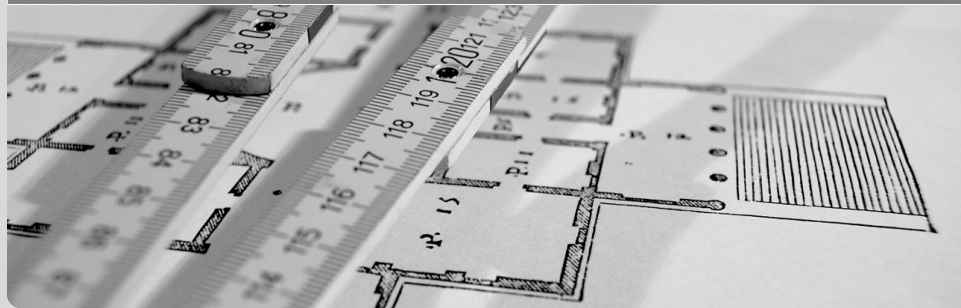


View-based Model-driven Software Development with ModelJoin

Arbeitskreistreffen AK-MDSD, Paderborn, Germany

Erik Burger, Jörg Henß, Steffen Kruse, Martin Küster | 02.07.2012

CHAIR FOR SOFTWARE DESIGN AND QUALITY



- 1 Motivation
- 2 The ModelJoin Approach
- 3 Implementation
- 4 Future Work/Conclusion

Motivation

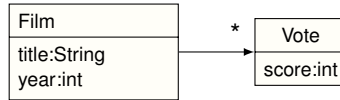
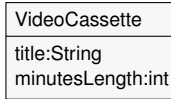


Motivation

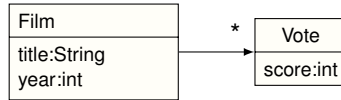
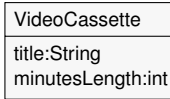


VideoCassette
title:String minutesLength:int

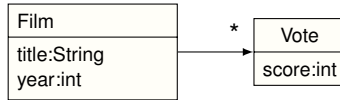
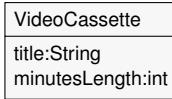
Motivation



Motivation

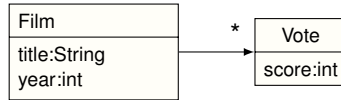
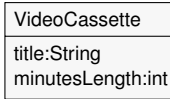


Motivation



Star Trek 1: 6.3
Star Trek 7: 6.5
Star Trek 8: 7.5

Motivation



Star Trek 1: 6.3
Star Trek 7: 6.5
Star Trek 8: 7.5

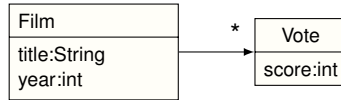
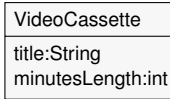


User A



User B

Motivation



Star Trek 1: 6.3
Star Trek 7: 6.5
Star Trek 8: 7.5

“What is the length and year of *Star Trek I*?”

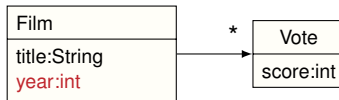
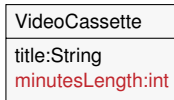


User A



User B

Motivation



Star Trek 1: 6.3
Star Trek 7: 6.5
Star Trek 8: 7.5

“What is the length and year of *Star Trek I*?”

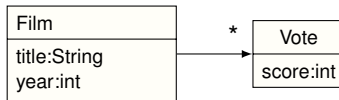
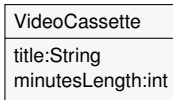


User A



User B

Motivation



Star Trek 1: 6.3
Star Trek 7: 6.5
Star Trek 8: 7.5

“What is the length and year of *Star Trek I*?”



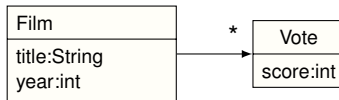
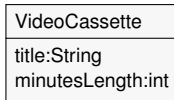
User A



User B

“What are the user votes for *Star Trek* movies in the library?”

Motivation



Star Trek 1: 6.3
Star Trek 7: 6.5
Star Trek 8: 7.5

“What is the length and year of *Star Trek I*?”



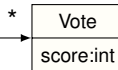
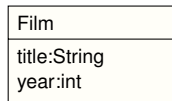
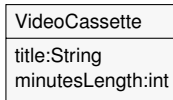
User A



User B

“What are the user votes for *Star Trek* movies in the library?”

Motivation



Star Trek 1: 6.3
Star Trek 7: 6.5
Star Trek 8: 7.5

“What is the length and year of *Star Trek I*?”



User A



User B

“What are the user votes for *Star Trek* movies in the library?”

