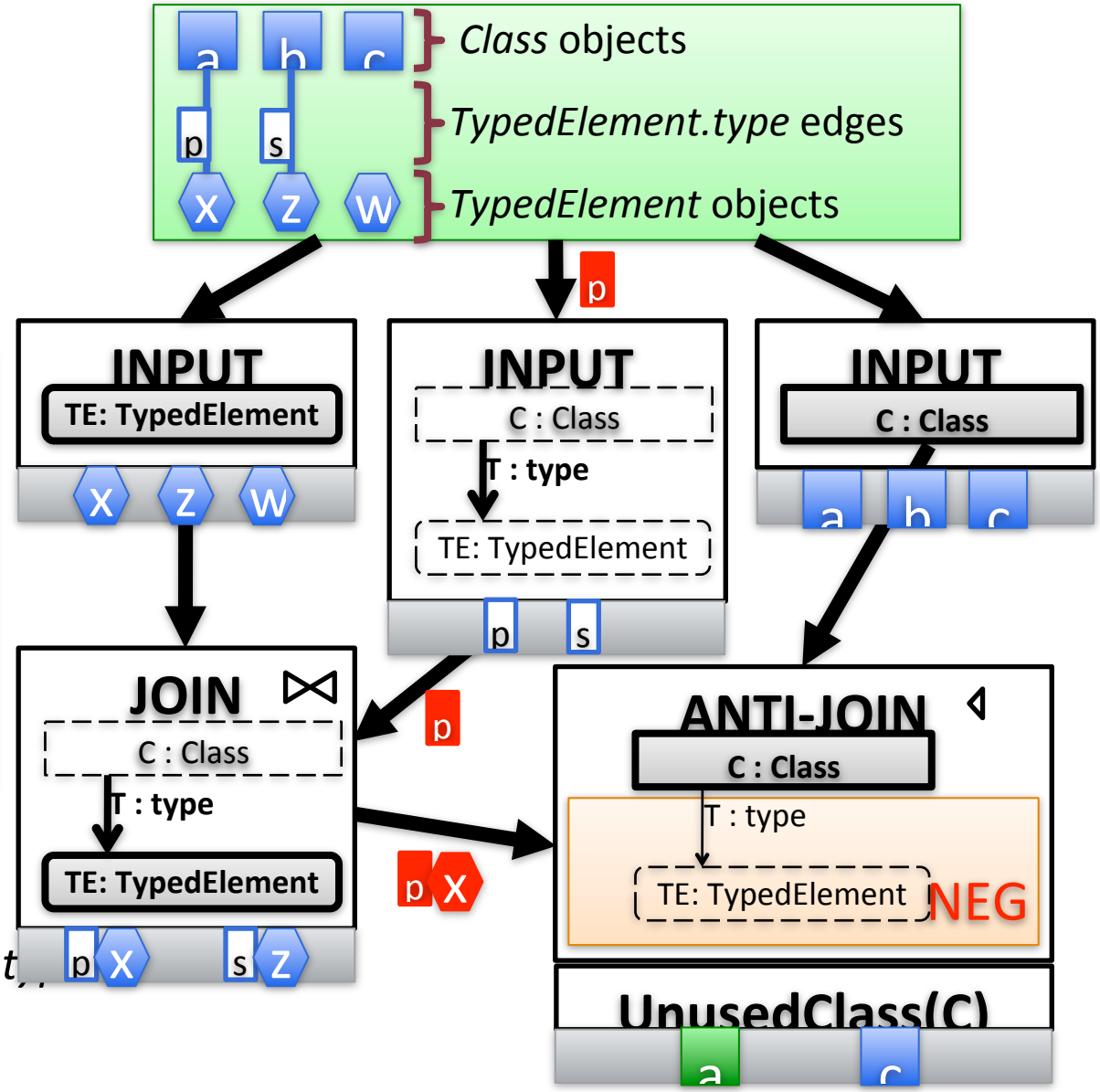


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *t*

RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation





- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *t*

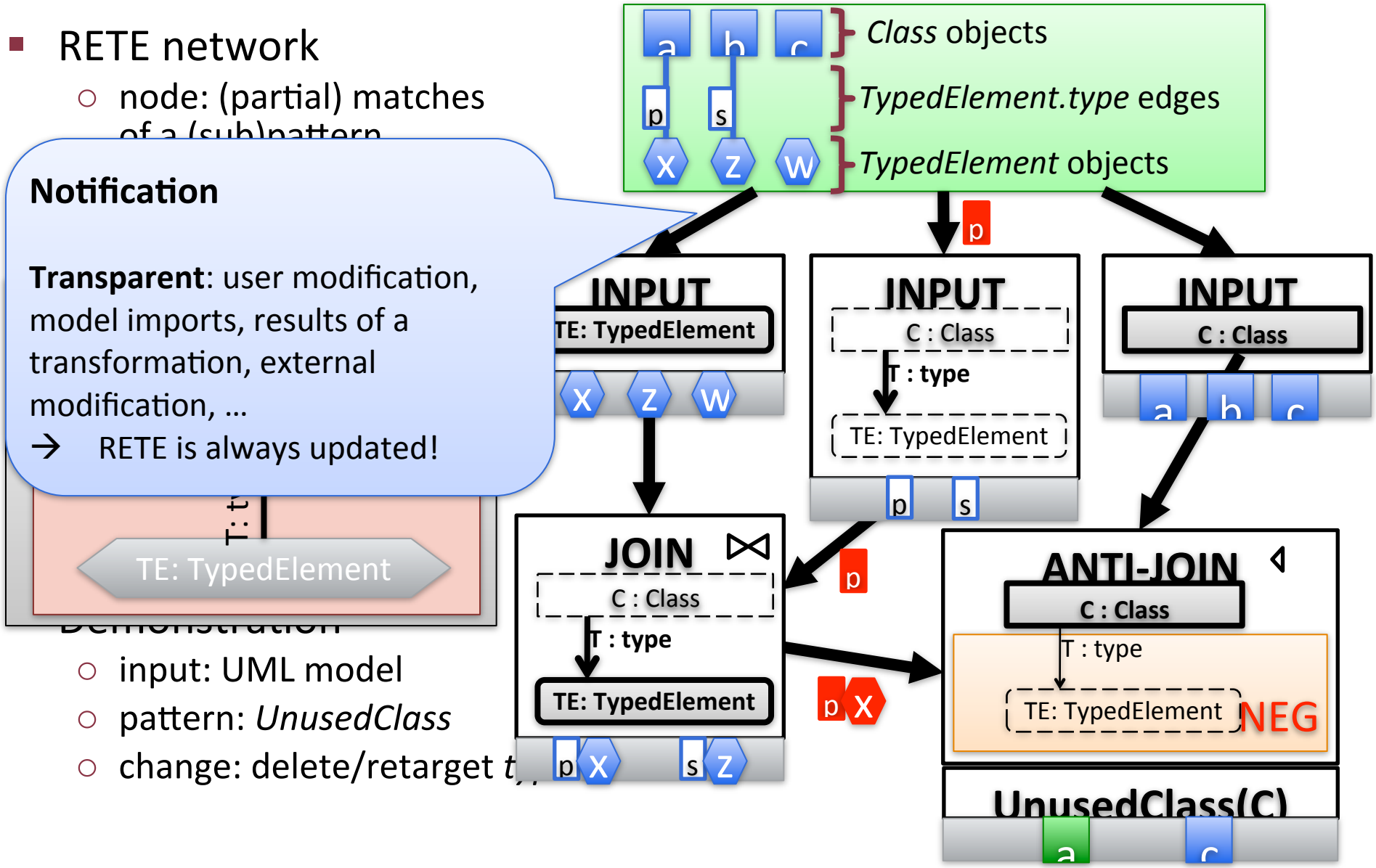
RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern
  - Demonstration
    - input: UML model
    - pattern: *UnusedClass*
    - change: delete/retarget *t*

Notification

Transparent: user modification, model imports, results of a transformation, external modification, …

→ RETE is always updated!



RETE nets

- RETE network
  - node: (partial) matches of a (sub)pattern

Notification

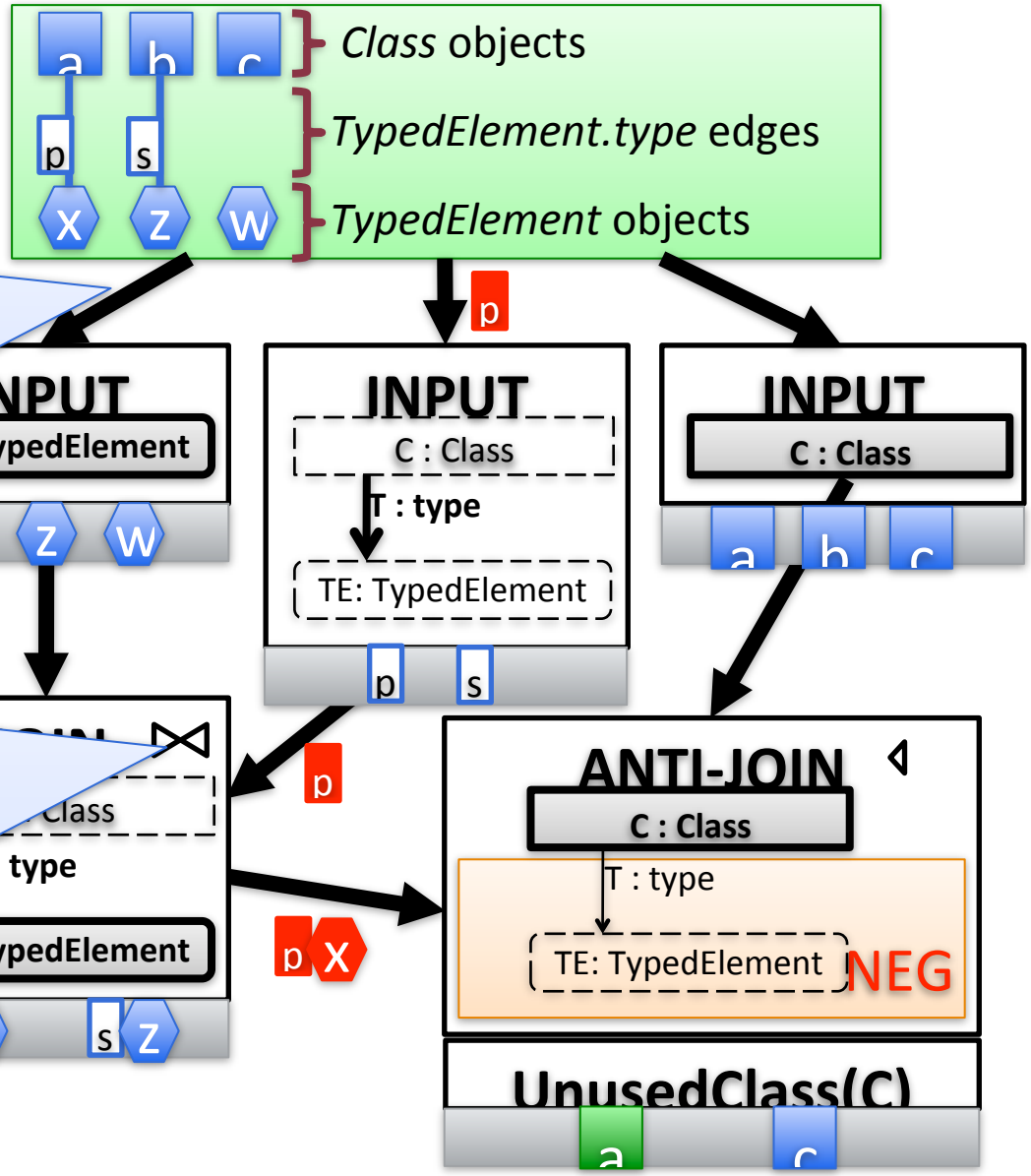
Transparent: user modification, model imports, results of a transformation, external modification, …

→ RETE is always updated!

