Naturwissenschaftlich Technische Fakultät
Department Elektrotechnik und Informatik

UNIVERSITÄT SIEGEN

Forschungspraktikum

Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

Manuel Ohrndorf

27.09.2016

Inhalt

- **Einführung**
  - Motivation & Beispiel
  - Konsistenz von Modellen
  - Analyse von Inkonsistenzen
- **Komplementierung inkonsistenter Editierschritte**
  - Differenzdarstellung
  - unvollständige Editierschritte
  - komplementierende Editierregel
  - Reparaturberechnung
- **Matching-Algorithmus**
  - aufbauen des Arbeitsgraphen
  - Matching-Algorithmus
  - Atomic-Patterns
- **Schlussfolgerung**

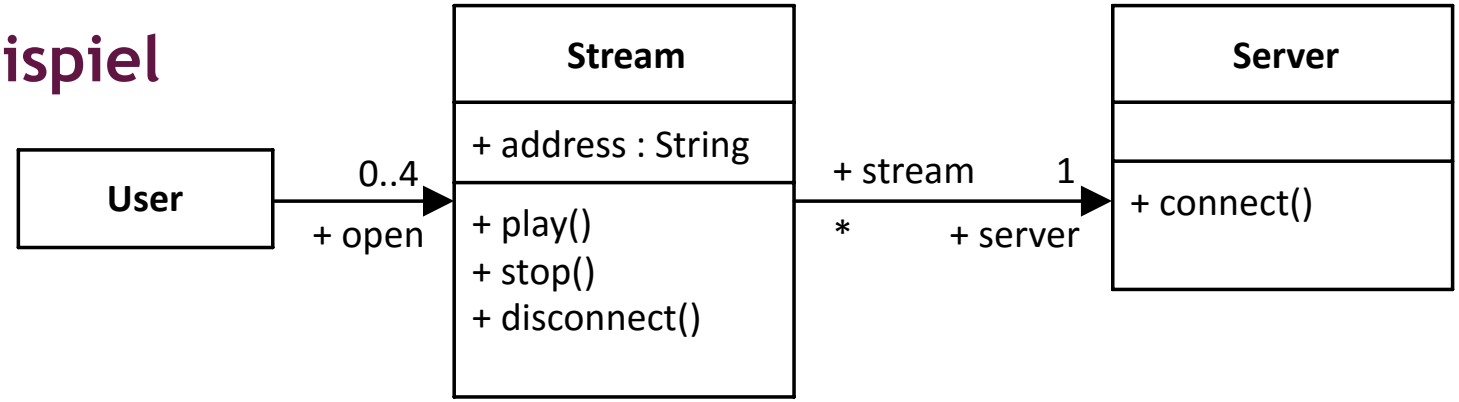
Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

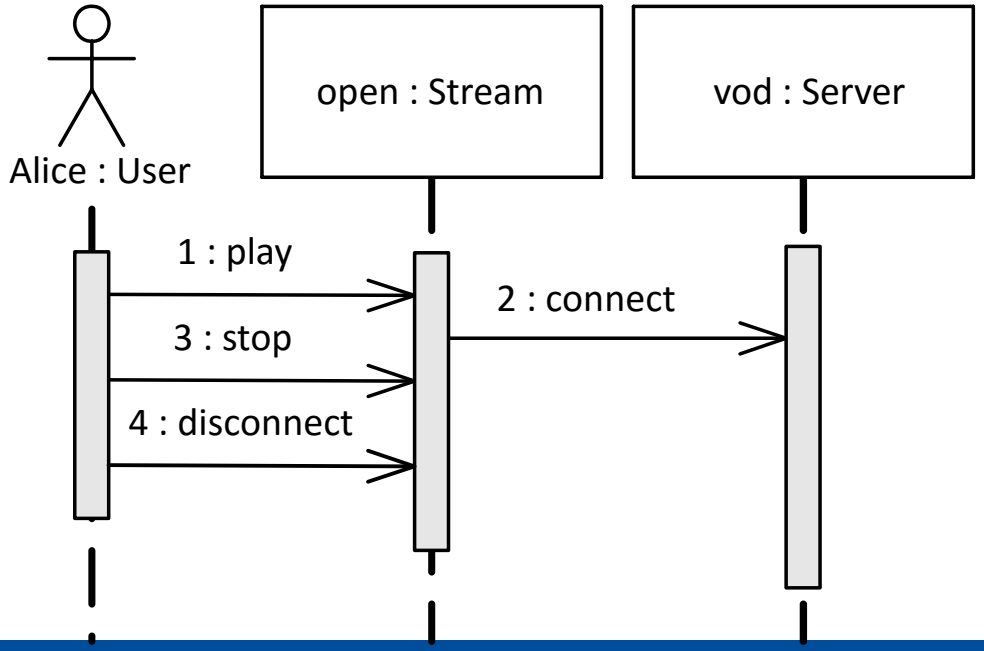
EINFÜHRUNG

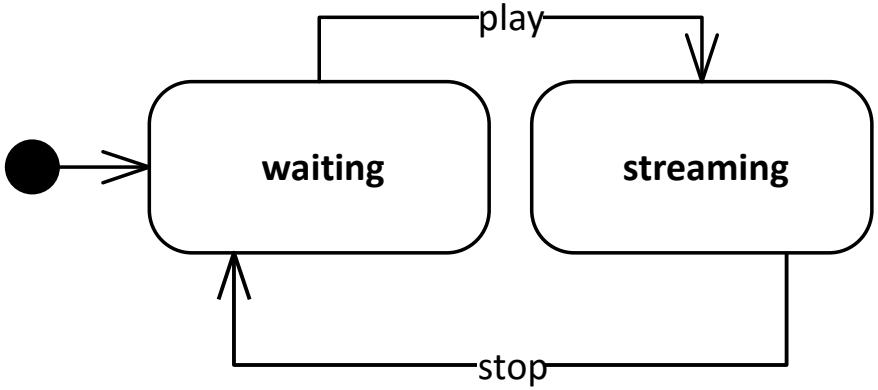
Motivation

- **modelbasierte Softwareentwicklung**
  - Ein Softwaresystem wird zunächst auf einem hohen Abstraktionsniveau entworfen.
  - Einsatz domänenspezifischer Modellierungssprachen
  - Modelle sind die zentralen Entwicklungsartefakte.
    - werden ggf. gleichzeitig durch mehrere Entwickler bearbeitet
  - Ein System kann aus verschiedenen Sichten beschrieben werden:
    - z.B. statische Sicht – Modellierung der Struktur
    - dynamische Sicht – Modellierung des Verhaltens
  - Modelle bilden die formale Grundlage für die Implementierung.
    - Prüfung und Wiederherstellung der Korrektheit der Modelle

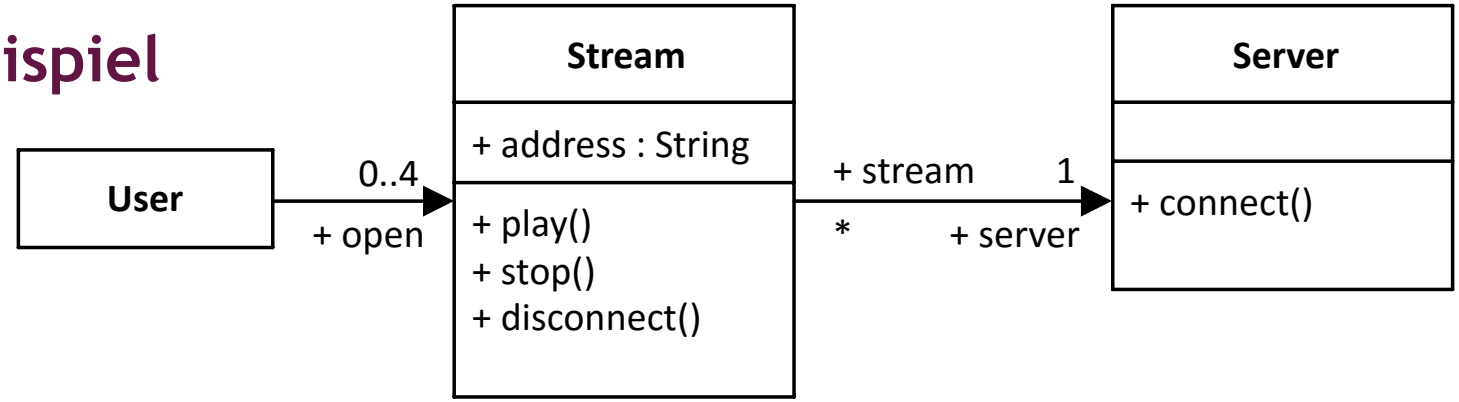
Beispiel

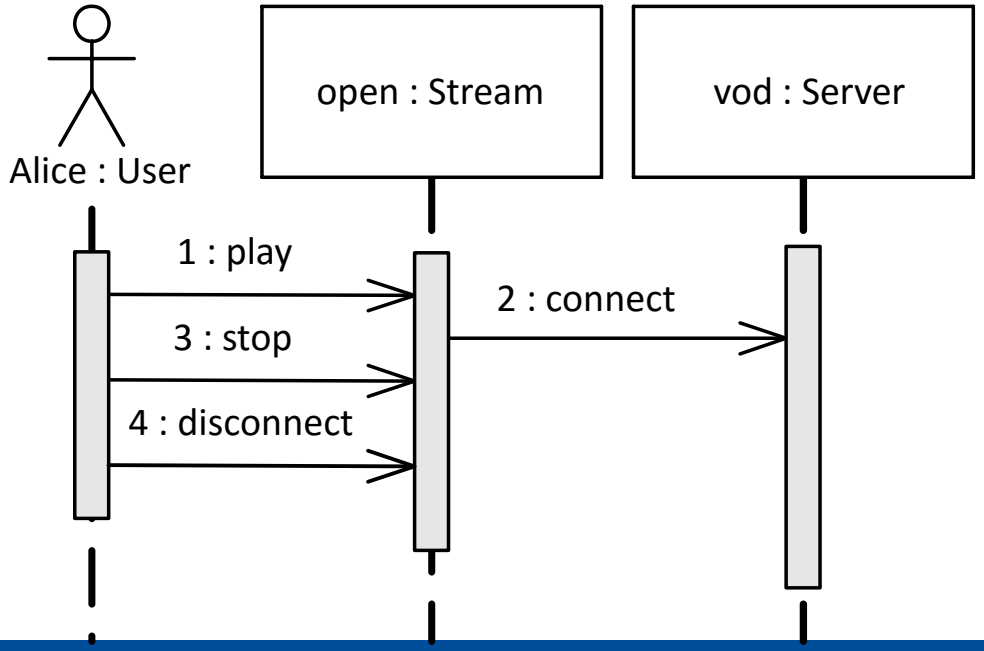

User
0..4
+ open
Stream
+ address : String
+ play()
+ stop()
+ disconnect()
+ stream
*
1
+ server
Server
+ connect()

