Diagram Predicate Framework
– A Formal Approach to MDE

Adrian Rutle

Faculty of Engineering, Bergen University College, Norway

29 November 2010
Vilvite, Bergen, Norway
Outline

Introduction

Metamodelling

Model Transformation

Version Control

Summary
• In general: not the subject of this presentation ;-(
• In engineering disciplines
  • Used for calculations, tests, cost estimation, etc.
• In engineering disciplines
  • Used for calculations, tests, cost estimation, etc.
  • ... before building the real “system”
Modelling ...

- In engineering disciplines
  - Used for calculations, tests, cost estimation, etc.
  - ... before building the real “system”
  - Aim: avoid finding errors the hard way!
Modelling in Software Engineering

- Means to understand the domain
- Abstract representation of software systems
- Usually diagrammatic (graph based structures)
Modelling in Software Engineering

- Means to understand the domain
- Abstract representation of software systems
- Usually diagrammatic (graph based structures)
  - We first learnt to draw
Modelling in Software Engineering

- Means to understand the domain
- Abstract representation of software systems
- Usually diagrammatic (graph based structures)
  - “Things” and “Relations” between them: Graphs

```
Employee * employees 1..* departments Department
```
Modelling in Software Engineering

• Means to understand the domain
• Abstract representation of software systems
• Usually diagrammatic (graph based structures)
  • “Things” and “Relations” between them: Graphs
• Guideline for software design and implementation
• Usually out-of-synch when development proceeds
  • New, compared to other engineering disciplines

![Diagram](image-url)
Modelling in Software Engineering

- Model Driven Engineering (MDE)
  - Reasoning at a higher level of abstraction
  - Models as primary artefacts of software development process
  - Generation of software by means of model transformations
-⇒ Bringing modelling in software engineering to the same level as modelling in other engineering disciplines
Modelling in Software Engineering

- This thesis:
  - One step in the “right” direction
Modelling in Software Engineering

• This thesis:
  • One step in the “right” direction
  • *One small step for a computer scientist, one giant leap for computer science*
MDE Standards: State-of-the-art

- Modelling languages
  - Unified Modeling Language (UML)
  - Eclipse Modeling Framework (EMF)
  Usually graph-based languages

- Constraint languages
  - Object Constraint Language (OCL)
  Usually text-based languages
MDE Standards: State-of-the-art

- Modelling languages
  - Unified Modeling Language (UML)
  - Eclipse Modeling Framework (EMF)
  Usually graph-based languages
- Constraint languages
  - Object Constraint Language (OCL)
  Usually text-based languages
- Standardisation: Object Management Group (OMG)
Challenges

- Shift of paradigm: code-centric to model-centric
- Formal foundation?
  - Semantics of models
  - Metamodelling
  - Conformance relation
  - Model transformation
  - Model management
- Tool support?
  - Mature tools and frameworks
  - Reliable standards
- Well-understood methodologies?
- Well-proved practises?
- etc.
Contribution

- Formalisation of conformance
  - Formalisation of metamodelling hierarchy
  - Integration of constraints in (meta)modelling
- Constraint-aware model transformation
  - Source constraints transformed to target constraints
  - Constraints used to control model transformation
- Version control of models
  - Model merging
  - Resolution patterns
- Tool design
Outline

Introduction

Metamodelling

Model Transformation

Version Control

Summary
Modelling approach

Sample model

- Model: abstract representation of a software system
Modelling approach

Sample model

- Req 1: “a person must be child of exactly two persons”
Modelling approach

- Model: specified by a modelling language
Modelling approach

- Modelling language: corresponding metamodel + semi-formal semantics
Modelling approach

- Req 2: “no person is his/her own child”
Modelling approach

- Attached constraint: specified by means of text-based OCL expressions

Sample model
Modelling approach

- Diagram Predicate Framework (DPF)
Modelling approach

- Specification: graph + set of constraints

Sample model

- Specification: graph + set of constraints
Modelling approach

- Specification: graph + set of constraints
Modelling approach

- Specification: specified by a modelling formalism
Modelling approach

- Modelling formalism: meta-specification + ...