Experimental results:

good, if…

- There is enough memory
- Transactional model manipulation

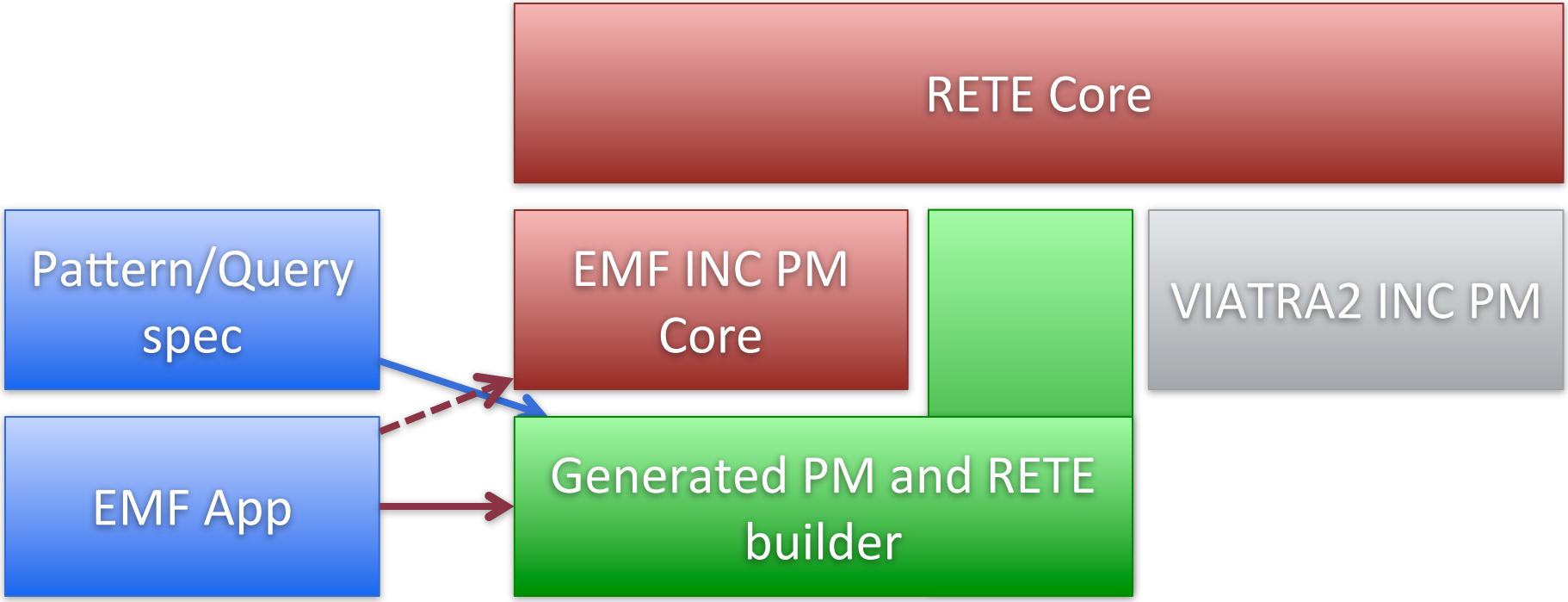


EMF-INCQUERY

Overview

- What is EMF-INCQuery?
  - Query language + incremental pattern matcher + development tools for EMF models
    - Works with any *(pure)* EMF application
    - Reusability by pattern composition
    - Arbitrary recursion, negation
    - Generic and parameterized model queries
    - Bidirectional navigability
    - Immediate access to all instances of a type
    - Complex change detection
- Benefits
  - Fully declarative + Scalable performance

Architectural overview

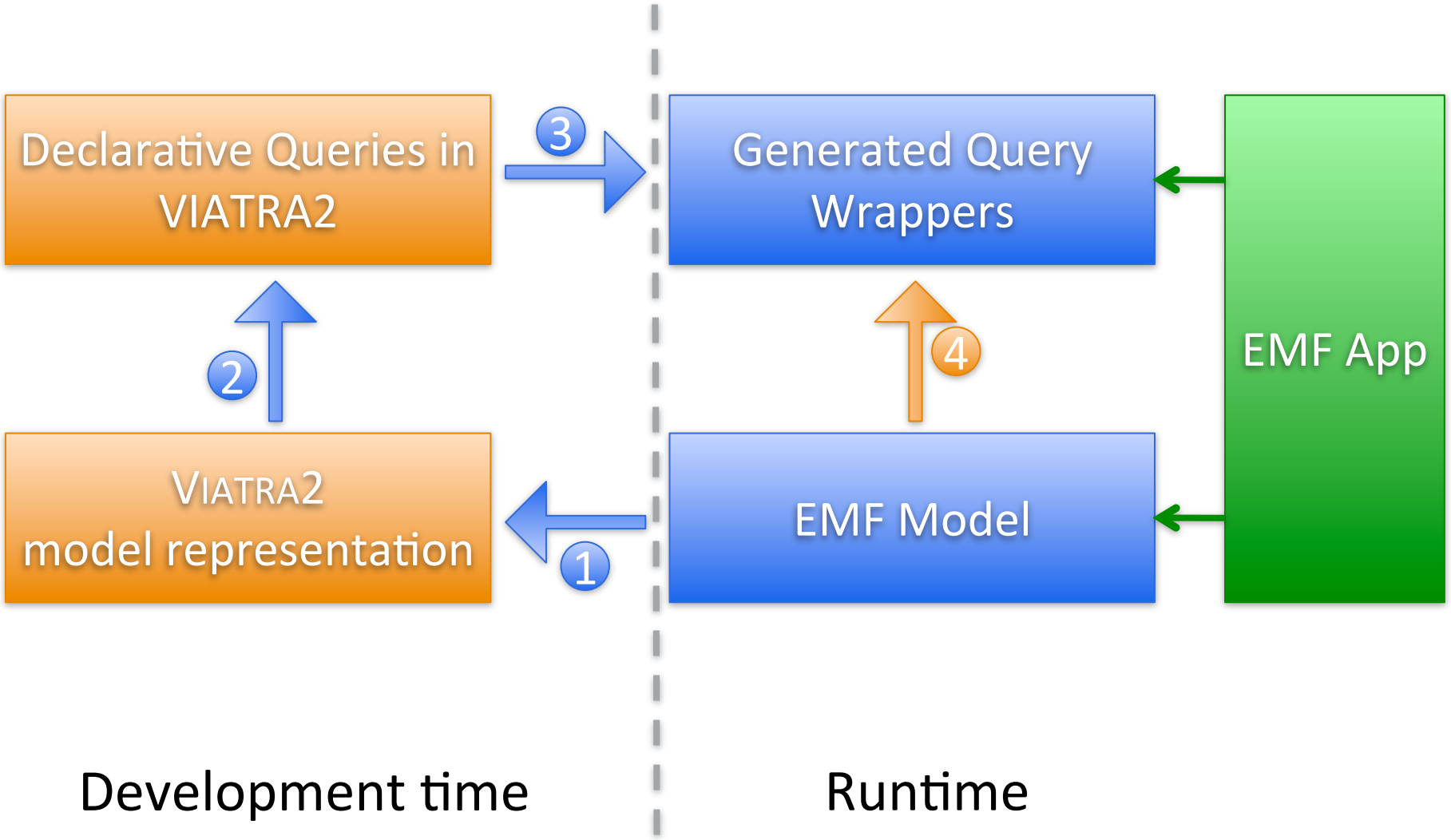


EMF-INCQUERY TOOLING

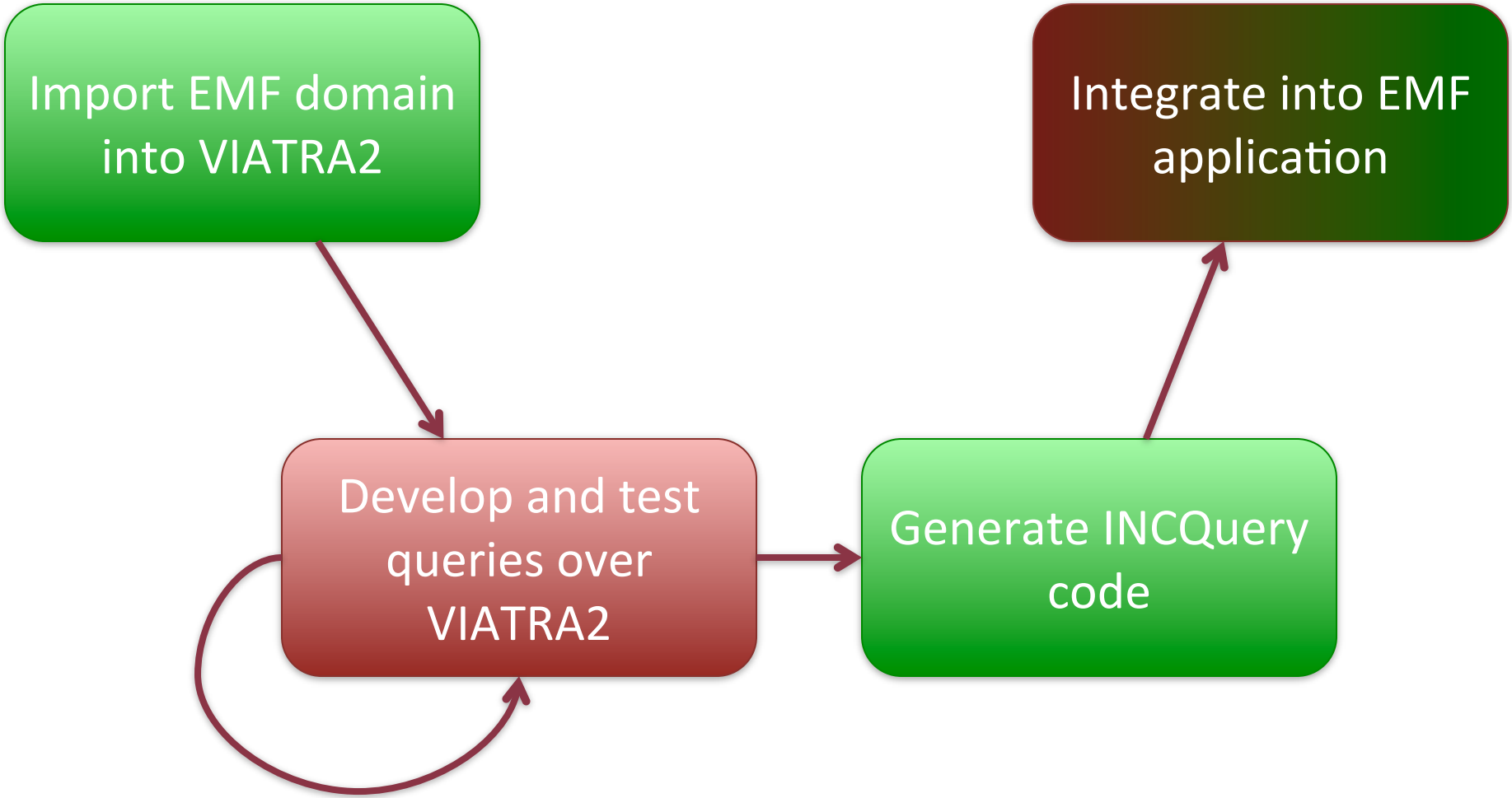
EMF-IncQuery Tooling

- Generative approach
  - compile queries into
    - RETE network builders
    - Query interface wrappers
  - end-to-end solution: generates Eclipse plug-in projects
- Relies on the VIATRA2 environment
  - query development
  - debugging