Metamodel

- information spread across heterogeneous metamodels which are structurally different but share a semantic overlap
- metamodels are only loosely coupled

Instances

- matching of elements necessary
- information need based on properties of instance level

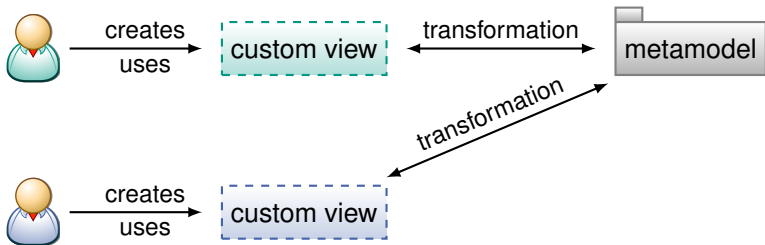
Metamodel

- information spread across heterogeneous metamodels which are structurally different but share a semantic overlap
- metamodels are only loosely coupled

Instances

- matching of elements necessary
- information need based on properties of instance level

- custom *views* fulfill users' information need
- generation of on-demand views
- a view is also a model
- view generation = model transformation



Related Work



VideoCassette
title:String minutesLength:int



Film
title:String year:int

Related Work



VideoCassette
title:String minutesLength:int



Film
title:String year:int

Related Work



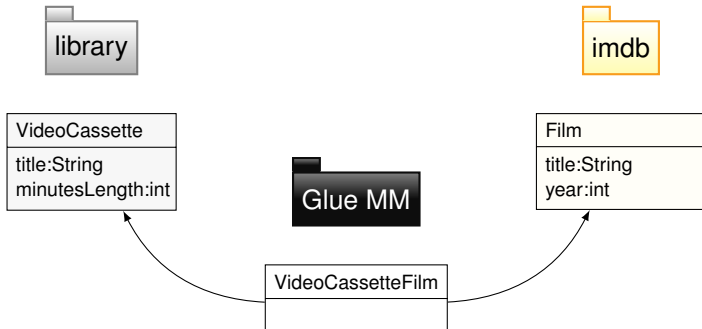
VideoCassette
title:String minutesLength:int

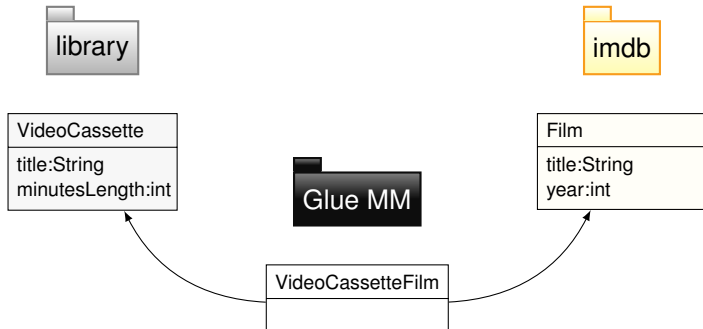


Film
title:String year:int

VideoCassetteFilm

Related Work





- definition of a glue/weaving model (*VirtualEMF, AMW*)
- matching of instances: weaving rules/heuristics (*EMFCompare, Epsilon*)

Related Work



VideoCassette
title:String minutesLength:int



Film
title:String year:int

Related Work



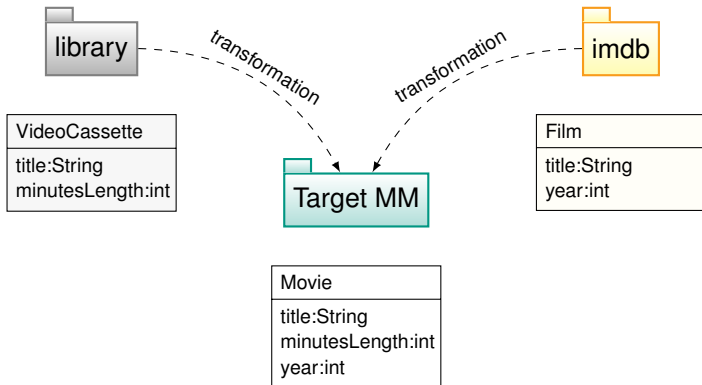
VideoCassette
title:String minutesLength:int