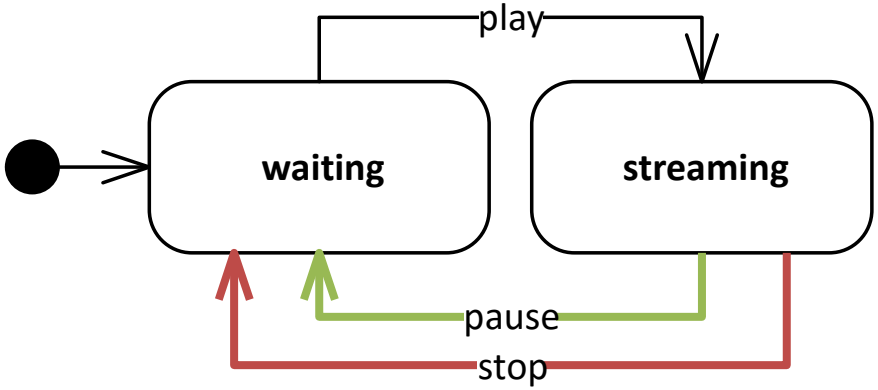

Alice : User
open : Stream
vod : Server
1 : play
2 : connect
3 : stop
4 : disconnect


play
waiting
streaming
stop

Beispiel

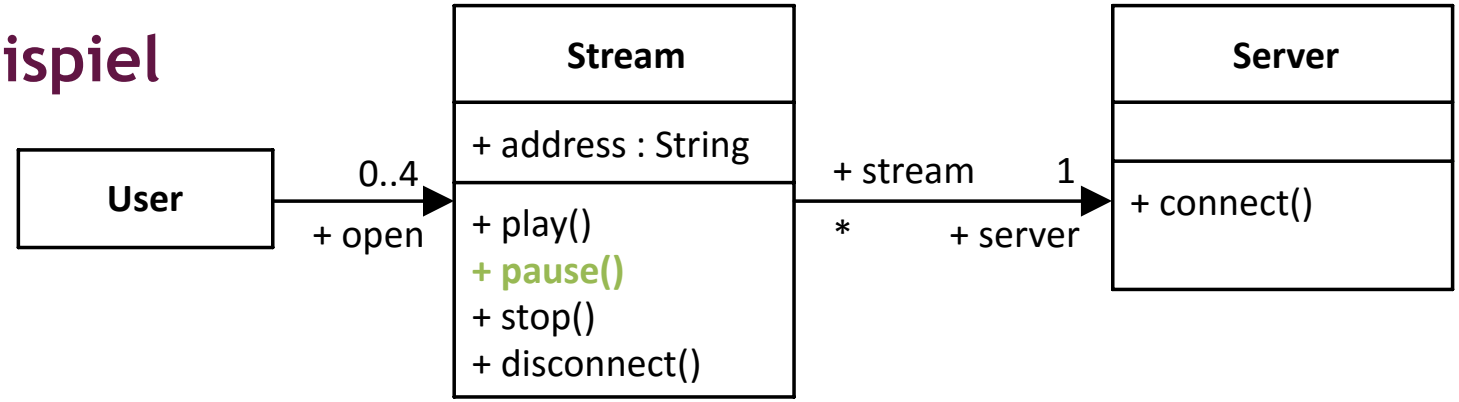


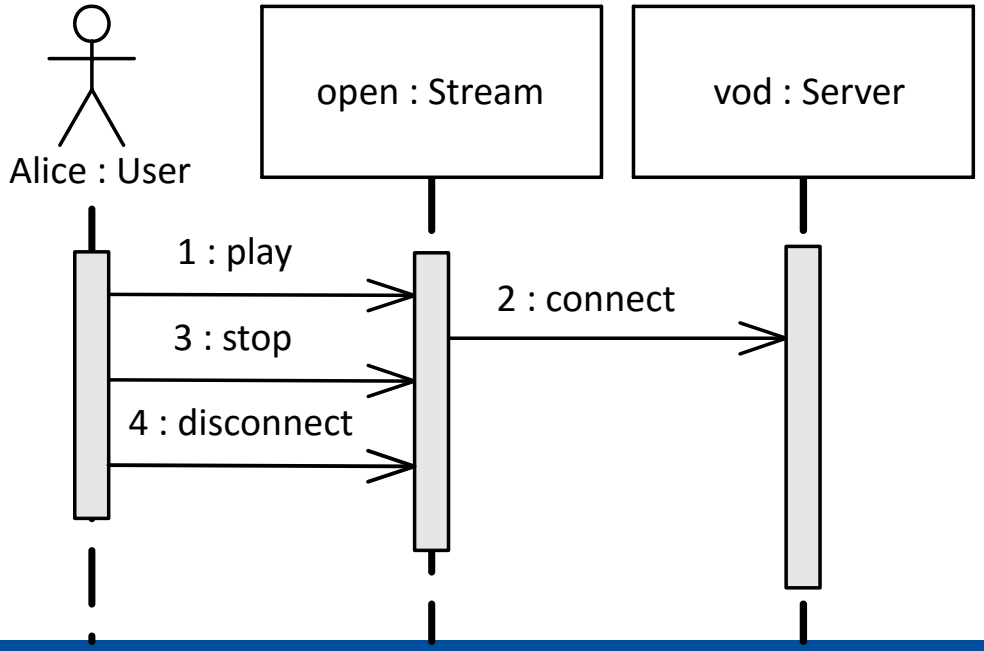


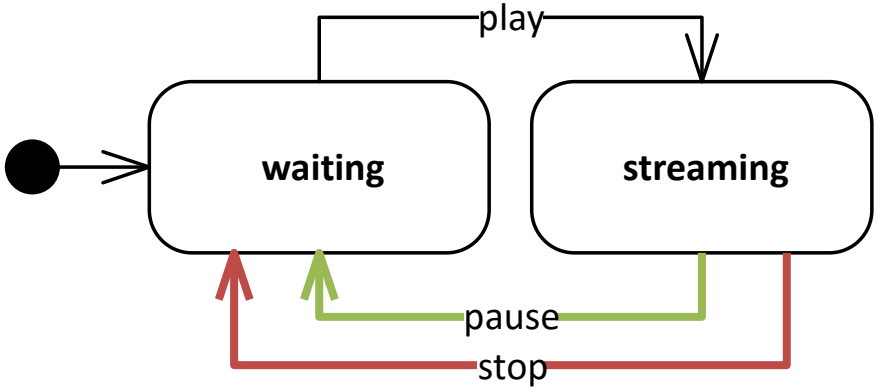


Legende: eingefügt @ Element-ID
gelöscht @ Element-ID

Beispiel

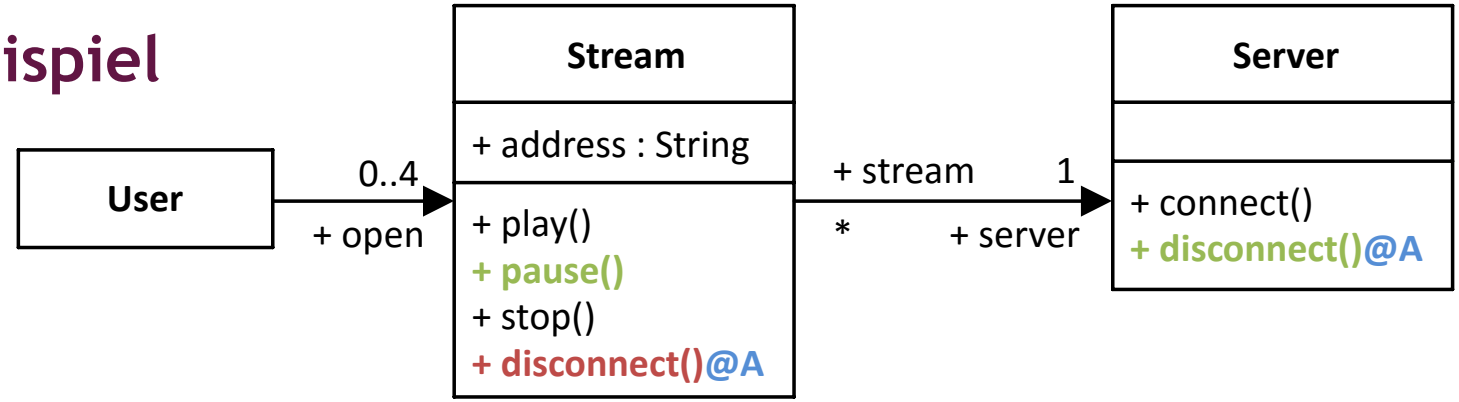


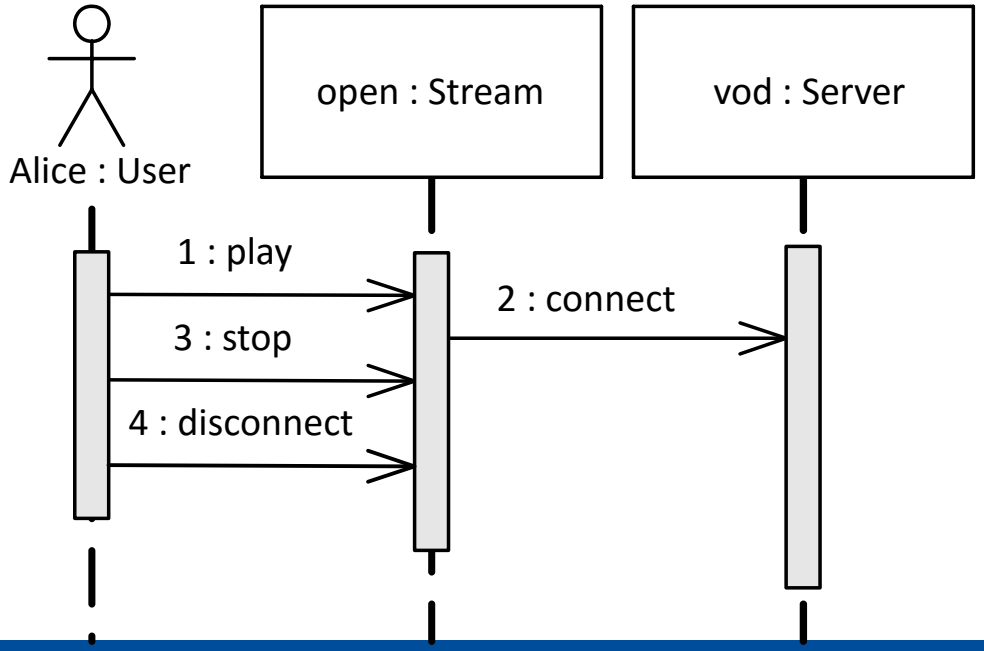


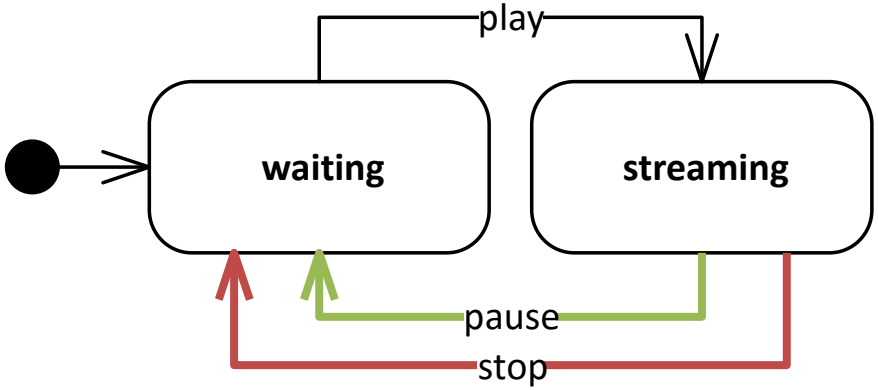


Legende: eingefügt @ Element-ID
gelöscht @ Element-ID

Beispiel

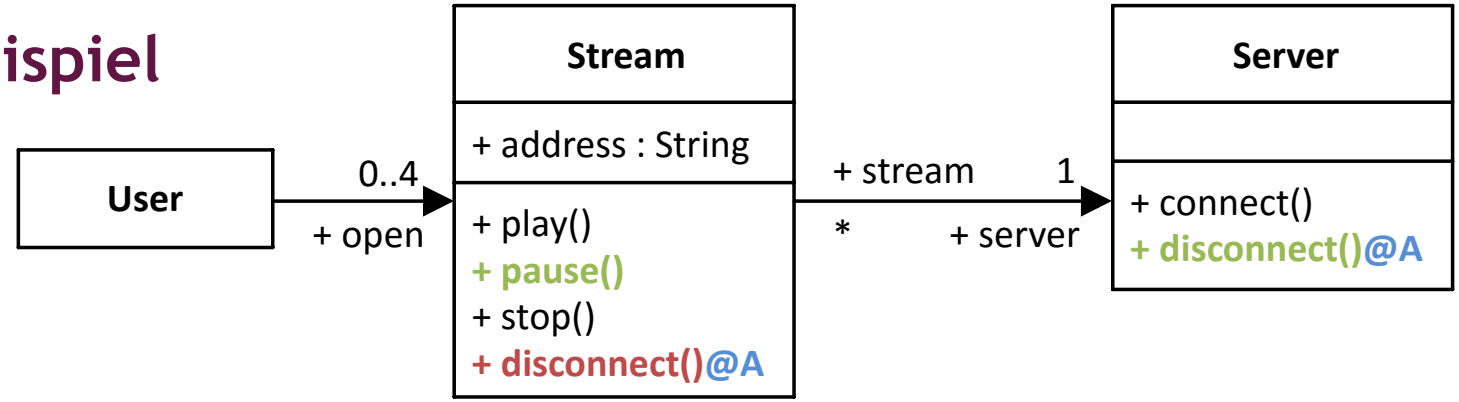


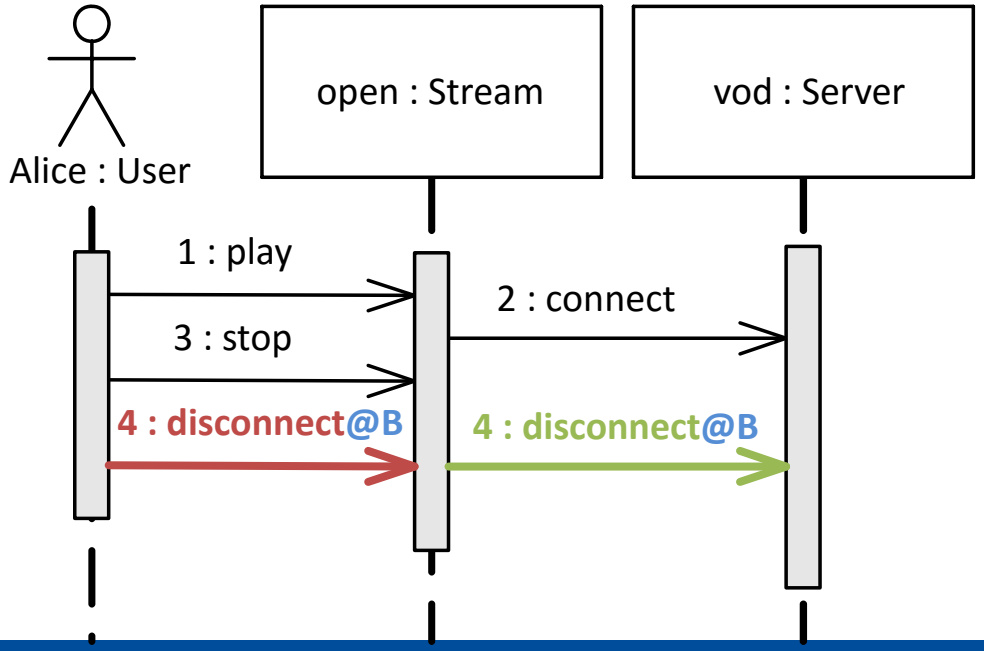


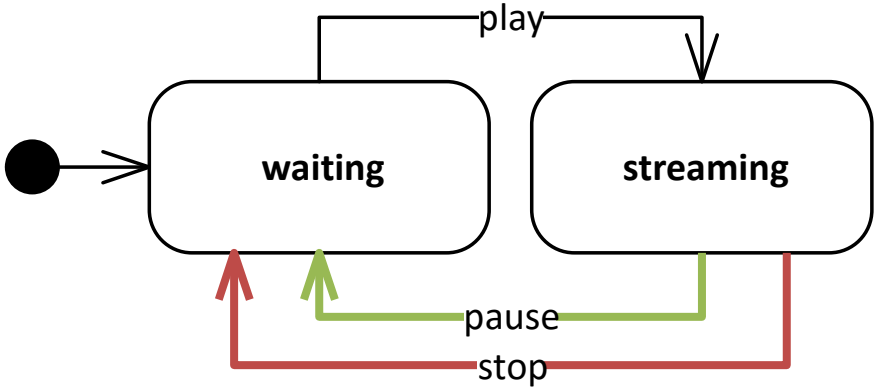


Legende: eingefügt @ Element-ID
gelöscht @ Element-ID

Beispiel







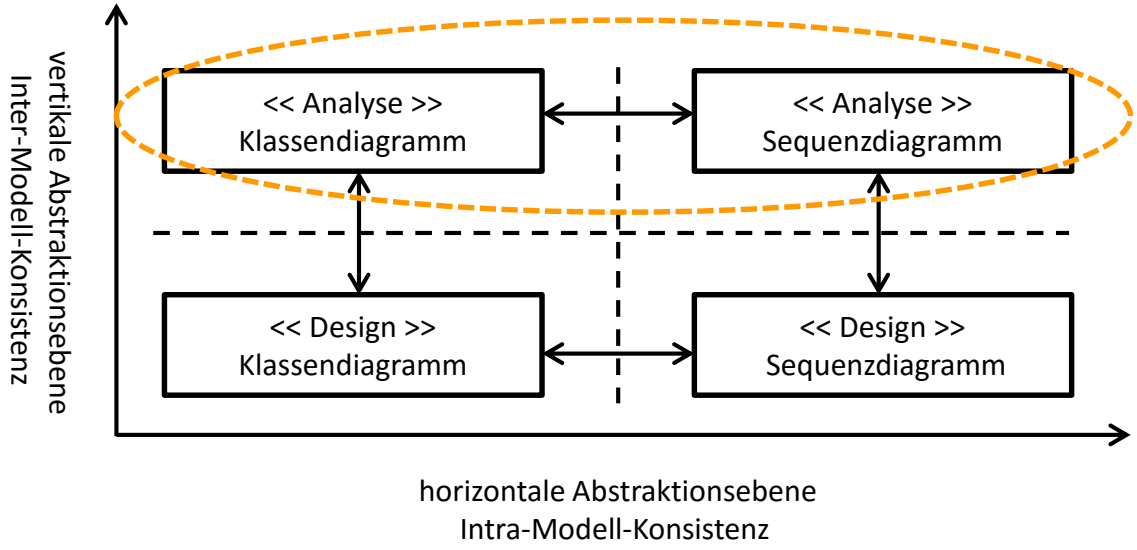
Legende:
- eingefügt @ Element-ID
- gelöscht @ Element-ID

Konsistenz von Modellen (semantisch)

- **Überschneidung** *(overlapping elements / correspondences)*
  - Zwei Elemente beschreiben den selben Teilaspekt eines Systems.
- **Inkonsistenz**
  - Überschneidende Elemente formulieren nicht zu vereinbarende Aussagen (über den selben Teilaspekt eines Systems).