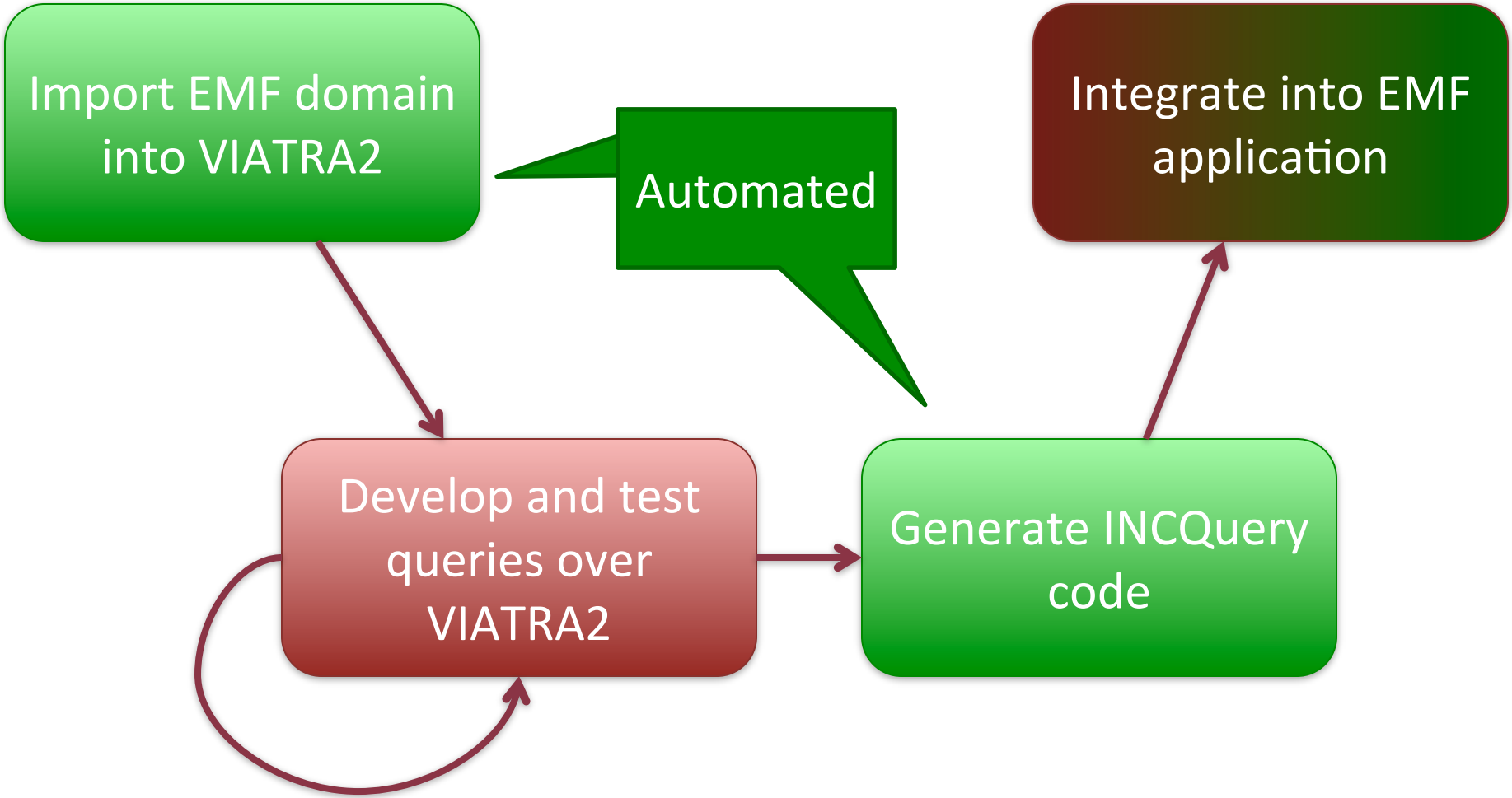
Tooling architecture



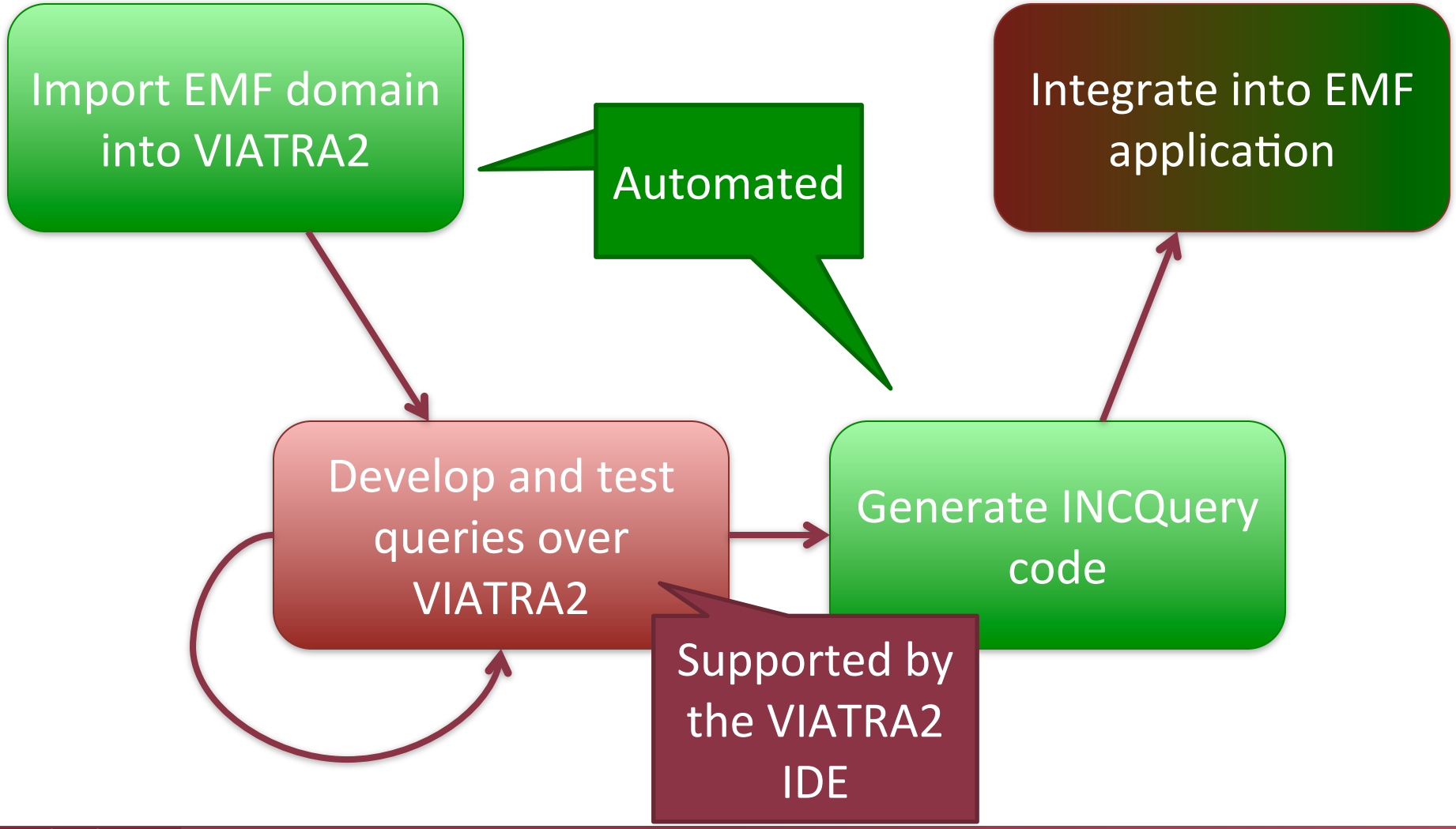
Development workflow



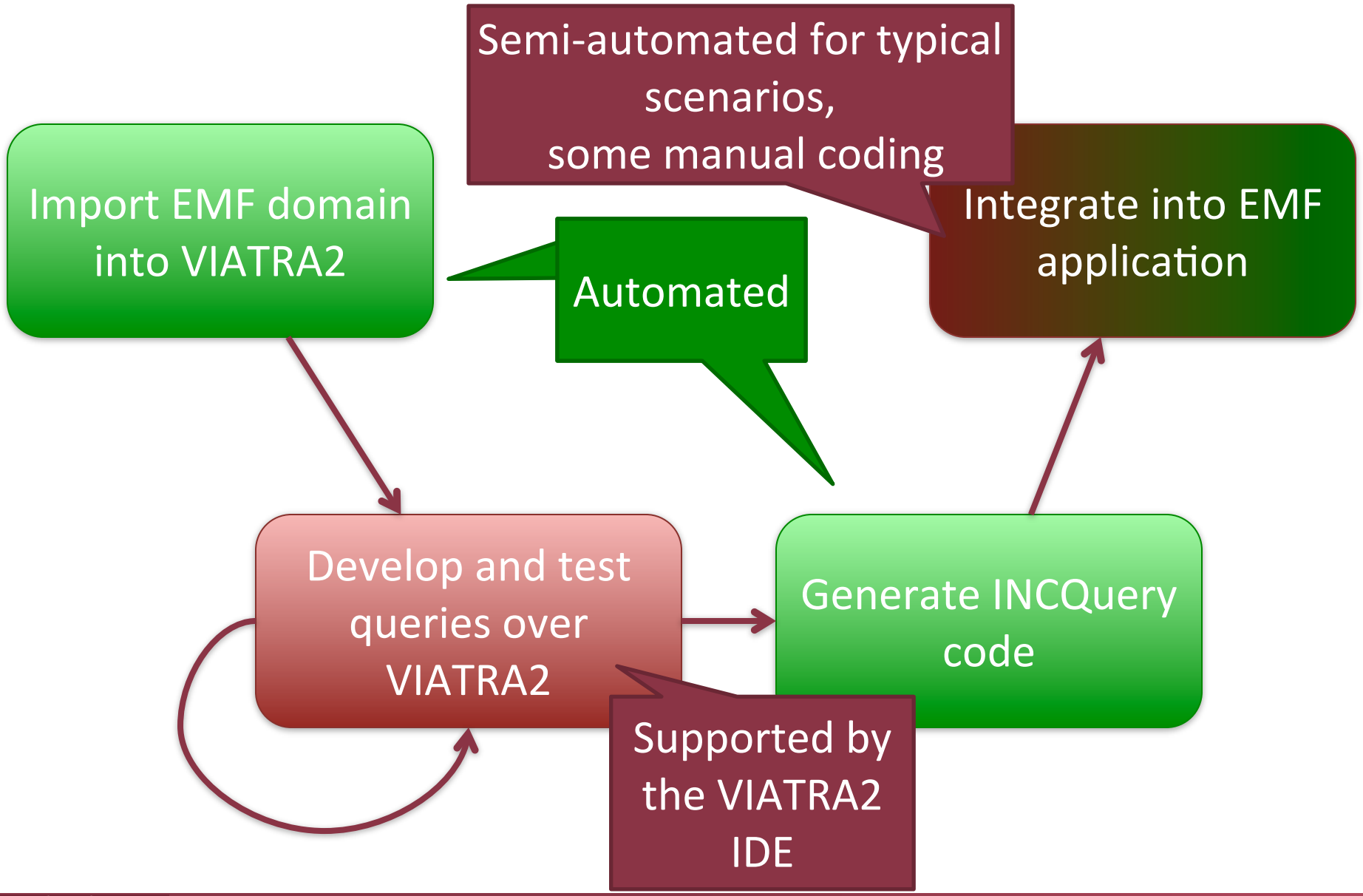
Development workflow



Development workflow



Development workflow

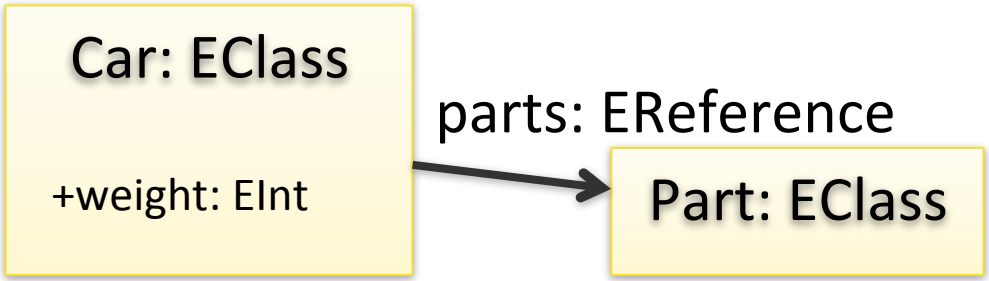


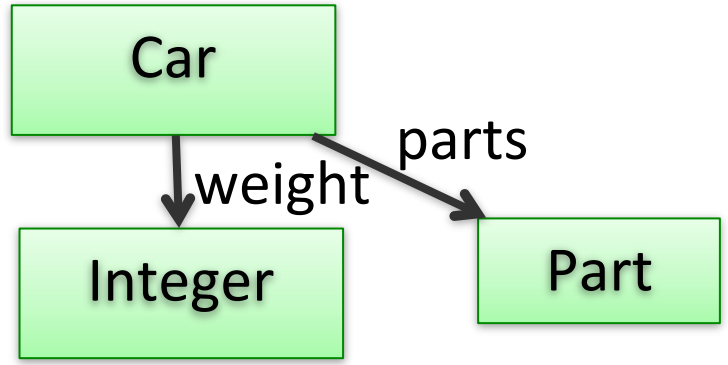
Importing EMF domains

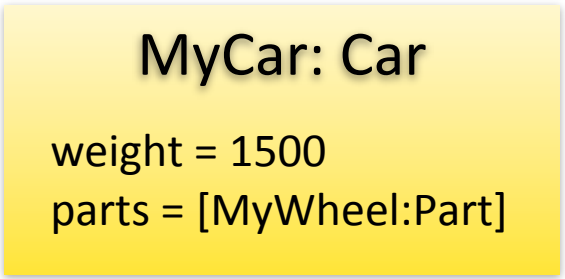
- Automated by the EMF2VIATRA plug-in (by OptXware Ltd.)
- Generic support to process
  - ECore metamodels and
  - EMF-based instance models
  - within the VIATRA2 model space
- Available as an independent open source add-on to VIATRA2

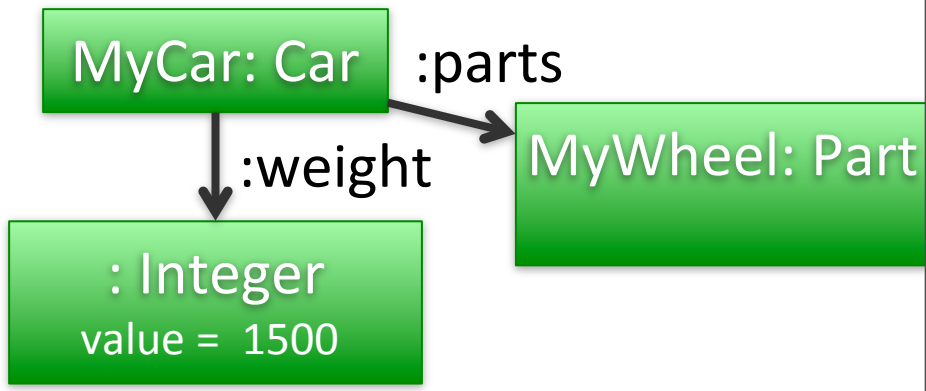
ECore models in VIATRA2

- Ecore
  - EClass
  - EReference
  - EAttribute
- VIATRA2
  - Entity (Node)
  - Relation (Edge)
  - Node+Edge









Developing and testing queries

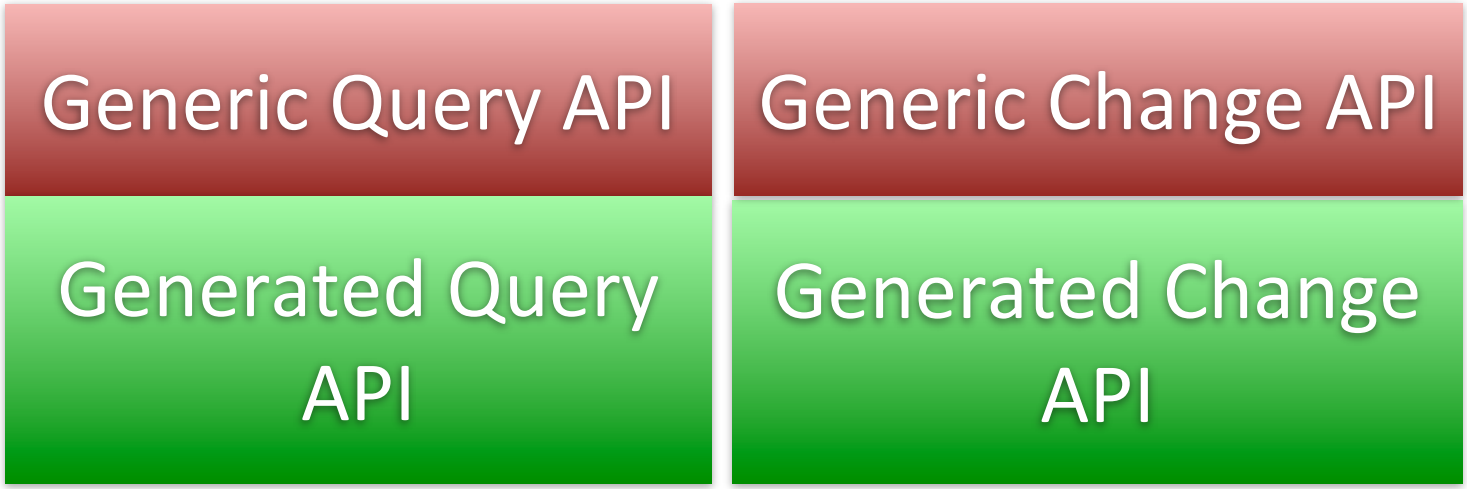
- Supported by the VIATRA2 framework
- LPG-based parser
  - Recovery parsing, error highlighting
  - Light-weight static analysis for patterns (type and metamodel conformance)
- Development features
  - Pattern graph visualization
  - Pattern match set visualization
  - Execute patterns (in transformations)

Generating INCQuery code

- Just a few clicks necessary
- Generated components: INCQuery runtime
  - Eclipse plugin
    - Depends only on EMF and the INCQuery core
    - No VIATRA2 dependency!
  - Private code: pattern builders
    - Parameterize the RETE core and the generic EMF PM library
  - Public API: Pattern matcher access layer
    - Query interfaces
    - Data Transfer Objects (DTOs)
    - Used to integrate to EMF applications

EMF-INCQUERY RUNTIME

Facilities


Generic Query API
Generic Change API
Generated Query API
Generated Change API

Generated Query API

- Basic queries
  - Uses tuples (object arrays) corresponding to pattern parameters
  - `Object[] getOneMatch()`
  - `Collection<Object[]> getAllMatches()`
- Parameterized queries
  - `getOneMatch(Object X, Object Y, …)`
  - `getAllMatches(Object X, Object Y, …)`
  - Null input values = unbound input variables

Based on pattern signature

Query DTOs

- **D**ata **T**ransfer **O**bjects generated for pattern signatures
- DTO query methods
  - `SiblingClassDTO getOneMatch()`
  - `SiblingClassDTO getOneMatch (Object C1, Object C2)`
  - `Collection<SiblingClassDTO> getAllMatches()`
  - `Collection<SiblingClassDTO> getAllMatches(Object C1, Object C2)`