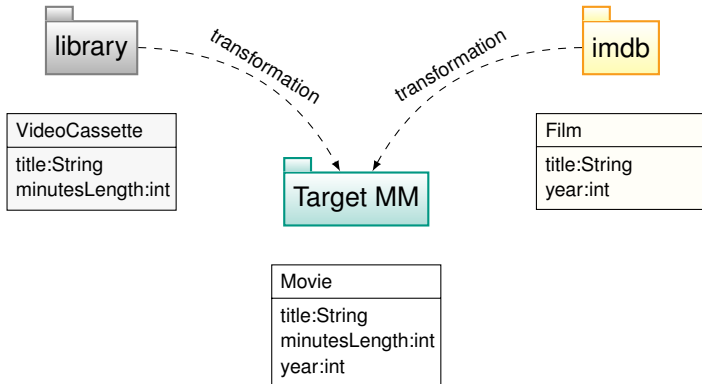


Film
title:String year:int

Movie
title:String minutesLength:int year:int

Related Work





- create target/composition metamodel (or use fixed metamodel)
- define model transformation rules (*ModelBus*, *DuALLY*)

- 1 Motivation
- 2 The ModelJoin Approach
- 3 Implementation
- 4 Future Work/Conclusion

ModelJoin

- DSL with human-readable textual concrete syntax similar to SQL
- joining of instances of heterogeneous metamodels into custom views
- declarative definition of element and feature selection
- rapid definition and execution at “run-time”
- no need to define target metamodel or transformations manually
- non-intrusive approach (source metamodels and instances remain unchanged)

ModelJoin

- DSL with human-readable textual concrete syntax similar to SQL
- joining of instances of heterogeneous metamodels into custom views
- declarative definition of element and feature selection
- rapid definition and execution at “run-time”
- no need to define target metamodel or transformations manually
- non-intrusive approach (source metamodels and instances remain unchanged)

ModelJoin

- DSL with human-readable textual concrete syntax similar to SQL
- joining of instances of heterogeneous metamodels into custom views
- declarative definition of element and feature selection
- rapid definition and execution at “run-time”
- no need to define target metamodel or transformations manually
- non-intrusive approach (source metamodels and instances remain unchanged)

ModelJoin

- DSL with human-readable textual concrete syntax similar to SQL
- joining of instances of heterogeneous metamodels into custom views
- declarative definition of element and feature selection
- rapid definition and execution at “run-time”
- no need to define target metamodel or transformations manually
- non-intrusive approach (source metamodels and instances remain unchanged)

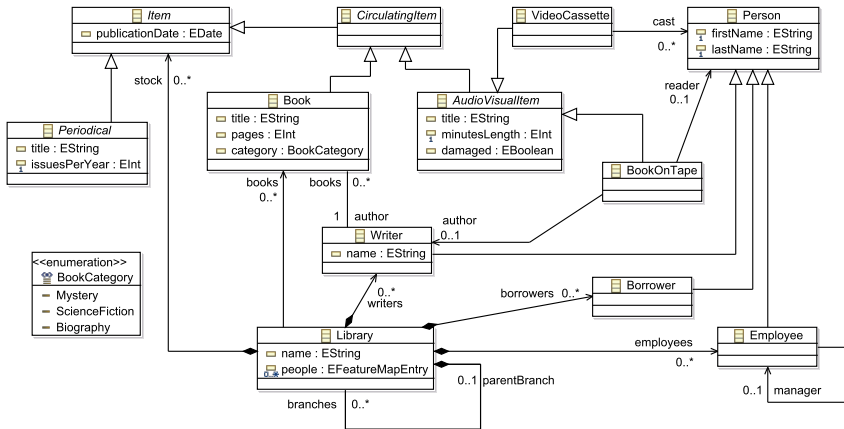
ModelJoin

- DSL with human-readable textual concrete syntax similar to SQL
- joining of instances of heterogeneous metamodels into custom views
- declarative definition of element and feature selection
- rapid definition and execution at “run-time”
- no need to define target metamodel or transformations manually
- non-intrusive approach (source metamodels and instances remain unchanged)

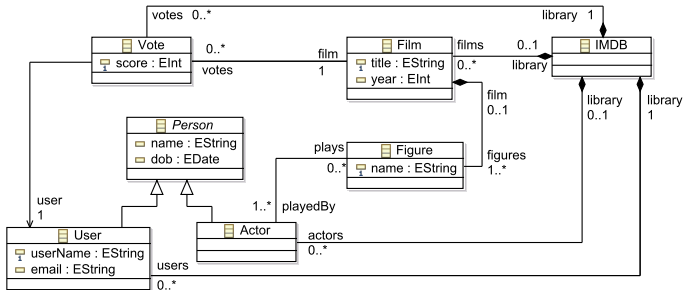
ModelJoin

- DSL with human-readable textual concrete syntax similar to SQL
- joining of instances of heterogeneous metamodels into custom views
- declarative definition of element and feature selection
- rapid definition and execution at “run-time”
- no need to define target metamodel or transformations manually
- non-intrusive approach (source metamodels and instances remain unchanged)