Sample model

UML

- Class
- Property
- Association

OCL

- Person

OMG DPF

- Structual constraints: Person
  - inv: Irreflexive
  - self.ChildOf->excluding(self)

OMG DPF

- Class
- Reference

- Person
Modelling approach

- Modelling formalism: meta-specification + signature
Modelling approach

- Integration of constraints
Modelling Formalism

Example

<table>
<thead>
<tr>
<th>p</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>(mult(n,m)]</td>
<td>∀x ∈ X : n ≤</td>
</tr>
<tr>
<td>[irreflexive]</td>
<td>∀x ∈ X : x ∉ f(x)</td>
</tr>
</tbody>
</table>
Modelling Formalism

Example

```
Modelling Formalism

Σ₂

S₂

C

S₁

S₁

Class

Reference

Person

Diagrammatic signature

<table>
<thead>
<tr>
<th>p</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[mult(n,m)]</td>
<td>∀x ∈ X : n ≤</td>
</tr>
<tr>
<td>[irreflexive]</td>
<td>∀x ∈ X : x ∉ f(x)</td>
</tr>
</tbody>
</table>
```

childOf
Modelling Formalism

Example

Diagrammatic signature

<table>
<thead>
<tr>
<th>p</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\forall x \in X : n \leq</td>
<td>f(x)</td>
</tr>
<tr>
<td>$\forall x \in X : x \notin f(x)$</td>
<td>$[\text{irreflexive}]$</td>
</tr>
</tbody>
</table>
Outline

Introduction

Metamodelling

Model Transformation

Version Control

Summary
MDE and Model Transformation

- Generation of target models from source models
- Model Transformation (MT) plays a central role in MDE
- Application areas in MDE:
  - Development process: code generation, refinement etc
  - Model management: integration, decomposition etc
  - Migration: from a platform, an implementation technology etc, to another
  - Language translation: from a modelling language to another
Introduction

Model transformation in MDE
Introduction

Model transformation in MDE
Introduction

Model transformation in MDE

DPF

- Integration of constraints in (meta)modelling
- Constraint-aware model transformation
  - Source constraints transformed to target constraints
  - Constraints used to control
    - which structure to create in the target model
    - which constraints to add to the created structure
Modelling approach

- Recall models together with attached constraints
Modelling approach

- Transformation rules: defined over the metamodel
Modelling approach

- Challenge: transformation of attached constraints
Modelling approach

- Diagram Predicate Framework (DPF)

Sample model

- Diagram Predicate Framework (DPF)
- **Transformation rules:** defined over the modelling formalism
Model Transformation Approach

- Given source and target modelling formalisms and a source model ...
Model Transformation Approach

- ... we want to generate a target model
Model Transformation Approach

- How we get the target model from the source model?
Model Transformation Approach

- We have to relate the modelling formalisms.
First: Define morphisms to an appropriate joined modelling formalism
• Second: Define constraint-aware transformation rules
Model Transformation Approach

- Third: Apply the model transformation
Third: Apply the model transformation
  • A: Convert the source model to an intermediate model
Model Transformation Approach

- Third: Apply the model transformation
  - B: Apply the transformation rules (Pushout)
Model Transformation Approach

- Third: Apply the model transformation
  - C: Project out the target model (Pullback)
Model Transformation Approach

- Heterogeneous, out-place model transformation
Step 1: Joining Modelling Formalisms

- Source and target modelling formalisms
Step 1: Joining Modelling Formalisms

- Joined metamodel \( \mathcal{J}_2 := \mathcal{S}_2 \cup \mathcal{K}_2 \cup \mathcal{T}_2 \)
Step 1: Joining Modelling Formalisms

- Joined metamodel $\mathcal{J}_2 := \mathcal{G}_2 \cup \mathcal{K}_2 \cup \mathcal{T}_2$
Step 1: Joining Modelling Formalisms

- Signature $\Xi_2$ used for constraining $\mathcal{J}_2$
Step 1: Joining Modelling Formalisms

- Source and target modelling formalisms

Modelling Formalisms
Step 1: Joining Modelling Formalisms

- Joined modelling formalism \((Γ_2, J_2, Γ_3)\)
Step 1: Joining Modelling Formalisms