- **Ursachen für Inkonsistenzen**
  - Mehrere Modelle *(Multi-View)* werden parallel entwickelt.
  - Die Überschneidungen zwischen den Modellen ist unklar.
  - Missverstandene Anforderungen in frühen Entwicklungsphasen.
  - …

Konsistenz von Modellen

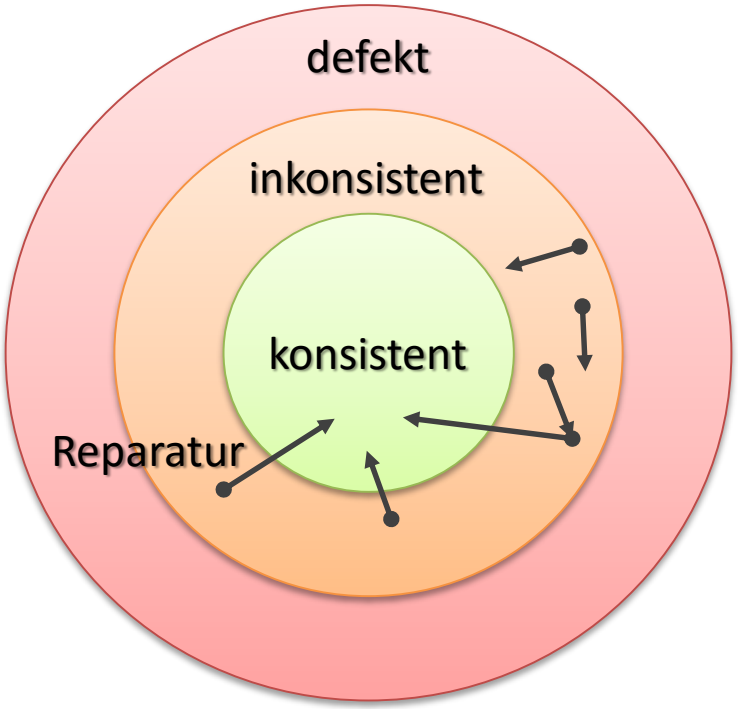
- **horizontale bzw. Intra-Modell-Konsistenz**
  - Modelle (ggf. Sichten) auf einer Abstraktionsebene
- **vertikale bzw. Inter-Modell-Konsistenz**
  - Verfeinerung oder Transformation der Modelle



Konsistenz von Modellen (syntaktisch)

- **Definition: horizontale Konsistenz**
  - Eine horizontale Abstraktionsebene kann aus mehreren (strukturell verbundenen*) Metamodellen bestehen.
  - Für jedes Metamodell, *sowie Metamodell übergreifend, können Konsistenzregeln formuliert werden.
  - (ggf. mehrere Modellierungssichten pro Metamodell)
- **Definition: Konsistenzregel** (oder Constraints)
  - Wird ausgehend von einem Kontextelement formuliert.
  - Wir betrachten First-Order-Logic basierte Regeln (z.B. OCL).

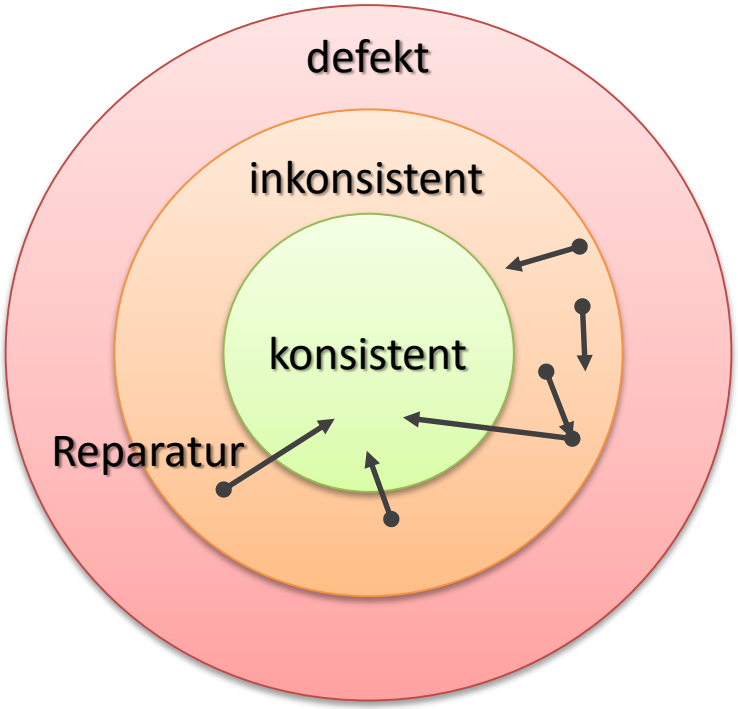
Zustandsraum eines Modells



- Definition der Ränder?
- Definition der Metrik?