Example: Library Metamodel



Example: IMDB Metamodel



ModelJoin expression example

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
  with keep attributes imdb.Film.year
keep outgoing imdb.Film.votes as type jointarget.Vote
  with keep attributes imdb.Vote.score
keep supertype library.AudioVisualItem as type jointarget.MediaItem
  with keep attributes library.AudioVisualItem.minutesLength
where {{ library.Videocassette.cast->forall (p |
  imdb.Film.figures->playedBy->exists (a | p.firstname.concat("_")
  .concat(p.lastName) == a.name)) }}
```

ModelJoin expression example

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
  with keep attributes imdb.Film.year
keep outgoing imdb.Film.votes as type jointarget.Vote
  with keep attributes imdb.Vote.score
keep supertype library.AudioVisualItem as type jointarget.MediaItem
  with keep attributes library.AudioVisualItem.minutesLength
where {{ library.Videocassette.cast->forall (p |
  imdb.Film.figures->playedBy->exists (a | p.firstname.concat("_")
  .concat(p.lastName) == a.name)) }}
```

ModelJoin expression example

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
  with keep attributes imdb.Film.year
keep outgoing imdb.Film.votes as type jointarget.Vote
  with keep attributes imdb.Vote.score
keep supertype library.AudioVisualItem as type jointarget.MediaItem
  with keep attributes library.AudioVisualItem.minutesLength
where {{ library.Videocassette.cast->forall (p |
  imdb.Film.figures->playedBy->exists (a | p.firstname.concat("_")
  .concat(p.lastName) == a.name)) }}
```

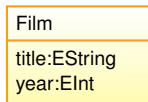
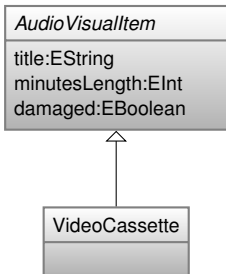
ModelJoin expression example

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
  with keep attributes imdb.Film.year
keep outgoing imdb.Film.votes as type jointarget.Vote
  with keep attributes imdb.Vote.score
keep supertype library.AudioVisualItem as type jointarget.MediaItem
  with keep attributes library.AudioVisualItem.minutesLength
where {{ library.Videocassette.cast->forAll (p |
  imdb.Film.figures->playedBy->exists (a | p.firstname.concat("_")
  .concat(p.lastName) == a.name)) }}
```

- declarative approach
- a ModelJoin expression defines
 - the selection of elements in the target model
 - the layout of the target metamodel
- operators similar to SQL:
 - join
 - keep (Projection)
 - where (Selection)

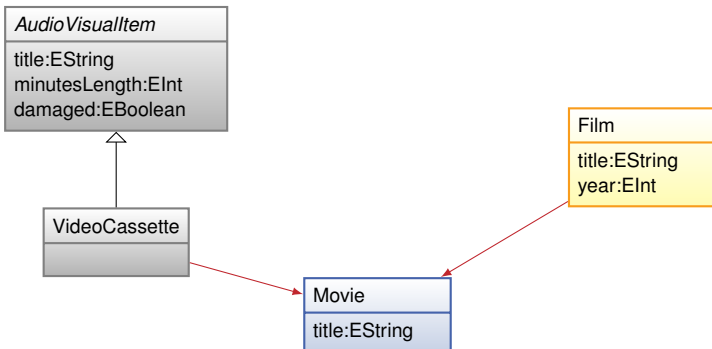
Natural Join

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
```



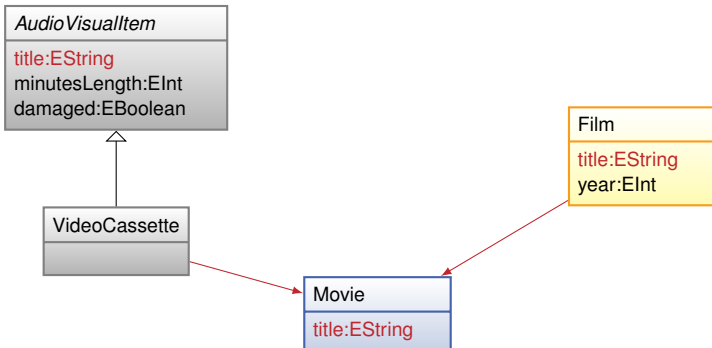
Natural Join

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
```



