A TUTORIAL ON EMF-INCQUERY

Incremental evaluation of model queries over EMF models

Gábor Bergmann, Ákos Horváth,
Ábel Hegedüs, Zoltán Ujhelyi, Balázs Polgár,
István Ráth, Dániel Varró

Budapest University of Technology and Economics
Fault Tolerant Systems Research Group

Contents

- Motivation
  - Model queries…?
  - State of the art of EMF model queries
  - Goals
- Technology overview
- Hands-on part 1
  - Basics (UML model statistics)
- Break
- Hands-on part 2
  - Advanced well-formedness validation for UML
  - Integration with Papyrus UML
- Conclusion
  - Performance benchmarks
  - Q&A
- **Feel free to ask questions along the way!**

Downloads

- http://viatra.inf.mit.bme.hu/incquery/events/ecmfa2011#Files
- What you'll need
  - Eclipse Helios (Modeling)
    - Make sure to allocate enough heap by passing "-Xmx1024m" through eclipse.ini
  - Optionally: Papyrus UML http://www.eclipse.org/modeling/mdt/papyrus/updates/index.php
  - Everything from the update site http://viatra.inf.mit.bme.hu/update/ecmfa11/
  - In your workspace:
    - Sample IncQuery project http://viatra.inf.mit.bme.hu/downloads/demo/ecmfa11/ecmfa-incquery-project.zip
    - Sample Papyrus UML models http://viatra.inf.mit.bme.hu/downloads/demo/ecmfa11/ecmfa-umlsamples-project.zip

MOTIVATION

While you’re downloading ☺

First of all…

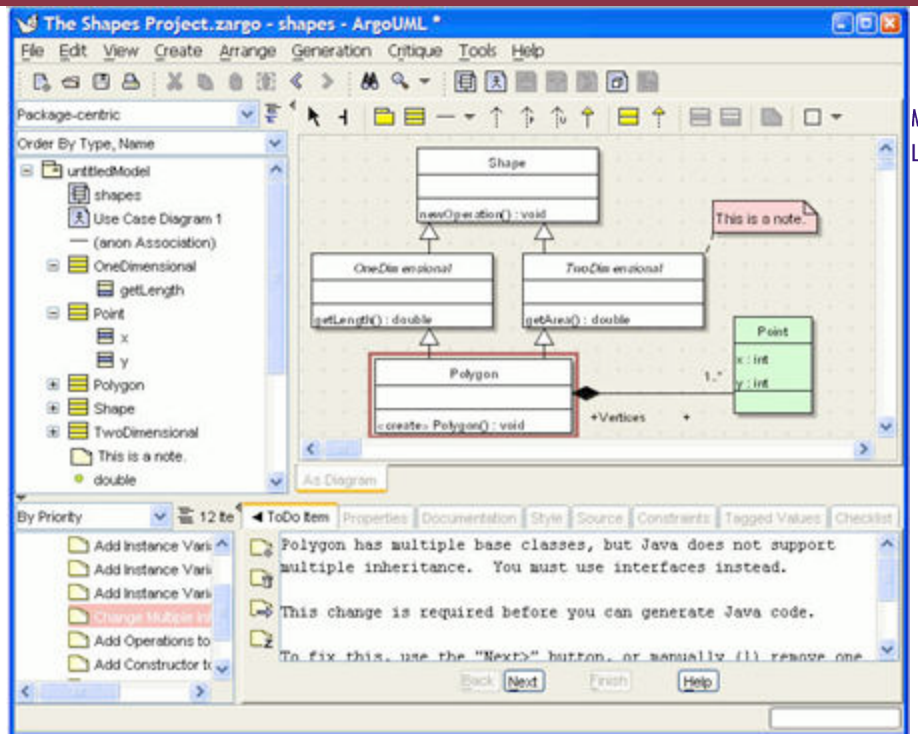
- What is a model query?
  - A piece of code that looks for certain parts of the model.
- "Mathematically"
  - Query = set of constraints that have to be satisfied by (parts of) the model.
  - Result = set of model elements (element configurations) that satisfy the constraints of the query.
- A query engine?
  - Supports the definition/execution of model queries.

Hi Jane, what do you do at work?

Jane

Hi Jane, what do you do at work?

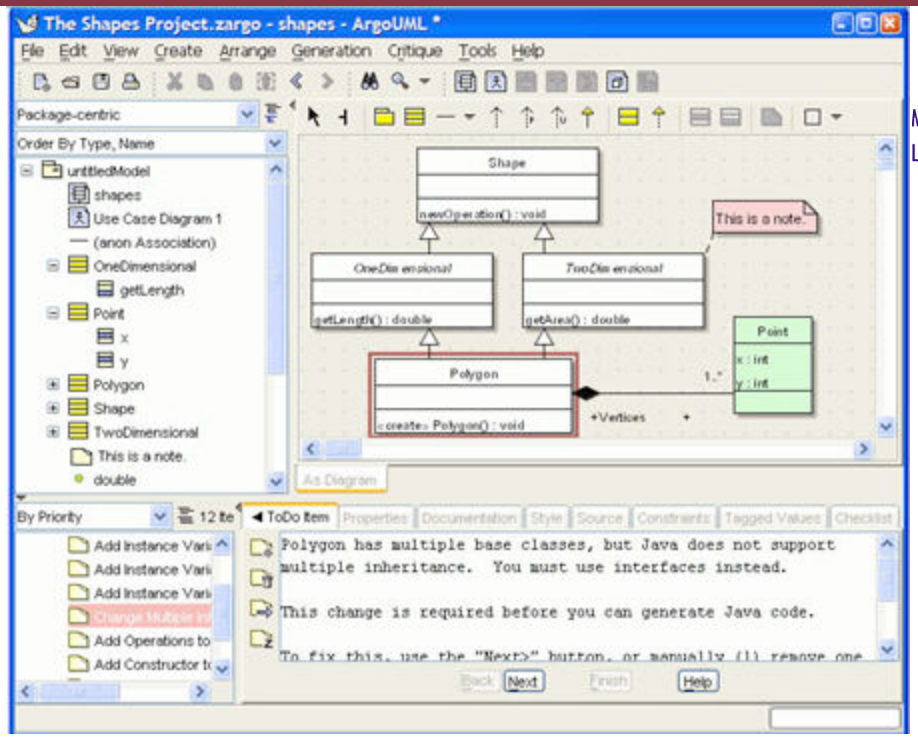
Jane



UNIFIED MODELING LANGUAGE

Hi Jane, what do you do at work?

Jane

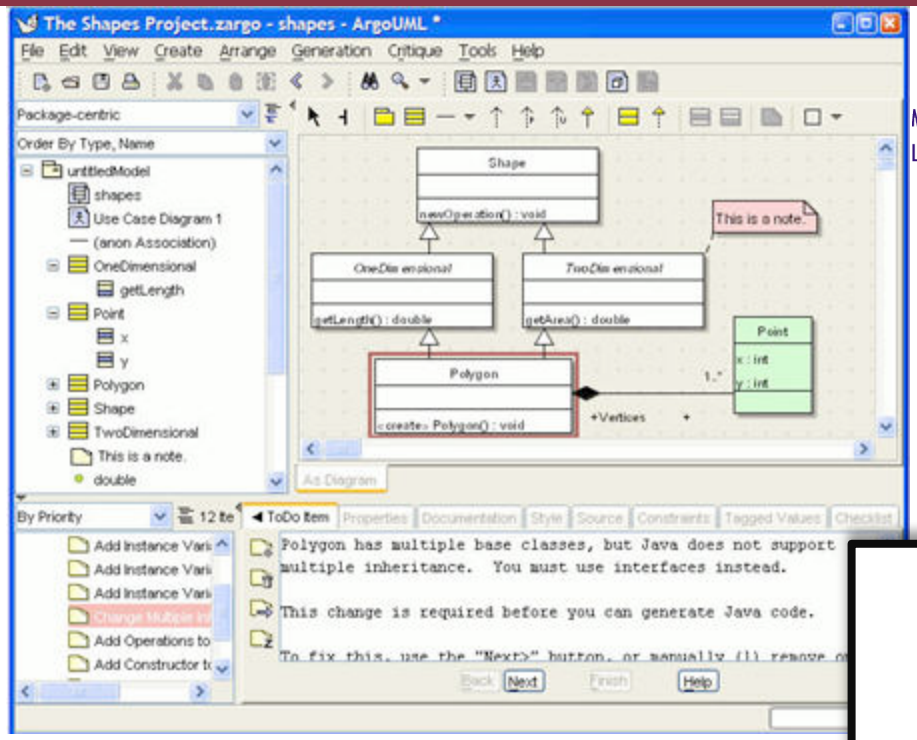


UNIFIED MODELING LANGUAGE

Boss

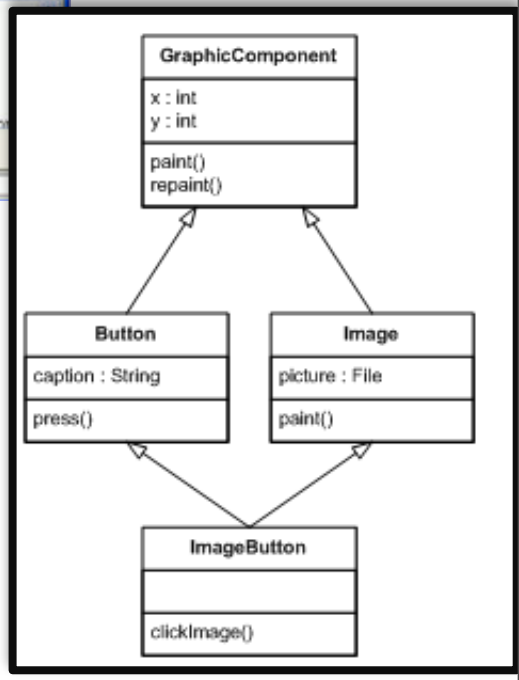
Hi Jane, what do you do at work?

Jane



UNIFIED MODELING LANGUAGE

Detect

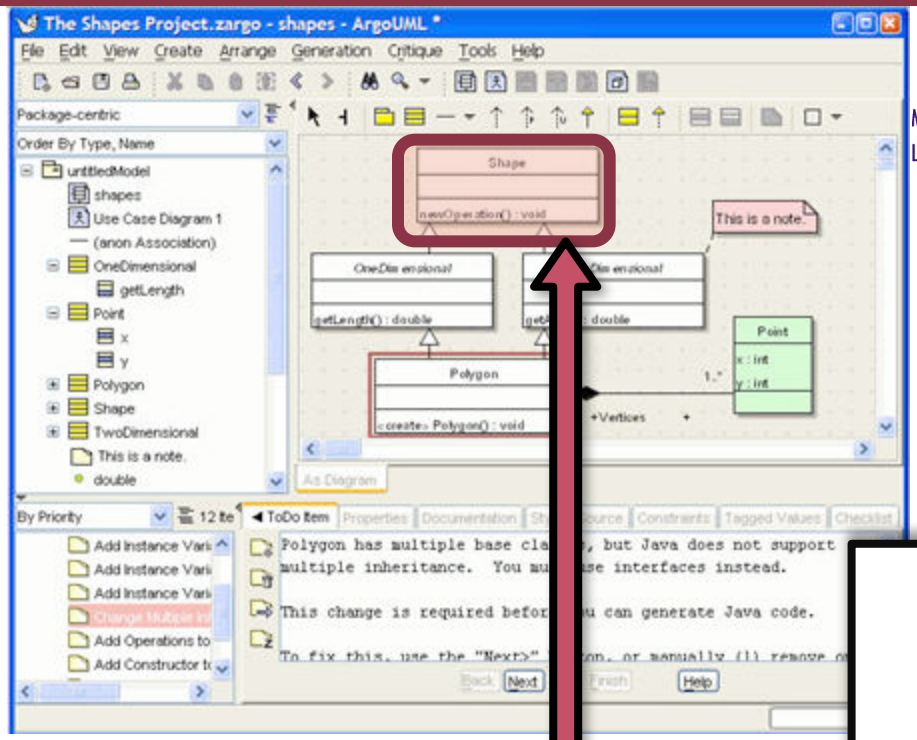


Boss

Constraint

Hi Jane, what do you do at work?

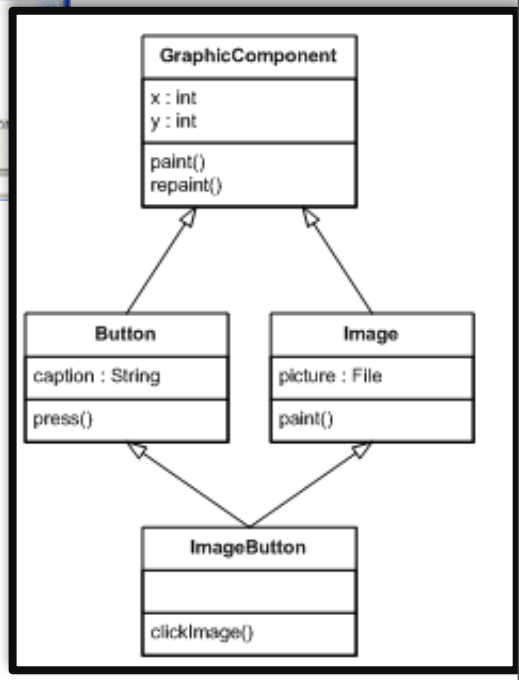
Jane



UNIFIED MODELING LANGUAGE

Detect

Report

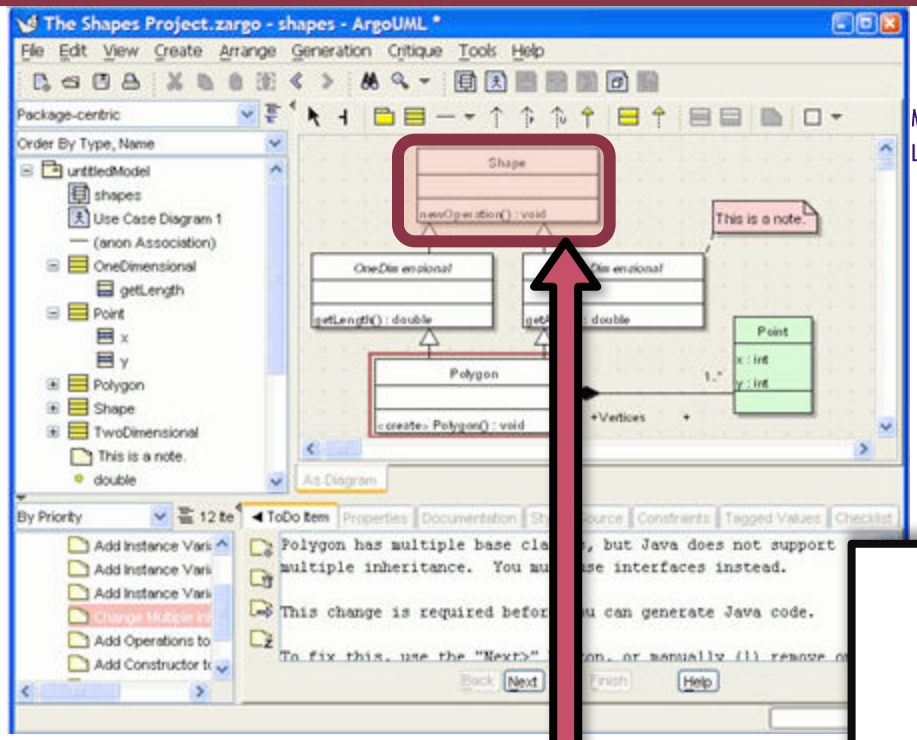


Boss

Constraint

Hi Jane, what do you do at work?

Jane



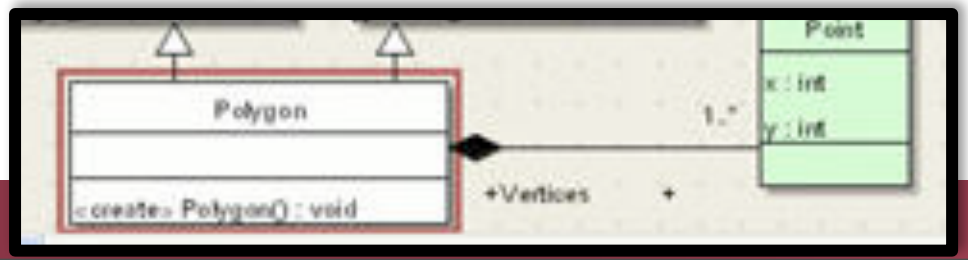
UNIFIED MODELING LANGUAGE

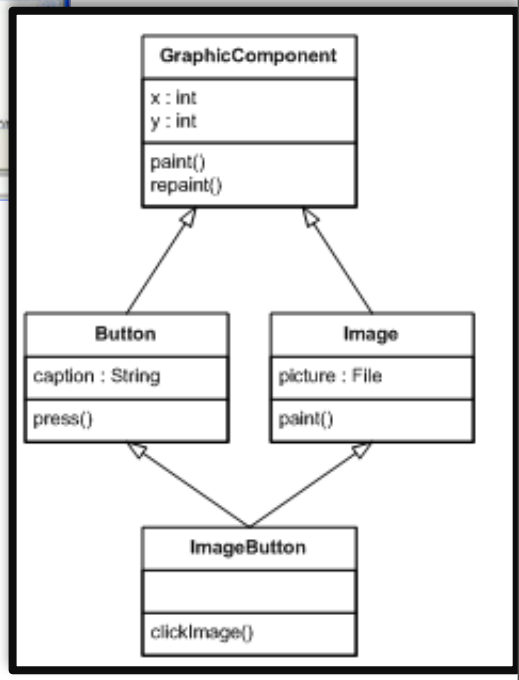
Detect

View

Report

Boss

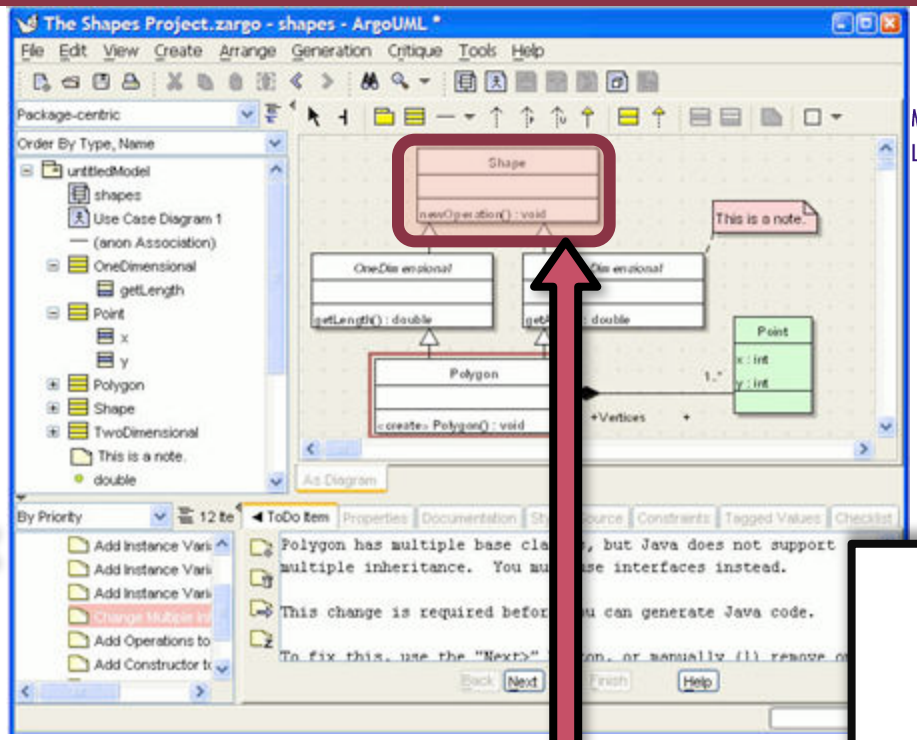




Constraint

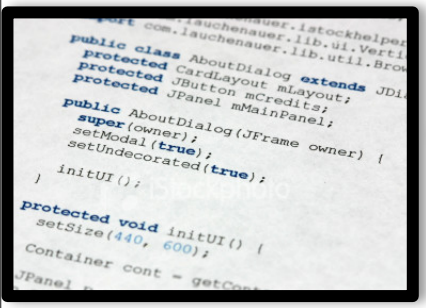
Hi Jane, what do you do at work?