- **nicht defekt:**
  - Metamodell konform:
    - Container / Containment
    - typkonforme Referenzen
    - konsistente Opposites
    - typkonforme Attribute
    - Listen [0..*] / Felder [0..1]
- **defekt:**
  - nicht auflösbare Referenzen
  - unbekannte Typen/Metaklassen
  - XML-Parserfehler … usw.

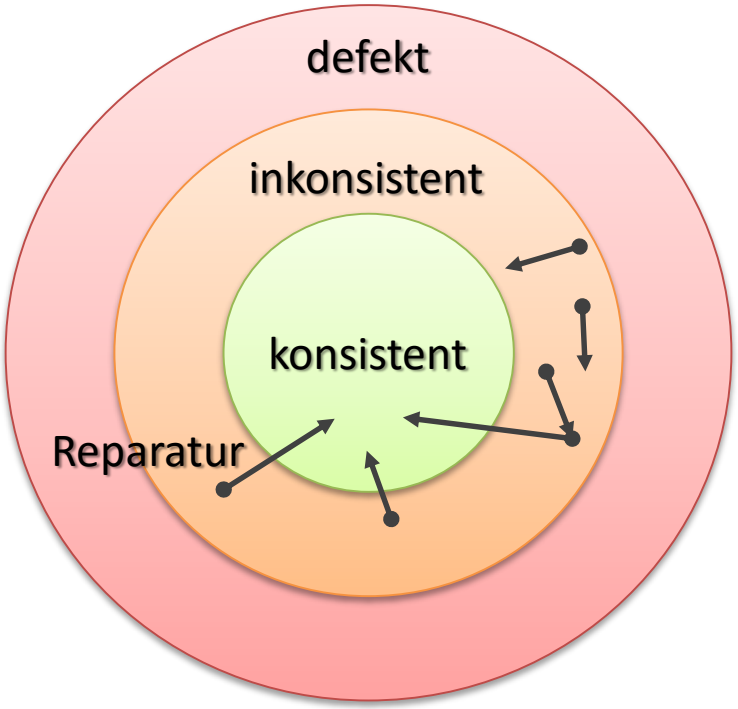
Zustandsraum eines Modells



- Definition der Ränder?
- Definition der Metrik?

- **konsistent:**
  - Einhaltung der Multiplizitäten
  - alle Konsistenzregeln sind erfüllt
- **Problem:**
  - Nicht jede Inkonsistenz kann sofort behoben werden.
  - Reparaturen sind ggf. ein inkrementeller Prozess.
  - Reparaturen können (temporäre) negative Seiteneffekte haben.

Zustandsraum eines Modells



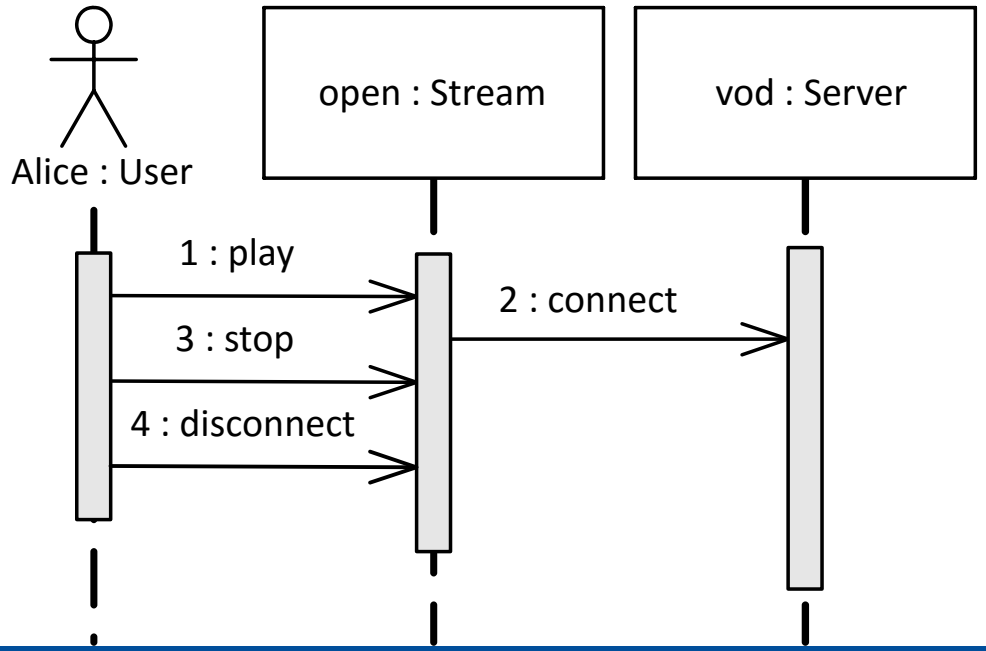
- Definition der Ränder?
- Definition der Metrik?

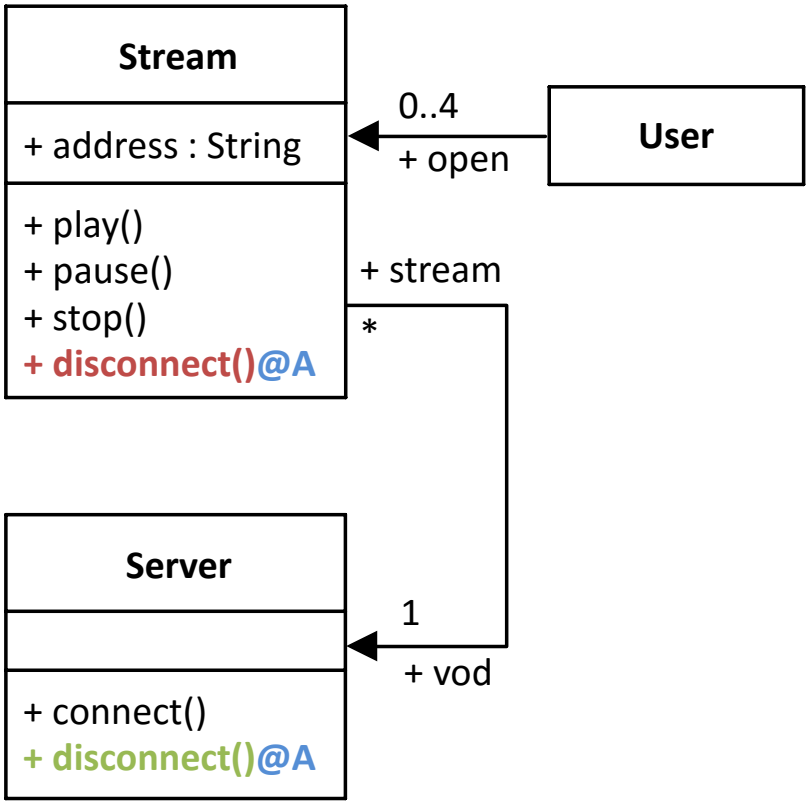
- **konsistent bzgl.** $V = \{v_1, \ldots, v_n\}$
  - sei v_i die Validierung einer Konsistenzregel C_x: $v_i = (C_x, \kappa, \varepsilon)$
    - κ : Kontextelement
    - ε : Validierungsergebnis
- **Reparatur:**
  - *Intuition:* Eine Reparatur muss eine Änderung durchführen, so dass sich die Validierung der Konsistenzregel (partiell*) verbessert.
  - *… nicht zwingend das Ergebnis ε

Analyse von Inkonsistenzen

» Jede Konsistenzregel wertet einen Teil des Modells aus.

$$Message\ m: \forall l \in m.receiveEvent.covered: \exists o \in l.represents.type.ownedOperation: o.name = m.name$$ [RE12]



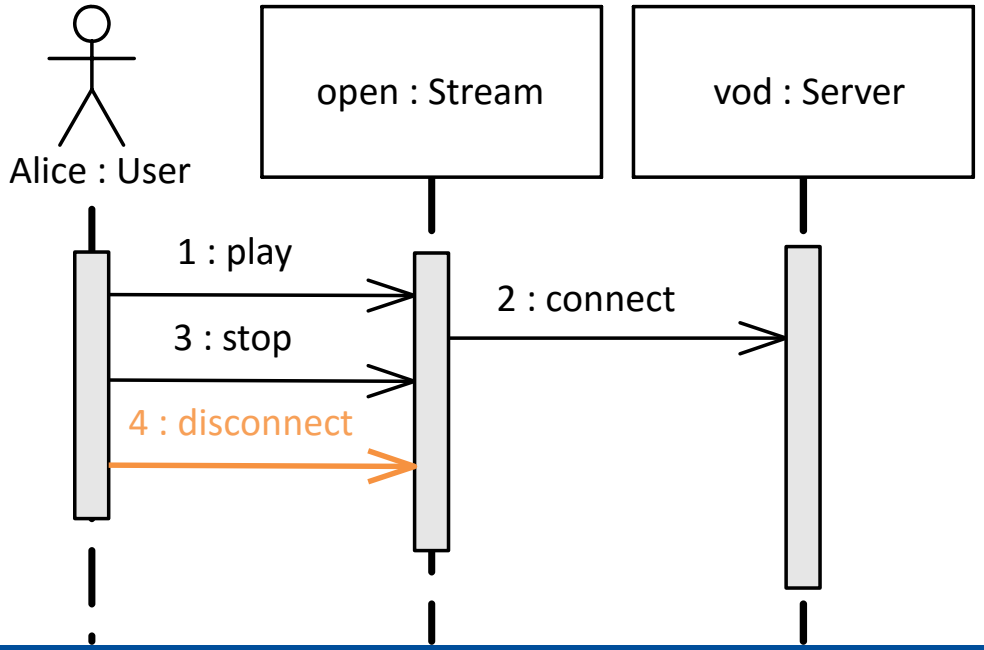


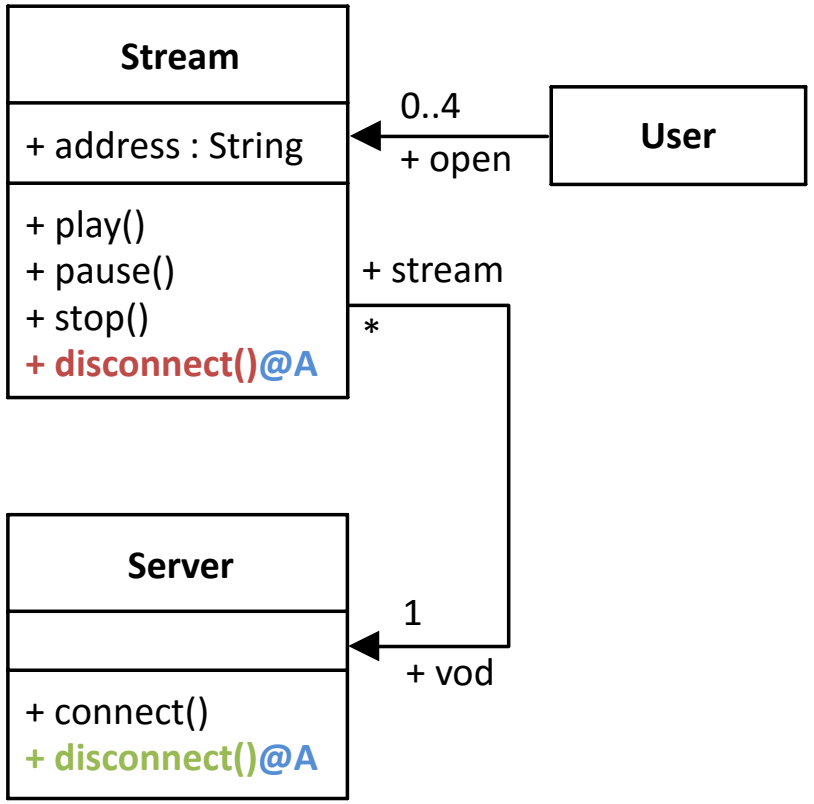
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

Analyse von Inkonsistenzen

» Jede Konsistenzregel wertet einen Teil des Modells aus.