Natural Join

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
```



Natural Join

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
```

st1:VideoCassette

title = "Star Trek I"
minutesLength = 132

trek1:Film

title = "Star Trek I"
year = 1979

trek7:Film

title = "Star Trek VII"
year = 1994

st8:VideoCassette

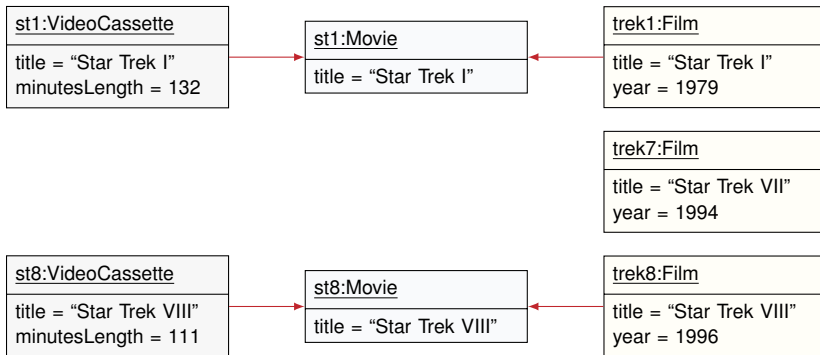
title = "Star Trek VIII"
minutesLength = 111

trek8:Film

title = "Star Trek VIII"
year = 1996

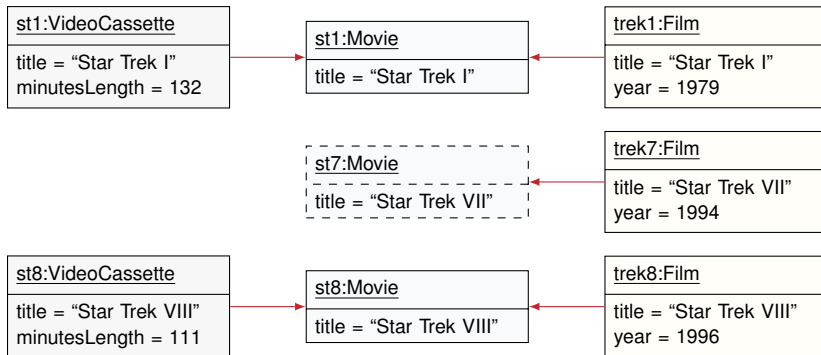
Natural Join

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie
```



Left outer Join

```
left outer join imdb.Film with library.VideoCassette as jointarget.Movie
```



- elements are created in the target metamodel in any case – even if there are no elements that fulfill the join condition
- common attributes (compatible in type and cardinality) are created in the target metamodel and are used for the calculation of the join condition
- element type and name is by default the same as in the left metamodel (rename is possible)
- in contrast to SQL, the join does not degenerate to the cartesian product if there are no common attributes, but delivers an empty result set

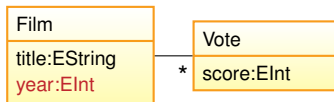
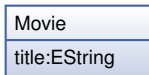
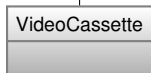
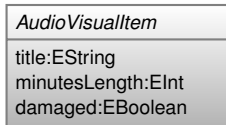
- elements are created in the target metamodel in any case – even if there are no elements that fulfill the join condition
- common attributes (compatible in type and cardinality) are created in the target metamodel and are used for the calculation of the join condition
- element type and name is by default the same as in the left metamodel (rename is possible)
- in contrast to SQL, the join does not degenerate to the cartesian product if there are no common attributes, but delivers an empty result set

- elements are created in the target metamodel in any case – even if there are no elements that fulfill the join condition
- common attributes (compatible in type and cardinality) are created in the target metamodel and are used for the calculation of the join condition
- element type and name is by default the same as in the left metamodel (rename is possible)
- in contrast to SQL, the join does not degenerate to the cartesian product if there are no common attributes, but delivers an empty result set

- elements are created in the target metamodel in any case – even if there are no elements that fulfill the join condition
- common attributes (compatible in type and cardinality) are created in the target metamodel and are used for the calculation of the join condition
- element type and name is by default the same as in the left metamodel (rename is possible)
- in contrast to SQL, the join does not degenerate to the cartesian product if there are no common attributes, but delivers an empty result set

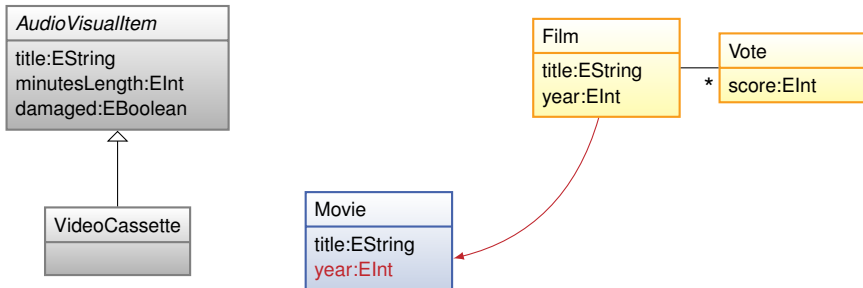
Keep Attributes

with keep attributes imdb.Film.year



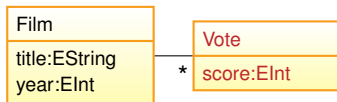
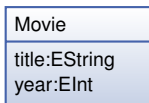
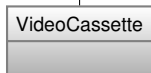
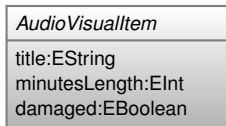
Keep Attributes

with keep attributes imdb.Film.year



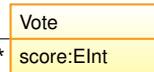
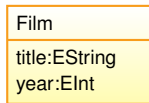
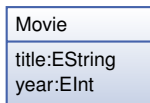
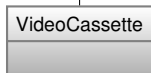
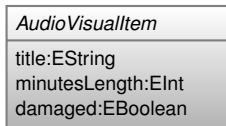
Keep Outgoing