Jane



UNIFIED MODELING LANGUAGE

Detect

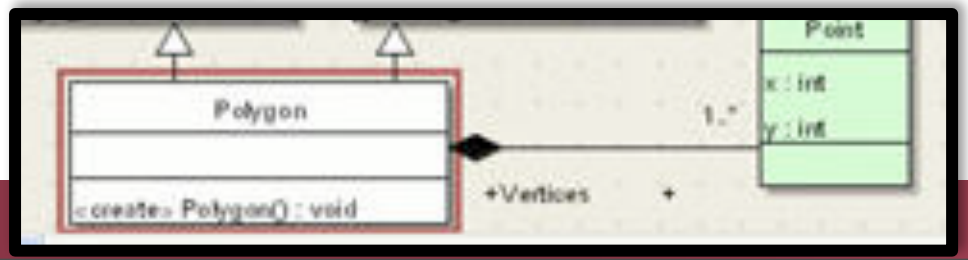


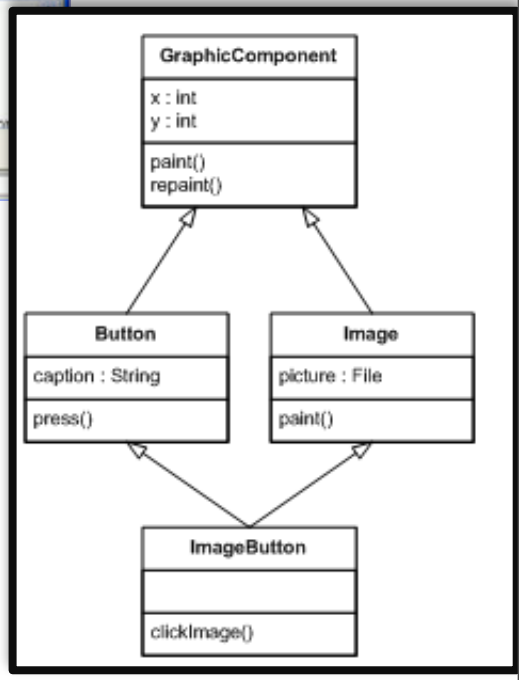
Gen

View

Report

Boss





Constraint

Model queries

- Queries are at the heart of MDD.
  - Views
  - Reports
  - Generators
  - Validators
  - …
- Development issues
  - Complex queries are hard to write

Issues with query development

- Hard to write?
- Your options
  - Java (or C/C++, C#, …)
  - Declarative languages (OCL, EMF Query 1-2, …)

	Imperative query languages	Declarative query languages
Expressive power	☹ (you write lots of code)	☺ (very concise)
Safety	☺☺ (precise control over what happens at execution)	☺☹ (unintended side-effects)
Learning curve	☺ (you already know it)	☹ (may be difficult to learn)
Reusability	☺ (standard OO practices)	☹☹ (???)
Performance	☹☺ (considerable manual optimization necessary)	☺☹ (depends on various factors)

Issues with query execution

- Query performance
  - = Execution time, as a function of
    - Query complexity
    - Model size / contents
    - Result set size
- Incrementality
  - Don't forget previously computed results!
  - Models changes are usually small, yet up-to-date query results are needed all the time.
  - Incremental evaluation is an essential, but not a very well supported feature.

Model query engine wish list

- Declarative query language
  - Easy to learn
  - Good bindings to the imperative world (Java)
  - Safe yet powerful
  - Reusable
- High performance
  - Fast execution for complex queries over large models
  - First-class support for incremental execution
- Technology
  - Works with EMF out-of-the-box

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 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

EMF-IncQuery

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

INCQUERY TECHNOLOGY OVERVIEW

Technology 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

Contributions

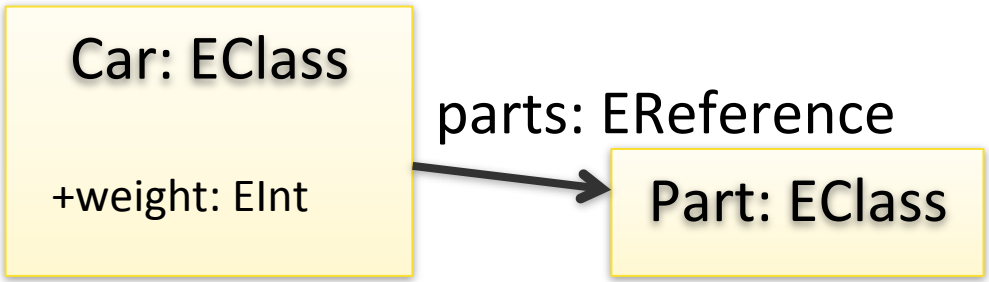
- **Expressive** declarative query language by graph patterns
  - Capture local + global queries
  - Compositionality + Reusabilility
  - „Arbitrary" Recursion, Negation
- **Incremental** cache of matches (materialized view)
- **High performance** for large models

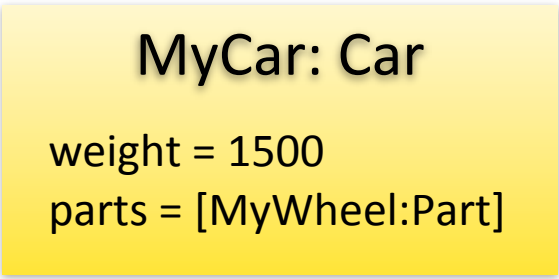
Origins of the idea

- 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)
- Developed by BUTE and OptXware Ltd.
- More info
  - http://eclipse.org/gmt/VIATRA2
  - http://wiki.eclipse.org/VIATRA2

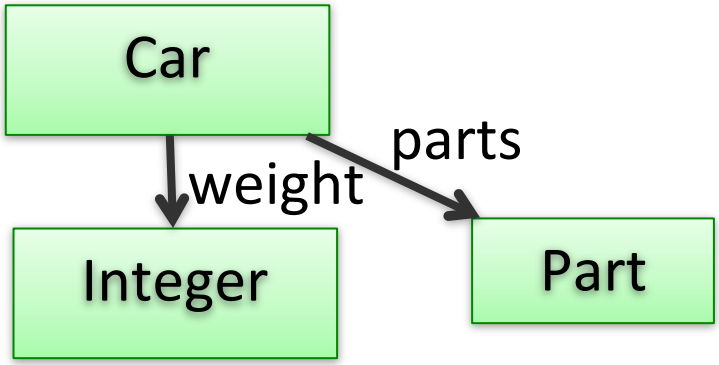
ECore models as graphs

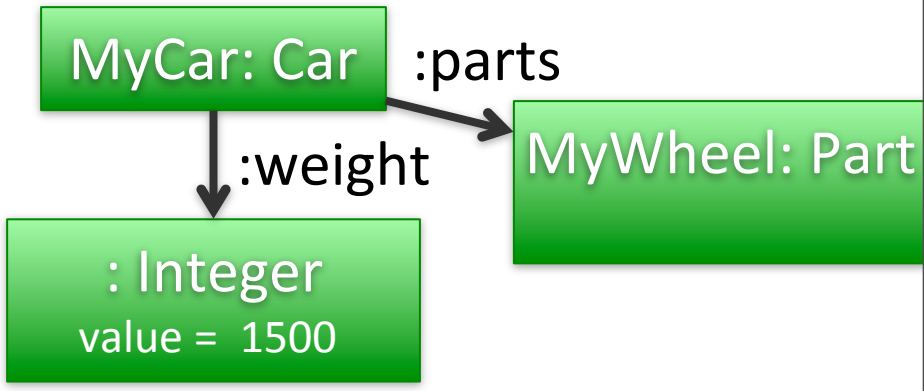
- Ecore
  - EClass
  - EReference
  - EAttribute



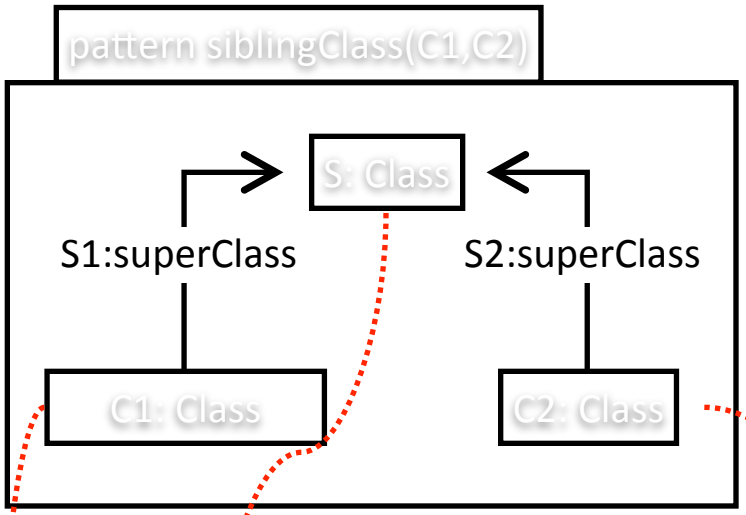


- VIATRA2
  - Entity (Node)
  - Relation (Edge)
  - Node+Edge

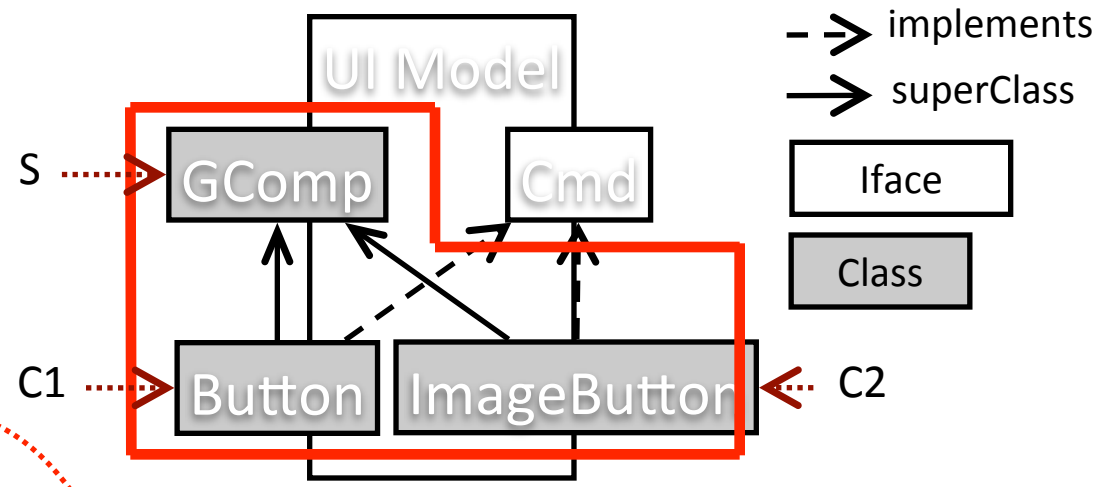


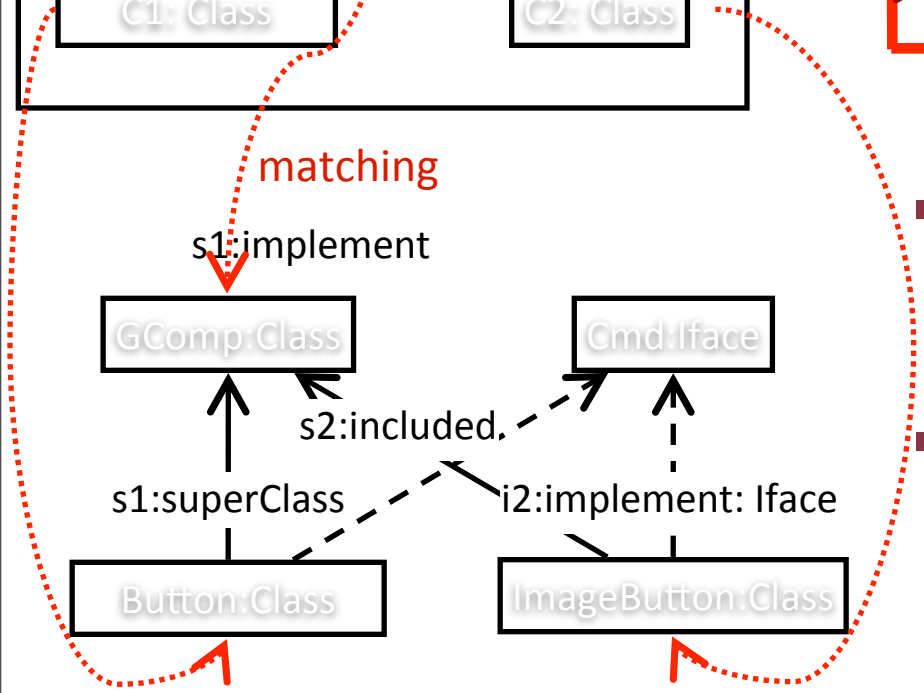


Pattern definition



Graphical notation

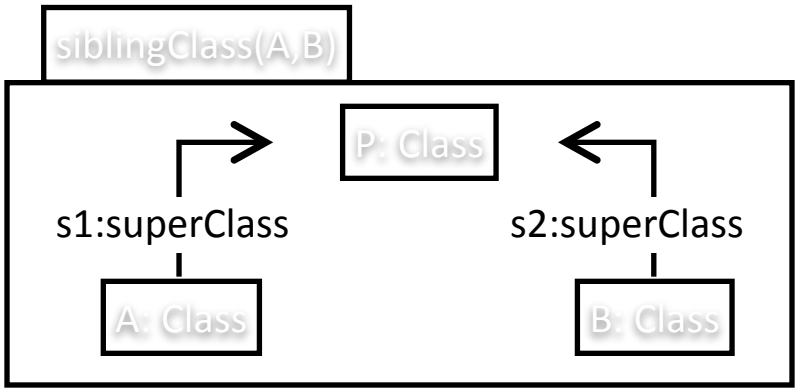




- Graph Pattern:
  - Structural condition that have to be fulfilled by a part of the model space
- Graph pattern matching:
  - A model (i.e. part of the model space) c satisfy a graph pattern,
  - if the pattern can be matched to a subgraph of the model

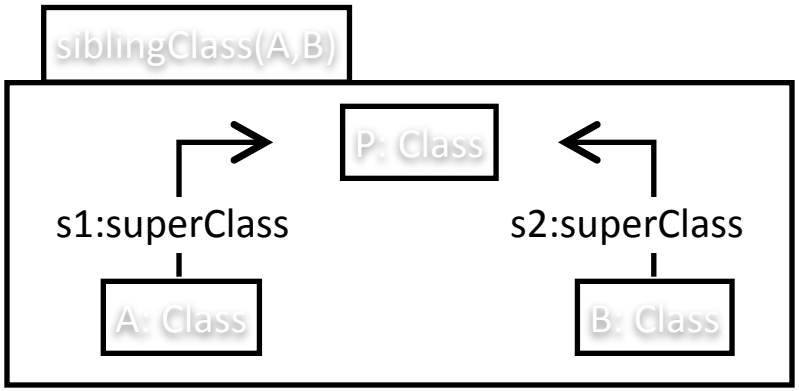
Instance Model

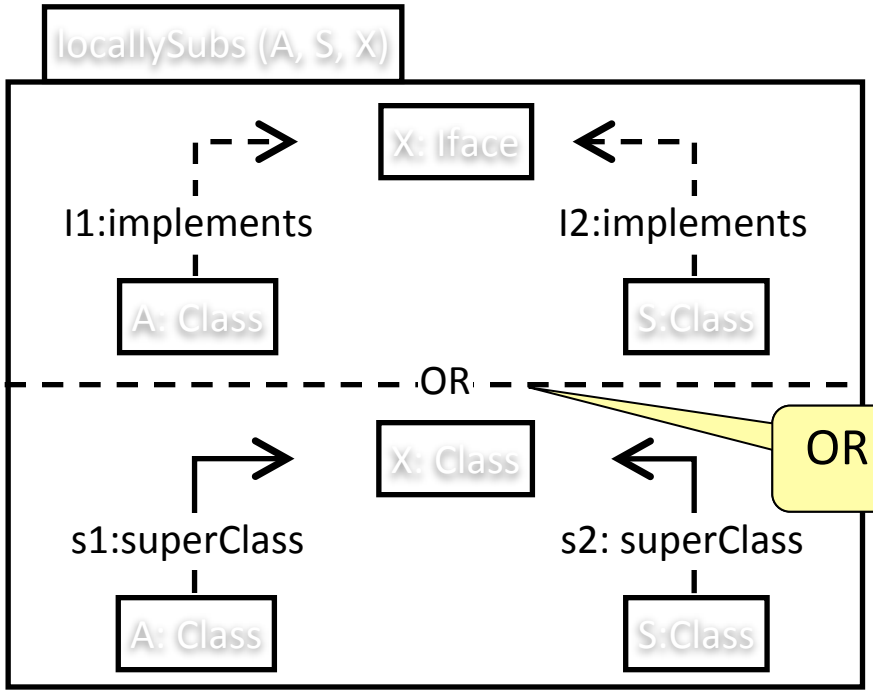
Graph patterns (VTCL)




```
// 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);
```