- For any specification $\hat{J}_1$ specified by the joined modelling formalism
Step 1: Joining Modelling Formalisms

- Possible to project out a source specification $S_1$...
Step 1: Joining Modelling Formalisms

- Possible to project out a source specification $S_1$ and a target specification $T_1$
Step 2: Define Constraint-Aware Rules

Source Modelling Formalism \(\xrightarrow{\text{join}}\) Joined Modelling Formalism \(\xrightarrow{\text{join}}\) Target Modelling Formalism

Source Model \(\xrightarrow{\text{specified by}}\) Constraint-aware Rules \(\xrightarrow{\text{specified by}}\) Target Model
Step 2: Define Constraint-Aware Rules

- Rule $r_1$. Class to table and primary key

<table>
<thead>
<tr>
<th>$\mathcal{L}$</th>
<th>$\mathcal{R}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:Class</td>
<td>1:Class $\rightarrow$ 1:Table $\xrightarrow{[\text{pk}]}$ Int:DT $^t$</td>
</tr>
</tbody>
</table>
Step 2: Define Constraint-Aware Rules

- Rule $r_2$. Attribute to column
Step 2: Define Constraint-Aware Rules

- Rule $r_3$. Many-to-one references to foreign key
Step 2: Define Constraint-Aware Rules

- Rule $r_4$. Many-to-many references to link table and foreign key

<table>
<thead>
<tr>
<th>L</th>
<th>R</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="image1.png" alt="Diagram L" /></td>
<td><img src="image2.png" alt="Diagram R" /></td>
</tr>
</tbody>
</table>
Step 3: Applying Model Transformation

- A: Type conversion of source model
A: Type Conversion of Source Model

Class

Data Type

Employee

Department

Project

S₂

S₁

Reference

Attribute

empDepts [1..∞]

[surj]
depEmps

[inv]

[comp]

proEmps' [1..∞]

proEmps

proDeps
A: Type Conversion of Source Model
Step 3: Applying Model Transformation

- B: Applying transformation rules

Diagram:
- Source Modelling Formalism
- Source Model
- Intermediate Model
- Joined Modelling Formalism
- Target Modelling Formalism
- Target Model

Relationships:
- Join: Source Modelling Formalism → Joined Modelling Formalism
- Join: Joined Modelling Formalism → Target Modelling Formalism
- Conversion: Source Model → Intermediate Model
- Specified by: Source Modelling Formalism → Constraint-aware Rules
- Specified by: Joined Modelling Formalism → Intermediate Model
- Specified by: Target Modelling Formalism → Target Model
- Iterative rule application: Intermediate Model → Intermediate Model
B: Applying the Rules (after $r_5$)

Blue: Matched, Green: Added
B: Applying the Rules (after $r_6$)

Blue: Matched, Green: Added
Applied $r_1$ three times, $r_5$ three times, $r_6$ once, and $r_7$ once
Step 3: Applying Model Transformation

- C: Projection of target model

Source Modelling Formalism \(\rightarrow\) Source Model

\[\text{conversion} \rightarrow \text{Intermediate Model} \]

\[\text{constraint-aware Rules} \rightarrow \text{Joined Modelling Formalism} \]

\[\text{specification by} \rightarrow \text{Target Modelling Formalism} \]

\[\text{join} \rightarrow \text{Target Model} \]

\[\text{projection} \rightarrow \text{Target Model} \]

\[\text{iterative rule application} \rightarrow \text{intermediate model} \]
C: Projection

introduction

model transformation

version control

summary
C: Projection

Introduction Metamodelling Model Transformation Version Control Summary

Class --> Table
Reference
Attribute
Data Type
Table
Column
Data Type
Table
Column
Data Type

J2
J1

TEmployee
Int
TEmpDep
Int
TDepartment

TProEmp
Int
TProDep
Int

TProEmp'

Int
TProject

Data Type
Data Type
Data Type

[1..1][inv]
[=]
[1..∞]
[1..∞]

[1..1][inv]
[=]
[1..∞]
[1..∞]