- C1, C2: EObjects or datatype instances (String, Boolean, …)

```
// B is a sibling class of A
pattern siblingClass(C1, C2) =
{
   /* contents */
}
```

```
public class SiblingClassDTO
{
   Object C1;
   Object C2;
}
```

Query DTOs

- **D**ata **T**ransfer **O**bjects generated for pattern signatures
- DTO query methods
  - `SiblingClassDTO getOneMatch()`
  - `SiblingClassDTO getOneMatch (UMLClass C1, UMLClass C2)`
  - `Collection<SiblingClassDTO> getAllMatches()`
  - `Collection<SiblingClassDTO> getAllMatches(UMLClass C1, UMLClass C2)`
- C1, C2: EObjects or datatype instances (String, Boolean, …)

```
// B is a sibling class of A
pattern siblingClass(C1, C2) =
{
    /* contents */
}
```

```
public class SiblingClassDTO
{
   UMLClass C1;
   UMLClass C2;
}
```

Ongoing work: use domain-specific types in generated code

Change API

- Track changes in the match set of patterns (new/lost)
- Delta monitors
  - May be initialized at any time
  - `DeltaMonitor.matchFoundEvents` / `DeltaMonitor.matchLostEvents`
    - Queues of matches (tuples/DTOs) that have appeared/disappeared since initialization
- Typical usage
  - Listen to model manipulation (transactions)
  - After transaction commits:
    - Evaluate delta monitor contents and process changes
    - Remove processed tuples/DTOs from .matchFound/LostEvents

Integrating into EMF applications

- Pattern matchers may be initialized for
  - Any EMF Notifier
    - e.g. Resources, ResourceSets
  - TransactionalEditingDomains
- Initialization
  - Possible at any time
  - Involves a **single** exhaustive model traversal (independent of the number of patterns, pattern contents etc.)

Typical programming patterns

1. Initialize EMF model
   - Usually already done by your app ☺
2. Initialize incremental PM whenever necessary
3. Use the incremental PM for queries
   - Model updates will be processed transparently, with minimal performance overhead
   - Delta monitors can be used to track complex changes
4. Dispose the PM when not needed anymore
   - + Frees memory
   - - Re-traversal will be necessary

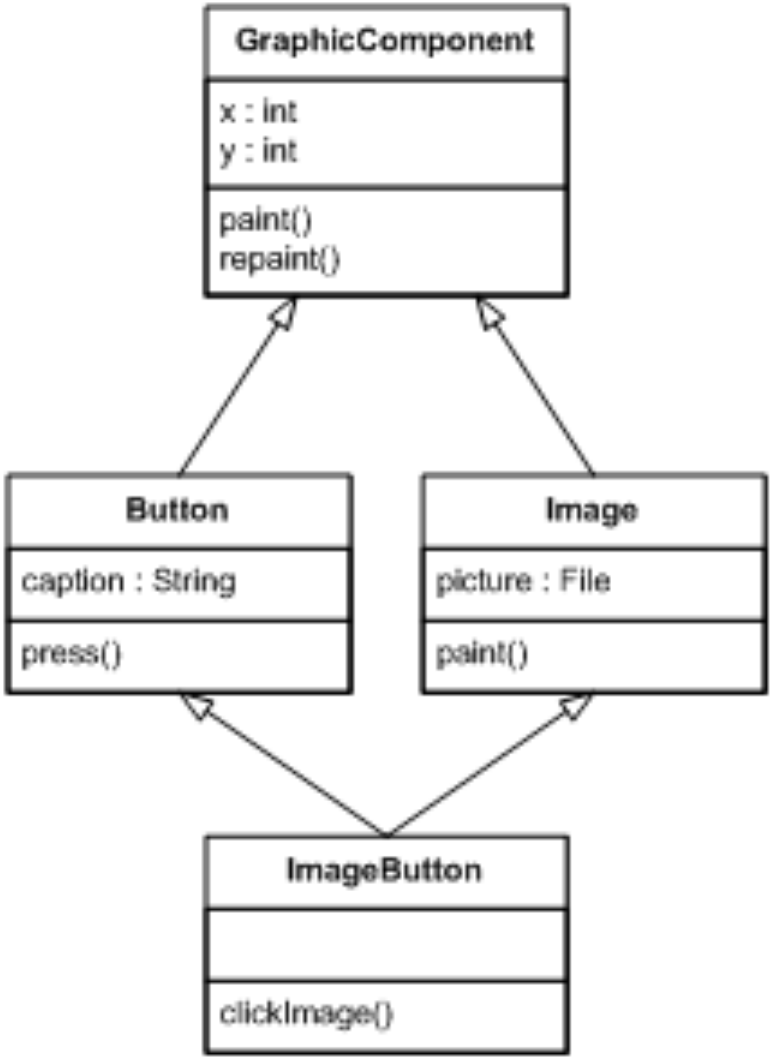
CASE STUDY

Case study: UML validation

- Goal:
  well-formedness constraint validation over UML2
- Requirements
  - Live / on-the-fly:
    instantaneous feedback as the user is editing
  - Incremental: fast for large models
- Ingredients
  - Papyrus UML
  - VIATRA2 R3.2
  - EMF-INCQuery

