Version Control in MDE

Adrian Rutle\textsuperscript{1}, Alessandro Rossini\textsuperscript{2}, Yngve Lamo\textsuperscript{1}, Uwe Wolter\textsuperscript{2}

\textsuperscript{1}Faculty of Engineering, Bergen University College, Norway
\textsuperscript{2}Department of Informatics, University of Bergen, Norway

20 November 2008
Outline

1. Introduction and Motivation

2. Version Control in MDE
   - Concurrent Development in MDE
   - Common Model
   - Merge Model
   - Synchronisation and Commit
   - Difference and Conflict

3. Summary and Future Work
Model-Driven Engineering

- Model-Driven Engineering (MDE)
  - models as primary artefacts of the software development process
- Models undergo a complex evolution during their life cycles
  - need for techniques and tools to support model evolution activities
- Version control enables concurrent and distributed software development
Version Control Paradigms

- **Lock-Modify-Unlock**
  - lock an artefact
  - modify it
  - release the lock

- **Copy-Modify-Merge**
  - create a local copy of the repository
  - modify the local copy independently
  - detect conflicting changes
  - merge the modifications into the repository
Motivation

- Lock-modify-unlock is workable if the developers always know who is planning to do what
- Traditional copy-modify-merge systems are focused on the management of text-based files
  - techniques based on per-line comparison
  - not suitable for graph-based structures
- Adoption of the copy-modify-merge paradigm enables effective version control in MDE
Motivation

- Solutions proposed by research are not sufficient to enable version control in MDE
  - lack of mature techniques targeting graph-based structures
  - same operations are given different semantics in different works/tools
  - concepts are described only semiformally
  - terminology is not precise and unique
Our contribution

- Formalisation of the copy-modify-merge paradigm in MDE
- Definition of common models and merge models by means of pullback and pushout constructions respectively
- Representation of model differences and detection of syntactic conflicts by using Diagram Predicate Framework (DPF)
Outline

1. Introduction and Motivation

2. Version Control in MDE
   - Concurrent Development in MDE
   - Common Model
   - Merge Model
   - Synchronisation and Commit
   - Difference and Conflict

3. Summary and Future Work
Example

Repository

Student [studies [1]] [Educates] University

Alice’s local copy

Bob’s local copy

Timeline

Repository $V_1$

Alice

Bob

Checkout $V_1$ $V_1^A$ $V_1^A$ $V_1^B$ $V_2^A$ $V_2^B$

Checkout $V_2$ $V_1^C$ $V_2^A$ $V_2^B$

$e_1^A$ $e_2^A$ $e_2^B$
Example

Repository

Alice's local copy

Bob's local copy

Timeline

Repository $V_1$

Alice

Bob

checkout

$V_1$ $V_1^A$ $V_1^A$ $V_2$ $V_1$

$e_1^A$

$e_2^A$

$V_2$ $V_2^A$ $V_2^B$ $V_3$

$V_3^B$

$V_3^A$

$V_2^A_3$
Example

Repository

Student -> University
  studies [1]
  [INV]
  educates

Alice’s local copy

Student -> University
  studies [1]
  [INV]
  educates

PhDStudent

Bob’s local copy

Timeline

Repository

Alice

checkout

V1

V1^A

e1^A

c1^A

V2

V2^A

e2^A

V3

V2^B

Bob

checkout

V2

V2^B

e2^B

V3

V3^A

V2^A

V2^A,3

✓
**Example**

**Repository**

- **Student** → **University**
  - Student is a **PhDStudent**
  - University educates Student

**Alice’s local copy**

- **Student** → **University**
  - Student is a **PhDStudent**
  - University educates Student

**Bob’s local copy**

- **Student** → **University**
  - Student is a **PhDStudent**
  - University educates Student

**Timeline**

- **Repository**
  - **V₁** to **V₂**
    - Alice checks out **e₁₁** from **V₁**
    - Bob checks out **e₂₂** from **V₂**
  - **V₂** to **V₃**
    - Alice checks out **e₂₁** from **V₂**
    - Bob checks out **e₂₂** from **V₂**