$$Message\ m: \forall l \in m.receiveEvent.covered: \exists o \in l.represents.type.ownedOperation: o.name = m.name$$ [RE12]





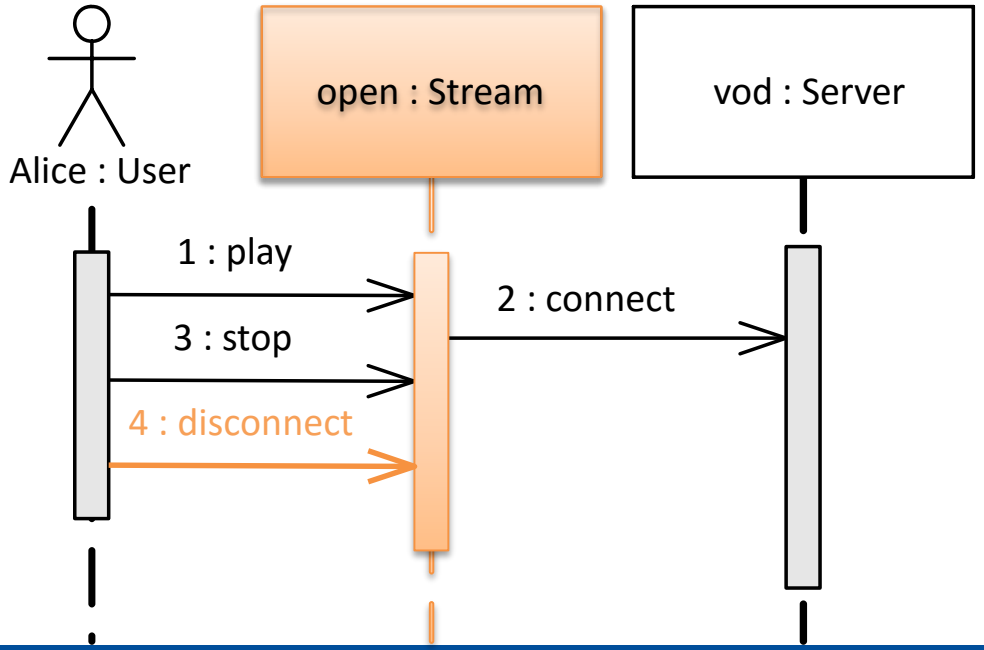
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

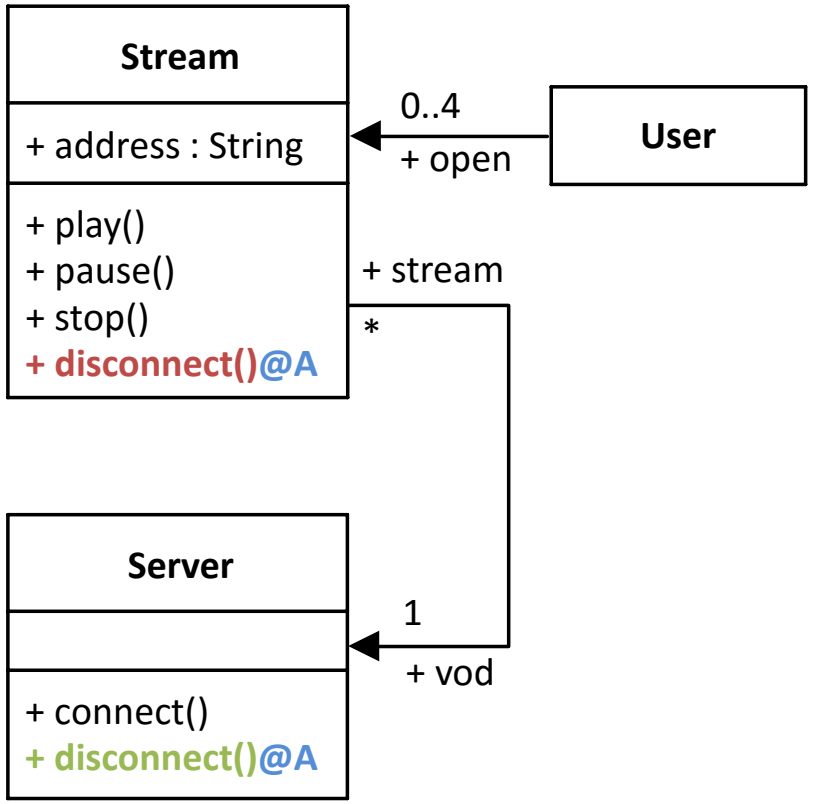
Analyse von Inkonsistenzen

» Jede Konsistenzregel wertet einen Teil des Modells aus.

$$Message\ m: \forall l \in m.receiveEvent.covered: \exists o \in l.represents.type.ownedOperation: o.name = m.name$$

[RE12]



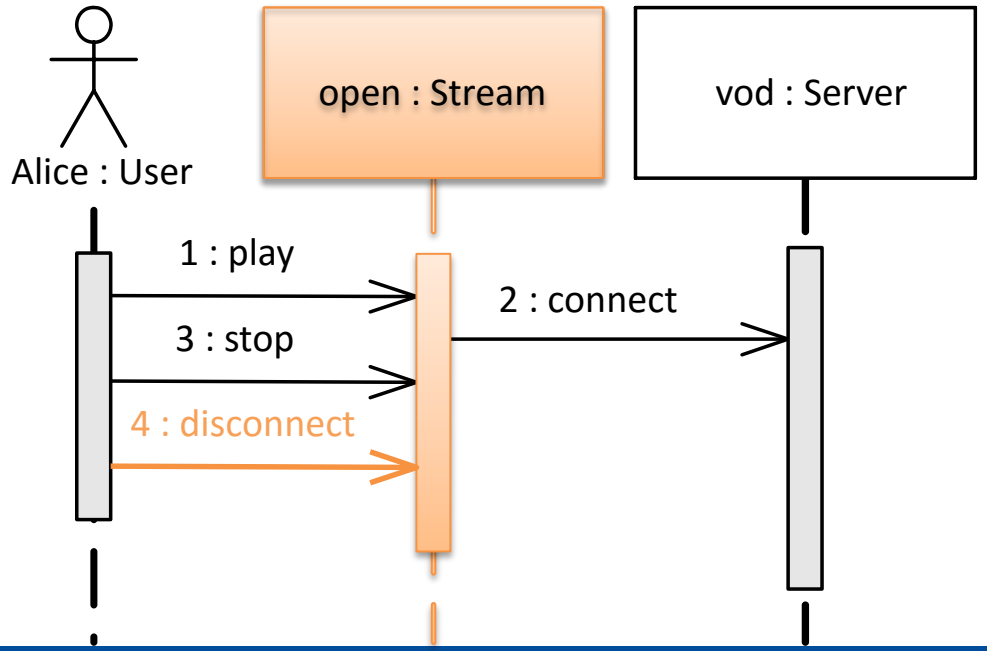


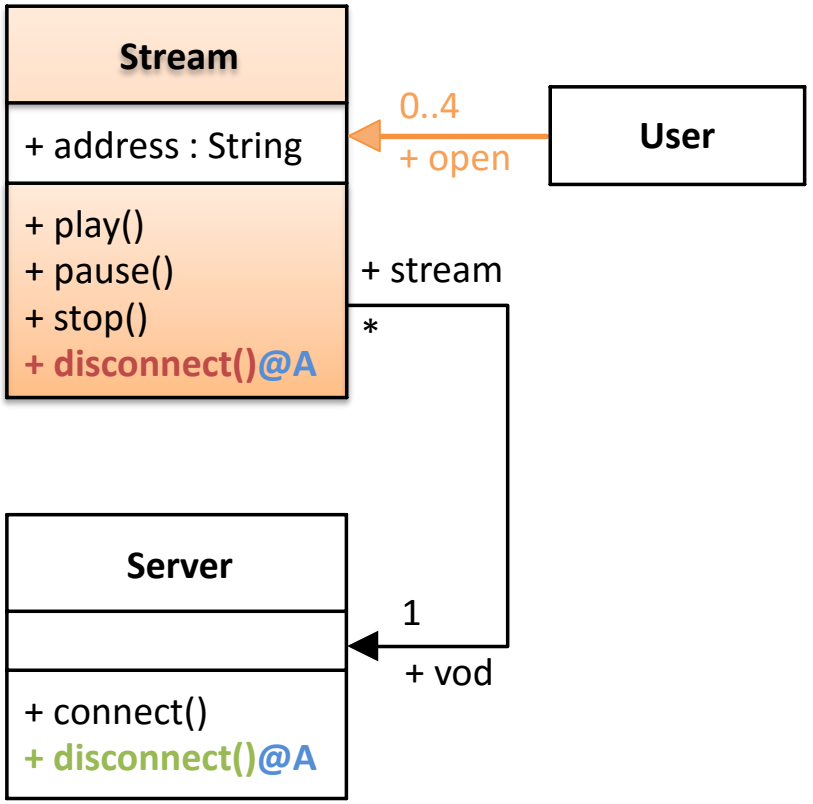
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

Analyse von Inkonsistenzen

» Jede Konsistenzregel wertet einen Teil des Modells aus.

$$Message\ m: \forall l \in m.receiveEvent.covered: \exists o \in l.represents.type.ownedOperation: o.name = m.name$$ [RE12]





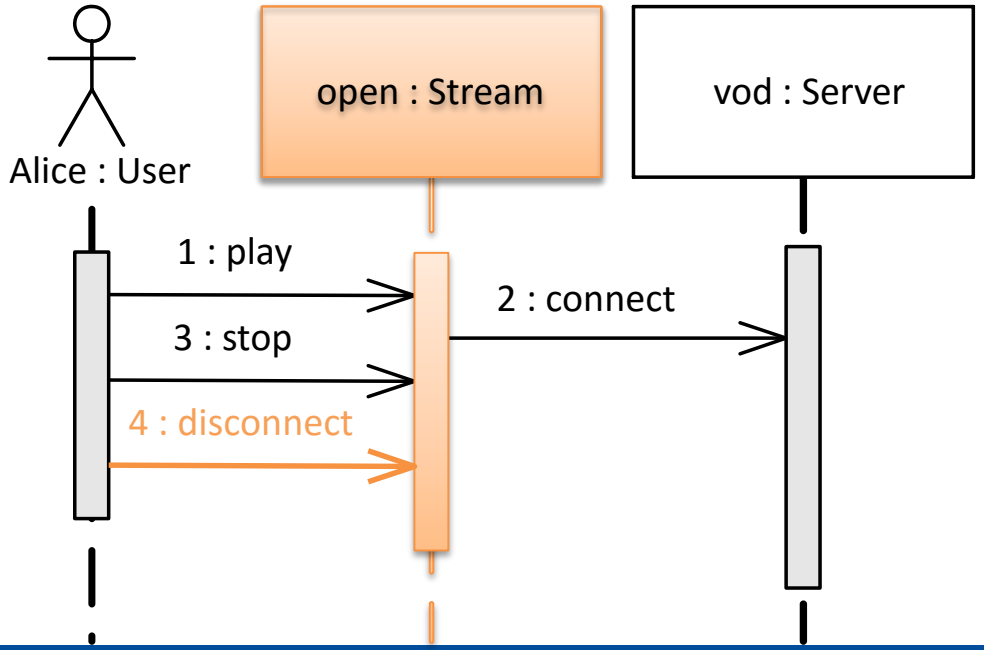
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