```
keep outgoing imdb.Film.votes as type jointarget.Vote  
with keep attributes imdb.Vote.score
```

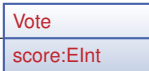


Keep Outgoing

```
keep outgoing imdb.Film.votes as type jointarget.Vote  
with keep attributes imdb.Vote.score
```



*

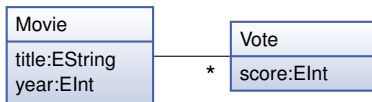
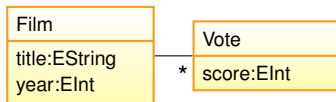
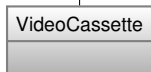
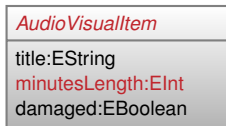


*



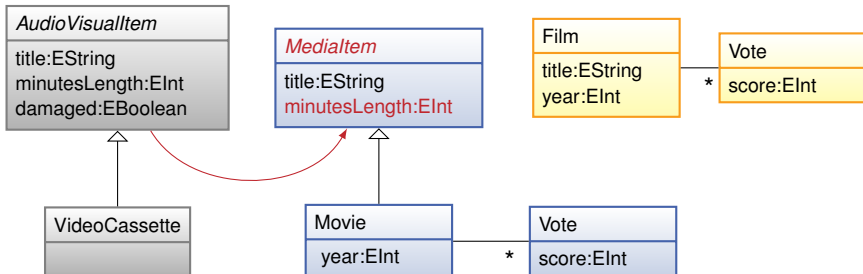
Keep Supertype

```
keep supertype library.AudioVisualItem as type jointarget.MediaItem  
with keep attributes library.AudioVisualItem.minutesLength
```



Keep Supertype

```
keep supertype library.AudioVisualItem as type jointarget.MediaItem  
with keep attributes library.AudioVisualItem.minutesLength
```



- if the feature does not exist in the source metamodel (e.g. in the case of `outer join`), it is set to \perp in the target instances
- for this reason, the lower multiplicity of features is set to 0 if they are created because of a `keep`-statement
- if `keep supertype` is invoked, attributes that are contained in a supertype in the source metamodel are pushed to the parent element automatically
- the `keep outgoing` operator creates a reference in the metamodel and links the joined instances with the respective elements

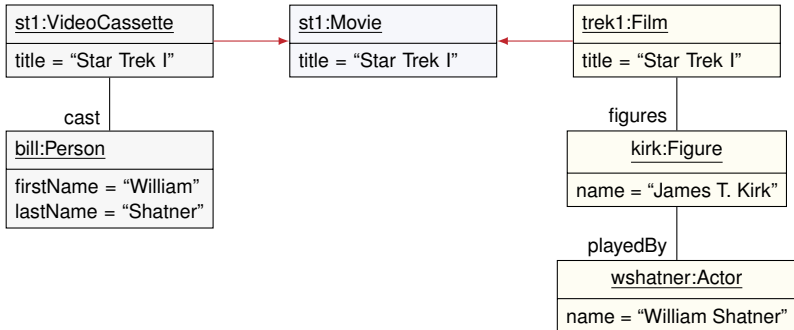
- if the feature does not exist in the source metamodel (e.g. in the case of `outer join`), it is set to \perp in the target instances
- for this reason, the lower multiplicity of features is set to 0 if they are created because of a `keep`-statement
- if `keep supertype` is invoked, attributes that are contained in a supertype in the source metamodel are pushed to the parent element automatically
- the `keep outgoing` operator creates a reference in the metamodel and links the joined instances with the respective elements

- if the feature does not exist in the source metamodel (e.g. in the case of `outer join`), it is set to \perp in the target instances
- for this reason, the lower multiplicity of features is set to 0 if they are created because of a `keep`-statement
- if `keep supertype` is invoked, attributes that are contained in a supertype in the source metamodel are pushed to the parent element automatically
- the `keep outgoing` operator creates a reference in the metamodel and links the joined instances with the respective elements

- if the feature does not exist in the source metamodel (e.g. in the case of `outer join`), it is set to \perp in the target instances
- for this reason, the lower multiplicity of features is set to 0 if they are created because of a `keep`-statement
- if `keep supertype` is invoked, attributes that are contained in a supertype in the source metamodel are pushed to the parent element automatically
- the `keep outgoing` operator creates a reference in the metamodel and links the joined instances with the respective elements

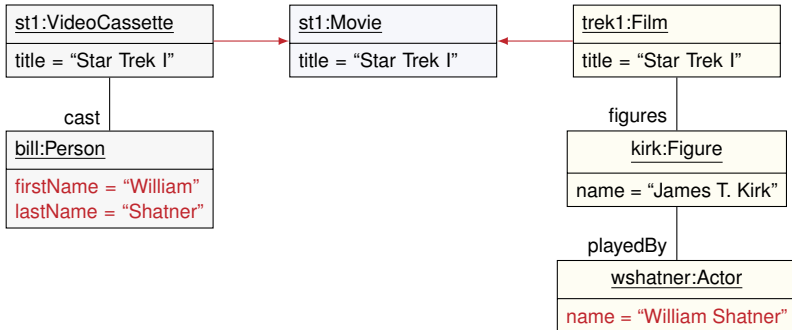
Where