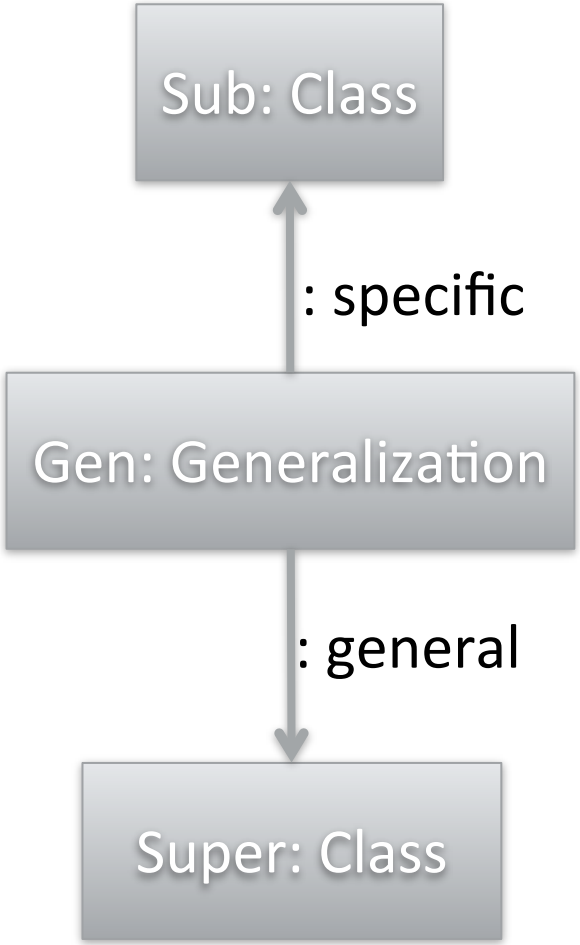
Example UML well-formedness constraint

- **Diamonds** are the girl's best friend
  - But our system architect's worst enemy ☺
- The "diamond" is a (local) anti-pattern
  - Multiple inheritance (and polymorphism) can be problematic in some programming languages
- Goal: detect diamond structures in UML class diagrams



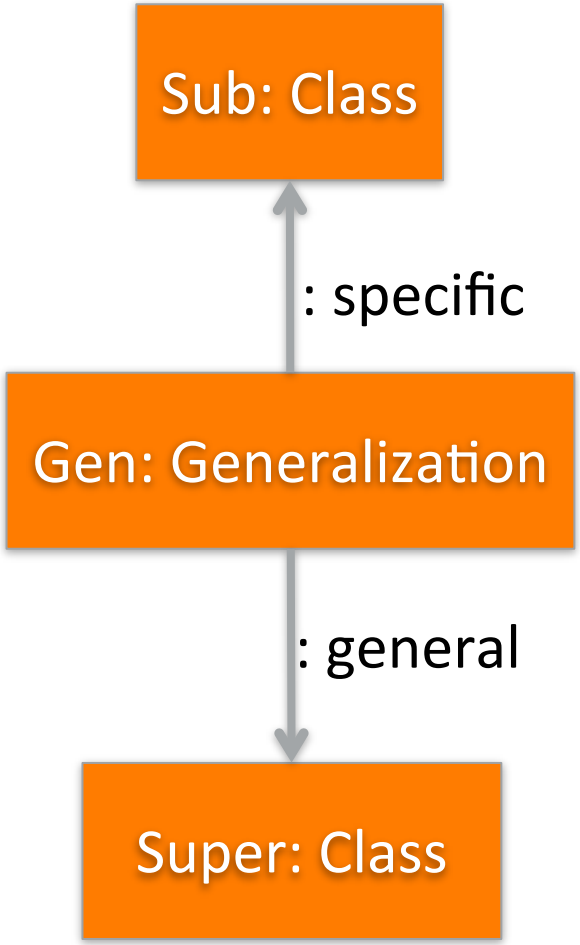
Formalization

```
pattern SuperType(Sub, Super) =
{
    Class(Sub);
    Generalization.specific(GS, Gen, Sub);
    Generalization(Gen);
    Generalization.general(GG, Gen, Super);
    Class(Super);
}
```



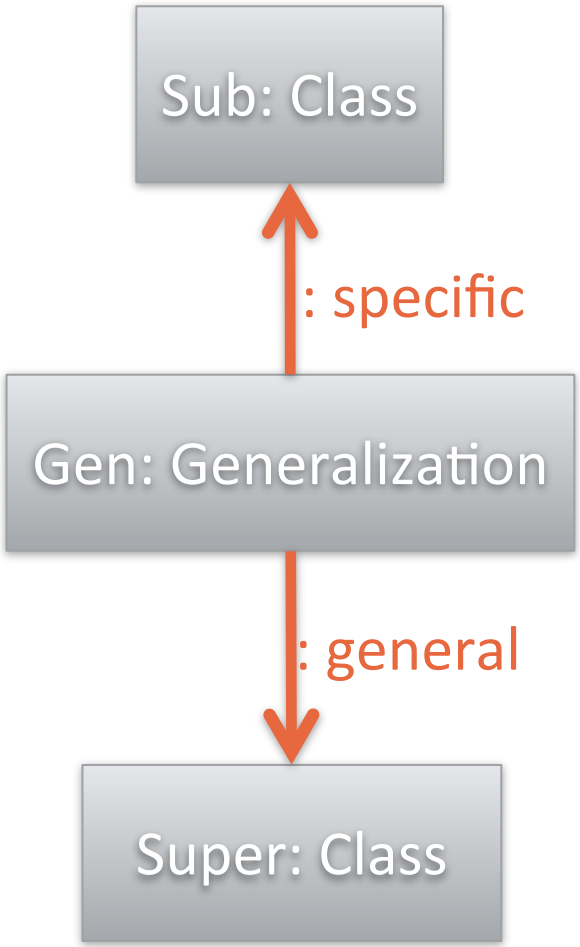
Formalization

```
pattern SuperType(Sub, Super) =
{
    Class(Sub);
    Generalization.specific(GS, Gen, Sub);
    Generalization(Gen);
    Generalization.general(GG, Gen, Super);
    Class(Super);
}
```



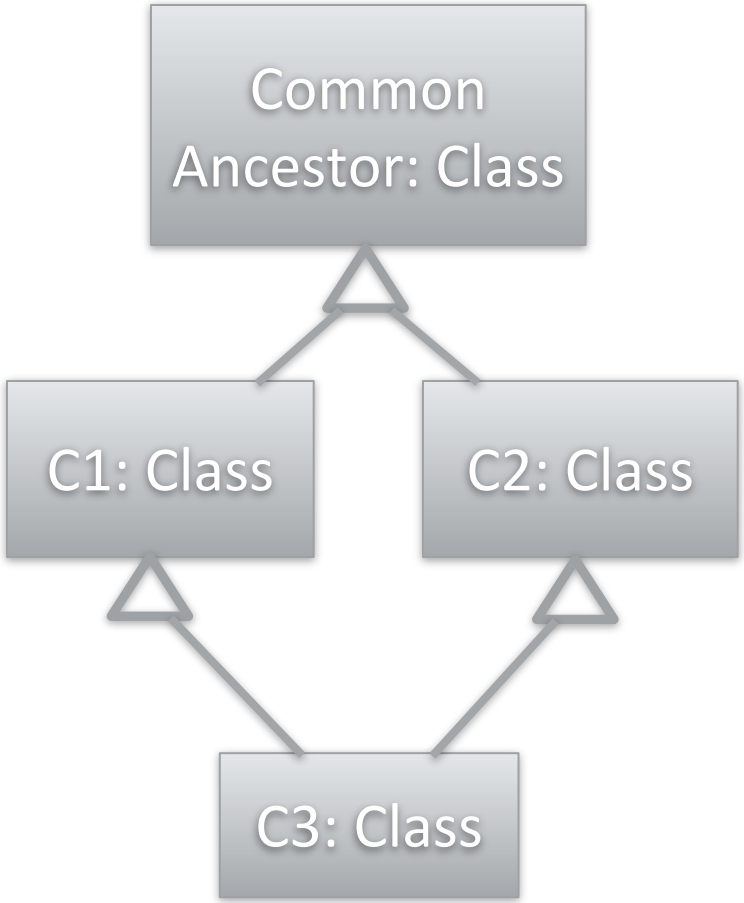
Formalization

```
pattern SuperType(Sub, Super) =
{
    Class(Sub);
    Generalization.specific(GS, Gen, Sub);
    Generalization(Gen);
    Generalization.general(GG, Gen, Super);
    Class(Super);
}
```



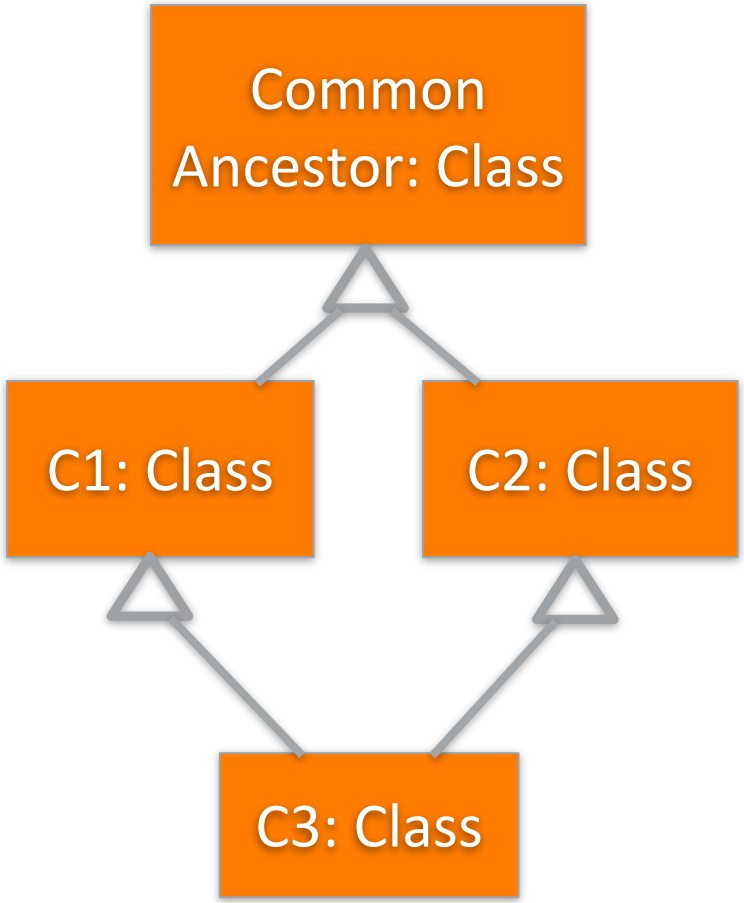
Formalization

```
pattern diamond(CA,C1,C2,C3) =
{
    Class(CA); Class(C1); Class(C2); Class(C3);
    find SuperType(C1,CA);
    find SuperType(C2,CA);
    find SuperType(C3,C1);
    find SuperType(C3,C2);
    neg find SuperType(C1,C2);
    neg find SuperType(C2,C1);
}

pattern SuperType(Sub, Super) =
{
    Class(Sub);
    Generalization.specific(GS, Gen, Sub);
    Generalization(Gen);
    Generalization.general(GG, Gen, Super);
```