Int
Int
Int

[pk]
[fk]
[fp]
[fp]
[pk]
[fk]
[fk]
[fk]
[pk]

[inj]
[ie]
[ji]
[ji]
[bfj]
[bfj]
[bfj]
[bfj]
[bfj]
[bfj]
[bfj]

[1..1]
[rcomp]
[bij]
[bij]
Controlling Rule Application

Layers and NACs

\[ l_0 \quad r_1 \quad r_2 \]

\[ r_3 \quad r_4 \quad r_5 \quad r_6 \]

\[ l_1 \]

\[ l_2 \quad r_7 \]

Only Layers
Controlling Rule Application

Layers and NACs

\[
\begin{array}{ccc}
l_0 & r_1 & r_2 \\
\hspace{1cm} & \hspace{1cm} & \\
r_3 & r_4 & r_5 \\
\hspace{1cm} & \hspace{1cm} & \\
l_1 & r_6 & \\
\hspace{1cm} & \hspace{1cm} & \\
l_2 & r_7 & \\
\end{array}
\]

Only Layers

\[
\begin{array}{ccc}
l_0 & r_1 & r_2 \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
l_1 & r_3 & r_6 \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
l_3 & r_4 & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
l_4 & r_5 & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
l_5 & r_7 & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
\hspace{1cm} & \hspace{1cm} & \\
\end{array}
\]
Outline

Introduction

Metamodelling

Model Transformation

Version Control

Summary
Version Control in MDE

- Complex evolution of models
  - Need for techniques and tools to support model evolution
- Version control (copy-modify-merge)
  - A local copy of artefacts for each developer
  - Independent and parallel modifications
  - Merging of modifications
  - Detection of conflicts
Outline

Introduction

Metamodelling

Model Transformation

Version Control

Summary
Summary

- Formalisation of conformance relation
- Diagrammatic formalisation of (meta)modelling hierarchy
- Integration of constraints in (meta)modelling
- Constraint-aware model transformation
  - Source constraints transformed to target constraints
  - Constraints used to control
    - which structure to create in the target model
    - which constraints to add to the created structure
- Version control of models
  - Model merging
  - Conflict detection
Related Work

• **Generalized Sketches (GS)** *Zinovy Diskin et al*
  - Software engineer friendly predicates
  - Metamodelling hierarchy, fibred semantics, specification entailments, transformation, etc.

• **Graph Transformation (GT)** *Hartmut Ehrig et al*
  - Model: graph + diagrammatic constraints
  - Conformance: typing + satisfaction of constraints
  - Transformation: also considering diagrammatic constraints

• **Triple Graph Grammar (TGG)** *Schürr et al*
  - Diagrammatic constraints in the joined modelling formalism
Future Work

- Complete comparison to OCL
- Logic for relations between atomic constraints (and universal constraints)
  - Deduction calculus
- Model transformation system: termination and functional properties
- Constraint-aware version control
- DPF Tool support
  - Comparison to other tools: AGG, GReAT, PROGRESS, VMTS, Epsilon, Fujaba
  - Real-size case studies: to prove usefulness of DPF
Thank you!

Questions?
Formalisation approach

- Based on category theory
  - Sketches formalism: define semantics of diagrams (thus graph-based models)
    - models: graphs (nodes and edges)
    - model properties: universal properties (limit, colimit, commutative diagrams)
  - Generalized sketches formalism
    - not only universal properties
    - user-defined diagrammatic predicate signatures
- DPF: specification formalism based on generalized sketches
### Sample Signature for Relational Data Models

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\alpha^2(p)$</th>
<th>Proposed vis.</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[primary-key]</td>
<td>(f) 1 (\rightarrow) 2 ([pk]) (X) (\rightarrow) (Y)</td>
<td>$f$ is [total] and [injective]</td>
<td></td>
</tr>
<tr>
<td>[foreign-key]</td>
<td>(f) 1 (\rightarrow) 2 (g) 3 ([fk]) (X) (\rightarrow) (Y) (\rightarrow) (Z)</td>
<td>$f(X) \subseteq g(Y)$</td>
<td></td>
</tr>
<tr>
<td>[image-equal]</td>
<td>(f) 1 (\rightarrow) 2 (g) 3 ([ie]) (X) (\rightarrow) (Y) (\rightarrow) (Z)</td>
<td>$f(X) = g(Z)$</td>
<td></td>
</tr>
</tbody>
</table>