- **Alice**
  - **V₁**
    - **e₁₁**
  - **V₂**
    - **e₂₁**
  - **V₃**
    - **e₂₂**

- **Bob**
  - **V₂**
    - **e₂₂**
  - **V₃**
    - **e₂₂**
Example

Repository

Student [isA] PhDStudent

students [1] [INV] educates

University

Alice’s local copy

Student [isA] PhDStudent

studies [1] [INV] educates

University

Bob’s local copy

Student [isA] PhDStudent

studies [1] [INV] educates

University

Timeline

Repository $V_1$

Alice

checkout

$V_1$

$e_1^A$

$V_1^A$

$V_1^A$

$V_1^A$

checkout

$V_1^A$

$e_2^A$

$V_2^A$

$V_2^A$

$V_3$

$V_2^A$

$V_2^A$

$V_2^A$

Bob

$V_2$

$e_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$

$V_2^B$
Example

Repository

**Student**

PhDStudent 

studies [1]

educes [INV]

**University**

Alice’s local copy

**Student**

PhDStudent 

studies [1]

educes [INV]

**University**

Bob’s local copy

**Student**

PhDStudent 

studies [1]

educes [INV]

**University**

**Enrolment**

Type

{Stud, PhDStud, Postdoc}

Timeline

Repository $V_1$

Alice

checkout

$V_1$

$V_1^A$

$V_1^A$

$e_1^A$

$V_2$

$V_1$

$V_1^A$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$

$V_2$
Example

Repository

- **Student**
- **University**
- **Enrolment**

**Type**
{Stud, PhDStud, Postdoc}

Alice’s local copy

- **Student**
- **University**
- **PhDStudent**

Bob’s local copy

- **Student**
- **University**
- **Enrolment**

**Type**
{Stud, PhDStud, Postdoc}

Timeline

Repository $V_1$

Alice

- checkout $V_1$
- $e_1^A$
- checkout $V_1^A$

Bob

- $e_2^B$

$V_2$

$V_2^A$

$V_3$

$V_2^A, V_3$
Example

Repository

```
Student

Enrolment

Type
{Stud, PhDStud, Postdoc}

University
```

Alice’s local copy

```
Student

Enrolment

Type
{Stud, PhDStud, Postdoc}

PhDStudent

Project

University
```

Bob’s local copy

```
Student

Enrolment

Type
{Stud, PhDStud, Postdoc}

University
```

Timeline

```
Repository

V_1

checkout

V_1

V_1^A

V_1^A

Alice

V_2

checkout

V_2

V_2^A

V_2^A

V_3

V_3

V_3^A

V_3^A

Bob

V_2

V_2^B

V_2^B

V_3

V_3^B
```

HØGSKOLEN I BERGEN
Outline

1. Introduction and Motivation

2. Version Control in MDE
   - Concurrent Development in MDE
   - Common Model
   - Merge Model
   - Synchronisation and Commit
   - Difference and Conflict

3. Summary and Future Work
Common Model

- The development environment must keep track over what is common between subsequent models.

- Existing approaches to the identification of mappings/differences:
  - *soft-linking*: structurally equivalent elements imply equal elements.
  - *hard-linking*: elements with equal identifiers are seen as equal elements.
Common Model

- We tackle the identification of mappings by specifying them explicitly in common models
- “Recording” which elements are kept unmodified addresses the problems of the soft- and hard-linking approaches

Definition
A model $C$ together with the injective morphism $\text{inj}$ and the inclusion morphism $\text{inc}$ is a common model for $V_i$ and $V_j$. 

\[
\begin{array}{ccc}
V_i & \xrightarrow{\text{inj}} & C & \xleftarrow{\text{inc}} & V_j
\end{array}
\]
Example
Composition of Common Models

- Common model of non-subsequent models can be obtained by the composition of the “intermediate” common models

Definition

The *composition of commons* $C_{i,k}$ is a special pullback $(C_i,k, \text{inj}_{C_i,j}, \text{inc}_{C_j,k})$ of the diagram $C_i,j \xrightarrow{\text{inc}_{V_j}} V_j \leftarrow^{\text{inj}_{V_j}} C_j,k$ such that $\text{inc}_{C_j,k}$ is an inclusion.
Outline

1. Introduction and Motivation

2. Version Control in MDE
   - Concurrent Development in MDE
   - Common Model
   - Merge Model
   - Synchronisation and Commit
   - Difference and Conflict

3. Summary and Future Work
Merge Model

The merge model must contain the information which is needed to distinguish which model elements come from which “source” model.