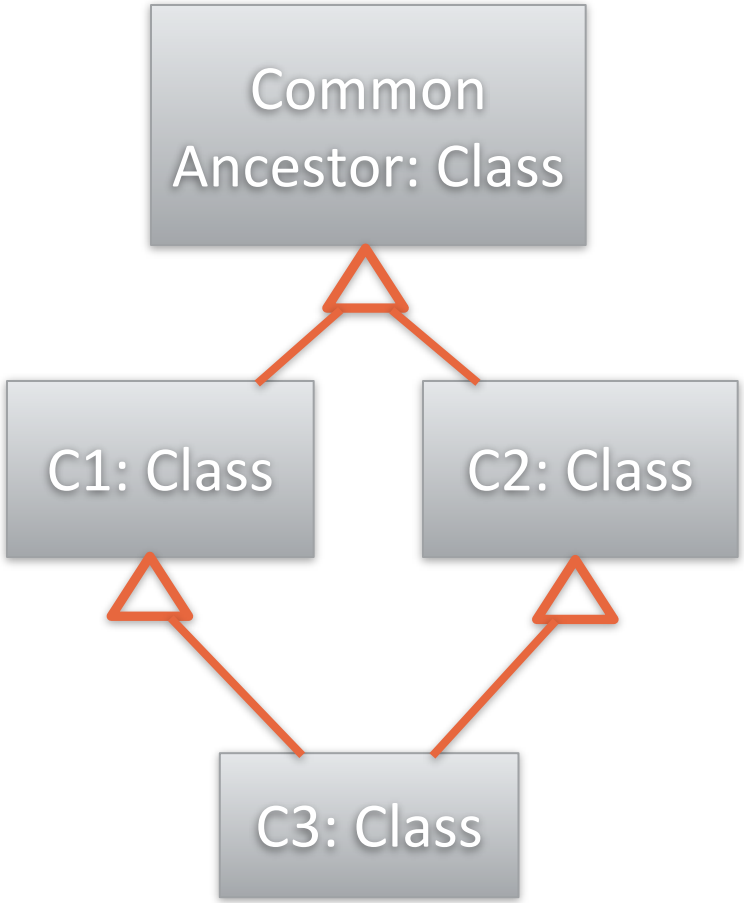
Formalization

```
pattern diamond(CA,C1,C2,C3) =
{
    Class(CA); Class(C1); Class(C2); Class(C3);
    find SuperType(C1,CA);
    find SuperType(C2,CA);
    find SuperType(C3,C1);
    find SuperType(C3,C2);
    neg find SuperType(C1,C2);
    neg find SuperType(C2,C1);
}

pattern SuperType(Sub, Super) =
{
    Class(Sub);
    Generalization.specific(GS, Gen, Sub);
    Generalization(Gen);
    Generalization.general(GG, Gen, Super);
```



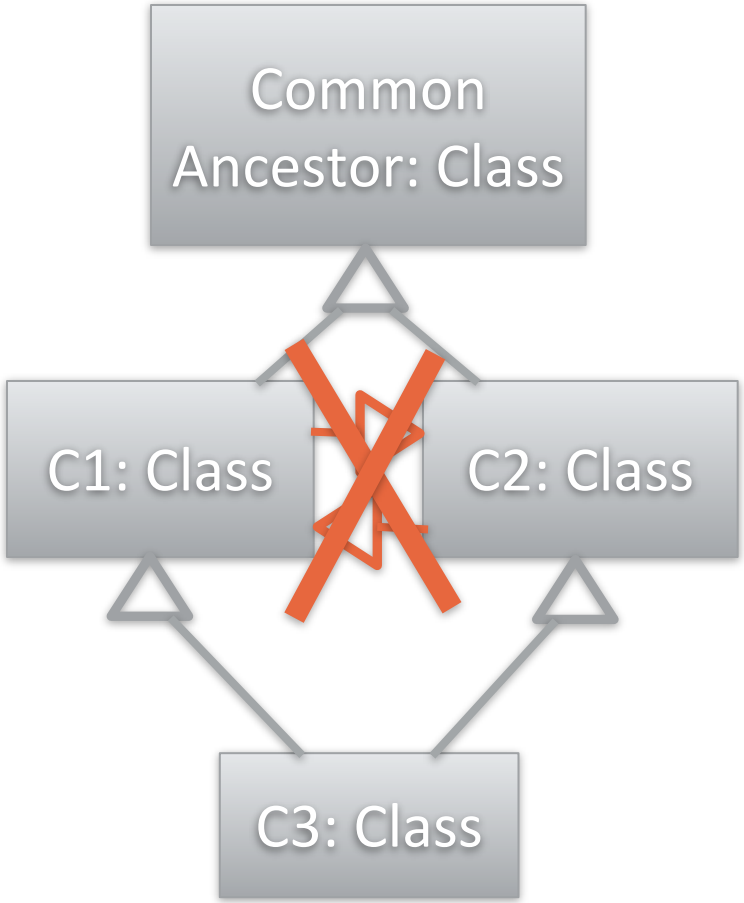
Formalization

```
pattern diamond(CA,C1,C2,C3) =
{
    Class(CA); Class(C1); Class(C2); Class(C3);
    find SuperType(C1,CA);
    find SuperType(C2,CA);
    find SuperType(C3,C1);
    find SuperType(C3,C2);
    neg find SuperType(C1,C2);
    neg find SuperType(C2,C1);
}

pattern SuperType(Sub, Super) =
{
    Class(Sub);
    Generalization.specific(GS, Gen, Sub);
    Generalization(Gen);
    Generalization.general(GG, Gen, Super);
```



Formalization

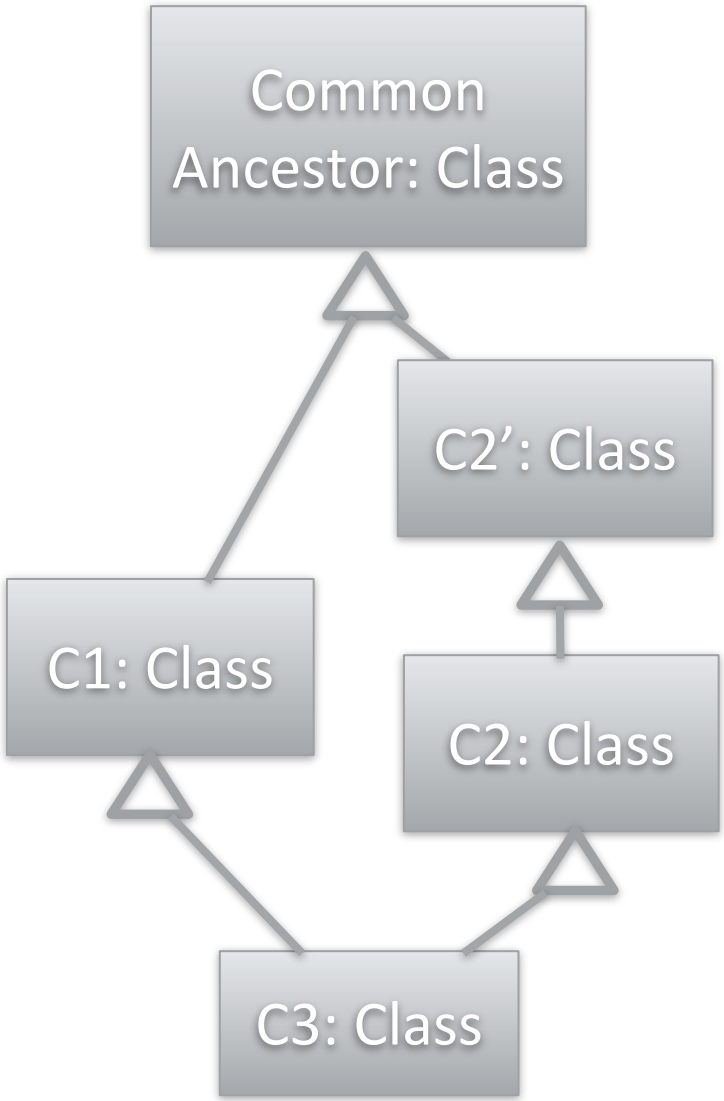
```
pattern diamond(CA,C1,C2,C3) =
{
    Class(CA); Class(C1); Class(C2); Class(C3);
    find SuperType(C1,CA);
    find SuperType(C2,CA);
    find SuperType(C3,C1);
    find SuperType(C3,C2);
    neg find SuperType(C1,C2);
    neg find SuperType(C2,C1);
}

pattern SuperType(Sub, Super) =
{
    Class(Sub);
    Generalization.specific(GS, Gen, Sub);
    Generalization(Gen);
    Generalization.general(GG, Gen, Super);
```



Formalization

- Problem: transitivity needs to be taken into account
- Solution: recursion

```
pattern transitiveSuperType(Sub, Super) =
{ find SuperType(Sub,Super); } OR {
    find transitiveSuperType(Sub, Middle);
    find SuperType(Middle,Super);
}
pattern diamond(CA,C1,C2,C3) =
{   Class(CA); Class(C1); Class(C2); Class(C3);
    find transitiveSuperType(C1,CA);
    find transitiveSuperType(C2,CA);
    find transitiveSuperType(C3,C1);
    find transitiveSuperType(C3,C2);
    neg find transitiveSuperType(C1,C2);
    neg find transitiveSuperType(C2,C1);
```



Pattern development in VIATRA2

- Import the UML metamodel
- Import UML instance models
- VTCL development environment
  - Create and debug patterns
  - Visualize patterns and pattern matches
  - Test queries/patterns in transformations

Generating INCQuery wrappers

- Create INCQuery project
- INCQuery project set-up and customization
- Generating code

Integration into Papyrus UML

- Papyrus uses org.eclipse.uml2 models internally
  - Managed in a standard ResourceSet
  - No TransactionalEditingDomain support
- Validation is facilitated using EMF-Validation
  - Inflexible
  - Manages live validation by a different paradigm
  - Constraints rely on manually coded model traversal

Not suitable for our purposes

Integration into Papyrus UML

- We implemented manually:
  - Custom light-weight validation extension (200 LOC)
    - Based on Eclipse resource markers
    - Generically reusable for any EMF model
  - UI integration (150 LOC)
    - Contributed menu item
    - Relevant part: 15 LOC
    - Generically reusable
  - Constraint-specific code
    - Graph patterns (18 LOC in VTCL)
    - Adapter class (~15 LOC in Java)
      - extends the light-weight validation feedback framework
- All the rest is automatically generated.