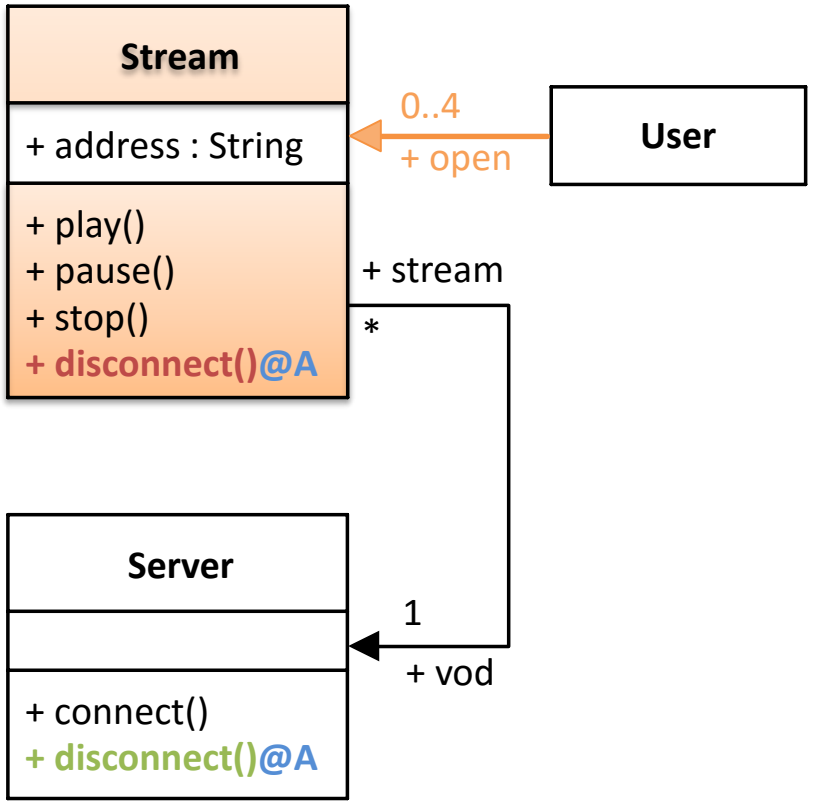
Reparatur von Inkonsistenzen

» Jede Konsistenzregel wertet einen Teil des Modells aus.

$$Message\ m: \forall l \in m.receiveEvent.covered: \exists o \in l.represents.type.ownedOperation: o.name = m.name$$

[RE12]



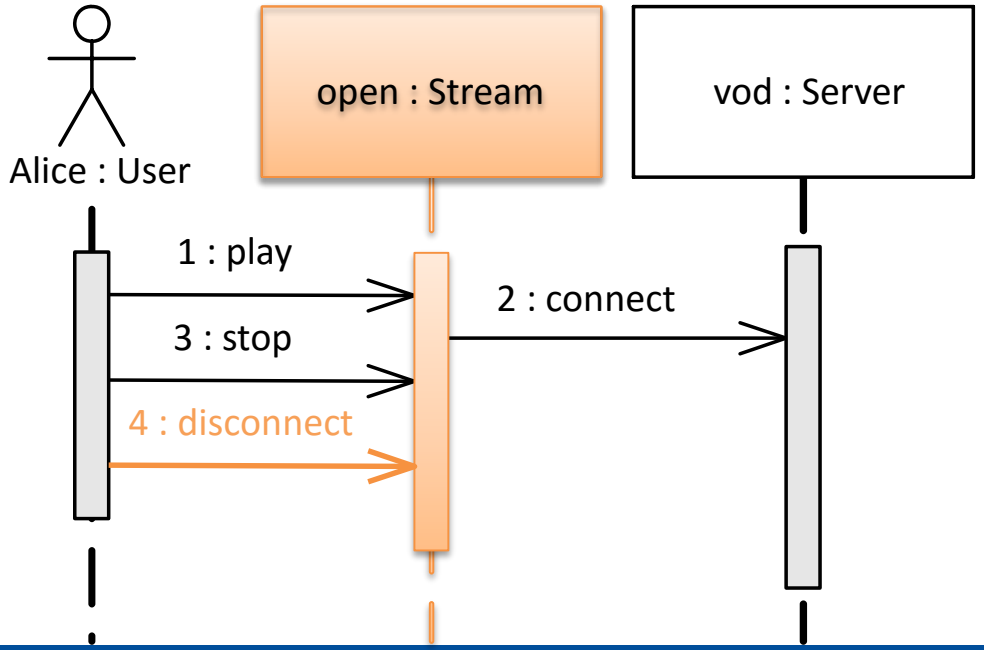


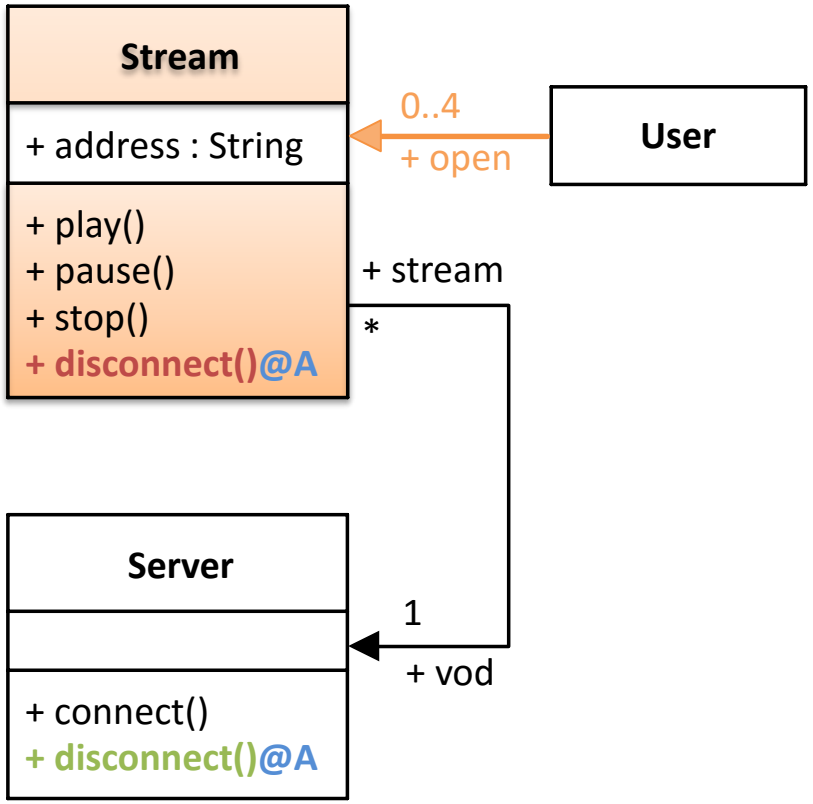
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

Reparatur von Inkonsistenzen

» Reparaturen lassen sich erzeugen, indem man die Auswertung rückwärts betrachtet und sich alle möglichen Modifikationen merkt.

$o.name = m.name$
$\Rightarrow z.B.\ umbenennen: stop() \rightarrow disconnect()$
$\Rightarrow Nachricht\ disconnect() \rightarrow pause()$ [RE12]





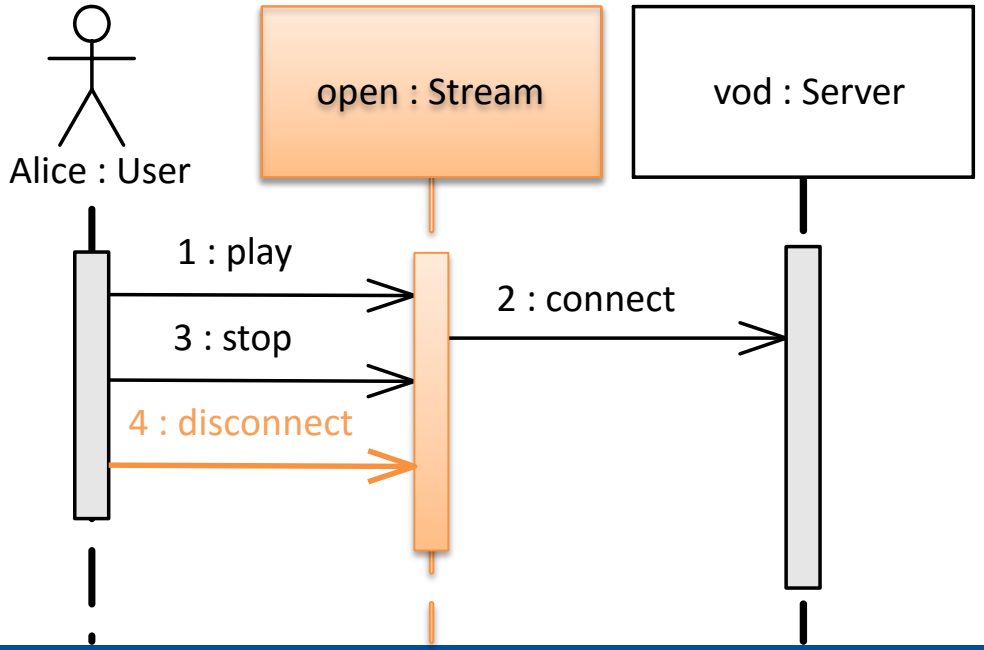
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

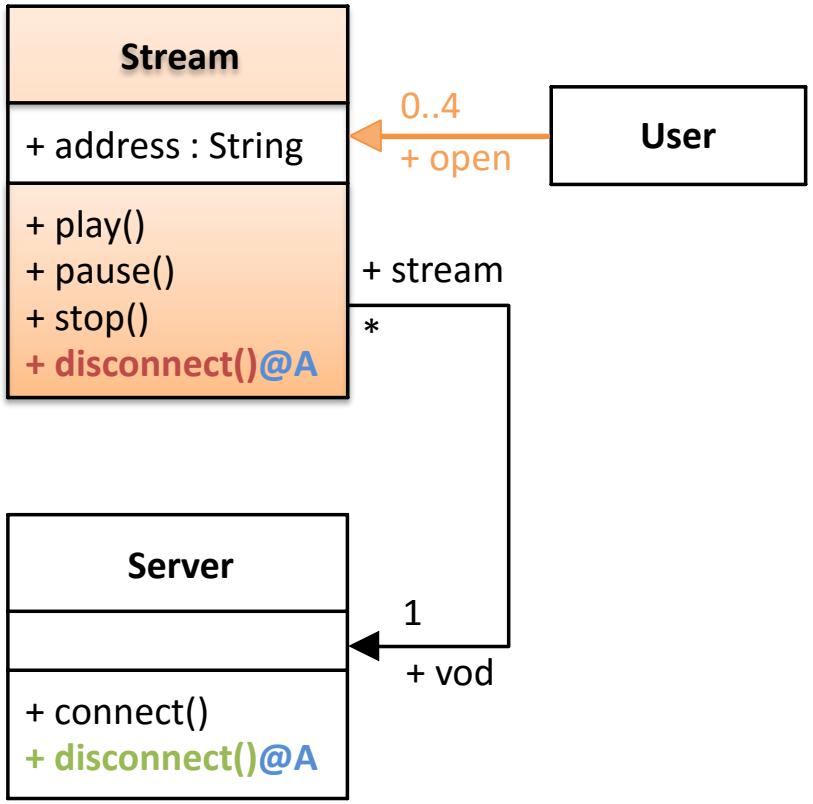
Reparatur von Inkonsistenzen

» Reparaturen lassen sich erzeugen, indem man die Auswertung rückwärts betrachtet und sich alle möglichen Modifikationen merkt.

