An Optimistic Three-way Merge Based on a Meta-Model Independent Modularization of Models to Support Concurrent Evolution

2nd Workshop on Model-Driven Software Development (MODSE08)

Christian Bartelt
Agenda

• Team Modeling Environment
• Three-way Merge of Documents
• Modularization of Model Representation
• Synchronization of Concurrent Model Changes – An Example
Team Modeling Environment

What is the context of the approach?

Scenarios to support:

b) Sequential Evolution

c) Concurrent Evolution

d) Management of Variants

Modeling Workbench supports:

b) Local Workspaces

c) Change Logging

d) Unique Identifiable Model Elements (UUIDs)

e) Meta-Model Independency (Universal applicable for several Modeling Languages)

⇒ How can we design an universal configuration management which supports the concurrent development of models? (in consideration to syntax and semantics)
State-of-the-Art / Benefits

- **Tools/Approaches**
  - Model merge and comparison
    - SiDiff (University of Siegen, Germany)
    - Fujaba/Coobra (University of Kassel, Germany)
    - 4everedit (our group)

- **Benefits**
  - A well-founded technique for the description of concurrent, collaborative model development
  - Well-founded semantics of an “intuitive” merge of modeling branches
  - Basis of a comprehensive model consistency management in concurrent development processes
Information Items

What are documents?

- Atomic manageable entities of documents – *information items*
- Documents are represented by sets of information items

**Definition 1 (information item).** An information item \( i \) is atomic information which is unspecified. \( I \) is the set of all information items.

**Definition 2 (document).** A document \( d \subseteq I \) is a finite set of information items. The relationships between information items are not considered knowingly in this definition. \( D = \varnothing(I) \) is the set of all documents.
"Intuitive" Optimistic Merge

Merged Document Instances as Information Sets

A: initial situation

B: changes in V2a/b

6: items added in V2a and V2b

1: items added only in V2a

2: items added only in V2b

3: items deleted only in V2b

4: items deleted only in V2a

5: items deleted in V2a and V2b

C: Merge Result

items untouched in V2a and V2b
**Primitive Changes**

How are documents changed?

---

**Definition 3 (primitive change).** A primitive (atomic) change \( pc \in PC \) describes a change operation on one information item. There are two types of changes: Instances of \( \text{ADD} \subseteq PC \) mean additions and instances of \( \text{DELETE} \subseteq PC \) mean deletions of information items. \( PC \) is the set of all primitive change descriptions.

**Definition 4 (\text{changedItem}).** \text{changedItem}: \( PC \rightarrow I \) is a mapping which assigns each primitive change with its information item.
Revision Tasks

How are documents changed?

- **Revision Tasks** encapsulate **Primitive Changes** and
- **Revision Tasks** with their references **previousRT** and **successive RT** span an acyclic version graph

**Definition 5 (revision task).** A revision task \( rt \in \varnothing(\text{PC}) \) with:

\[
rt \in RT : (pc_1, pc_2 \in rt) \land \text{changedItem}(pc_1) = \text{changedItem}(pc_2) \rightarrow pc_1 = pc_2
\]

represents a further development of a document by processing the consisted primitive changes. \( RT \) is the set of all revision tasks.

**Definition 6 (successiveRT / previousRT).** **successiveRT** and **previousRT** : \( RT \rightarrow \varnothing(\text{RT}) \) are mappings which represent not reflexive, transitive relations with:

\[
rt \in RT : rt_{\text{prev}} \in \text{previousRT}(rt) \rightarrow rt \in \text{successiveRT}(rt_{\text{prev}})
\]

and

\[
rt \not\in \text{previousRT}^+(rt)
\]

The term **successiveRT**\(^+\) means the transitive closure.
**Merge Semantics**

How are concurrent branches synchronized?

**Definition 7** *(changeState)*. \(\text{changeState}: \mathbb{I} \times \mathbb{RT} \rightarrow \{A,D,U,C\}\) is a function with:

\[
\text{changeState}(i,rt) = \begin{cases} 
A, & \exists pc \in rt : \text{changedItem}(pc) = i \land pc \in ADD \\
D, & \exists pc \in rt : \text{changedItem}(pc) = i \land pc \in DELETE \\
\text{mergeState}(i,rt), & \text{else}
\end{cases}
\]

\(A\) – means added, \(D\) – means deleted, \(U\) – means unknown and \(C\) – means in conflict with another primitive change.