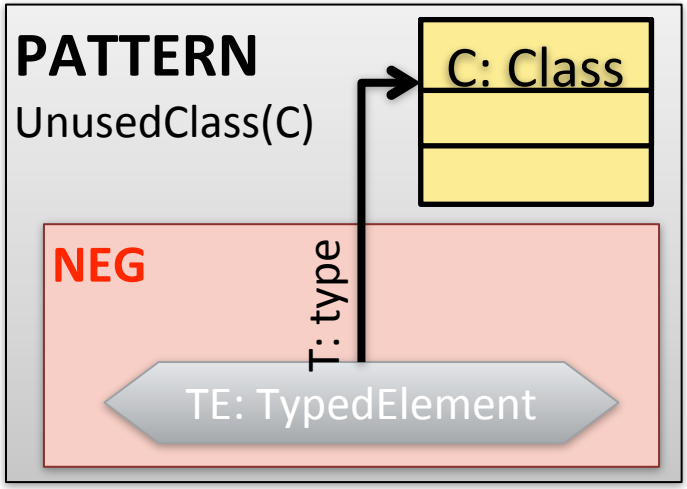
```
// S is locally substitutable by A
pattern localSubs(A, S, X) = {
    Class(A);
    Iface(X);
    Class.implements(I1, A, X);
    Class(S);
    Class.implements(I2, S, X);
  } or {
    Class(A);
    Class(X);
    Class.superClass(P1, X, X);
    Class(S);
```

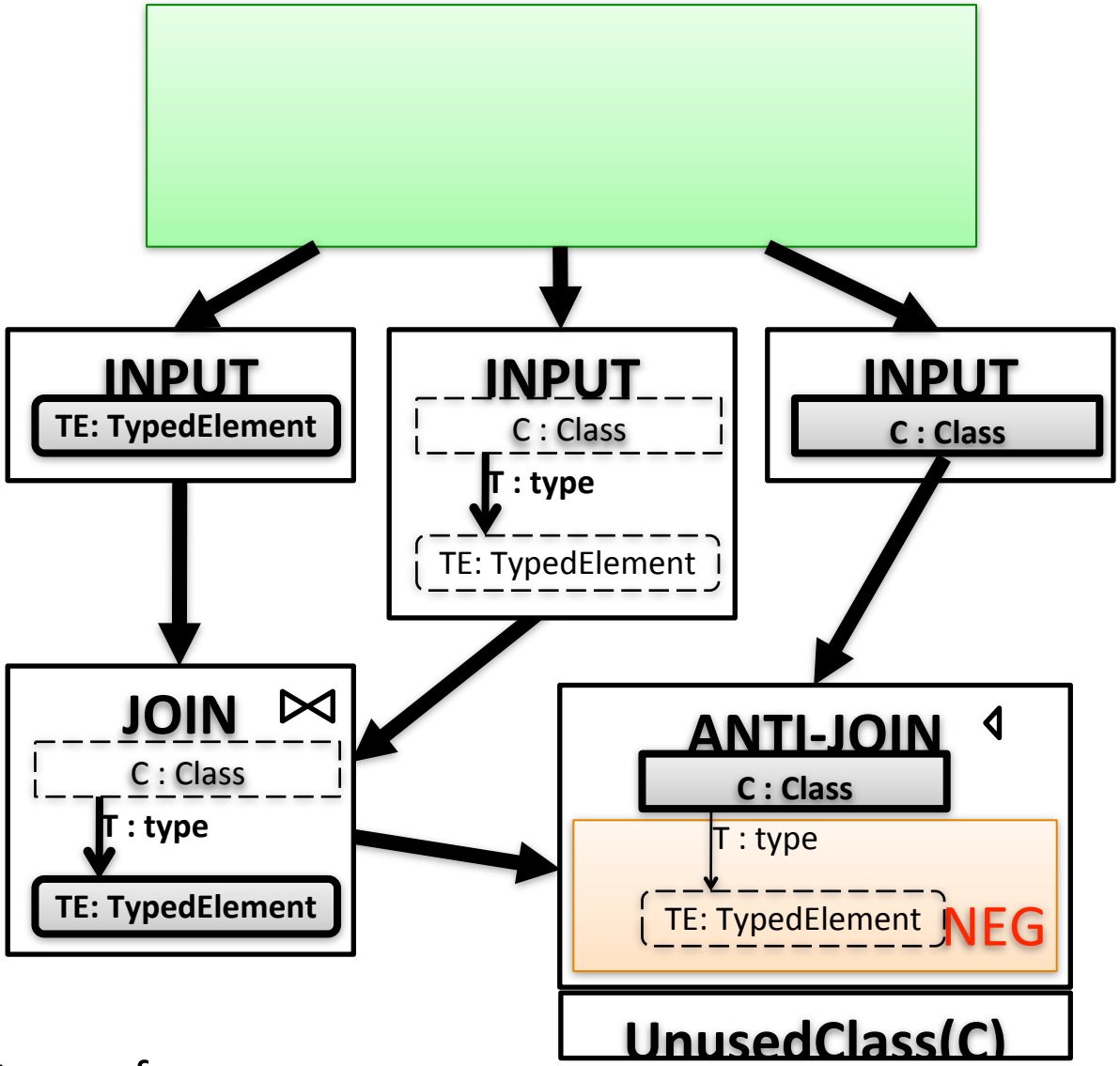
Contributions

- **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

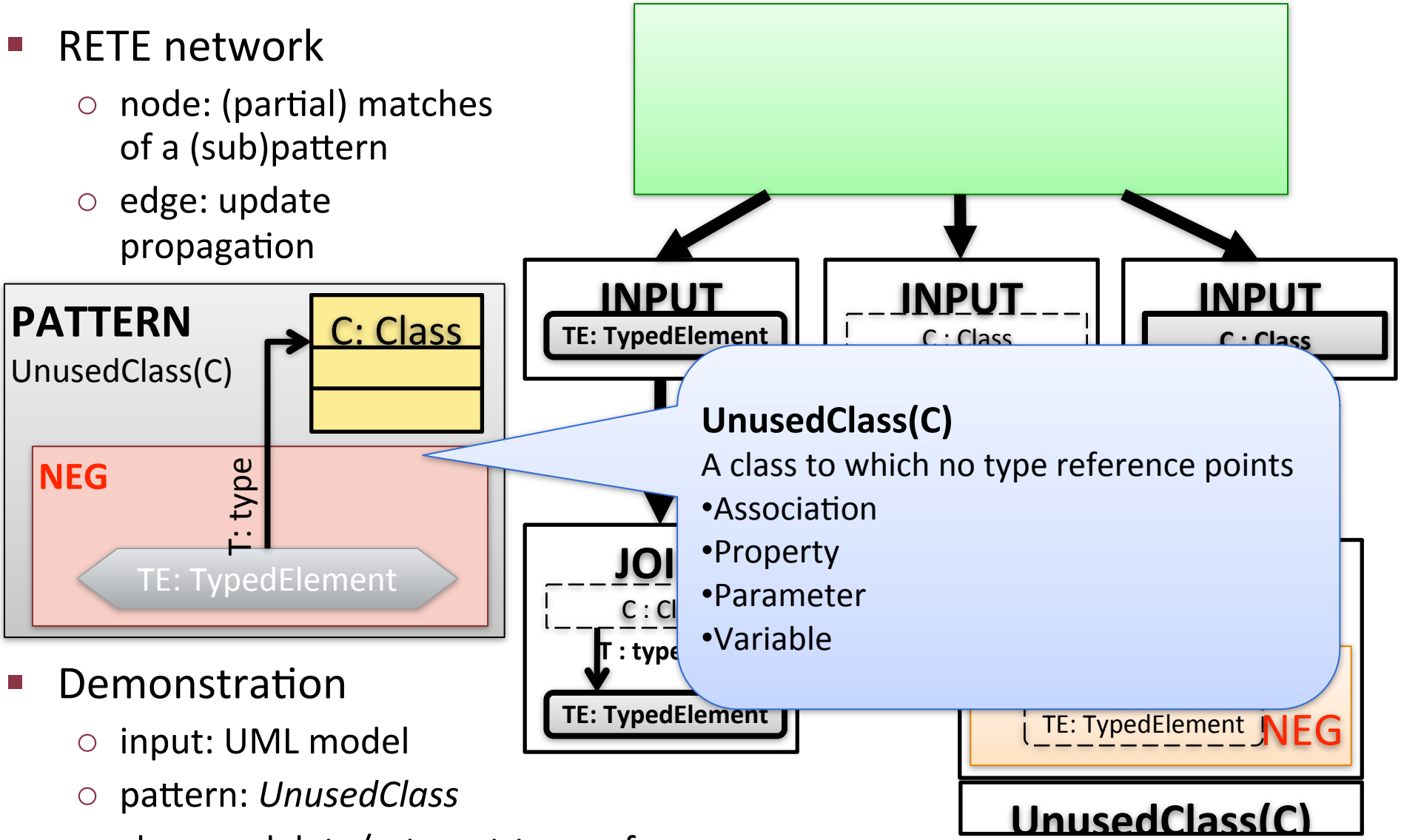




- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

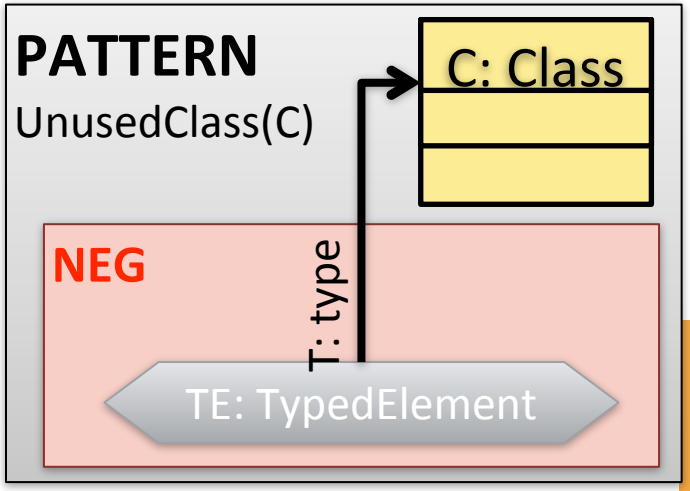
- RETE network
  - node: (partial) matches of a (sub)pattern
  - edge: update propagation