$\exists o \in l.represents.type.\mathbf{ownedOperation}$:
$\Rightarrow$ *einfügen einer neuen Operation*

[RE12]





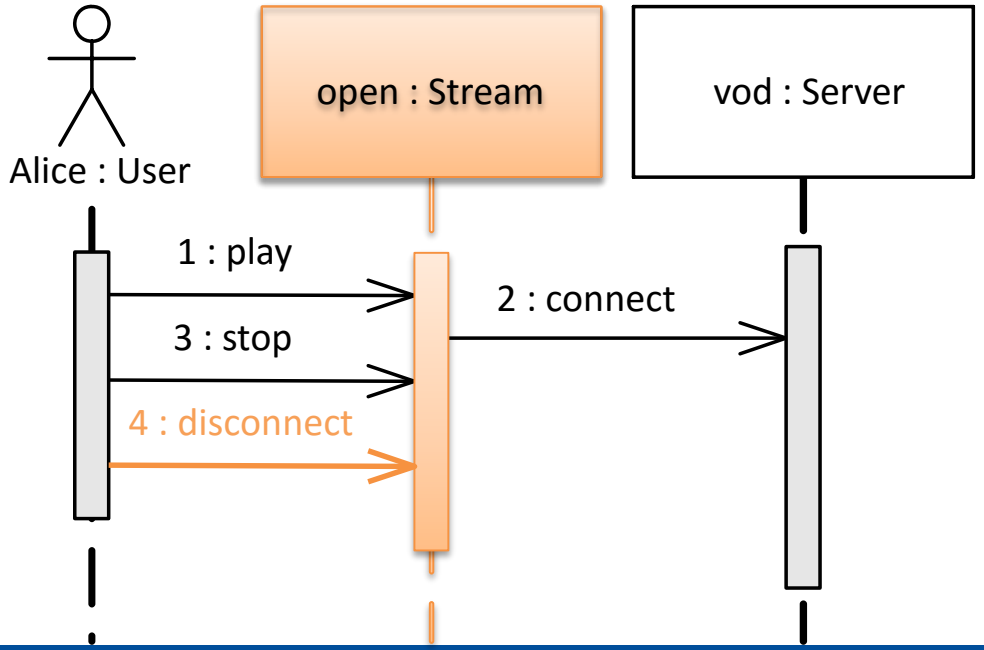
Legende: eingefügt @ Element-ID ausgewertet
gelöscht @ Element-ID

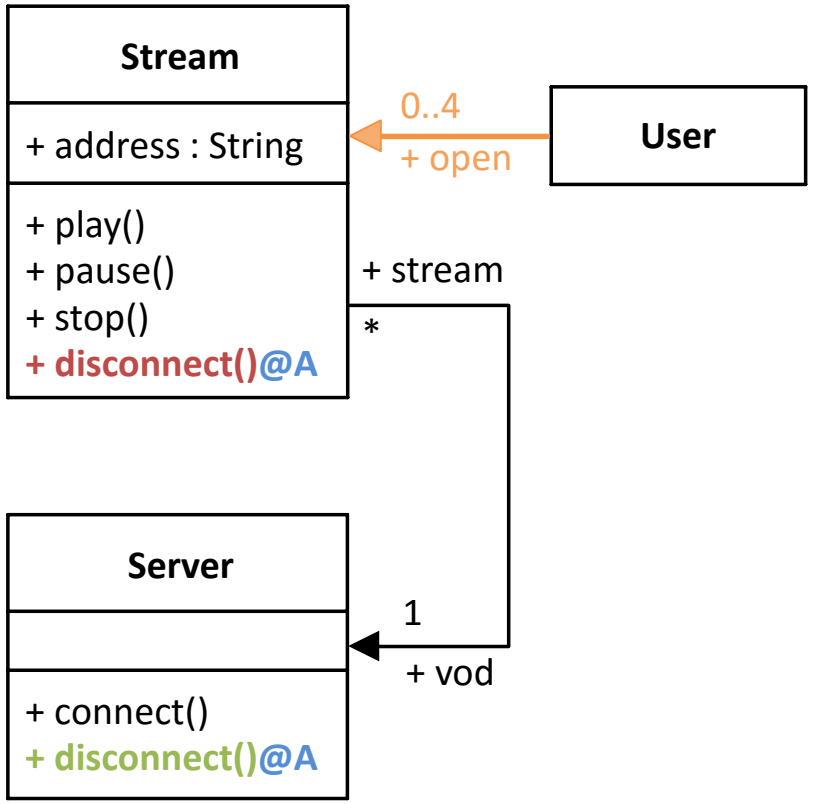
Reparatur von Inkonsistenzen

» Reparaturen lassen sich erzeugen, indem man die Auswertung rückwärts betrachtet und sich alle möglichen Modifikationen merkt.

$\exists o \in l.represents.\mathbf{type}.ownedOperation$:
$\Rightarrow$ *z.B. setze den Typ der Property 'open' auf Server*

[RE12]





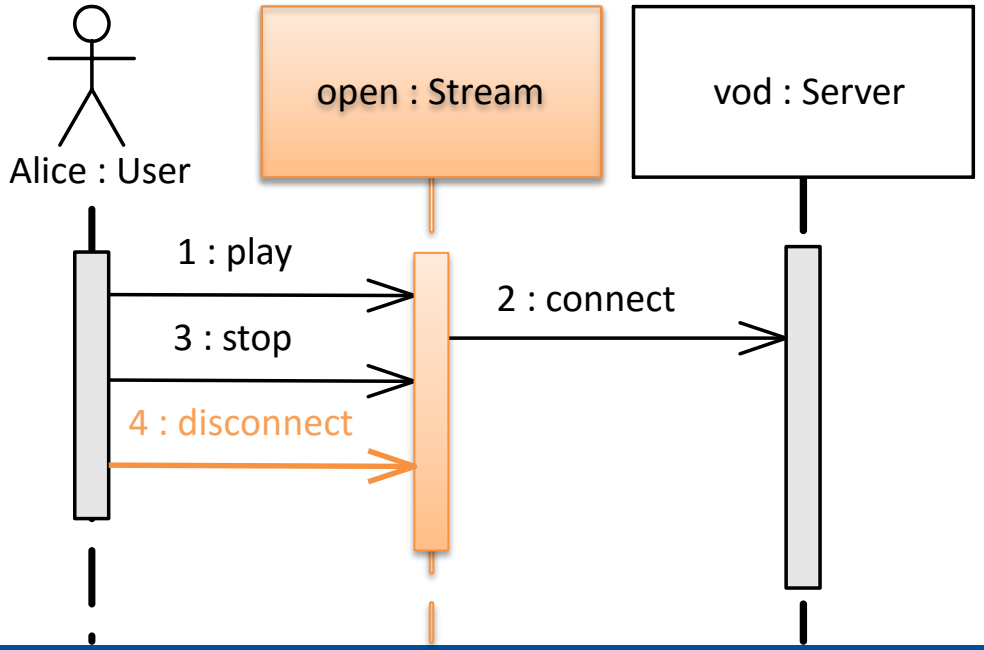
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

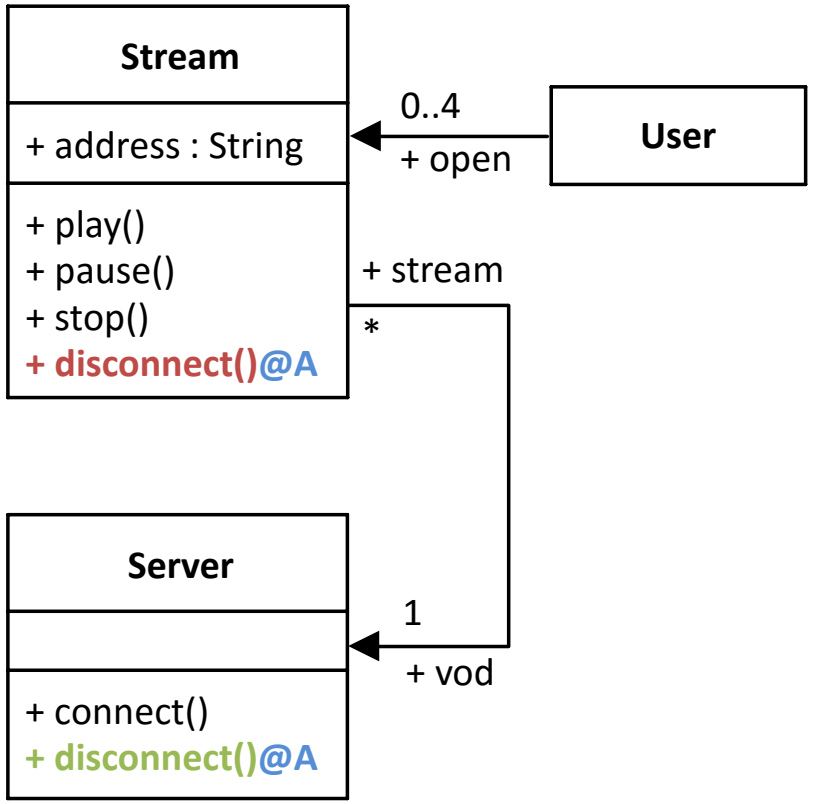
Reparatur von Inkonsistenzen

» Reparaturen lassen sich erzeugen, indem man die Auswertung rückwärts betrachtet und sich alle möglichen Modifikationen merkt.

$\exists o \in l.\boldsymbol{represents}.type.ownedOperation :$
$\Rightarrow$ *z.B. ersetze 's : Stream' durch 'vod : Server'*

[RE12]



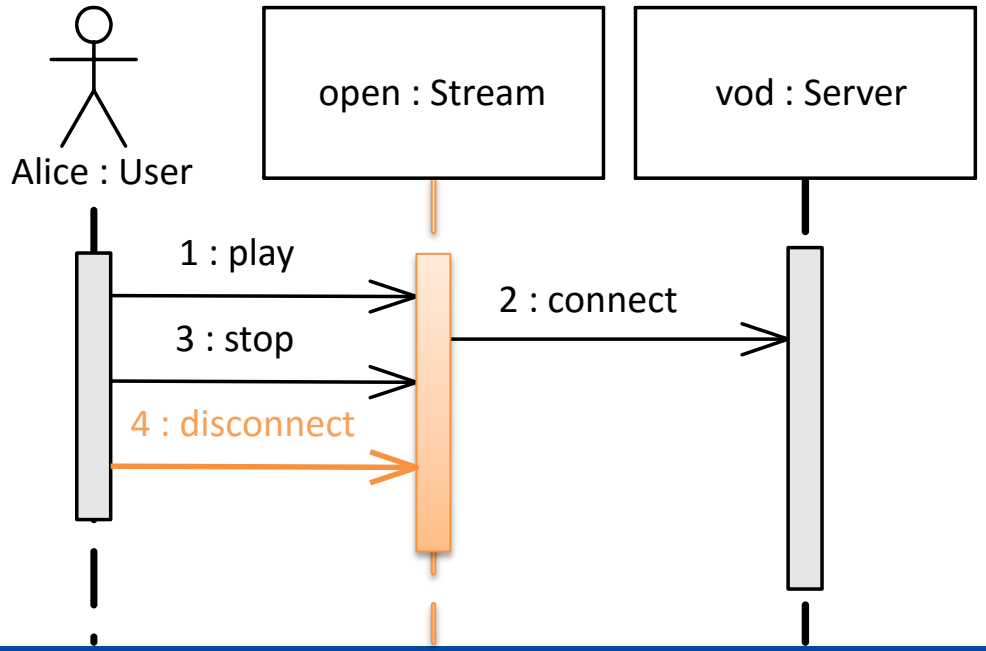


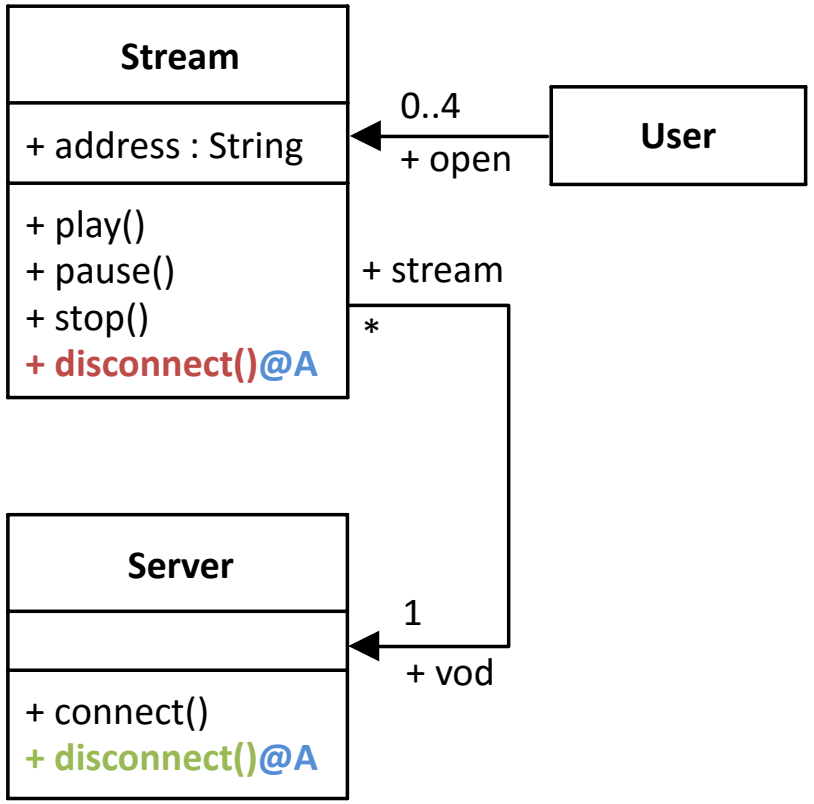
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

Reparatur von Inkonsistenzen

» Reparaturen lassen sich erzeugen, indem man die Auswertung rückwärts betrachtet und sich alle möglichen Modifikationen merkt.