**Definition 8** *(mergeState)*. \(\text{mergeState}: \mathbb{I} \times \mathbb{RT} \rightarrow \{A,D,U,C\}\) is a function with:

\[
\text{mergeState}(i,rt) = \begin{cases} 
U, & (\text{previousRT}(rt) = \emptyset) \lor (\forall rtp \in \text{previousRT}(rt) : \text{changeState}(i,rt_p) = U) \\
A, & \text{previousRT}(rt) \neq \emptyset \land \bigcup_{rtp \in \text{previousRT}(rt)} \text{changeState}(i,rt_p) / \{U\} = \{A\} \\
D, & \text{previousRT}(rt) \neq \emptyset \land \bigcup_{rtp \in \text{previousRT}(rt)} \text{changeState}(i,rt_p) / \{U\} = \{D\} \\
C & \text{else}
\end{cases}
\]

**Definition 9** *(documentRevision)*. \(\text{documentRevision} : \mathbb{RT} \rightarrow \mathbb{D}\) is a function which determines the information items of a document after processing all changes of a revision task \(rt\) and its predecessors.

\[
\text{documentRevision}(rt) = \{ \text{changeState}(i,rt) = A \}
\]
Repository Structure – Overview

**package** ModelRepository[ ConcurrentEvolution ]

**Document**
- `documentRevision( rt : RevisionTask ) : Document`

**InformationItem**
- `+changeState( rt : RevisionTask ) : ItemState`
- `+mergeState( rt : RevisionTask ) : ItemState`

**<<enumeration>>**

**Item State**
- A
- D
- U
- C

```
Document

revisionTask 1

InformationItem

previousRT *

successiveRT *

RevisionTask

primitiveChange 1..*

primitiveChange *

PrimitiveChange

Add

Delete

Create

Recreate

changedItem 1
```

```
ADUC

primitiveChange

previousRT *

successiveRT *

RevisionTask

primitiveChange 1..*

primitiveChange *

PrimitiveChange

Add

Delete

Create

Recreate

changedItem 1
```

```
ADUC

primitiveChange

previousRT *

successiveRT *

RevisionTask

primitiveChange 1..*

primitiveChange *

PrimitiveChange

Add

Delete

Create

Recreate

changedItem 1
```
Modularization of Model Representation

How can the introduced technique be applied to models?

- **Modeling items** represent instances of types defining by the meta-model (e.g. UML: Class, Association, Data type)
- **Item properties** represent instantiated properties of the meta-model entities
- **Type information items** represent the relationship between a model element and its type defined in the meta-model

Indirect Referencing with Couplings

- **Couplings** don't represent model information
- **Couplings** are placeholders for ends of links between information items
Example UML - Model Library

- Introduction of a small Example – Library
- Library Model is specified on basis of the UML meta-model

UML Meta Model

Association

Property

name : UML::String

Class

ame : UML::String

<<dataType>>

String

memberEnd

2..*

ownedAttribute

*

RT:A

RT:B

V1

Library

Book

V2a

PrintMedia

V2b

Library

Book

PrintMedia

V2ab

Library

PrintMedia

author

author

Christian Bartelt/ MODSE08
How is the meta model independency guaranteed?