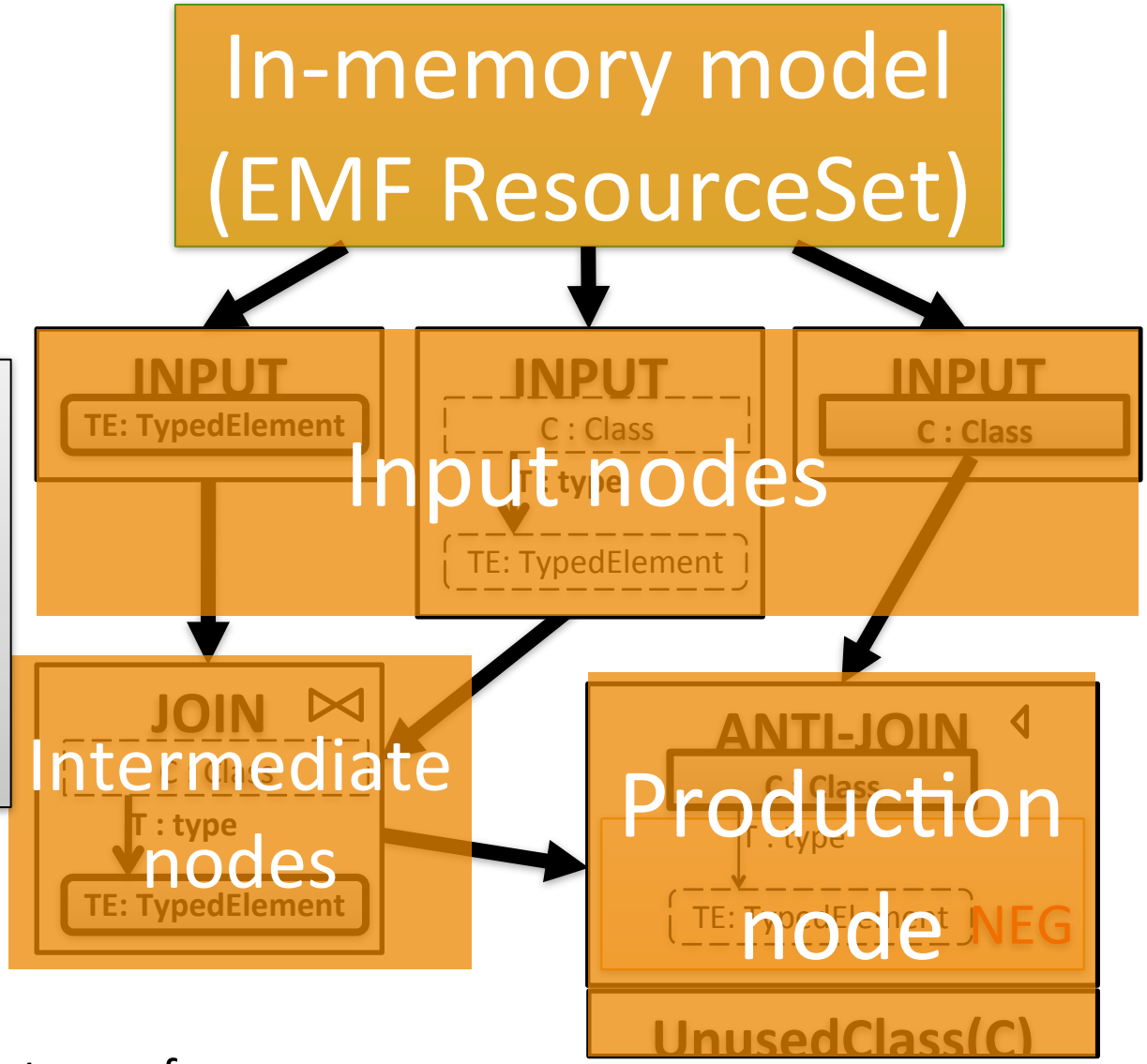


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

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

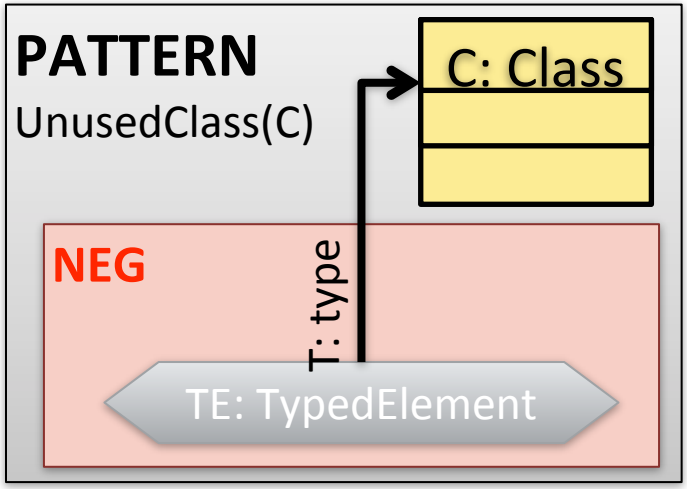


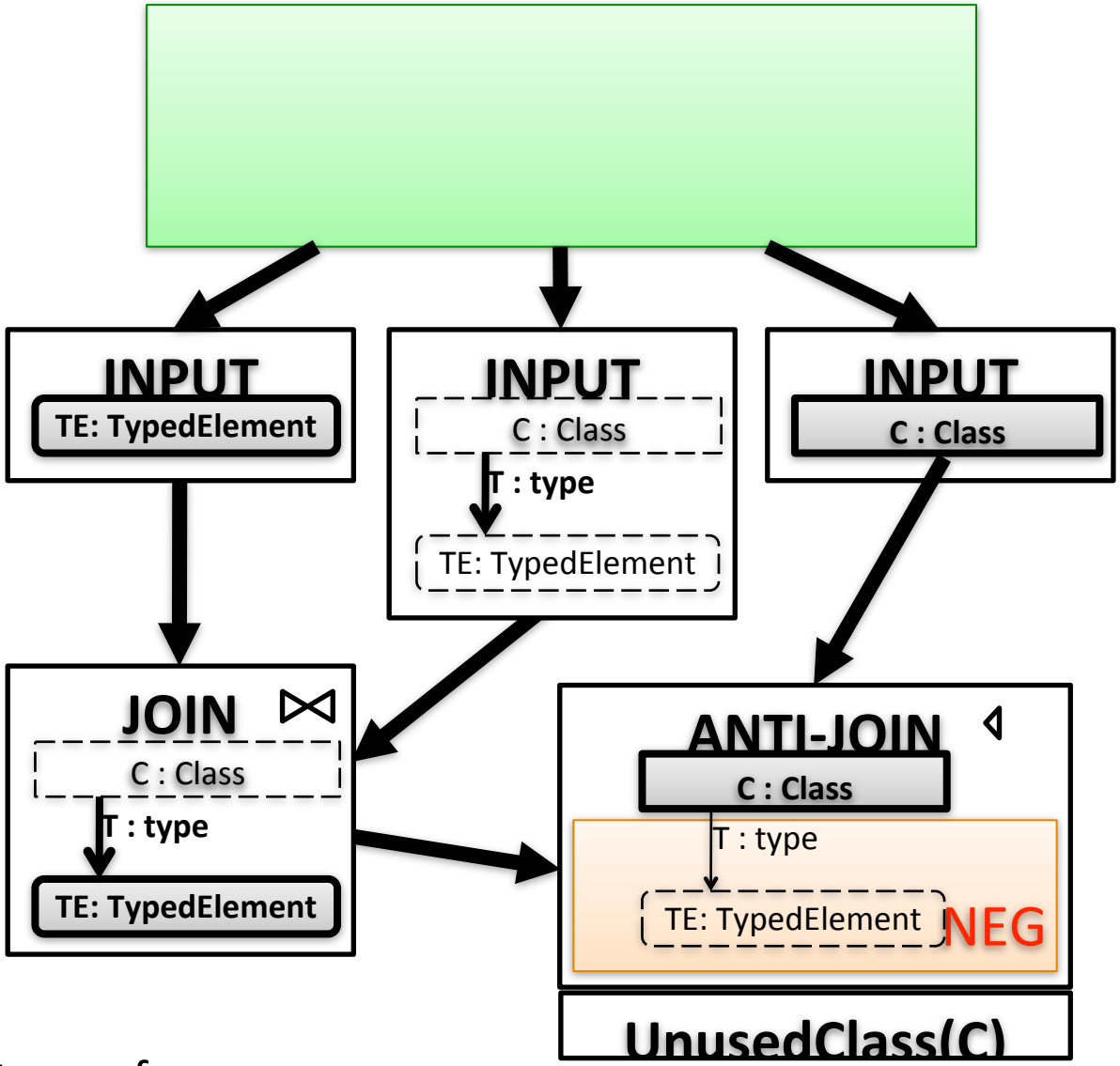


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

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

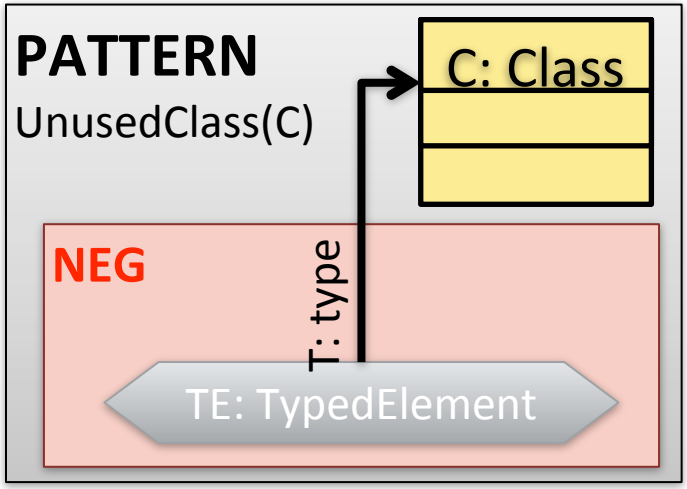


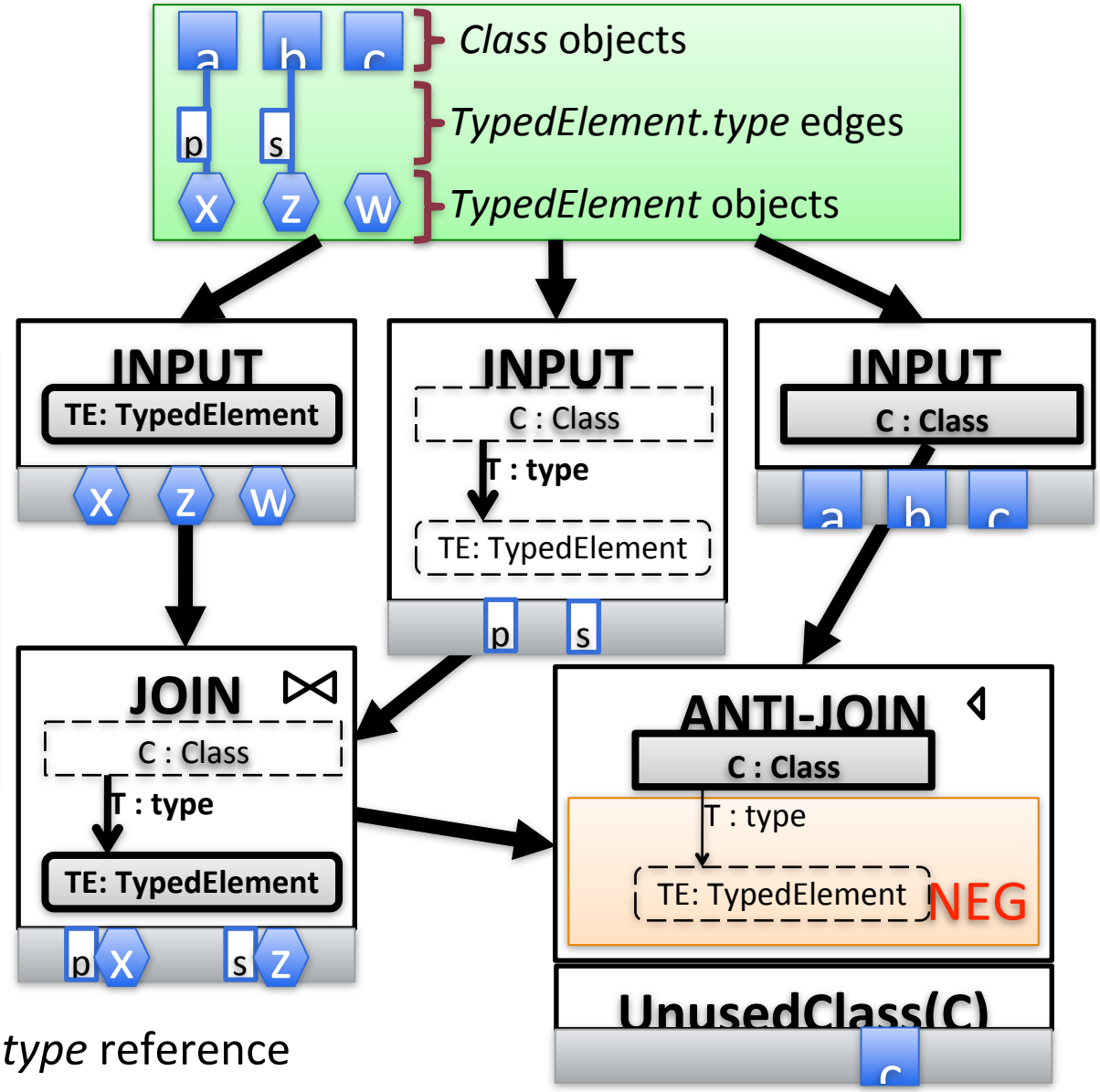


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

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

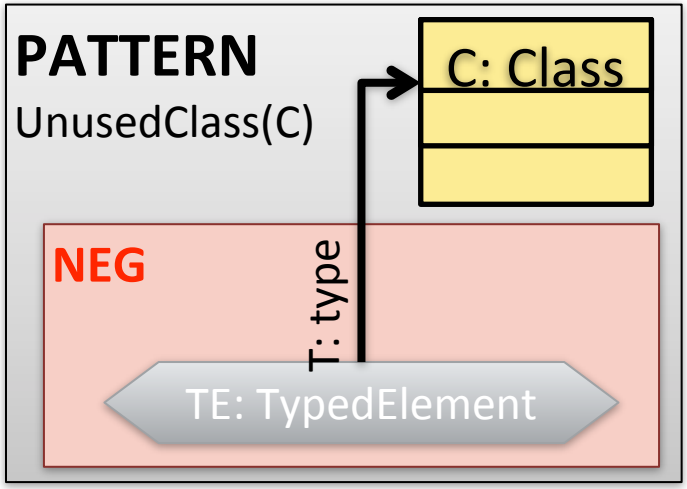


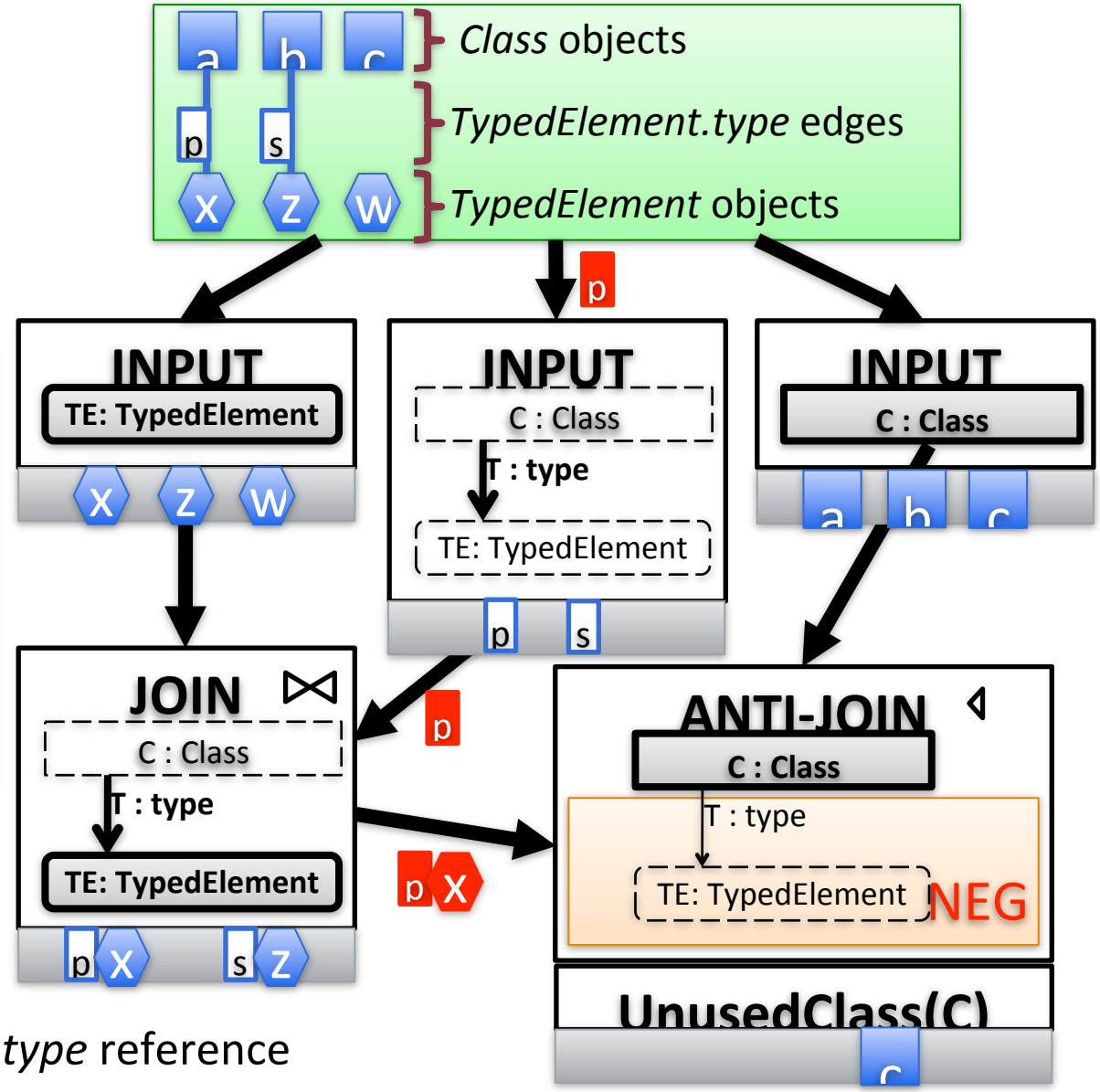


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

RETE nets

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

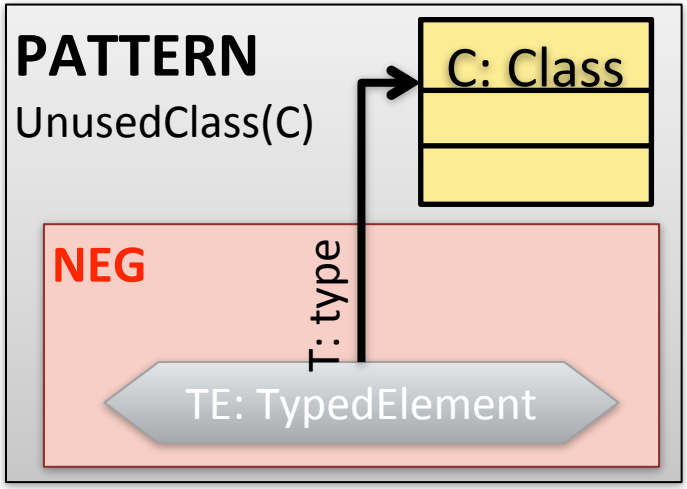


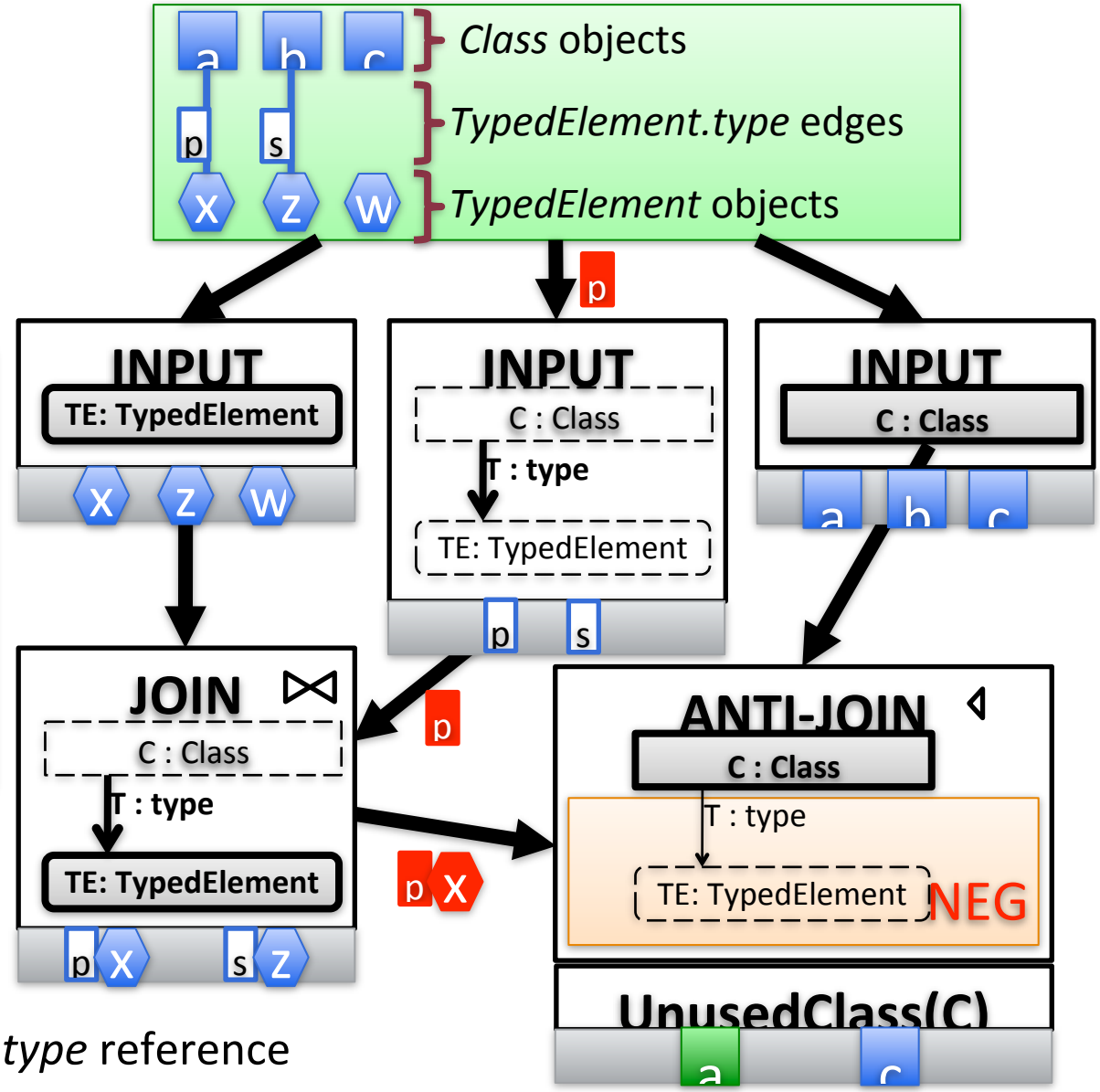


- Demonstration
  - input: UML model
  - pattern: *UnusedClass*
  - change: delete/retarget *type* reference

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

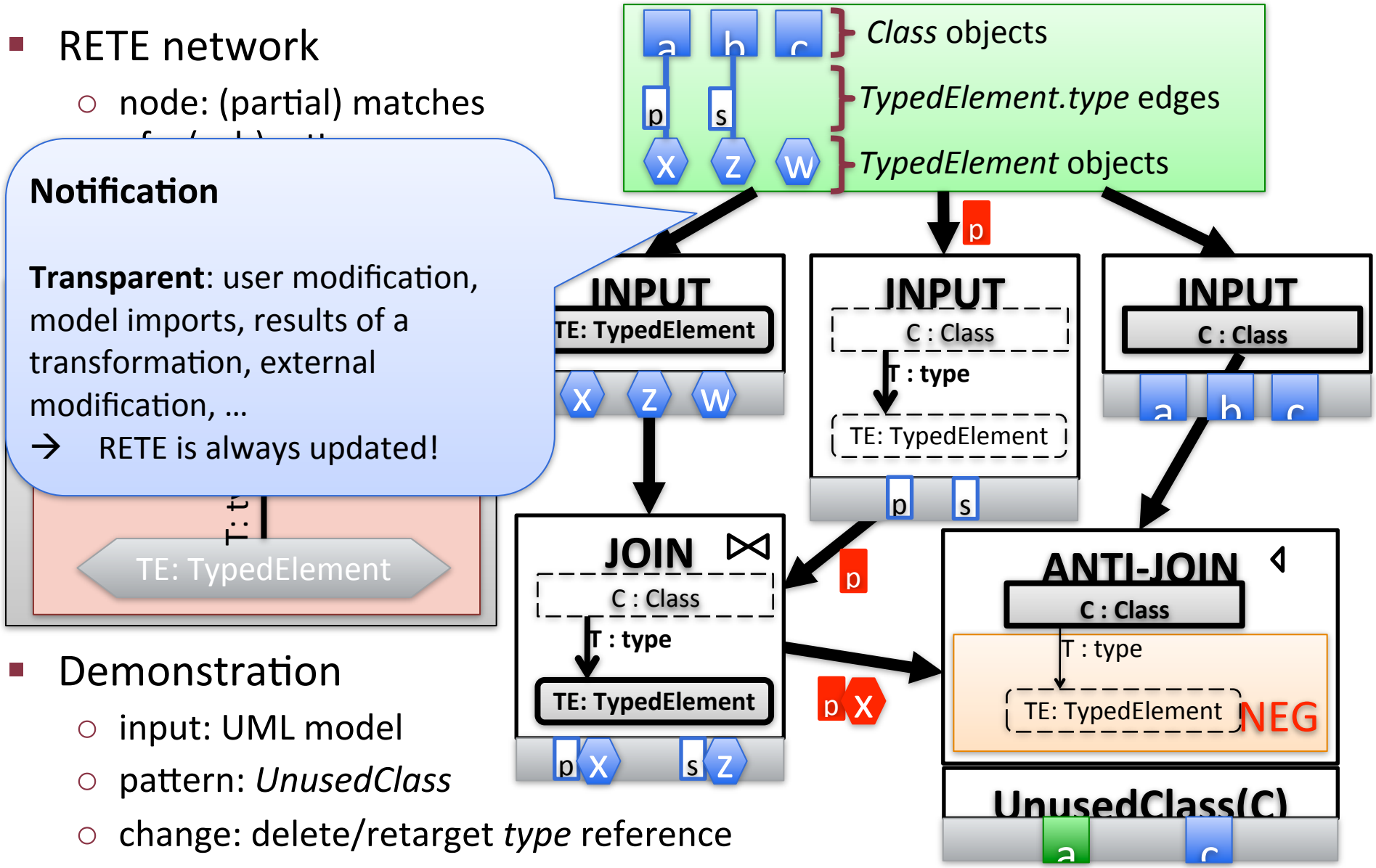
RETE nets

- RETE network
  - node: (partial) matches

Notification

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

→ RETE is always updated!