```
where {{ library.Videocassette.cast->forall (p |  
  imdb.Film.figures->playedBy->exists (a | p.firstname.concat("_")  
  .concat(p.lastName) == a.name)) }}
```



Where

```
where {{ library.Videocassette.cast->forall (p |  
  imdb.Film.figures->playedBy->exists (a | p.firstname.concat("_")  
  .concat(p.lastName) == a.name)) }}
```



- OCL operators offer complex join conditions in addition to default join operators
- in conjunction with a *keep*-statement, θ -joins can be emulated
- no impact on the generated metamodel

- OCL operators offer complex join conditions in addition to default join operators
- in conjunction with a keep-statement, θ -joins can be emulated
- no impact on the generated metamodel

- OCL operators offer complex join conditions in addition to default join operators
- in conjunction with a keep-statement, θ -joins can be emulated
- no impact on the generated metamodel

- declarative language for metamodel and model merging
- construction of target metamodel and matching rules combined in one expression
- few automatisms; explicit definition of classes and features target metamodel by the user
- naming conflicts have to be dealt with manually

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- not a query language
- not a transformation language
- DSL for model merging with a focus on view-based development
- not a data integration framework
- no heuristics or automatic matching of similar models
- difference from SQL:
 - SQL: reduction of a base set (cartesian product) via rules
 - ModelJoin: constructive approach

- no joining over references
- un-joinable cases not described yet

- no joining over references
- un-joinable cases not described yet

- 1 Motivation
- 2 The ModelJoin Approach
- 3 Implementation**
- 4 Future Work/Conclusion

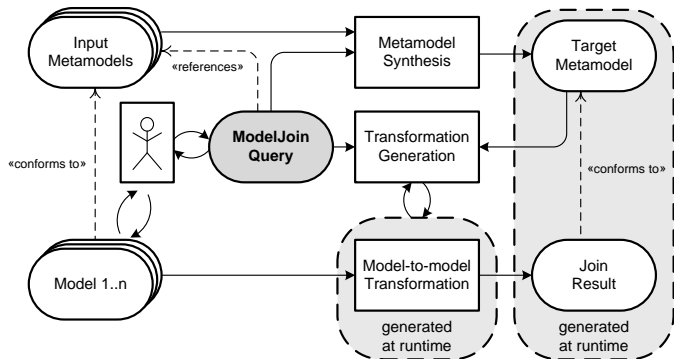


Figure: Model Workflow in FMC notation (Knopf, Grone, and Tabeing 2006)

» Skip details

- *Xtext* textual syntax
- *MWE2* workflow
- metamodel synthesis by Java transformation
 - ① join
 - ② keep supertype
 - ③ keep reference
 - ④ keep attribute
- target metamodel and instances generated at runtime (on save)
- target metamodel does not contain references to source models

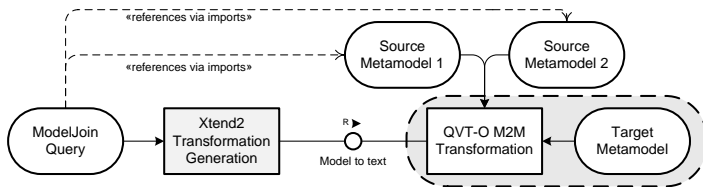


Figure: Transformation generation process

» Skip details

ModelQuery expression

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie  
with keep attributes imdb.Film.year
```

