A Change Metamodel for the Evolution of MOF-based Metamodels

Erik Burger, Boris Gruschko | 26 March 2010
Scenario

Metamodel version 1.0
Scenario

Metamodel version 1.0

instances
Scenario

Metamodel version 1.0

instances

conforms to
Scenario

Metamodel version 1.0 \[\sim\] evolution \[\sim\] Metamodel version 1.1

instances

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels

26 March 2010

Background

Change Metamodel

Classification of Changes

Conclusion
Scenario

Metamodel version 1.0 \(\sim\) evolution \(\sim\) Metamodel version 1.1

\[
\begin{align*}
&\text{instances} \\
\text{A} &\sim\text{conforms to}\sim\text{conforms to}\text{B} &\sim\text{conforms to}\sim\text{C}
\end{align*}
\]
Scenario

Metamodel version 1.0 \(\sim\) evolution \(\sim\) Metamodel version 1.1

\[\text{instances}\]

\[B\] \(\sim\) conforms to

\[C\] \(?\) conforms to

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels

26 March 2010
**Contribution**

**Change Metamodel**

- Metamodel for the formalisation of changes to MOF 1.4 models
- Classification of changes into change severities
- Formal derivation rules for severities
Difference of Metamodels – Determination

Direct Comparison

+ minimal set of changes
+ independent from metamodel editing tool
  - algorithm needed to find sequence of single changes
  - high granularity may lead to loss in semantics

Tracing of Changes

+ determines change steps
  - needs tool support
  - redundant changes have to be eliminated
Difference of Metamodels – Determination

Direct Comparison

+ minimal set of changes
+ independent from metamodel editing tool
– algorithm needed to find sequence of single changes
– high granularity may lead to loss in semantics

Tracing of Changes

+ determines change steps
– needs tool support
– redundant changes have to be eliminated

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels

26 March 2010
Difference of Metamodels – Representation

Metamodel version 1.0

Metamodel version 1.1

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels

26 March 2010
Difference of Metamodels – Representation

Change Description

Metamodel version 1.0

references

Change Metamodel

Metamodel version 1.1

references

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels
Difference of Metamodels – Representation

Metamodel version 1.0 + Change Metamodel Instance

References
Difference of Metamodels – Representation

Metamodel version 1.0 + references Instance

Change Metamodel Instance

creates

Metamodel version 1.1
change metamodel instance contains elements that are added

similar to the approach of EMFCompare

Metamodel version 1.0 + references Change Metamodel Instance creates Metamodel version 1.1
Change Metamodel

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels
Properties of the Change Metamodel

- formal representation of metamodel changes
- automatic derivation of change severities by OCL expressions
- ... it is itself specified in MOF 1.4
Example

Type1
attr1:int [1..*]
Example

Type2

Type1

attr1:int [1..*]

Change1: ExistenceChange
kind = ADD
affectedElement = Type2
Example

Type2

attr1:int [1..*]

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1
Example

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1
Example

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

Change4: ContainsChange
kind = ADD
container = Type2
containedElement = attr1
**Change Severities**

```xml
<ModelElementChange
  /severity:SeverityKind

«enumeration»
SeverityKind
NON_BREAKING
BREAKING_RESOLVABLE
BREAKING_NOT_RESOLVABLE
```

- **Nonbreaking**
  - does not invalidate M1 data

- **Breaking and Resolvable**
  - can invalidate M1 data
  - algorithm can be defined to resolve conflict

- **Breaking and not Resolvable**
  - can invalidate M1 data
  - manual interaction necessary
Change Severities

```
ModelElementChange
/severity:SeverityKind
```

<table>
<thead>
<tr>
<th>«enumeration» SeverityKind</th>
</tr>
</thead>
<tbody>
<tr>
<td>NON_BREAKING</td>
</tr>
<tr>
<td>BREAKING_RESOLVABLE</td>
</tr>
<tr>
<td>BREAKING_NOT_RESOLVABLE</td>
</tr>
</tbody>
</table>