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

RETE nets

- RETE network
  - node: (partial) matches

Notification

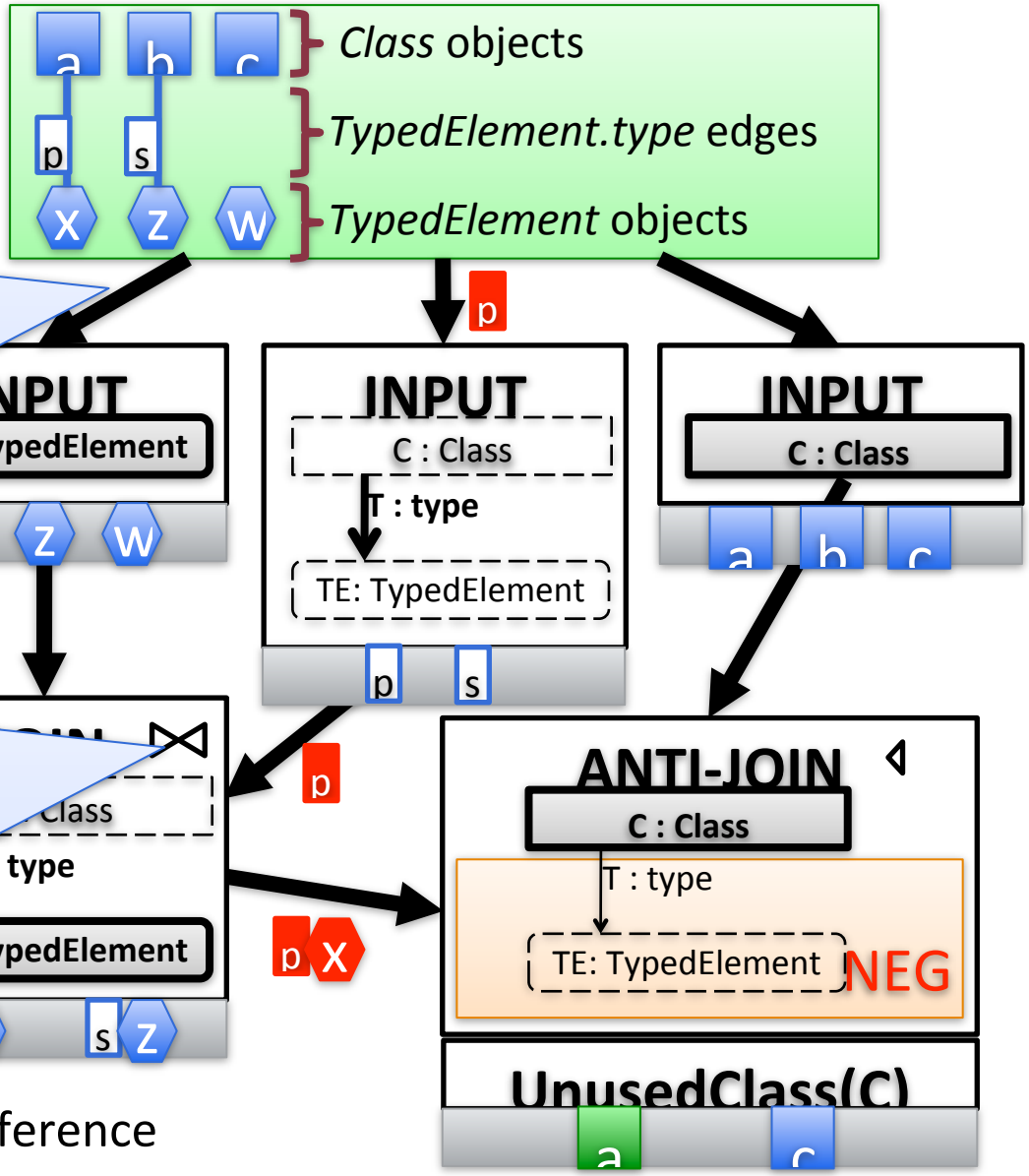
Transparent: user modification, model imports, results of a transformation, external modification, …

→ RETE is always updated!

Experimental results:

