Domain-specific Performance Optimizations of Pattern Matching in Henshin

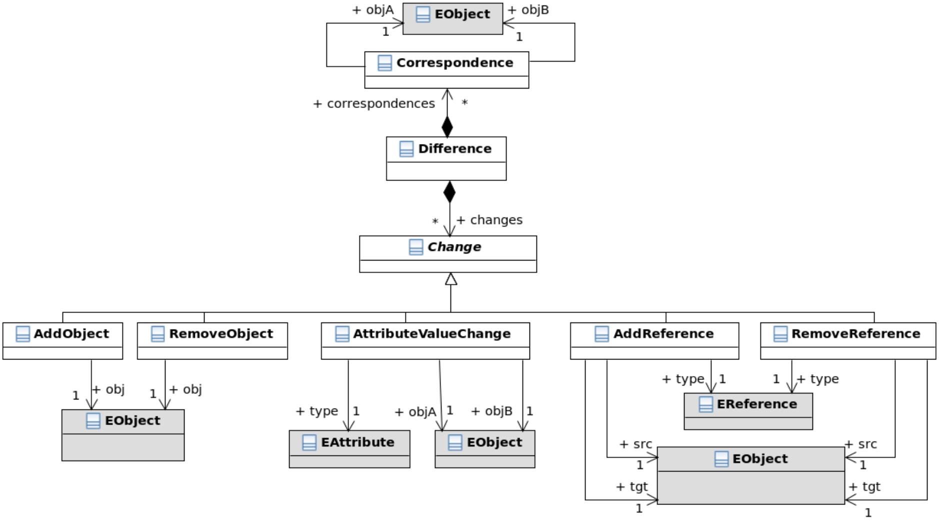
Timo Kehrer, Manuel Ohrndorf

Software Engineering Group
Univ. of Siegen

"Henshin Monthly"
3. Juni 2014

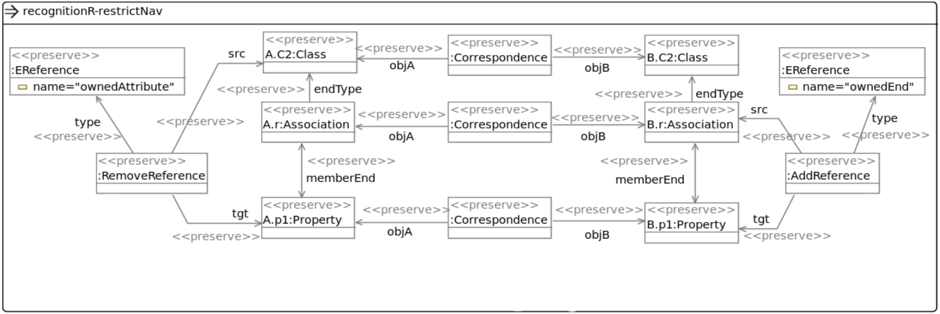
1. Application Domain
2. Performance Optimizations
3. Experiences and Conclusions

Meta-model (Representation of Model Differences)



Change Set Recognition Rules

- Example: “Restrict association navigability” in UML (clipping)
- We use Henshin as pattern matching engine for detecting such change patterns
- Involved changes are finally grouped to so called semantic change sets



Recognition of Semantic Change Sets

Problem description:

- Given a rule set R and a difference D ("difference model")
- For each rule r in R, find all matches in D

Assumptions:

- None of the rules in R modifies our model D, i.e. all rules are parallel independent of each other
- The rule set R is fixed and none of the rules is modified at runtime

Overview

- Parallel execution with a pool of worker threads
- Sorting of LHS nodes
- Reduction of the pre-defined rule set
- Creation of minimal EGraphs

Sorting of LHS Nodes

Two (in general orthogonal) heuristics:

- Number of occurrences of objects of a certain type in a difference
- "Position" of LHS nodes within a rule

Remarks:

- Sorting determines the order in which the recursive constraint solving algorithm locks variables
- Heuristics similar to generic heuristics in Henshin matching engine?

Reduction of the Pre-defined Rule Set

Basic proceeding:

1. Create index on difference D:
   - EClass $\rightarrow$ int, t $\mapsto NoN_t^M$
   - NoN_t^M is the number of objects of type t (or one of its subtypes) in D
2. Create index ("signature") for each rule r in R
   - EClass $\rightarrow$ int, t $\mapsto NoN_t^R$
   - NoN_t^R is the number of objects of type t in R
3. A rule is matchable if its signature can be found in D

Extensions:

- Basic proceeding is extended to domain-specific type definitions
- For instance, tuples such as (AddObject,obj) or (AddReference,src,tgt,type) can be regarded as complex types

Creation of Minimal EGraphs

Basic idea:

- We create a separate EGraph for finding all matches of a rule.

General proceeding:

- Given a rule r, let T_r be the set of all node types used by r
- EGraph $G = \{o \in D \mid (o.type \in T_r) \vee (o.type.allSuperTypes \cap T_r \neq \emptyset)\}$

Remarks:

- Should be already handled by reducing the CSP solution space based on type constraints
- However, basic proceeding is extended to domain-specific definitions of complex types

1. Application Domain
2. Performance Optimizations
3. Experiences and Conclusions

Experiences and Conclusions

- **Parallel execution:**
  - Strongly depends on the underlying hardware
- **Sorting of LHS nodes:**
  - Biggest performance gain, even for very small models (from minutes to seconds)
  - Should now already be handled by generic sorting of CSP variables
- **Rule set reduction:**
  - Moderate performance gain for large rule sets ($>$ 500 rules)
  - Generalizable, but current rule set has to be continuously adapted
- **Minimal EGraphs:**
  - Similar to reducing the CSP solution space based on type constraints
  - Generalization option: API or config file for application-specific definitions of complex types