On-the-fly incremental validation by delta monitors

```
idm.getReteEngine().getAfterUpdateCallbacks().add(new Runnable() {

@Override
public void run() {
// after each model update, check the delta monitor
// FIXME should be: after each complete transaction, check the delta
   monitor

dm.matchFoundEvents.removeAll( processNewMatches(dm.matchFoundEvents,
   outline, f.getFile()) );
dm.matchLostEvents.removeAll( processLostMatches(dm.matchLostEvents,
   outline, f.getFile()) );

}
});
```

On-the-fly incremental validation by delta monitors

```
idm.getReteEngine().getAfterUpdateCallbacks().add(new Runnable() {

@Override
public void run() {
// after each model update, check the delta moni
// FIXME should be: after each complete transacti
   monitor

dm.matchFoundEvents.removeAll( processNewMatches(c
   outline, f.getFile()) );
dm.matchLostEvents.removeAll( processLostMatches(c
   outline, f.getFile()) );

}
});
```

Simple mechanism to register call-backs that execute after all updates have been propagated

On-the-fly incremental validation by delta monitors

```
idm.getReteEngine().getAfterUpdateCallbacks().add(new Runnable() {

@Override
public void run() {
// after each model update, check the delta moni
// FIXME should be: after each complete transacti
   monitor

dm.matchFoundEvents.removeAll( processNewMatches(c
   outline, f.getFile()) );
dm.matchLostEvents.removeAll( processLostMatches(c
   outline, f.getFile()) );

}
});
```

Simple mechanism to register call-backs that execute after all updates have been propagated

Core technique of processing changes

On-the-fly incremental validation by delta monitors

```
private Collection<InheritanceDiamondDTO> processNewMatches
   (Collection<InheritanceDiamondDTO> dtos, IFile f) throws
   CoreException {
Vector<InheritanceDiamondDTO> processed = new
   Vector<InheritanceDiamondDTO>();
for (InheritanceDiamondDTO diamond : dtos) {
   Vector<EObject> affectedElements = new Vector<EObject>();
   affectedElements.add((EObject)diamond.getA());
   affectedElements.add((EObject)diamond.getB());
   affectedElements.add((EObject)diamond.getC());
   affectedElements.add((EObject)diamond.getD());
   ValidationUtil.markProblem(f, affectedElements,
       ValidationProblemKind.DIAMOND);
   processed.add(diamond);
}
return processed;
}
```

On-the-fly incremental validation by delta monitors

```
private Collection<InheritanceDiamondDTO> processNewMatches
   (Collection<InheritanceDiamondDTO> dtos, IFile f) throws
   CoreException {
Vector<InheritanceDiamondDTO> processed = new
   Vector<InheritanceDiamondDTO>();
for (InheritanceDiamondDTO diamond : dtos) {
   Vector<EObject> affectedElements = new Vecto
   affectedElements.add((EObject)diamond.getA());
   affectedElements.add((EObject)diamond.getB());
   affectedElements.add((EObject)diamond.getC());
   affectedElements.add((EObject)diamond.getD());
   ValidationUtil.markProblem(f, affectedElements,
       ValidationProblemKind.DIAMOND);
   processed.add(diamond);
}
return processed;
}
```

Process pattern signature DTOs

On-the-fly incremental validation by delta monitors

```
private Collection<InheritanceDiamondDTO> processNewMatches
   (Collection<InheritanceDiamondDTO> dtos, IFile f) throws
   CoreException {
Vector<InheritanceDiamondDTO> processed = new
   Vector<InheritanceDiamondDTO>();
for (InheritanceDiamondDTO diamond : dtos) {
   Vector<EObject> affectedElements = new Vecto
   affectedElements.add((EObject)diamond.getA());
   affectedElements.add((EObject)diamond.getB());
   affectedElements.add((EObject)diamond.getC());
   affectedElements.add((EObject)diamond.getD());
   ValidationUtil.markProblem(f, affectedElements,
       ValidationProblemKind.DIAMOND);
   processed.add(diamond);
}
return processed;
}
```

Process pattern signature DTOs

Use light-weight validation facility to create error marker

On-the-fly incremental validation by delta monitors

```
private Collection<InheritanceDiamondDTO> processNewMatches
   (Collection<InheritanceDiamondDTO> dtos, IFile f) throws
   CoreException {
Vector<InheritanceDiamondDTO> processed = new
   Vector<InheritanceDiamondDTO>();
for (InheritanceDiamondDTO diamond : dtos) {
   Vector<EObject> affectedElements = new Vecto
   affectedElements.add((EObject)diamond.getA());
   affectedElements.add((EObject)diamond.getB());
   affectedElements.add((EObject)diamond.getC());
   affectedElements.add((EObject)diamond.getD());
   ValidationUtil.markProblem(f, affectedElements,
       ValidationProblemKind.DIAMOND);
   processed.add(diamond);
}
return processed;
}
```

Process pattern signature DTOs

Make sure to remove a processed change from the queue

Use light-weight validation facility to create error marker

DEMO How does it work?

PERFORMANCE

Challenges

- Performance evaluations are hard
  - Fair?
  - Reliable?
  - Reproducible?
  - Can the results be generalized?
- Benchmark example:
  on-the-fly constraint validation over AUTOSAR models
  - Conference presentation tomorrow, 11-12.30, Hall B
  - Motivation: the Embedded Architect Tool of OptXware Ltd.
  - AUTOSAR models can be very large (>>1m elements)

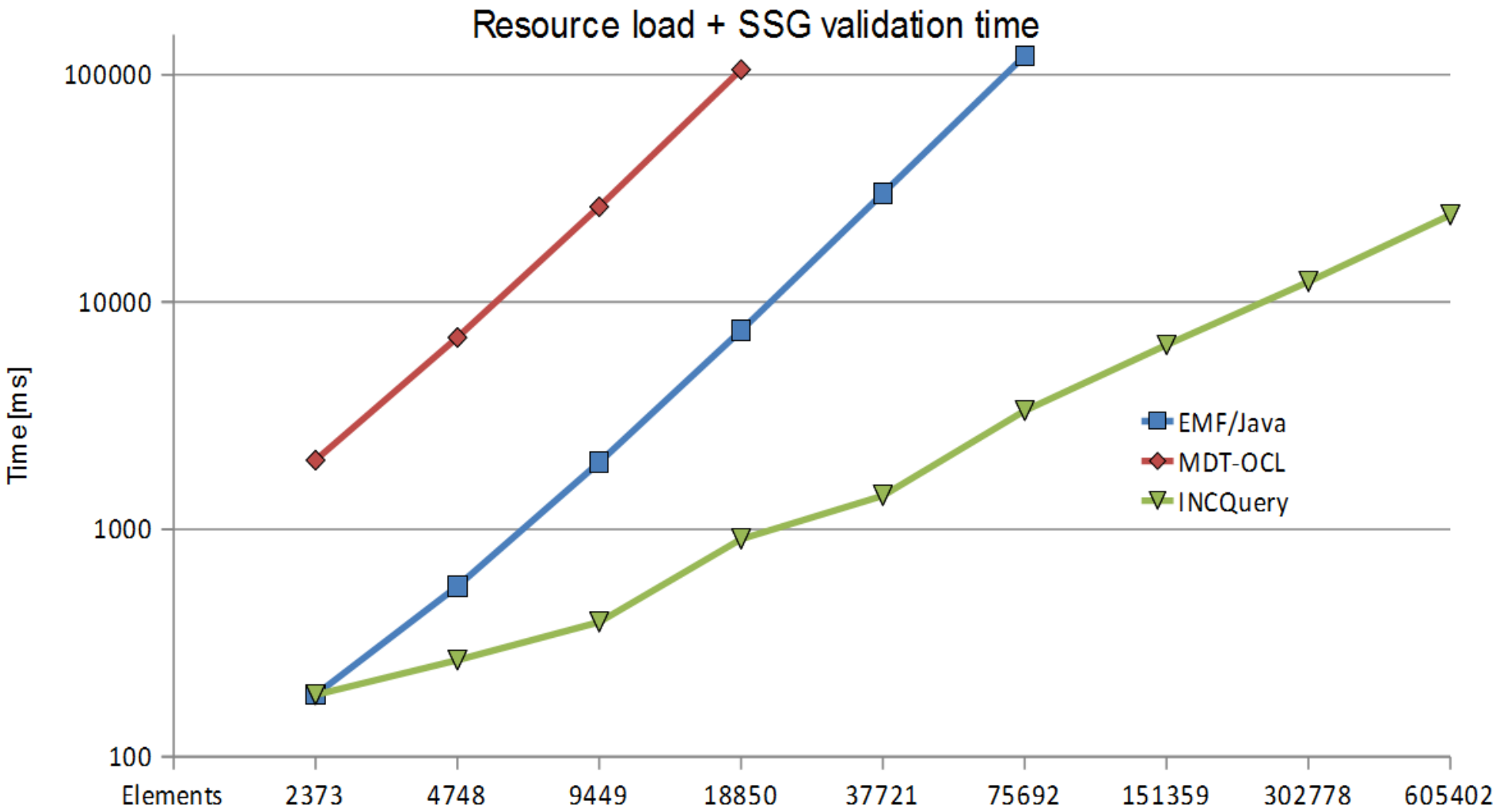
What is measured?

- Sample models were generated
  - matches are scarce relative to overall model size
- On-the-fly validation is modeled as follows:
  1. Compute initial validation results
  2. Apply randomly distributed, small changes
  3. Re-compute validation results
- Measured: execution times of
  - Initialization (model load + RETE construction)
  - Model manipulation operations (negligible)
  - Validation result (re)computation
- Compared technologies
  - MDT-OCL
  - Plain Java code that an average developer would write

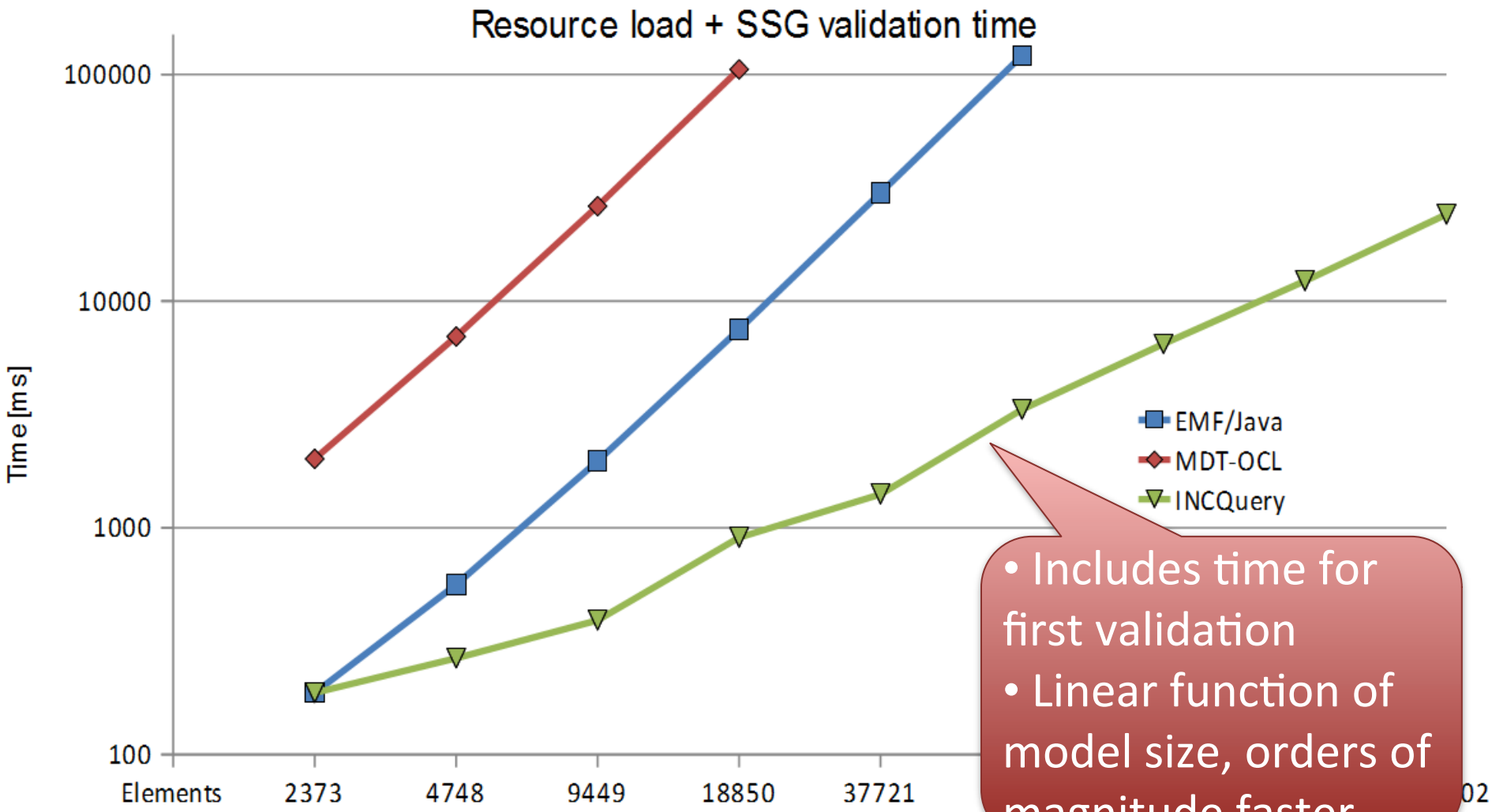
INCQuery Results

- Hardware: normal desktop PC (Core2, 4GB RAM)
- Model sizes up to 1.5m elements
- Initialization times (resource loading + first validation)
  - <1 sec for model sizes below 50000 elements
  - Up to 40 seconds for the largest model (grows linearly with the model size)
- Recomputation times
  - Within error of measurement (=0), **independent of model size**
  - **Retrieval of matches AND complex changes is instantaneous**
- Memory overhead
  - <50 MB for model sizes below 50000 elements
  - Up to 1GB for the largest model (grows linearly with model size)
- How does it compare to native code / OCL?
  - See the conference talk tomorrow ☺