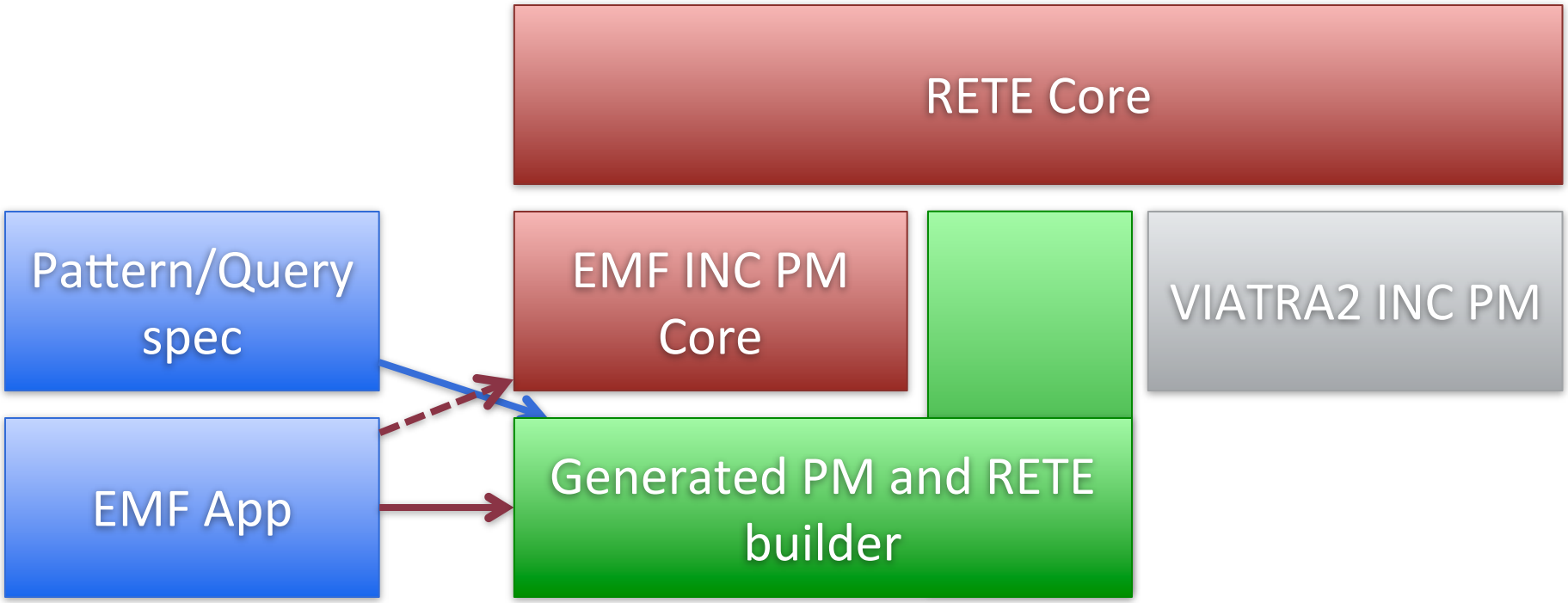
good, if…

- There is enough memory
- Transactional model manipulation



- change: delete/retarget *type* reference

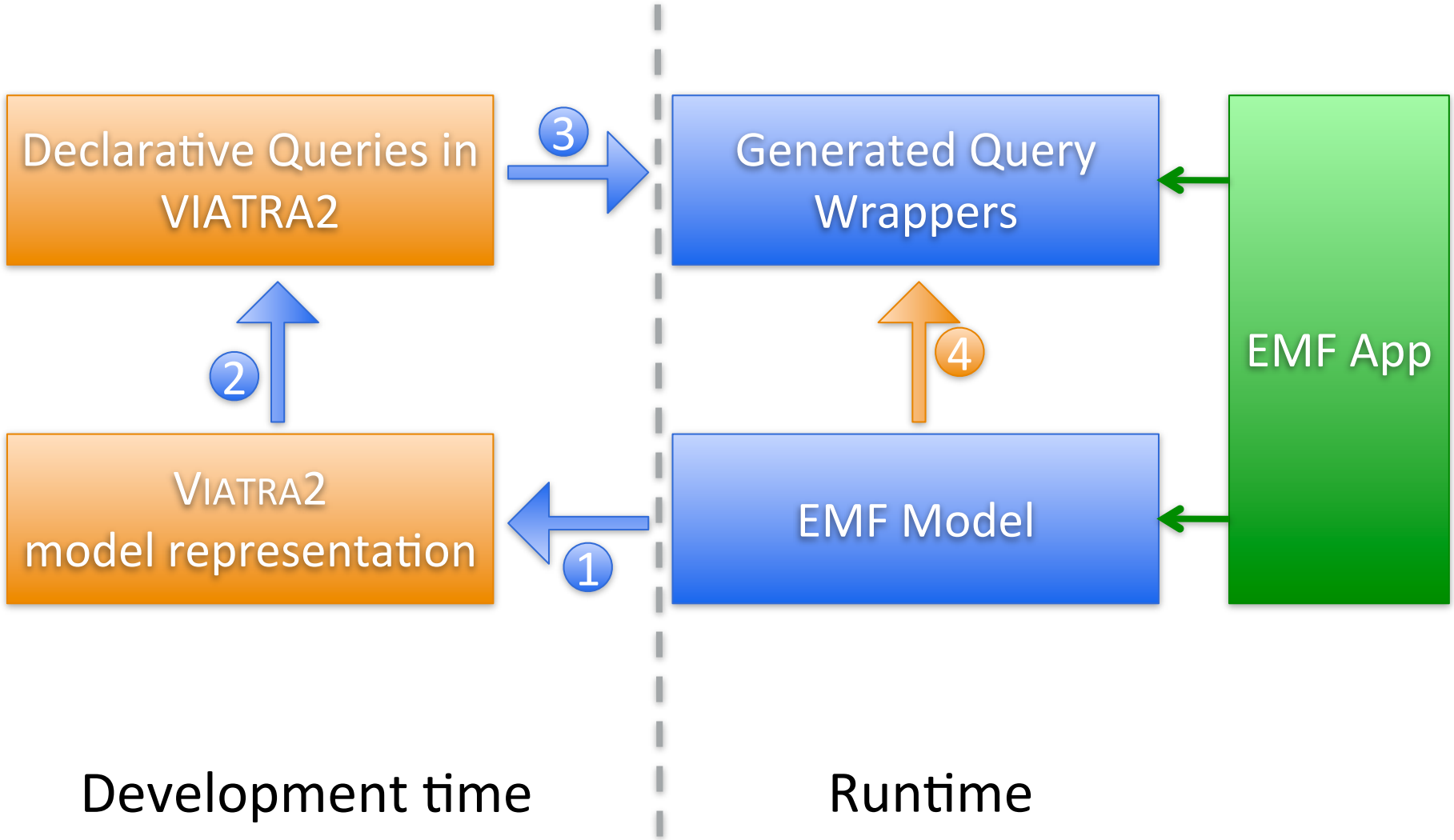
EMF-IncQuery Architecture



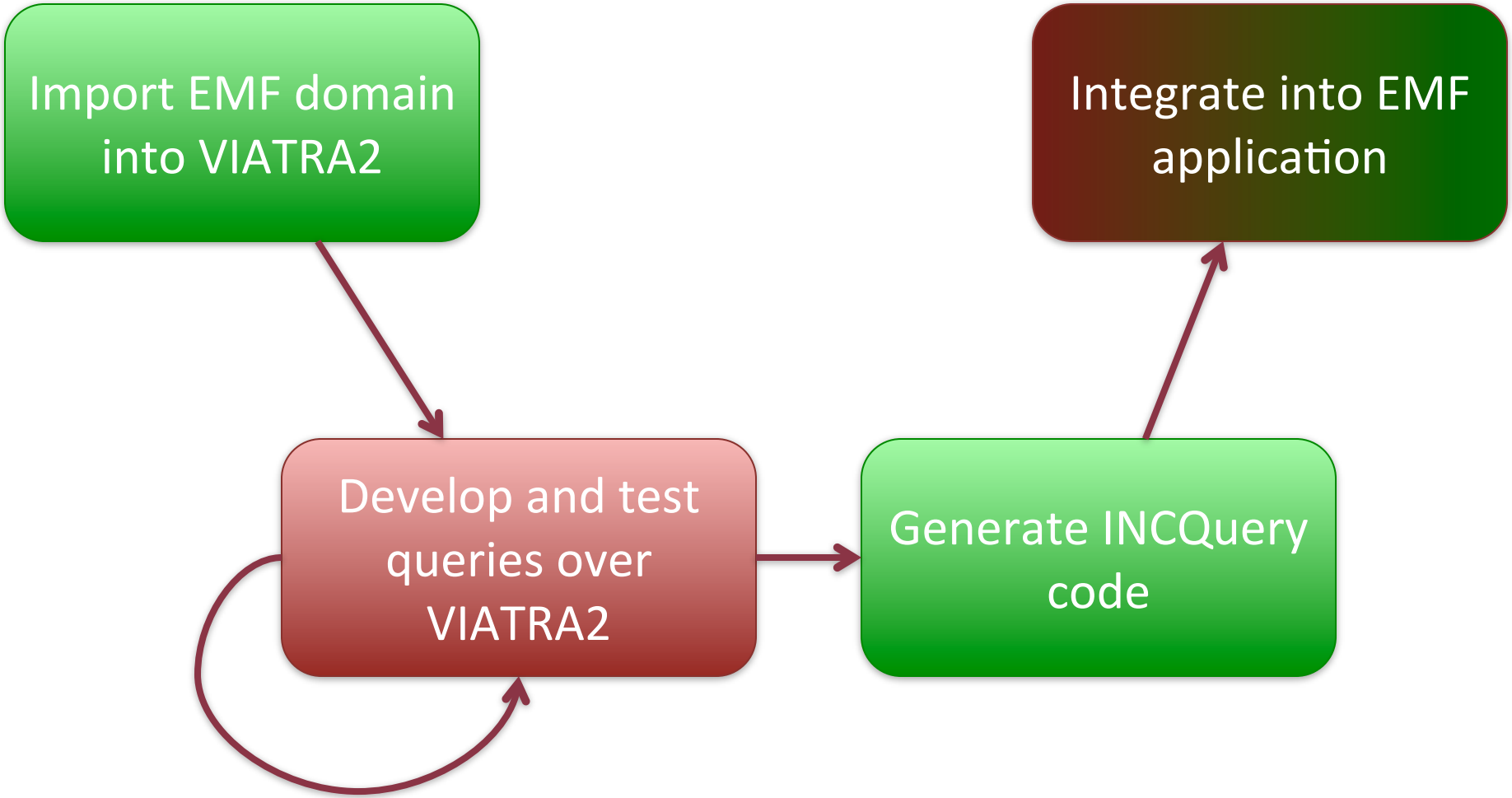
EMF-IncQuery Tooling

- 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

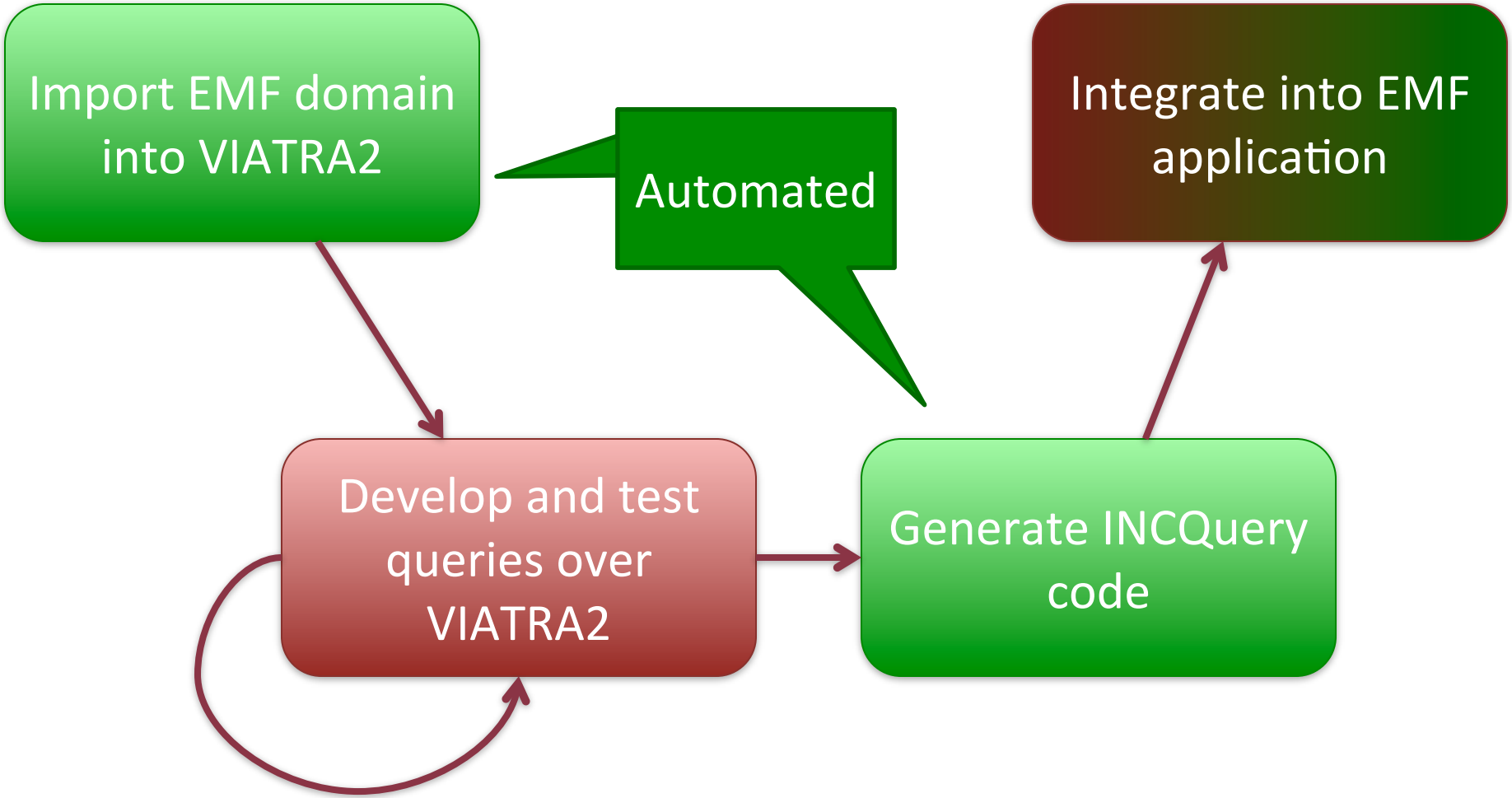
Tooling architecture



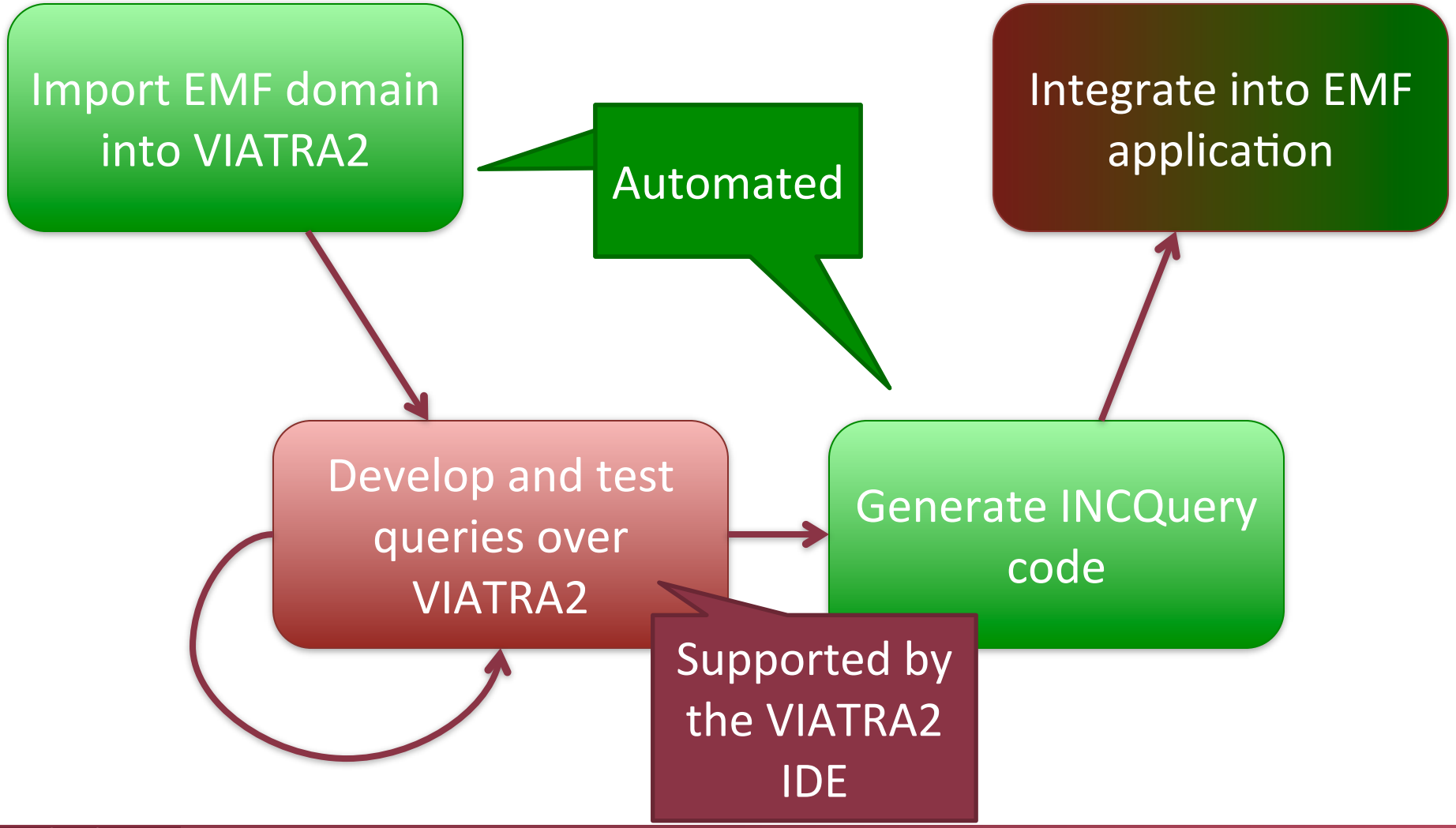
Development workflow



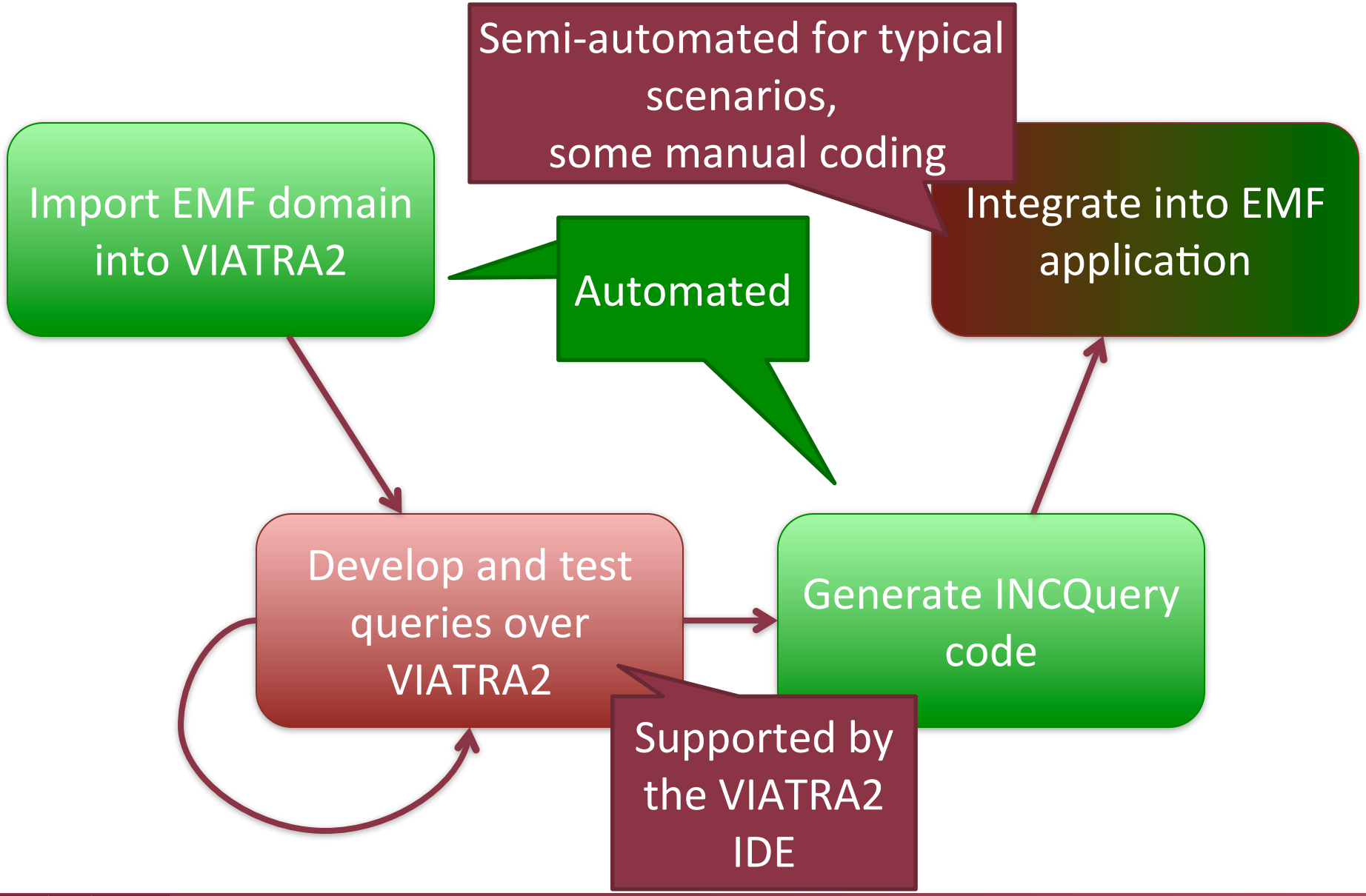
Development workflow



Development workflow



Development workflow



PART 1 - BASICS

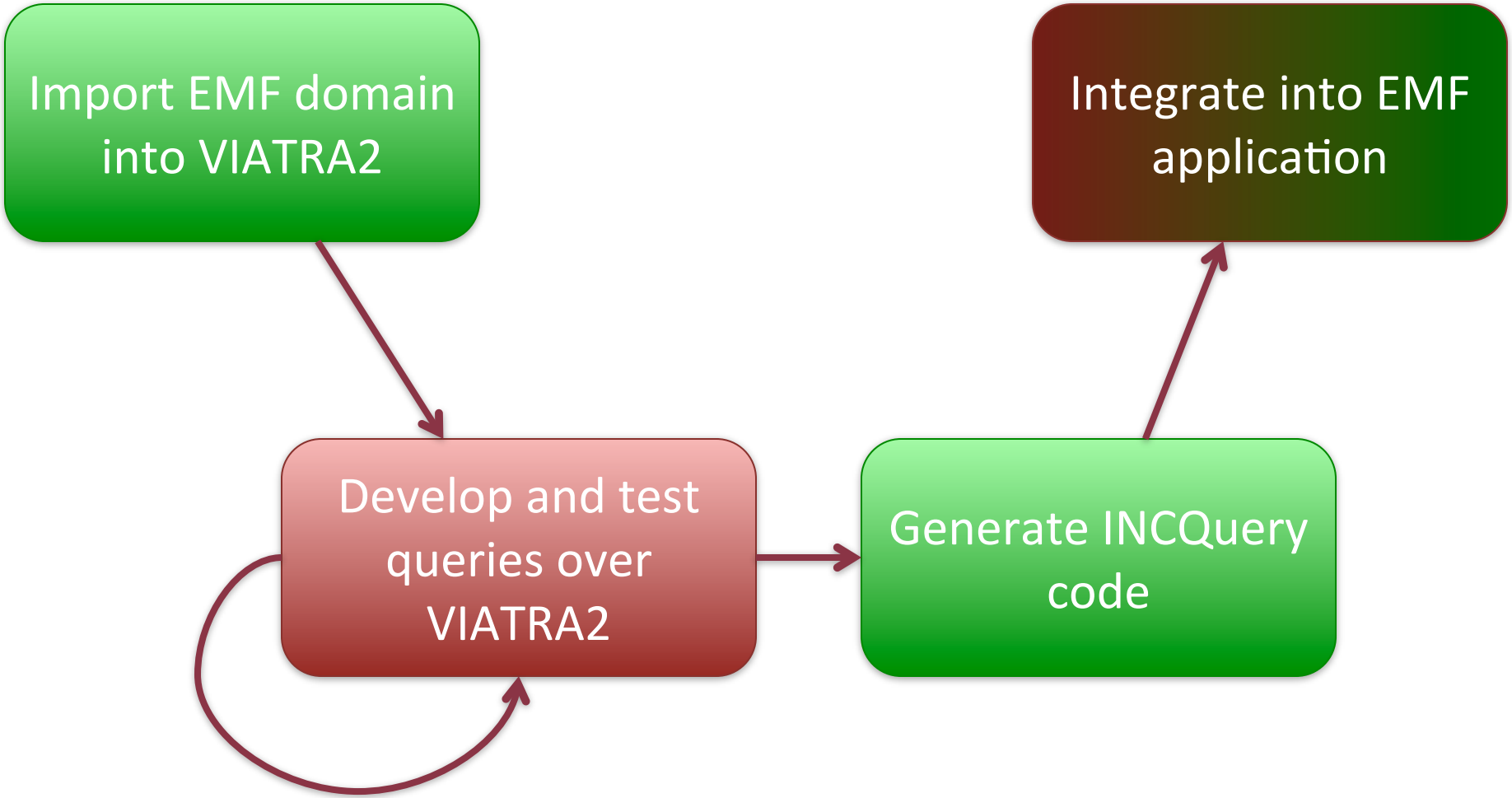
Example 1

- Model statistics
- Inspired by the "model measurement" ATL case studies
  - http://www.eclipse.org/m2m/atl/usecases/ModelsMeasurement/
  - http://www.eclipse.org/m2m/atl/usecases/Measuring_UML_models/

Model measures

- Total number of
  - Packages
  - Classes
  - Attributes
- Number of
  - Children
  - Attributes
  - Attributes inherited
- …

Solution: Development workflow



DEMO Pattern development in VIATRA2

- Create IncQuery project
- Import the UML metamodel
- Import UML instance models
- VTCL development environment
  - Create and debug patterns
  - Visualize pattern matches
  - Test queries/patterns in transformations
  - Create patterns using instance model “examples”

EXAMPLE

Our first IncQueries

```
namespace uml;
import nemf.packages.uml;

machine measurement {

    @SampleUI(mode="list") // <-- list all classes; default
    pattern classes(C) = {
        Class(C);
    }

    @SampleUI(mode="counter") // <-- enumerate all packages
    pattern packages(P) = {
        Package(P);
    }

    @SampleUI(mode="counter") // <-- enumerate all properties
    pattern properties(P) = {
        Property(P);
    }

    rule main() = skip;
}
```

DEMO Generating IncQuery code

- Generate
  - EMF-IncQuery Source code: Pattern matchers
  - Sample UI project: UI integration
- Run Sample UI project commands to see model measures

Generated pattern matchers

- 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

Generated Sample UI

- Command contributions
  - Project explorer, Navigation, Package Explorer
  - Perform model loading and query execution
  - Display the results on the UI
    - List (default)
      - Pretty prints a list of matches
    - Counter
      - Prints the number of matches

IncQuery Runtime

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 Signatures

- **D**ata **T**ransfer **O**bjects generated for
  pattern signatures
- Signature query methods
  - `SiblingClassSignature getOneMatch()`
  - `SiblingClassSignature getOneMatchAsSignature(Object C1, Object C2)`
  - `Collection<SiblingClassSignature> getAllMatchesAsSignature()`
  - `Collection<SiblingClassSignature> getAllMatchesAsSignature(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 SiblingClassSignature
{
    Object C1;
    Object C2;
}
```

Query Signatures

- **D**ata **T**ransfer **O**bjects generated for
  pattern signatures
- Signature query methods
  - `SiblingClassSignature getOneMatchAsSignature()`
  - `SiblingClassSignature getOneMatchAsSignature(UMLClass C1, UMLClass C2)`
  - `Collection<SiblingClassSignature> getAllMatchesAsSignature()`
  - `Collection<SiblingClassSignature> getAllMatchesAsSignature(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 SiblingClassSignature
{
    UMLClass C1;
    UMLClass C2;
}
```

Ongoing work: use domain-specific types in generated code

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
   - Typically: at loading time
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

PART 2 – VALIDATION CASE STUDY

Example 2

- Inspired by the "Verifying UML profiled models" ATL case study
  - http://www.eclipse.org/m2m/atl/usecases/Verifying_UML_profiled_models/
- Well-formedness checking of UML models, with a crucial difference:
  - Incremental, **on-the-fly** validation
  - Maintain error markers as the user is editing the model

UML well-formedness rules

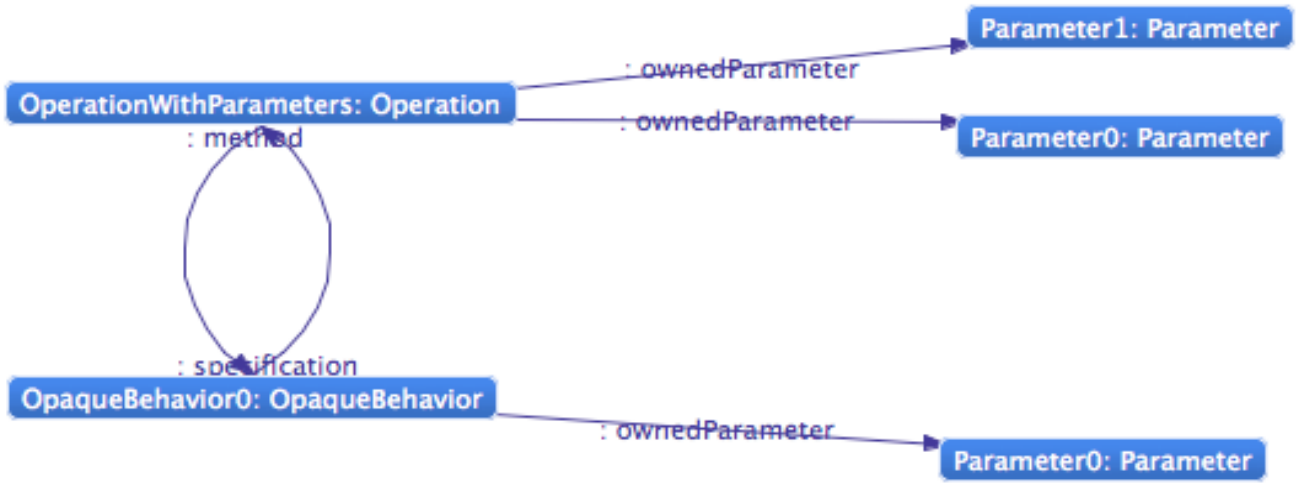
- Traditionally specified by OCL constraints
  - Note: OCL constraints included in UML models serve a different purpose
  - OCL constraints can be attached to any EMF instance model via EMF Validation
- Rules specified by
  - **Tool developers**
  - (End users)

Well-formedness checking from a tool developer's perspective

- Well-formedness rules
  - Express constraints not (easily) possible by metamodeling techniques
  - Ensure "sane" modeling conventions & best practices
  - Aid code generation by design-time validation
- Example

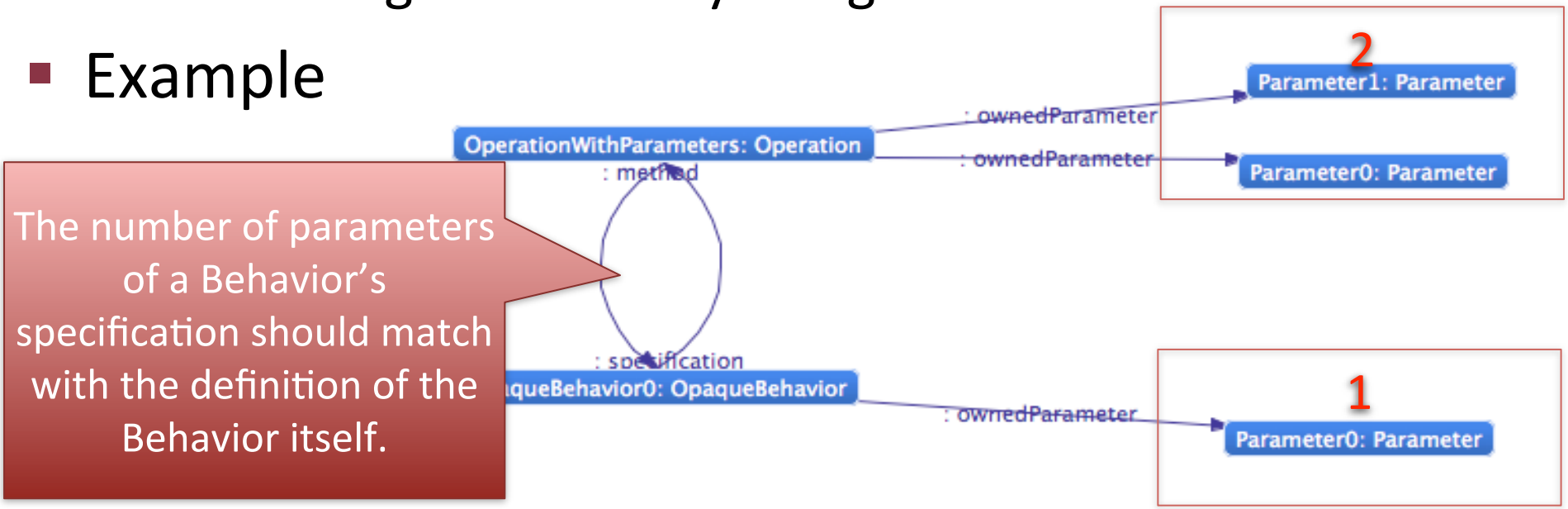
Well-formedness checking from a tool developer's perspective

- Well-formedness rules
  - Express constraints not (easily) possible by metamodeling techniques
  - Ensure "sane" modeling conventions & best practices
  - Aid code generation by design-time validation
- Example



Well-formedness checking from a tool developer's perspective

- Well-formedness rules
  - Express constraints not (easily) possible by metamodeling techniques
  - Ensure "sane" modeling conventions & best practices
  - Aid code generation by design-time validation
- Example



The number of parameters of a Behavior's specification should match with the definition of the Behavior itself.