- The meta-model is specified by the MOF (Meta-Meta-Model) language
- The instances of the meta-model elements (MOF::Class, MOF::Property) are referenced by Type Information Items
Repository - Content: Library Example

```
package LibraryRepository {

DataRepresentation

ownedAttribute : MOF::Property

Property : MOF::Class

Association : MOF::Class

Class : MOF::Class

S1 : ModelingItem
  symbol = "Library"
  typeCoupling = C4
  referenceSourceCoupling = C5

S2 : ModelingItem
  symbol = "Book"
  typeCoupling = C19
  referenceSourceCoupling = C18

T1 : TypeInformationItem
  type = Class
  instanceCoupling = C1

T2 : TypeInformationItem
  type = String
  instanceCoupling = C4

T3 : TypeInformationItem
  type = Property
  instanceCoupling = C6

T4 : TypeInformationItem
  type = Association
  instanceCoupling = C8

T5 : TypeInformationItem
  instanceCoupling = C13
  type = Property

T6 : TypeInformationItem
  instanceCoupling = C14
  type = Class

T7 : TypeInformationItem
  instanceCoupling = C19
  type = String

T8 : TypeInformationItem
  instanceCoupling = C15
  type = Property

T9 : TypeInformationItem
  instanceCoupling = C11
  type = Association

P1 : ModelingItem
  modelingItemCoupling = C2
  role = ownedAttribute
  referenceTargetCoupling = C7

IP1 : Item Property
  role = name
  modelingItemCoupling = C2
  referenceTargetCoupling = C7

IP2 : Item Property
  role = name
  modelingItemCoupling = C3
  referenceTargetCoupling = C5

IP3 : Item Property
  role = memberEnd
  modelingItemCoupling = C9
  referenceTargetCoupling = C10

A1 : ModelingItem
  propertyCoupling = C9, C11
  role = memberEnd

C1 : Coupling

C2 : Coupling

C3 : Coupling

C4 : Coupling

C5 : Coupling

C6 : Coupling

C7 : Coupling

C8 : Coupling

C9 : Coupling

C10 : Coupling

C11 : Coupling

C12 : Coupling

C13 : Coupling

C14 : Coupling

C15 : Coupling

C16 : Coupling

C17 : Coupling

C18 : Coupling

C19 : Coupling

C20 : Coupling
```

Christian Bartelt/ MODSE08

15/04/08
**Repository – Content: Change Book Name**

**B : RevisionTask**
- primitiveChange = Add5, Add8, Add4, Add6, Add7, Add3
- previousRT = AB

**A : RevisionTask**
- primitiveChange = Add1, Add2, Del1, Del2
- previousRT = AB

**Del1 : Delete**
- changedItem = S2
- revisionTask = B, A

**Add1 : Add**
- changedItem = S3
- revisionTask = B, A

**Add6 : Add**
- changedItem = S4
- revisionTask = B, A

**Del2 : Delete**
- changedItem = T7
- revisionTask = B, A

**Add2 : Add**
- changedItem = T8
- revisionTask = B, A

**S2 : ModelingItem**
- symbol = "Book"
- referenceSourceCoupling = C18
- typeCoupling = C19
- primitiveChange = Del1

**S3 : ModelingItem**
- symbol = "PrintMedia"
- referenceSourceCoupling = C18
- typeCoupling = C20
- primitiveChange = Add1

**T7 : TypeInformationItem**
- instanceCoupling = C19
- type = String
- primitiveChange = Del2

**String : MOF::Class**

**T8 : TypeInformationItem**
- instanceCoupling = C20
- type = String
- primitiveChange = Add2
Conclusion and Further Work

• What we have seen?
  – Formalization of document evolution focused on synchronizing concurrent development branches
  – Description of an appropriate model representation to manage concurrent evolved models

• What are the next steps?
  – Formalization of the modular model representation concept
  – Implementation/Integration of the approach in the integrated collaboration platform Jazz
  – Comprehensive approach of conflict management
Any questions?

Christian Bartelt

christian.bartelt@tu-clausthal.de
Think about universal conflict detection

How can the consistency of models be preserved?

Identification of causing changes (involved developers) is possible

Validation of syntactical structure is possible
Systems and Development Processes

What are the Problems?

• Characteristics of systems
  – Hard-/Software Systems being more and more \textit{large and complex} (Computer Science, Technical Engineering etc.)
  – SW-Systems being more and more used in \textit{safety critical environments}
  – SW-Systems being more and more \textit{customized}

• Characteristics of development processes
  – \textit{Many developers} of several engineering areas \textit{cooperate} in one team
  – \textit{Formalization} (Syntax/Semantics) of the specifications in the early phases of the software development process

• Methods and tools
  – \textit{Cooperative Development}
    • Distributed, concurrent Development, \textit{software configuration management}
  – \textit{Model based Development}
    • \textit{meta-models for the definition of} \textit{formal syntax (and semantics)}

⇒ \textit{How can we design an universal configuration management which supports the concurrent development of models? (in consideration to syntax and semantics)}