Initialization time

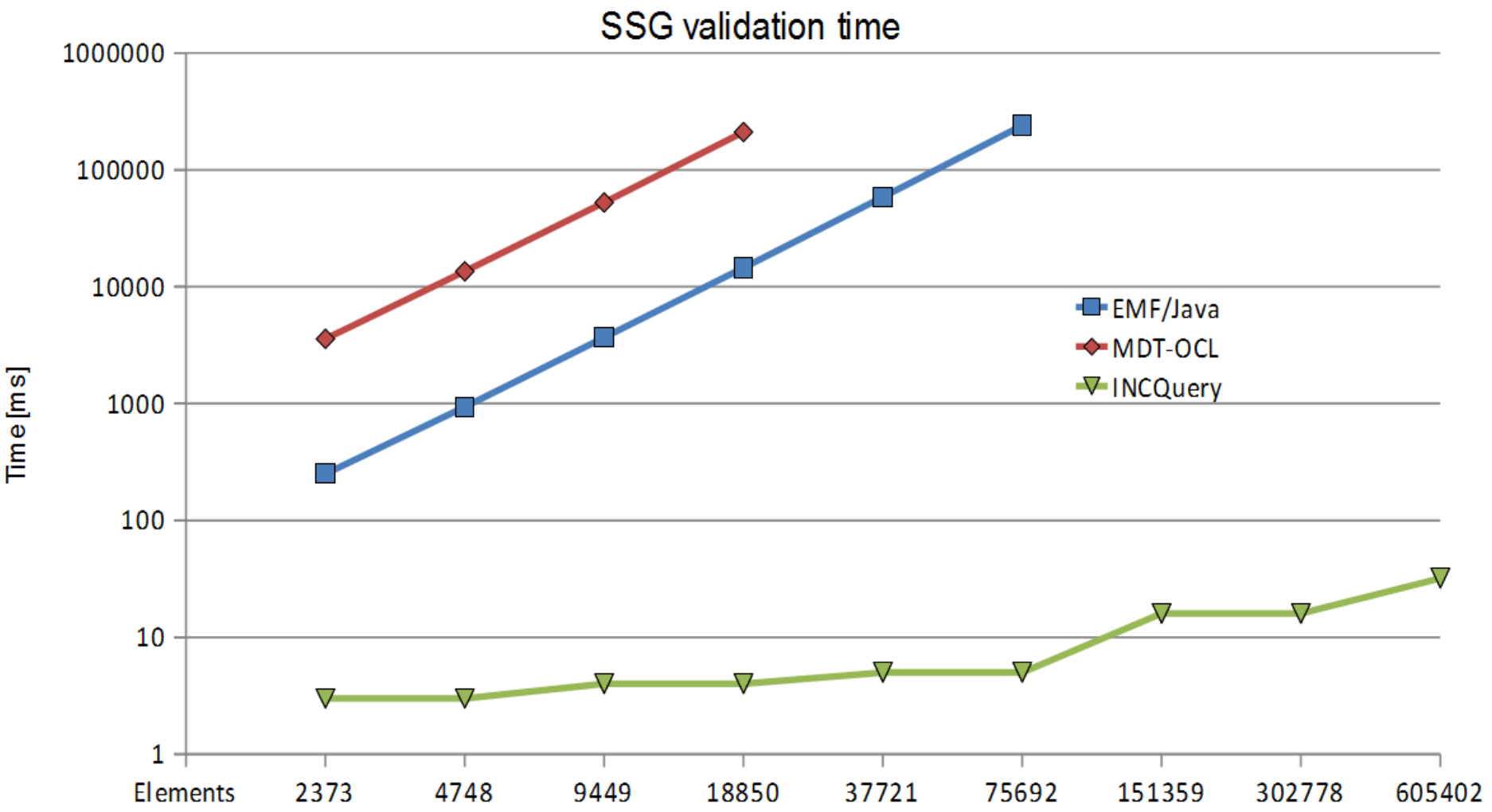


Initialization time

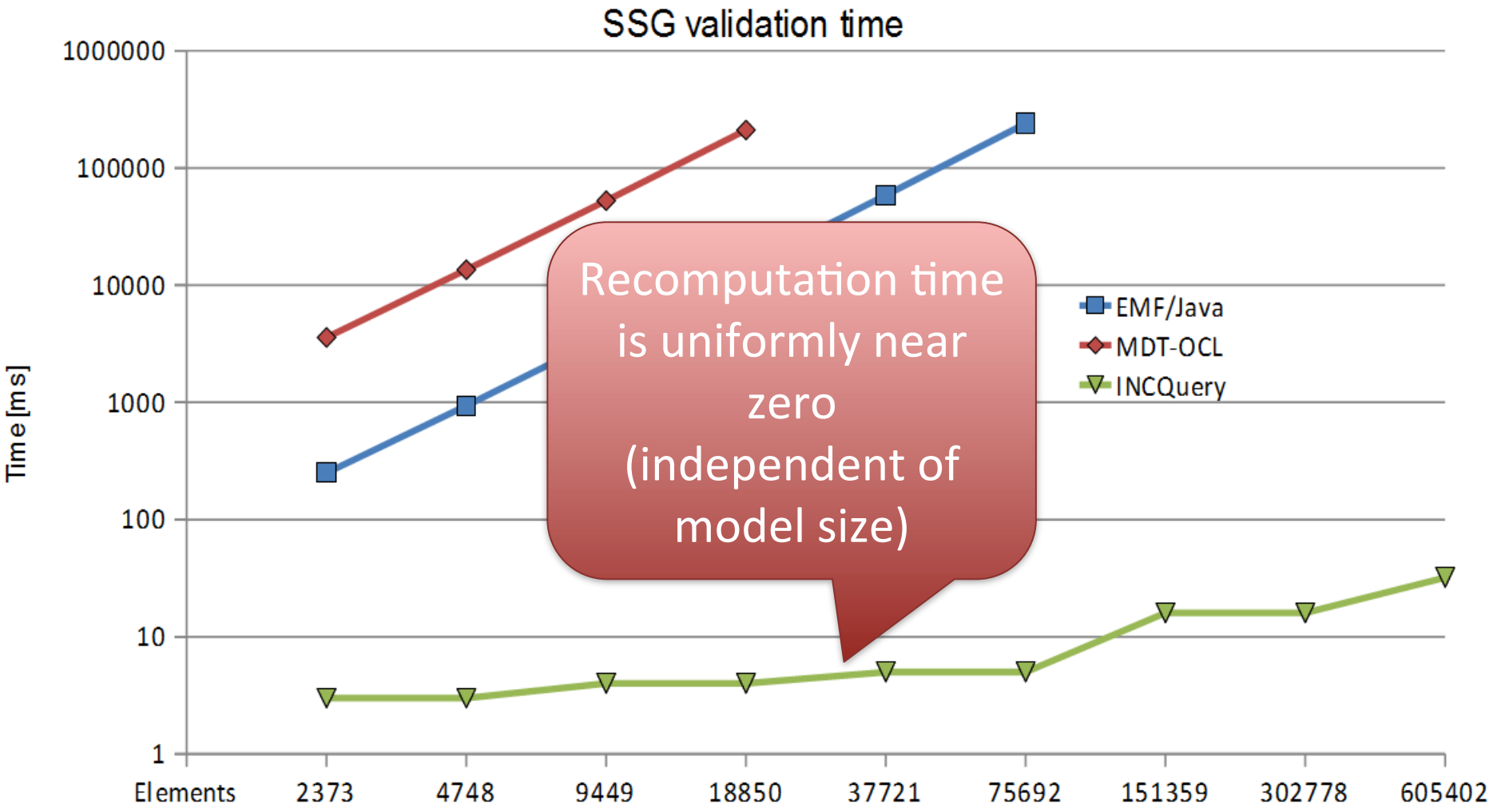


- Includes time for first validation
- Linear function of model size, orders of magnitude faster

Recomputation time



Recomputation time



Performance overview

SSG and iSignal validation pattern in model family A

		EMF/Java			MDT-OCL			INCQuery			
Model Elements #	Model size [MB]	Res [s]	iSignal [s]	SSG [s]	Res [s]	iSignal [s]	SSG [s]	Res [s]	iSignal [s]	SSG [s]	Mem OH [MB]
2 373	30	0.06	0.00	0.25	0.13	0.16	3.58	0.17	0.00	0.00	3
4 748	31	0.08	0.00	0.94	0.16	0.17	13.53	0.22	0.00	0.00	6
9 449	32	0.13	0.01	3.67	0.20	0.19	52.48	0.30	0.00	0.00	12
18 850	33	0.22	0.01	14.52	0.30	0.22	210.48	0.45	0.01	0.00	22
37 721	37	0.42	0.01	58.56	0.47	0.27		0.75	0.01	0.01	45
75 692	43	0.78	0.02	239.53	0.86	0.33		1.58	0.01	0.01	92
151 359	55	1.81	0.03		1.84	0.53		3.22	0.02	0.02	187
302 778	81	3.63	0.06		3.64	0.88		6.19	0.02	0.02	373
605 402	135	7.14	0.09		7.48	1.63		12.00	0.02	0.03	746

Channel validation pattern in model family B

		EMF/Java		MDT-OCL		INCQuery		
Model Elements #	Model size [MB]	Res [s]	Channel [s]	Res [s]	Channel [s]	Res [s]	Channel [s]	Mem OH [MB]
2 972	30	0.06	0.00	0.14	0.17	0.19	0.00	2
6 237	31	0.09	0.02	0.16	0.22	0.27	0.00	4
12 708	32	0.16	0.00	0.25	0.31	0.38	0.00	8
24 885	34	0.28	0.03	0.34	0.33	0.89	0.00	14
47 228	38	0.49	0.06	0.53	0.48	1.28	0.00	28
90 586	44	1.13	0.09	1.20	0.80	2.41	0.00	55
180 389	58	1.94	0.19	2.05	1.41	4.56	0.00	111
370 660	91	4.06	0.39	4.08	2.50	9.00	0.00	225
752 172	156	8.09	0.80	8.11	5.00	20.38	0.00	456
1 558 100	295	17.28	1.59	17.39	10.13	40.22	0.00	943

Legend: Res – resource loading time
Mem OH – memory overhead

Assessment of the benchmark

- On-the-fly validation is only one scenario
  - Early model-based analysis
  - Language engineering in graphical DSMLs
  - Model execution/analysis (stochastic GT)
  - Tool integration
  - Model optimization / constraint solving
- Example AUTOSAR queries
  - Do not make use of advanced features such as parameterized queries or complex structural constraints (recursion)

CONCLUSION

Contributions

- **Expressive** declarative query language by graph patterns
  - Capture local + global queries
  - Compositionality + Reusabilility
  - „Arbitrary” Recursion, Negation
- **Incremental** cache of matches (materialized view)
  - Cheap maintenance of cache (only memory overhead)
  - Notify about relevant changes
  - Enable reactions to complex structural events
- **High performance** for large models
  - Linear overhead for loading
  - Instant response for queries
  - > 1 million model elements (on desktop PC)

Pointers

- Details
  - http://viatra.inf.mit.bme.hu/incquery