IncQuery Validation Engine

- Simple validation engine
  - Supports on-the-fly validation through incremental pattern matching and problem marker management
  - Uses IncQuery graph patterns to specify constraints
- Simulates EMF Validation markers
  - To ensure compatibility and easy integration with existing editors
  - Doesn't use EMF Validation directly
    - Execution model is different

How it works – IncQuery 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/Signatures) 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/Signatures from .matchFound/LostEvents

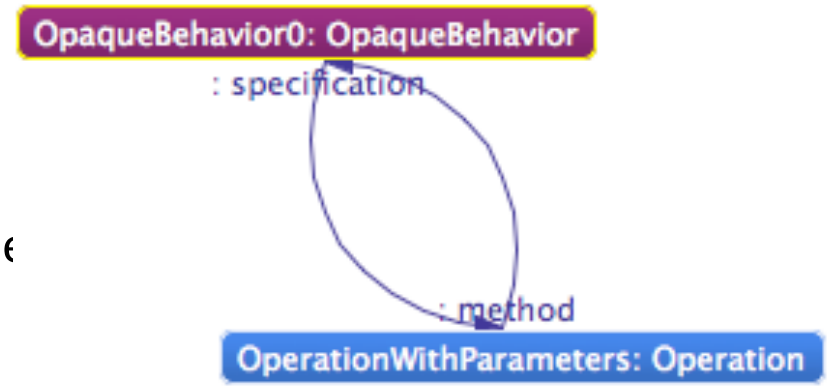
Well-formedness rule specification by graph patterns

- WFRs: *Invariants* which must hold at all times
- Specification = set of elementary constraints + context
  - Elementary constraints: Query (pattern)
  - Location/context: a model element on which the problem marker will be placed
- Constraints by graph patterns
  - Define a pattern for the "bad case"
    - Either directly
    - Or by negating the definition of the "good case"
  - Assign one of the variables as the location/context

Match:
A violation of the invariant

EXAMPLE A simple UML validation constraint

- "All Behaviors must have an Operation as their specification."
  - Otherwise they do not have any "interface" through which they could be accessed → "dead code"

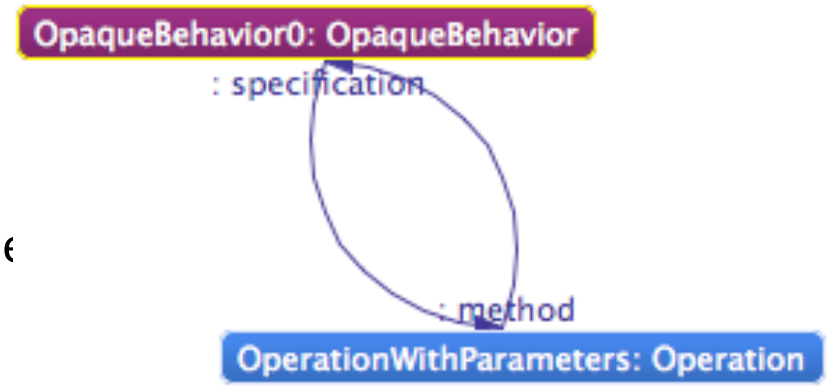


- Bad case:

```
@Constraint(mode="problem",location="Behavior",
    message="The behavior $Behavior$ has no specification operation.")
pattern OpaqueBehaviorWithoutOperation(Behavior) = {
    OpaqueBehavior(Behavior);

    neg pattern behaviorHasSpecification(Behavior) = {
        Behavior(Behavior);
        Behavior.specification(SpecRel,Behavior,Specification);
        BehavioralFeature.method(BFRel,Specification,Behavior);
        Operation(Specification);
    }
}
```

EXAMPLE A simple UML validation constraint

- "All Behaviors must have an Operation as their specification."
  - Otherwise they do not have any "interface" through which they could be accessed → "dead code"



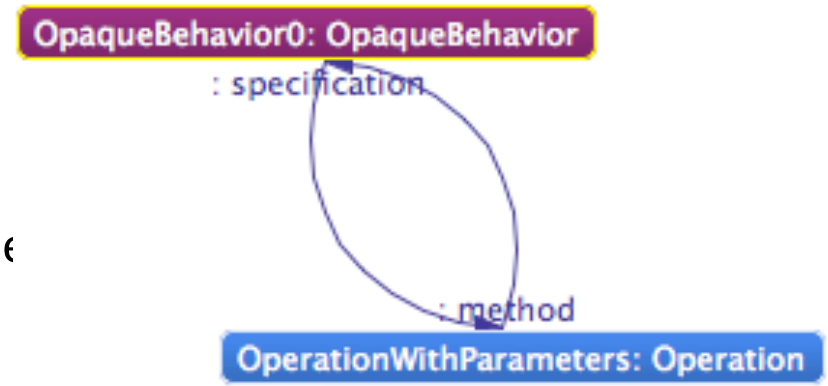
- Bad case:

```
@Constraint(mode="problem",location="Behavior",
    message="The behavior $Behavior$ has no specif
pattern OpaqueBehaviorWithoutOperation(Behavior) =
    OpaqueBehavior(Behavior);

    neg pattern behaviorHasSpecification(Behavior) = {
        Behavior(Behavior);
        Behavior.specification(SpecRel,Behavior,Specification);
        BehavioralFeature.method(BFRel,Specification,Behavior);
        Operation(Specification);
    }
}
```

Identify pattern variable "Behavior" as the location

EXAMPLE A simple UML validation constraint

- "All Behaviors must have an Operation as their specification."
  - Otherwise they do not have any "interface" through which they could be accessed → "dead code"
- Bad case:




```
@Constraint(mode="problem",location="Behavior",
    message="The behavior $Behavior$ has no specif
pattern OpaqueBehaviorWithoutOperation(Behavior) =
    OpaqueBehavior(Behavior)

    neg patter
        Behav
        Behav
        Behav
        Opera
    }
}
```

Identify pattern variable "Behavior" as the location

- Substitute the model element match to "Behavior".
- Syntax: $Behavior.featureName$
- $Behavior$: shortcut for $Behavior.name$

DEMO Constraint code generation

- Generate sample validation project
- Observe on-the-fly validation in action

Generated Sample Validation project

- Java classes: Constraint descendants
- Plugin.xml
  - Constraint registration
  - UI integration
    - Editor ID from genmodel

Validation lifecycle

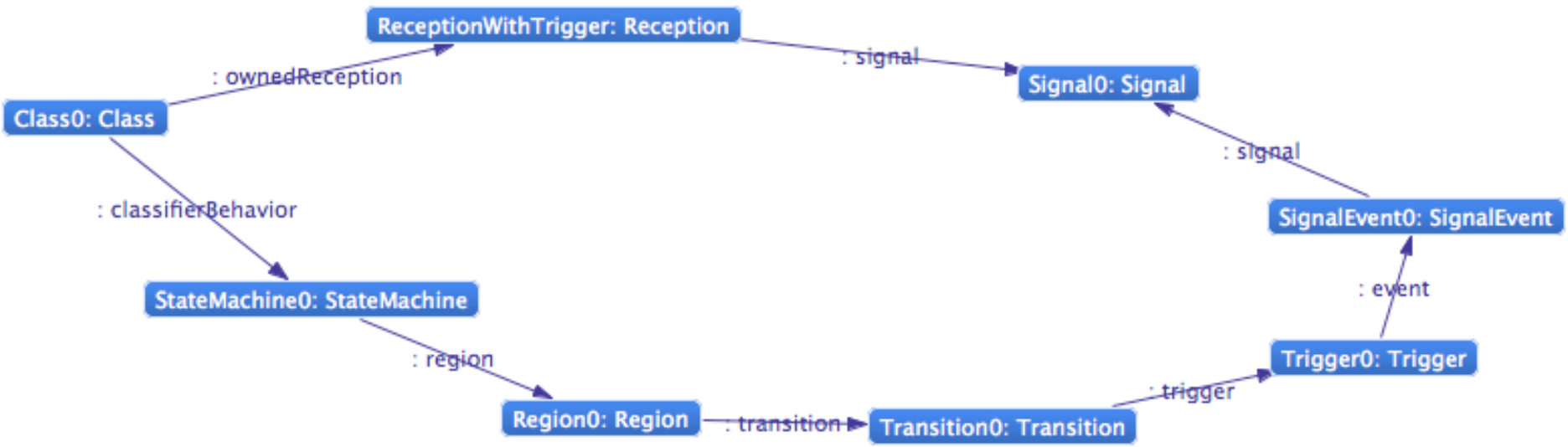
- Constraint violations
  - Represented by Problem Markers (Problems view)
  - Marker text is updated if affected elements are changed in the model
  - Marker removed if violation is no longer present
- Lifecycle
  - Editor bound validation (markers removed when editor is closed)
  - Incremental maintenance not practical outside of a running editor

Validation UI integration

- A menu item (command) to start the validation engine
- Generic (part of the IncQuery Validation framework)
  - GMF editor command
    - Appears in all GMF-based editor's context menu
  - Sample Reflective Editor command
    - Appears on the toolbar
- Generated
  - EMF generated tree editor command
    - Appears on the toolbar

EXAMPLE A complex UML constraint

- "Unused signal"
  - A Class defines a Reception for a Signal, but it doesn't use it in its StateMachine for triggering a transition.
- Good case:



EXAMPLE A complex UML constraint

- "All Receptions of a Class must use Signals that are used in the defining Class' StateMachine as a Transition Trigger."
- Bad case:

```
@Constraint(mode="problem",location="Reception",
    message="The reception $Reception$ lacks triggers.")
pattern receptionWithoutAction(Reception) = {

    Class(Class);
    Reception(Reception);
    Class.ownedReception(RecRel,Class,Reception);
    Reception.signal(S_R,Reception,Signal);
    Signal(Signal);

    neg pattern usedInStateMachine(Signal, Class) = {
        StateMachine(StMach);
        Class(Class);
        BehavioredClassifier.classifierBehavior(ClBehRel,Class,StMach);
        Trigger(Trigger) below StMach;

        SignalEvent(Event);
        Trigger.event(EventRel, Trigger, Event);
        Signal(Signal);
        SignalEvent.signal(SignalRel, Event, Signal);
    }
}
```

EXAMPLE A complex UML constraint

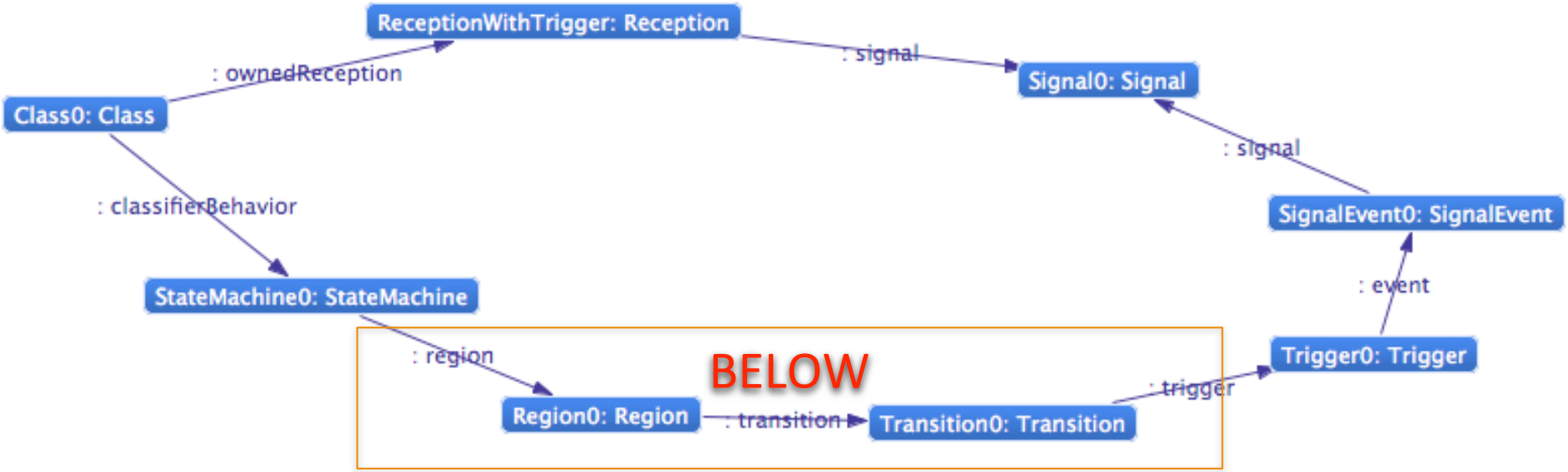
- "All Receptions of a Class must use Signals that are used in the defining Class' StateMachine as a Transition Trigger."
- Bad case:

```
@Constraint(mode="problem",location="Reception",
    message="The reception $Reception$ lacks triggers.")
pattern receptionWithoutAction(Reception) = {

    Class(Class);
    Reception(Reception);
    Class.ownedReception(RecRel,Class,Reception);
    Reception.signal(S_R,Reception,Signal);
    Signal(Signal);

    neg pattern usedInStateMachine(Signal, Class) = {
        StateMachine(StMach);
        Class(Class);
        BehavioredClassifier.classifierBehavior(ClBehRel,Class,StMach);
        Trigger(Trigger) below StMach;

        SignalEvent(Event);
        Trigger.event(EventRel, Trigger, Event);
        Signal(Signal);
        SignalEvent.signal(SignalRel, Event, Signal);
    }
}
```

below expresses a transitive containment relationship.

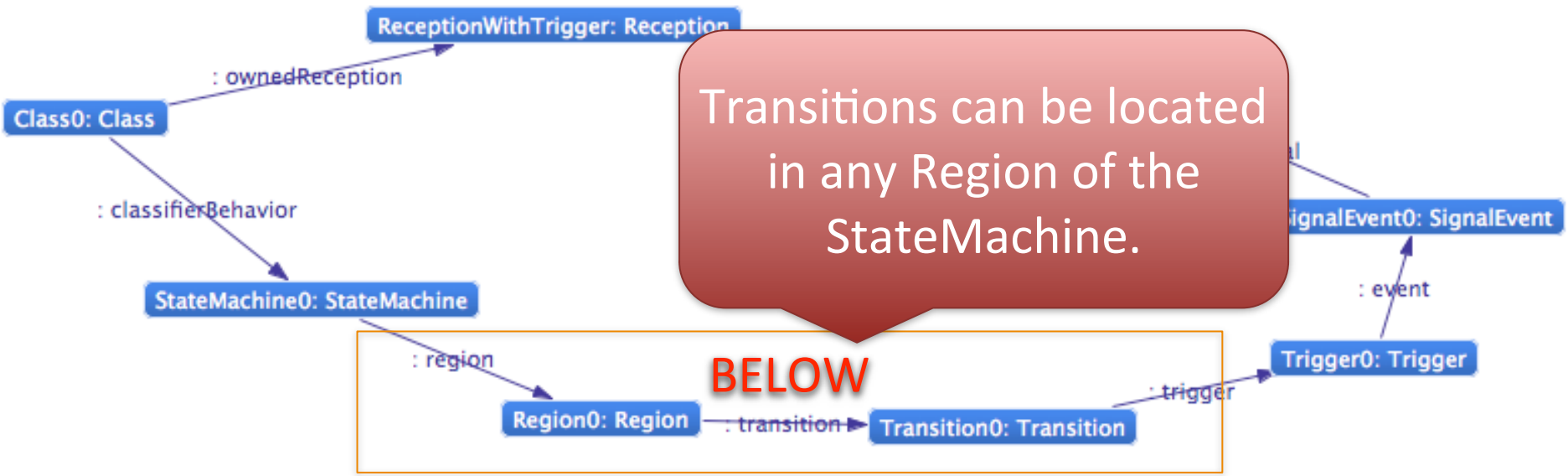
EXAMPLE A complex UML constraint

- "All Receptions of a Class must use Signals that are used in the defining Class' StateMachine as a Transition Trigger."



EXAMPLE A complex UML constraint

- "All Receptions of a Class must use Signals that are used in the defining Class' StateMachine as a Transition Trigger."

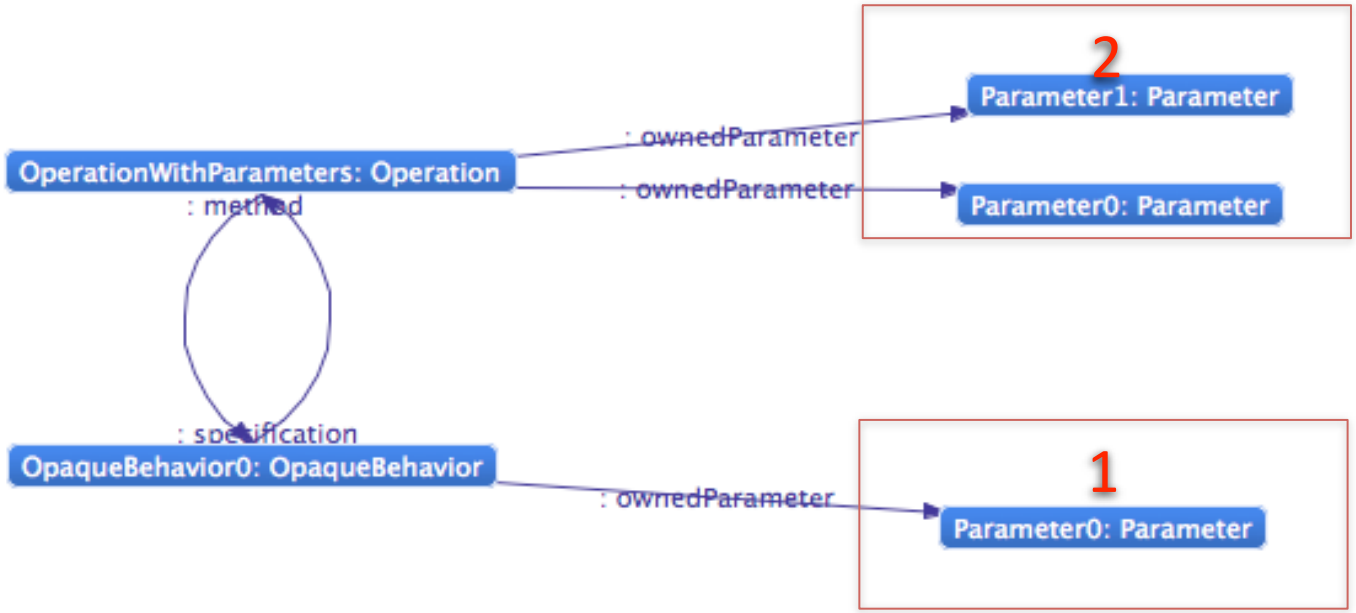


DEMO On-the-fly validation on a large model

- Generate Sample Validation code
- Load modelGenerationExample_2000.uml
- Observe on-the-fly validation
  - Performance is capped by the Eclipse Marker management layer.