**Nonbreaking**
- does not invalidate M1 data

**Breaking and Resolvable**
- can invalidate M1 data
- algorithm can be defined to resolve conflict

**Breaking and not Resolvable**
- can invalidate M1 data
- manual interaction necessary
**Change Severities**

**ModelElementChange**

/severity:SeverityKind

«enumeration»

SeverityKind

NON_BREAKING

BREAKING_RESOLVABLE

BREAKING_NOT_RESOLVABLE

**Nonbreaking**

- does not invalidate M1 data

**Breaking and Resolvable**

- can invalidate M1 data
- algorithm can be defined to resolve conflict

**Breaking and not Resolvable**

- can invalidate M1 data
- manual interaction necessary
Change Severities

- **Nonbreaking**: does not invalidate M1 data
- **Breaking and Resolvable**: can invalidate M1 data, algorithm can be defined to resolve conflict
- **Breaking and not Resolvable**: can invalidate M1 data, manual interaction necessary

**ModelElementChange**

/severity:SeverityKind

«enumeration» SeverityKind

- NON_BREAKING
- BREAKING_RESOLVABLE
- BREAKING_NOT_RESOLVABLE
Example

Type2
attr1:int [1..*]

Type1

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

Change4: ContainsChange
kind = ADD
container = Type2
containedElement = attr1
Example

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

Change4: ContainsChange
kind = ADD
container = Type2
containedElement = attr1
Example

Type2
attr1:int [1..*]

Type1

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

non-breaking

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

non-breaking

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

Change4: ContainsChange
kind = ADD
container = Type2
containedElement = attr1
Example

**Type2**

*attr1:* int [1..*]

**Type1**

---

**Change1**: ExistenceChange
kind = ADD
affectedElement = Type2

**Change2**: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

**Change3**: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

**Change4**: ContainsChange
kind = ADD
container = Type2
containedElement = attr1

---

**Background**

**Change Metamodel**

**Classification of Changes**

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels

26 March 2010

---

**Conclusion**
Example

Type2
attr1:int [1..*]

Type1

Change1 : ExistenceChange
kind = ADD
affectedElement = Type2

non-breaking

Change2 : GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

non-breaking

Change3 : ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

breaking and resolvable

Change4 : ContainsChange
kind = ADD
container = Type2
containedElement = attr1

breaking and not resolvable

Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels
26 March 2010
Determining overall Severity

Problem

- severity of a change sequence = maximum severity of contained changes
- changes can revert each other/weaken severity
- severities vary for different combinations of changes

Solution

- In order to calculate the severity of a single change, all other changes of the same change sequence have to be regarded.
- then, take maximum of single changes
- description of single severities as OCL expression
Determining overall Severity

Problem

- severity of a change sequence = maximum severity of contained changes
- changes can revert each other/weaken severity
- severities vary for different combinations of changes

Solution

- In order to calculate the severity of a single change, all other changes of the same change sequence have to be regarded.
- then, take maximum of single changes
- description of single severities as OCL expression
Example

Type2
attr1:int [1..*]

Type1

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

Change4: ContainsChange
kind = ADD
container = Type2
containedElement = attr1

non-breaking

breaking and resolvable

breaking and not resolvable

non-breaking

non-breaking

breaking and resolvable

not resolvable
Example

Type2
attr1:int [1..*]

Type1

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

non-breaking

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

non-breaking

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

breaking and resolvable

Change4: ContainsChange
kind = ADD
container = Type2
containedElement = attr1

breaking and not resolvable
Example

**Type2**
attr1:int [1..*]

**Type1**

---

**Change1**: ExistenceChange
kind = ADD
affectedElement = Type2

**Change2**: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

**Change3**: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

**Change4**: ContainsChange
kind = ADD
container = Type2
containedElement = attr1

---

**Classification of Changes**

- non-breaking
- breaking and not resolvable

---

**Erik Burger, Boris Gruschko – A Change Metamodel for the Evolution of MOF-based Metamodels**

26 March 2010
Example

Type2
attr1:int [1..*]