---

**Proposed vis.**
- **X**: A node representing a class or entity.
- **f**: A function or mapping from one node to another.
- **Y**: Another node indicating the target of the mapping.
- **Z**: A further node showing the relationship or constraint specified by the foreign key.
- **[pk]**: The primary key attribute.
- **[fk]**: The foreign key attribute.
- **[ie]**: Image equality constraint.
Challenges in modelling

- Mixing graph-based structures with textual constraints
  - Different technical spaces
    - checking models in two different engines/steps
    - model-constraint synchronisation problem
    - violation of “everything-is-a-model” vision of MDE
  - Challenge for domain experts who do not understand OCL
Sample Object-Oriented Modelling Hierarchy

- Modelling formalism: meta-specification + ...
Sample Object-Oriented Modelling Hierarchy

- Modelling formalism: meta-specification + signature
Sample Object-Oriented Modelling Hierarchy

- **Req 1**: “an employee must work for at least one department”

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\alpha^{\Sigma_2}(p)$</th>
<th>Proposed vis.</th>
<th>Intended semantics</th>
</tr>
</thead>
</table>
| $\{\text{mult}(m,n)\}$ | $1 \xrightarrow{a} 2$ | $\begin{array}{c} x \\
\xrightarrow{f_{\frac{m}{n}}} \\
Y \end{array}$ | $\forall x \in X : m \leq |f(x)| \leq n$, with $0 \leq m \leq n$ and $n \geq 1$ |
| $\{\text{surjective}\}$ | $1 \xrightarrow{a} 2$ | $\begin{array}{c} x \\
\xrightarrow{f_{\text{surj}}} \\
Y \end{array}$ | $f(X) = Y$ |
| $\{\text{inverse}\}$ | $1 \xrightarrow{a} 2$ | $\begin{array}{c} X \\
\xrightarrow{\text{inv}} \\
Y \end{array}$ | $\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$ |
| $\{\text{image-inclusion}\}$ | $1 \xrightarrow{a} 2$ | $\begin{array}{c} X \\
\xrightarrow{\subseteq} \\
Y \end{array}$ | $\forall x \in X : f(x) \subseteq g(x)$ |
| $\{\text{composition}\}$ | $1 \xrightarrow{a} 2$ | $\begin{array}{c} X \\
\xrightarrow{h_{\text{comp}}} \\
Y \end{array}$ | $\forall x \in X : h(x) = \{g(y) \mid y \in f(x)\}$ |
### Sample Object-Oriented Modelling Hierarchy

- **Req 2:** “a department may have none or many employees”

<table>
<thead>
<tr>
<th>Relationship</th>
<th>Mathematical Formulation</th>
<th>Proposed visual</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>[surjective]</td>
<td>( \forall x \in X : m \leq</td>
<td>f(x)</td>
<td>\leq n ) with ( 0 \leq m \leq n ) and ( n \geq 1 )</td>
</tr>
<tr>
<td>[inverse]</td>
<td>( \forall x \in X, \forall y \in Y : y \in f(x) \iff x \in g(y) )</td>
<td>( X \xrightarrow{\text{inv}} Y )</td>
<td></td>
</tr>
<tr>
<td>[image-inclusion]</td>
<td>( \forall x \in X : f(x) \subseteq g(x) )</td>
<td>( X \xrightarrow{\subseteq} Y )</td>
<td></td>
</tr>
<tr>
<td>[composition]</td>
<td>( \forall x \in X : h(x) = { g(y) \mid y \in f(x) } )</td>
<td>( X \xrightarrow{\text{comp}} Y )</td>
<td></td>
</tr>
</tbody>
</table>
Sample Object-Oriented Modelling Hierarchy

- **Req 3:** “a project may involve none or many employees”
• Req 4: “a project must be controlled by at least one department”
• Req 5: “an employee involved in a project must work in the controlling department”
Sample Object-Oriented Modelling Hierarchy