This is exactly one of the properties of pushout.

Definition

The merge model $V_{i,j}$ is the pushout $(V_{i,j}, m_i, m_j)$ of the diagram $V_i \xleftarrow{inj_{V_i}} C_{i,j} \xrightarrow{inc_{V_j}} V_j$ such that $m_j$ is an inclusion.
Example
Outline

1. Introduction and Motivation

2. Version Control in MDE
   - Concurrent Development in MDE
   - Common Model
   - Merge Model
   - Synchronisation and Commit
   - Difference and Conflict

3. Summary and Future Work
Synchronisation and Commit

- A *synchronisation* is a procedure which generates a synchronised local copy $V_j U$, where $V_j$ is the last version in the repository
  - conflicting changes are detected
- A *commit* is an operation which adds the model $V_j U$ to the repository and labels it $V_{j+1}$
Synchronisation Procedure

\[ V_{i,U} \rightarrow \begin{array}{c} C_{i,i} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{inc}_{V_{i,U}} \rightarrow \vspace{0.5cm} \end{array} V_{i,U} \rightarrow \begin{array}{c} \text{inj}_{V_{i}} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{e}_{i} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{mu}_{i} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{P.O.} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{P.O.} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{mr}_{i} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{mr}_{j} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{P.O.} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{mr}_{i,j} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{c}_{i} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{V}_{i,U,j} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{V}_{j,U} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{V}_{j+1} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{V}_{i,U} \rightarrow \vspace{0.5cm} \end{array} \]

\[ \begin{array}{c} \text{V}_{i,U} \rightarrow \vspace{0.5cm} \end{array} \]
Outline

1  Introduction and Motivation

2  Version Control in MDE
   - Concurrent Development in MDE
   - Common Model
   - Merge Model
   - Synchronisation and Commit
   - Difference and Conflict

3  Summary and Future Work
Difference

- Need to identify the differences between two models
- In a merge model $V_{iU,j}$ we can distinguish common elements, $V_iU$-elements and $V_j$-elements from each other
- We can identify which elements are common, added, deleted, renamed and moved
- We define a signature $\Sigma_\Delta$
  - predicates: [common], [add], [delete], [rename], [move], and [conflict]
Conflict

- The merge models will be decorated by predicates from the signature $\Sigma_\Delta$
  - detection of conflicts in the decorated merge model $V_{iU,j}$
  - reduction of $V_{iU,j}$ to $V_{jU}$
- Rules:
  - $[\text{common}] \land [\text{common}] \rightarrow [\text{common}]$
  - $[\text{delete}] \land [\text{delete}] \rightarrow [\text{delete}]$
  - $[\text{common}] \land [\text{delete}] \rightarrow [\text{delete}]$
  - $[\text{move}] \land [\text{move}] \rightarrow [\text{conflict}]$
  - $[\text{add}] (\text{arrow}) \land [\text{delete}] (\text{node}) \rightarrow [\text{conflict}]$
  - $[\text{rename}] \land [\text{rename}] \rightarrow [\text{conflict}]$
Example
Outline

1. Introduction and Motivation

2. Version Control in MDE
   - Concurrent Development in MDE
   - Common Model
   - Merge Model
   - Synchronisation and Commit
   - Difference and Conflict

3. Summary and Future Work
Summary

- The copy-modify-merge paradigm can be exploited for version control of graph-based structures
- Formalisation of the concepts of the copy-modify-merge paradigm in MDE by means of category-theoretical constructions
- Explicit specification of model elements’ mappings in common models
- Diagrammatic language for the representation of model differences
- Rules for the detection of syntactic conflicts
Future Work

- Semantic conflicts
- Prototype implementation
Thank you!

Questions?