Type1

Change1: ExistenceChange
kind = ADD
affectedElement = Type2

non-breaking

Change2: GeneralizesChange
kind = ADD
superElement = Type2
subElement = Type1

non-breaking

Change3: ContainsChange
kind = DELETE
container = Type1
containedElement = attr1

non-breaking

Change4: ContainsChange
kind = ADD
container = Type2
containedElement = attr1

breaking and not resolvable
**Type2**
attr1:int [1..*]

**Type1**

---

**Change1**: ExistenceChange

- **kind**: ADD
- **affectedElement**: Type2

**Change2**: GeneralizesChange

- **kind**: ADD
- **superElement**: Type2
- **subElement**: Type1

**Change3**: ContainsChange

- **kind**: DELETE
- **container**: Type1
- **containedElement**: attr1

**Change4**: ContainsChange

- **kind**: ADD
- **container**: Type2
- **containedElement**: attr1
OCL Expression

Change3 : ContainsChange

kind = DELETE
container = Type1
containedElement = attr1

Severity Determination Rule for ContainsChange

context ContainsChange
inv: if self.changeSequence.changes -> exists(c|
  c.oclIsTypeOf(ContainsChange) and c.kind = ADD and
  (c.containedElement = self.containedElement or
   c.containedElement.similar(self.containedElement)) and
  self.newSupertypesExtended(self.container) -> contains(c.container))
  -- feature moved to supertype
then self.severity = NONBREAKING
  -- feature deleted or moved elsewhere
else self.severity = BREAKING_RESOLVABLE
endif
**OCL Expression**

---

**Change3 : ContainsChange**

| kind = DELETE  |
| container = Type1 |
| containedElement = attr1 |

---

**Severity Determination Rule for ContainsChange**

**context** ContainsChange

**inv:** if self.changeSequence.changes -> exists(c| c.oclIsTypeOf(ContainsChange) and c.kind = ADD and (c.containedElement = self.containedElement or c.containedElement.similar(self.containedElement)) and self.newSupertypesExtended(self.container) -> contains(c.container))

- feature moved to supertype

**then** self.severity = NON_BREAKING

- feature deleted or moved elsewhere

**else** self.severity = BREAKING_RESOLVABLE

**endif**
Severity Determination Rule for ContainsChange

classification: severity

classification: severity

*context* ContainsChange

*inv: if* self.changeSequence.changes -> exists(c|
  c.oclIsTypeOf(ContainsChange) and c.kind = ADD and
  (c.containedElement = self.containedElement or
   c.containedElement.similar(self.containedElement)) and
  self.newSupertypesExtended(self.container) -> contains(c.container))
  -- feature moved to supertype
  then self.severity = NON_BREAKING
  -- feature deleted or moved elsewhere
else self.severity = BREAKING_RESOLVABLE
endif
Severity Determination Rule for `ContainsChange`

```ocl
class ContainsChange
  context ContainsChange
  inv: if self.changeSequence.changes -> exists(c|
    c.oclIsTypeOf(ContainsChange) and c.kind = ADD and
    (c.containedElement = self.containedElement or
    c.containedElement.similar(self.containedElement)) and
    self.newSupertypesExtended(self.container) -> contains(c.container))
    -- feature moved to supertype
    then self.severity = NON_BREAKING
    -- feature deleted or moved elsewhere
    else self.severity = BREAKING_RESOLVABLE
  endif
```
Severity Determination Rule for ContainsChange

```ocl
classification
context ContainsChange
inv: if self.changeSequence.changes -> exists(c | c.oclIsTypeOf(ContainsChange) and c.kind = ADD and (c.containedElement = self.containedElement or c.containedElement.similar(self.containedElement)) and self.newSupertypesExtended(self.container) -> contains(c.container))
  -- feature moved to supertype
  then self.severity = NON_BREAKING
  -- feature deleted or moved elsewhere
else self.severity = BREAKING_RESOLVABLE
endif
```
Severity Determination Rule for ContainsChange