QVT script I

```
leftAll->select(leftElement | rightAll->exists(right |  
  true and right.title = leftElement.title))  
->map JoinOperation(rightAll);
```

QVT script II

```
mapping _imdb::Vote::VoteToVote() : _jointarget::Vote {  
  score := self.score;  
}
```


ModelQuery expression

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie  
with keep attributes imdb.Film.year
```

QVT script I

```
leftAll->select(leftElement | rightAll->exists(right |  
  true and right.title = leftElement.title))  
->map JoinOperation(rightAll);
```

QVT script II

```
mapping _imdb::Vote::VoteToVote() : _jointarget::Vote {  
  score := self.score;  
}
```

ModelQuery expression

```
natural join imdb.Film with library.VideoCassette as jointarget.Movie  
with keep attributes imdb.Film.year
```

QVT script I

```
leftAll->select(leftElement | rightAll->exists(right |  
  true and right.title = leftElement.title))  
->map JoinOperation(rightAll);
```

QVT script II

```
mapping _imdb::Vote::VoteToVote() : _jointarget::Vote {  
  score := self.score;  
}
```

QVT script III

```
mapping _imdb::Film::JoinOperation(right : Set(_library::VideoCassette))
  : _jointarget::VideoCassette {
  title := self.title;
  year := self.year;
  right->forEach(vc) {
    if (vc.title != self.title) then continue endif;
    votes += self.votes->map VoteToVote(); // set the reference "votes"
  };
}
```

- 1 Motivation
- 2 The ModelJoin Approach
- 3 Implementation
- 4 Future Work/Conclusion**

Bidirectionality

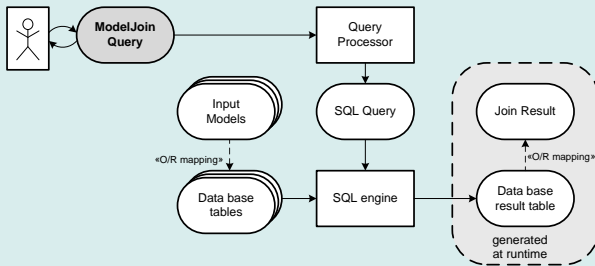
- reverse transformations
- editable views

Fixed target metamodel

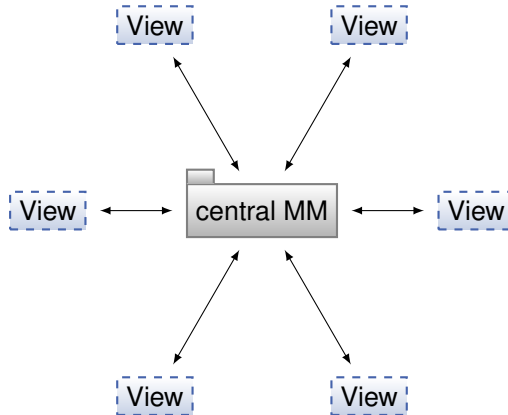
- for recurring join transformations
- necessary to check if ModelJoin expression conforms to target metamodel

Different implementations

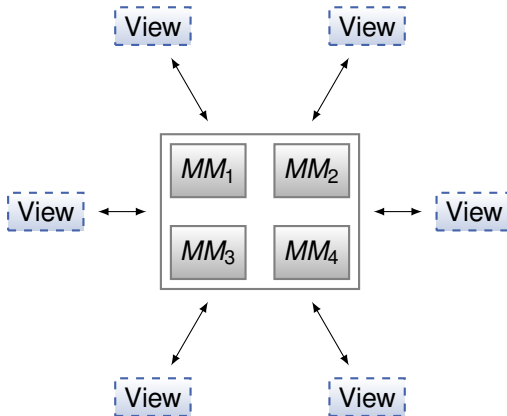
- metamodel generation with QVT (instead of Java)
- using object/relational mapping with Eclipse CDO



View-centric development



View-centric development



ModelJoin DSL

- custom views to answer questions not known during design of metamodels
- non-invasive, declarative approach

Implementation

- Prototypical implementation
- available for download
- `https://svnserver.informatik.kit.edu/i43/svn/code/MDSD`
- user: anonymous, password: anonymous

ModelJoin DSL

- custom views to answer questions not known during design of metamodels
- non-invasive, declarative approach

Implementation

- Prototypical implementation
- available for download
- <https://svnserver.informatik.kit.edu/i43/svn/code/MDSD>
- user: anonymous, password: anonymous

 Andreas Knopfel, Bernhard Grone, and Peter Tabeling.

Fundamental Modeling Concepts: Effective Communication of IT Systems. Wiley, 2006. ISBN: 978-0-470-02710-3.