- **Node**: set
- **Arrow**: multi-valued function

<table>
<thead>
<tr>
<th>$p$</th>
<th>$\alpha^{\Sigma_2}(p)$</th>
<th>Proposed vis.</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\pi_{\text{mult}(m,n)}$</td>
<td>$1 \xrightarrow{a} 2$</td>
<td>$X \xrightarrow{\text{surj}} Y$</td>
<td>$\forall x : X : m \leq</td>
</tr>
<tr>
<td>$\pi_{\text{surjective}}$</td>
<td>$1 \xrightarrow{a} 2$</td>
<td>$X \xrightarrow{\text{surj}} Y$</td>
<td>$f(X) = Y$</td>
</tr>
<tr>
<td>$\pi_{\text{inverse}}$</td>
<td>$1 \xrightarrow{a} 2$</td>
<td>$X \xrightarrow{\text{inv}} Y$</td>
<td>$\forall x \in X, \forall y \in Y : y \in f(x)$ iff $x \in g(y)$</td>
</tr>
<tr>
<td>$\pi_{\text{image-inclusion}}$</td>
<td>$1 \xrightarrow{a} 2$</td>
<td>$X \xrightarrow{\text{inclusion}} Y$</td>
<td>$\forall x \in X : f(x) \subseteq g(x)$</td>
</tr>
<tr>
<td>$\pi_{\text{composition}}$</td>
<td>$1 \xrightarrow{a} 2$</td>
<td>$X \xrightarrow{\text{comp}} Y$</td>
<td>$\forall x \in X : h(x) = {g(y)</td>
</tr>
</tbody>
</table>
## Sample Object-Oriented Modelling Hierarchy

### Diagram

- **Data Type**: Reference
- **Attribute**: Class
- **Employee**: 
  - **empDeps**: [inv] 
  - **depEmps**: [surj] 
- **Department**: 
  - **depEmps**: [surj] 
- **Project**: 
  - **proEmps**: [comp] 
- **Employee**: 
  - **proEmps**: [comp] 
- **Department**: 
  - **proDep**: [comp] 

### Table

<table>
<thead>
<tr>
<th>Principle</th>
<th>1</th>
<th>2</th>
<th>Proposed vis.</th>
<th>Intended semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>![mult](m, n)</td>
<td>![f](m, n)</td>
<td>![1..n]</td>
<td><img src="f" alt="X" /></td>
<td>![Y]</td>
</tr>
<tr>
<td>![surjective]</td>
<td>![f]</td>
<td>![surj]</td>
<td>![X]</td>
<td>![Y]</td>
</tr>
<tr>
<td>![inverse]</td>
<td>![f]</td>
<td>![inv]</td>
<td>![X]</td>
<td>![g]</td>
</tr>
<tr>
<td>![image-inclusion]</td>
<td>![f]</td>
<td>![⊑]</td>
<td>![X]</td>
<td>![g]</td>
</tr>
<tr>
<td>![composition]</td>
<td>![f]</td>
<td>![comp]</td>
<td>![X]</td>
<td>![Y]</td>
</tr>
</tbody>
</table>

### Instance

- **Alessandro**
- **Adrian**
- **Distech**
- **DPF**
- **DI-UiB**
- **DCE-HiB**
Sample Object-Oriented Modelling Hierarchy

- Invalid instance
Version Control Example

Repository

Alice’s copy

Bob’s copy

Timeline

Repository

Alice

Bob
Version Control Example

Repository

Alice’s copy

Bob’s copy

Timeline

Repository

Alice

Bob
Version Control Example

Repository

Alice’s copy

Bob’s copy

Timeline

Repository

Alice

Bob
Version Control Example

Repository

Alice's copy

Bob's copy

Timeline
Version Control Example

Repository

Alice’s copy

Bob’s copy

Timeline

Repository

Alice

Bob
Version Control Example

Repository

Alice’s copy

Bob’s copy

Timeline

Repository

Alice

Bob
Version Control Example

Repository

Alice’s copy

Bob’s copy

Timeline

Repository