**context** ContainsChange

**inv:** if self.changeSequence.changes -> exists(c|
  c.oclIsTypeOf(ContainsChange) and c.kind = ADD and
  (c.containedElement = self.containedElement or
   c.containedElement.similar(self.containedElement)) and
  self.newSupertypesExtended(self.container) -> contains(c.container))

  -- feature moved to supertype
  then self.severity = NON_BREAKING

  -- feature deleted or moved elsewhere

else self.severity = BREAKING_RESOLVABLE

endif
Severity Determination Rule for ContainsChange

c\text{context} \text{ContainsChange}
\text{inv: if} \ (\text{self.changeSequence.changes} \to \exists (c| c.\text{oclIsTypeOf(ContainsChange)} \ \text{and} \ c.\text{kind} = \text{ADD} \ \text{and} \ (c.\text{containedElement} = \text{self.containedElement} \ \text{or} \ c.\text{containedElement.similar(self.containedElement)}) \ \text{and} \ \text{self.newSupertypesExtended(self.container)} \to \text{contains(c.container)})
\text{then} \ \text{self.severity} = \text{NON\_BREAKING}
\text{else} \ \text{self.severity} = \text{BREAKING\_RESOLVABLE}
\text{endif}
Conclusion

Summary

- Change Metamodel for MOF 1.4
- formal classification of changes
- automatic determination of change severities

Future Work

- automatic determination of Change Metamodel instance
- semi-automatic resolve of conflicts
Conclusion

Summary

- Change Metamodel for MOF 1.4
- formal classification of changes
- automatic determination of change severities

Future Work

- automatic determination of Change Metamodel instance
- semi-automatic resolve of conflicts
Conclusion

Summary

- Change Metamodel for MOF 1.4
- formal classification of changes
- automatic determination of change severities

Future Work

- automatic determination of Change Metamodel instance
- semi-automatic resolve of conflicts
Conclusion

Summary

- Change Metamodel for MOF 1.4
- formal classification of changes
- automatic determination of change severities

Future Work

- automatic determination of Change Metamodell instance
- semi-automatic resolve of conflicts
Conclusion

Summary
- Change Metamodel for MOF 1.4
- formal classification of changes
- automatic determination of change severities

Future Work
- automatic determination of Change Metamodel instance
- semi-automatic resolve of conflicts
Conclusion

Summary

- Change Metamodel for MOF 1.4
- formal classification of changes
- automatic determination of change severities

Future Work

- automatic determination of Change Metamodel instance
- semi-automatic resolve of conflicts
Helper Function [M-2]: newSupertypes()

context ModelElementChange::newSupertypes(GeneralizableElement element) :
    Set(GeneralizableElement)

post:  result = element.supertypes
-> including(this.changeSequence.changes -> select(ch|
    ch.oclIsTypeOf(GeneralizesChange) and
    ch.subtype = element and ch.kind = ADD) -> collect(supertype))
-> excluding(this.changeSequence.changes -> select(ch|
    ch.oclIsTypeOf(GeneralizesChange) and
    ch.subtype = element and
    ch.kind = DELETE) -> collect(supertype)
)

context ModelElementChange::newSupertypesExtended
    (GeneralizableElement element) : Set(GeneralizableElement)

post:  result = self.newSupertypes(element) ->
    iterate(i:GeneralizableElement; a:Set(GeneralizableElement) = Set{} |
        a -> including(i) -> including(self.newSupertypesExtended(i))))
Helper Function [M-13]: accessorName()

```ts
context Model::Attribute::accessorName() : String
let isPrefix = "is" in
let getPrefix = "get" in
let sname = self.substitutedName() in
post: -- for single-valued boolean types, accessor starts with "is"
  if (self.type = PrimitiveTypes::Boolean and self.multiplicity.upper = 1) then
    if (self.name.substring(1,2) = isPrefix) then
      result = sname
    else
      result = isPrefix.concat(sname.substring(1,1).toUpper()).concat(
          sname.substring(2,sname.size()))
  endif
else -- for others, the accessor name starts with "get"
  result = getPrefix.concat(sname.substring(1,1).toUpper()).concat(
        sname.substring(2,sname.size()))
endif
```

Back to example