EXAMPLE Extra assignment

- The number of parameters of a Behavior's specification Operation should match with the Behavior's definition.
- Bad case:



EXAMPLE Solution

```
@Constraint(mode="problem",location="Behavior",
 message="The number of parameters of $Behavior$ are different from its specification $Operation$")
pattern incompatibleSpecification(Behavior, Operation) = {
    OpaqueBehavior(Behavior);
    Operation(Operation);
    Behavior.specification(S_R,Behavior,Operation);

    find parameterOfOperation(OperationParameter, Operation) # N;
    find parameterOfBehavior(BehaviorParameter, Behavior) # M;
    check(N != M);
}

@SampleUI(mode="ignore")
pattern parameterOfOperation(P, O) = {
    Operation(O);
    Parameter(P);
    BehavioralFeature.ownedParameter(O_R,O,P);
}

@SampleUI(mode="ignore")
pattern parameterOfBehavior(P, OB) = {
    OpaqueBehavior(OB);
    Behavior.ownedParameter(O_R,OB,P);
    Parameter(P);
}
```

Cardinality constraints expressing the "number of matches"

PERFORMANCE BENCHMARKING

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 at MODELS 2010
  - Motivation: the Embedded Architect Tool of OptXware Ltd.
  - AUTOSAR models can be very large (>>1m elements)

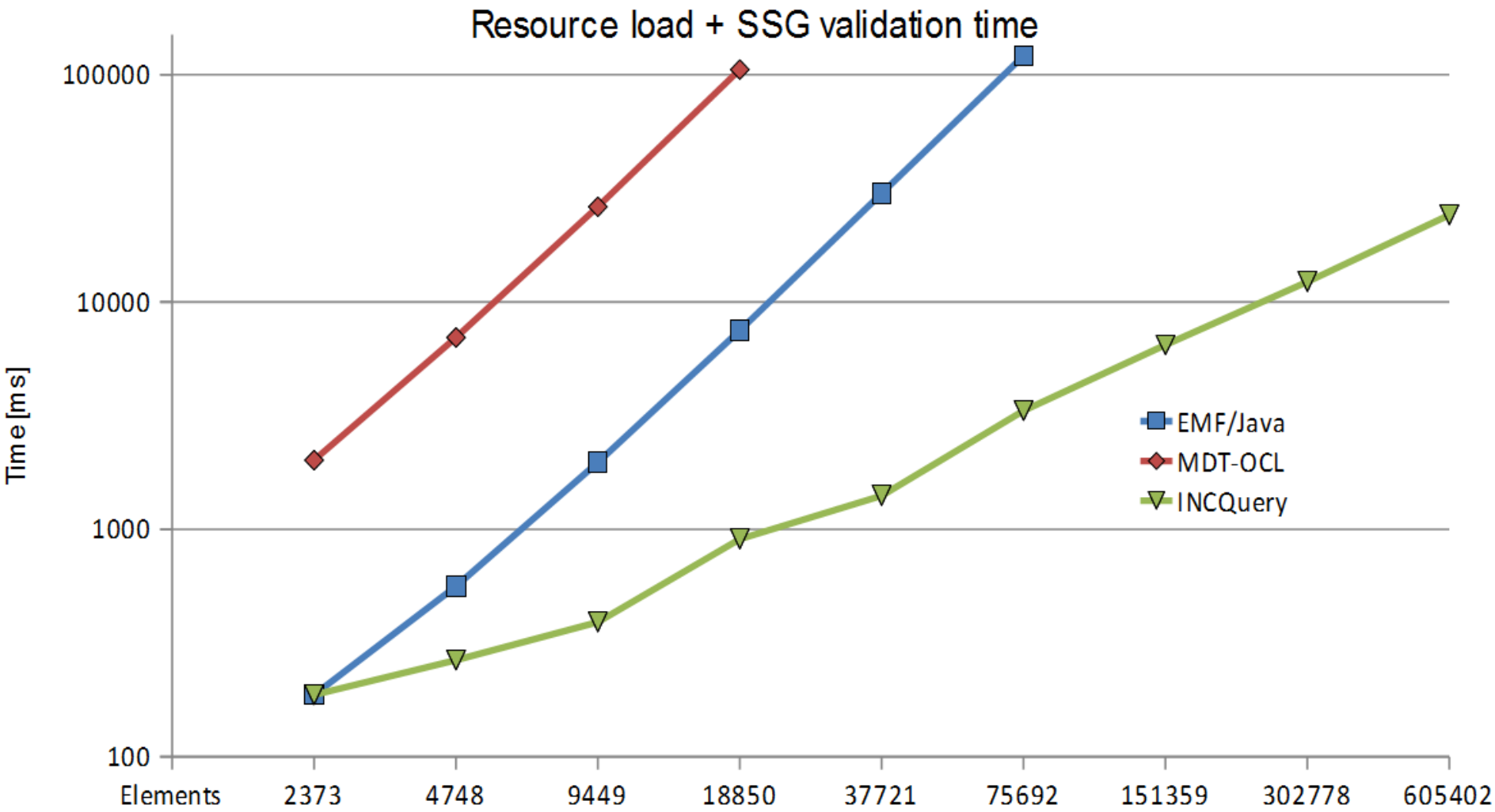
What is measured?

- 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

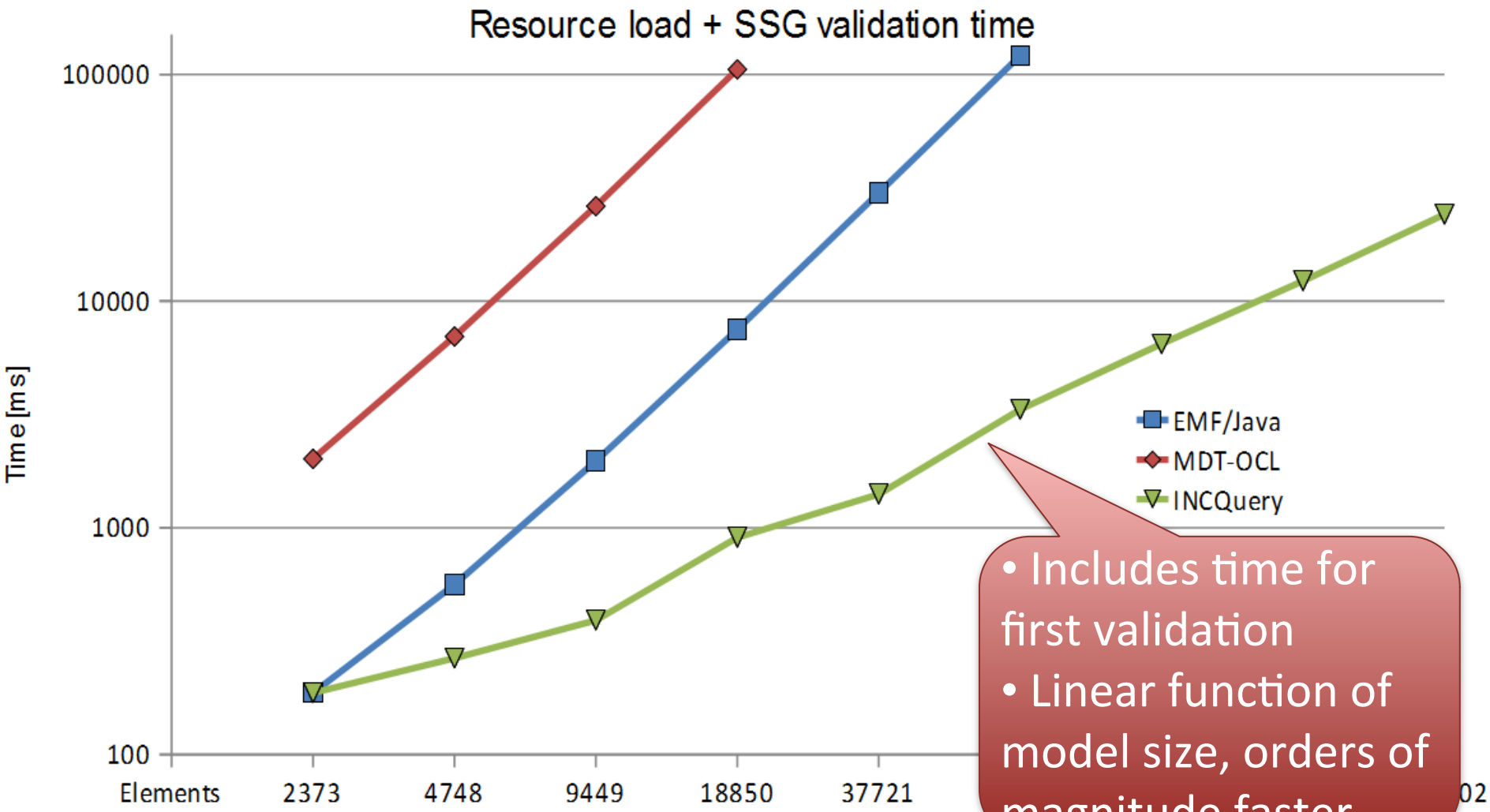
IncQuery Results

- 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?

Initialization time

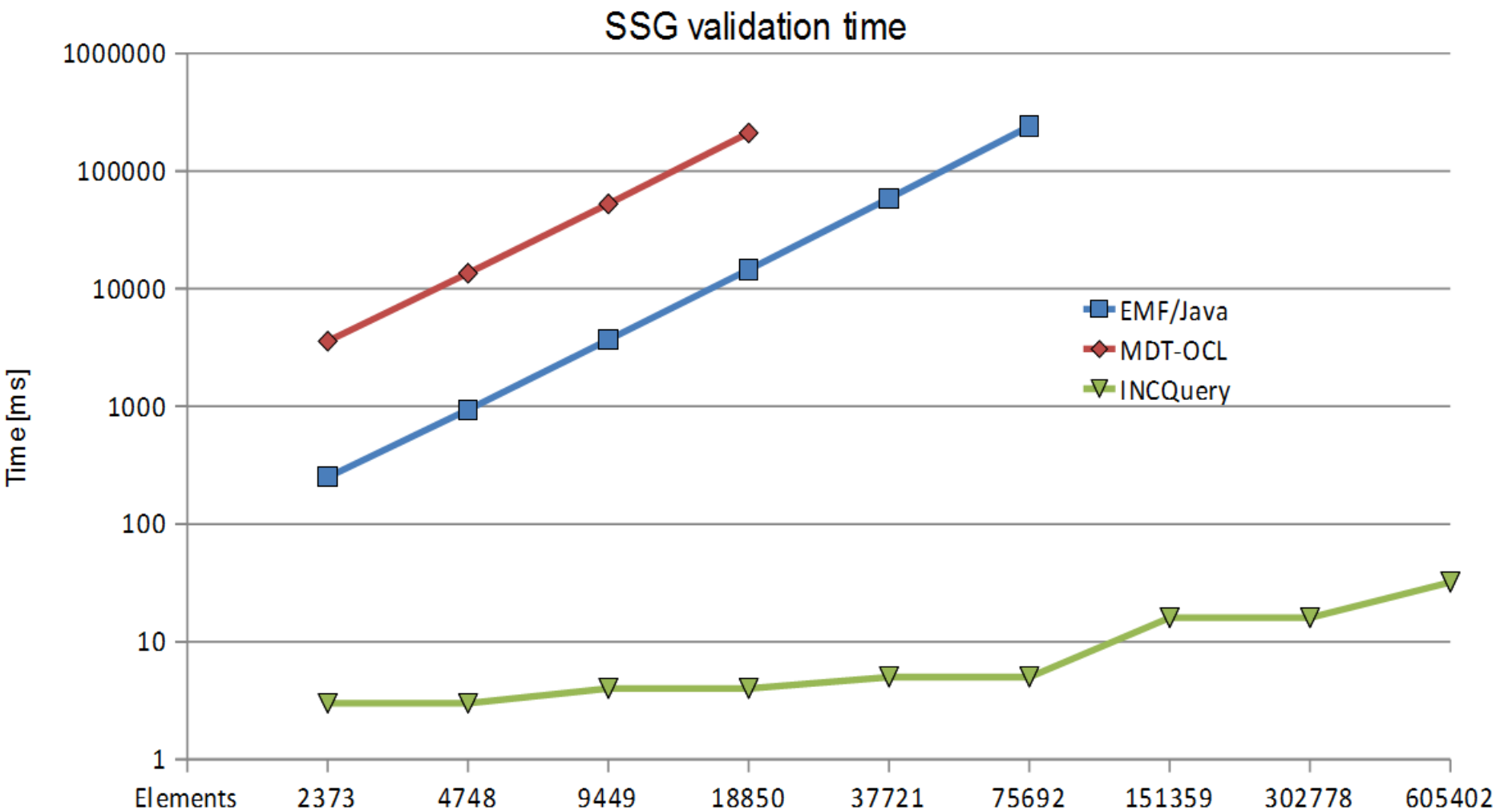


Initialization time

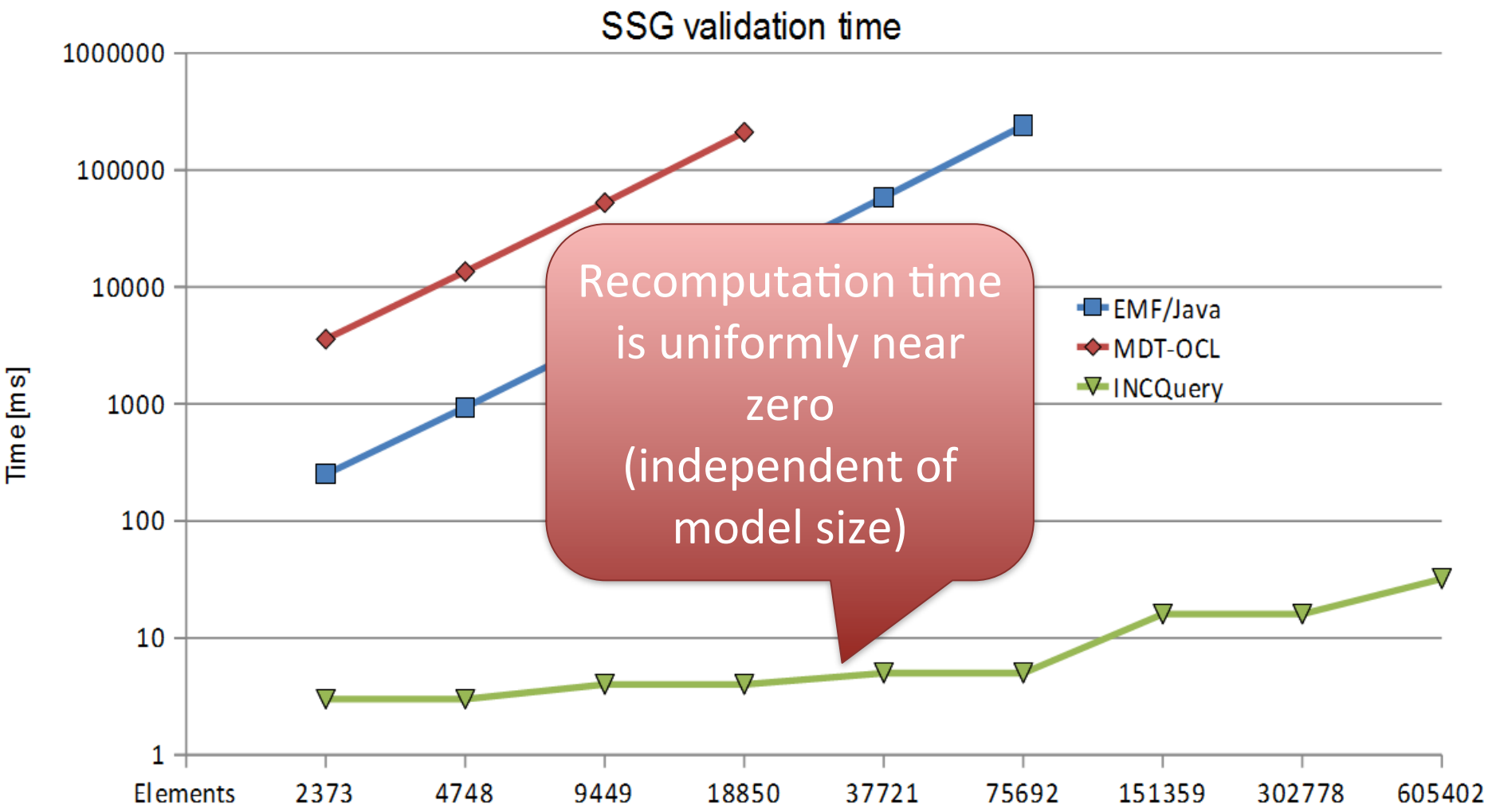


- Includes time for first validation
- Linear function of model size, orders of magnitude faster

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

- High performance complex queries are hard to write manually in Java.
- IncQuery can do the trick nicely as long as you have enough RAM.
- Metamodel structure has huge impact on performance when using “conventional” query technologies such as OCL, due to
  - Lack of reverse navigation
  - Lack of type enumeration (.allInstances())

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 a desktop PC)

CONCLUSION

Closing thoughts

- 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
  - …
- The tutorial examples
  - Do not make use of advanced features such as parameterized queries or complex structural constraints (recursion)
  - Are meant only as a starting point
  - The project website has many more examples!

Model transformations based on IncQuery

- High performance model transformations
  - Hybrid query approach
    - Use IncQuery for
      - Complex queries
      - Frequently used queries
      - Parameterized queries
    - Plain Java for simple queries (saves RAM)
  - Java for control structure & model manipulation
- High-level transformation languages (VIATRA2, ATL, Epsilon, …) could be “compiled” to run on this infrastructure
- Ongoing research: automatic mapping of SPARQL, OCL & others to the IncQuery language

~~Wish list~~ IncQuery features ☺

- Declarative query language
  - Easy to learn
  - Good bindings to the imperative world (Java)
  - Safe yet powerful
  - Reusable
- High performance
  - Fast execution for complex queries over large models
  - First-class support for incremental execution
- Technology
  - Works with EMF out-of-the-box

Pointers

- Open source project:
  - http://viatra.inf.mit.bme.hu/incquery
- VIATRA2 (grandfather)
  - http://eclipse.org/gmt/VIATRA2
  - http://viatra.inf.mit.bme.hu
- István
  - rath@mit.bme.hu
- The development team
  - viatra-dev@inf.mit.bme.hu
  - We're glad to help with any problems you might have.
  - Check the site for FAQ, examples & more.
- Questions & answers