$Message\ m: \forall l \in m.\mathbf{receiveEvent}.\mathbf{covered}:$
$\Rightarrow z.B.\ lasse\ die\ Nachricht\ von$
$'vod : Server'\ empfangen$ [RE12]





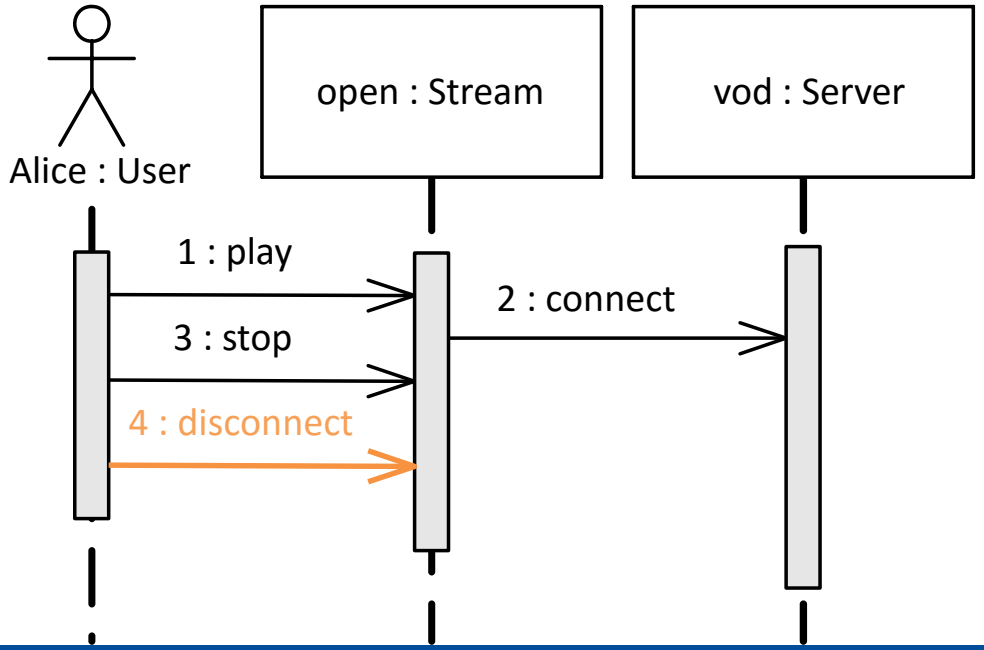
Legende: eingefügt @ Element-ID ausgewertet gelöscht @ Element-ID

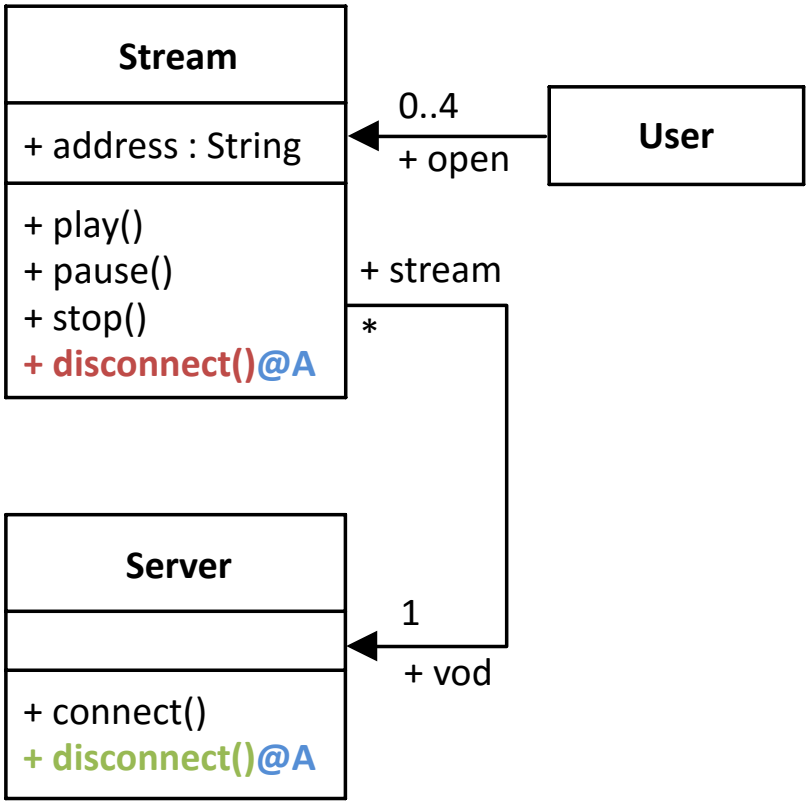
Reparatur von Inkonsistenzen

» Reparaturen lassen sich erzeugen, indem man die Auswertung rückwärts betrachtet und sich alle möglichen Modifikationen merkt.

$\boldsymbol{Message\ m}: \forall l \in m.receiveEvent.covered:$
$\Rightarrow$ *lösche Nachricht m*

[RE12]





Legende: eingefügt @ Element-ID ausgewertet
gelöscht @ Element-ID

Analyse von Inkonsistenzen – State of the Art

- Reder und Egyed [RE12] beschreiben die vollständige Berechnung aller *abstract repairs* bzgl. einer Konsistenzregel.

- **abstrakte Reparaturen** (bzgl. einer Inkonsistenz)
  - **Ausgangspunkt/Kontext:** Element und eine Eigenschaft (Attribut / Referenz des Metamodells) dieses Elements
  - **Art der Reparatur**:
    - *delete:* Struktur die verboten wird
    - *add:* Struktur die gefordert und nicht vorhanden ist
    - *modify*: Struktur oder Attribut das gefordert wird

  - z.B. < *add*, *Class[Stream]*, *ownedOperation* > macht keine Aussage darüber, ob die Operation verschoben oder neu erzeugt werden soll.

Analyse von Inkonsistenzen – State of the Art

- Zur Suche von konkreten Reparaturen wurden verschiedene **State-Space Exploration** Verfahren vorgeschlagen.
  - Starte mit dem **aktuellen Modellzustand und** suche eine **Sequenz von Editierschritten,** welche zu einem konsistenten Modellzustand führt.
  - Zum Teil werden benutzerdefinierte **Editierregeln** verwendet:
    - schränkt den Suchraum ein
    - schließt syntaktisch falsche Reparaturen aus
    - erhöht die Verständlichkeit der Reparaturen
- Es kann ggf. sehr viele mögliche konkrete Reparaturen geben.
  - Alternativen über Parameter zusammenfassen (Abstraktion)
  - Ranking der Reparatur-Alternativen durchführen

Naturwissenschaftlich Technische Fakultät
Department Elektrotechnik und Informatik

UNIVERSITÄT SIEGEN

Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

~~EIN~~ ZWEI VERFAHREN ZUR KOMPLEMENTIERUNG INKONSISTENTER EDITIERSCHRITTE

Motivation

- Die vorgestellten Verfahren beurteilen eine Inkonsistenz nur auf Basis des aktuellen Modellzustands.

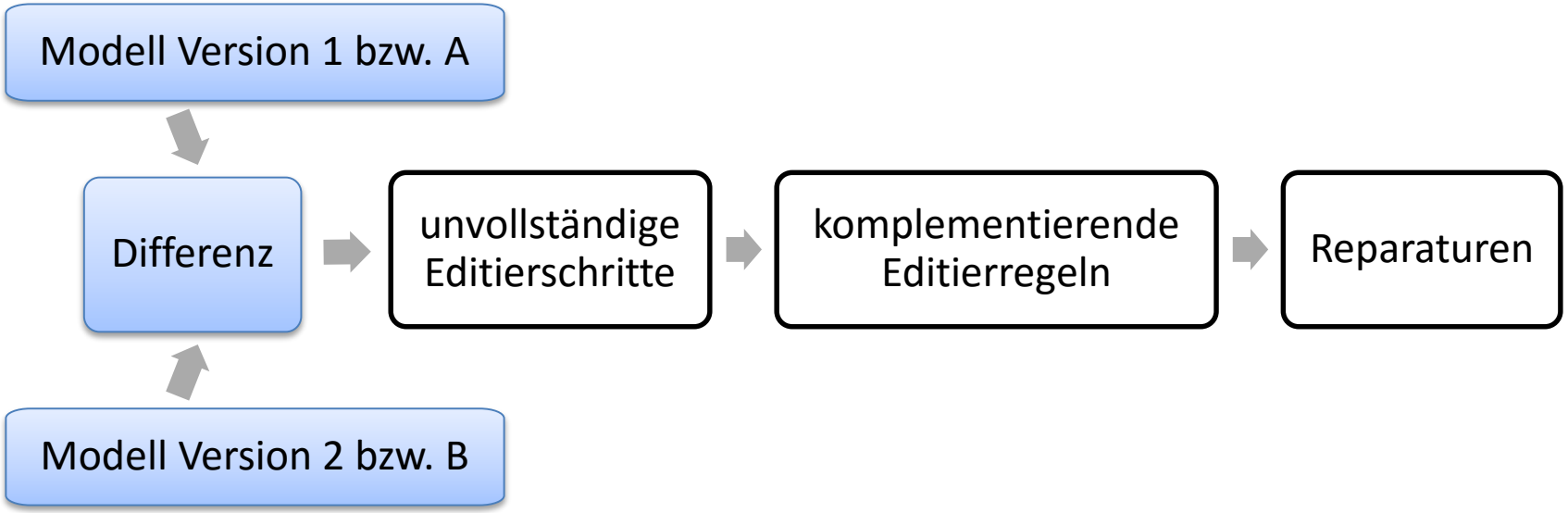
⇒ Idee: Benutze die Modell-Historie, um unvollständig durchgeführte Editierschritte zu finde, welche zu einer Inkonsistenz geführt haben.

  - Die **Ursache** für eine Inkonsistenz besser verstehen.
  - Die **Intention** eines Entwicklers besser verstehen.

- **Zum Beispiel:**
  - Es wurden Teile in einer Ansicht gelöscht/hinzugefügt aber in den anderen Ansichten vergessen.
  - Es wurden Teile inkonsistent umbenannt oder verschoben.
- **Ziel: Finde Reparaturen mit Hilfe von Editierregeln,**
  - die komplexe Veränderungen in einem Schritt vornehmen.
  - die mehrere Sichten gleichzeitig editieren.

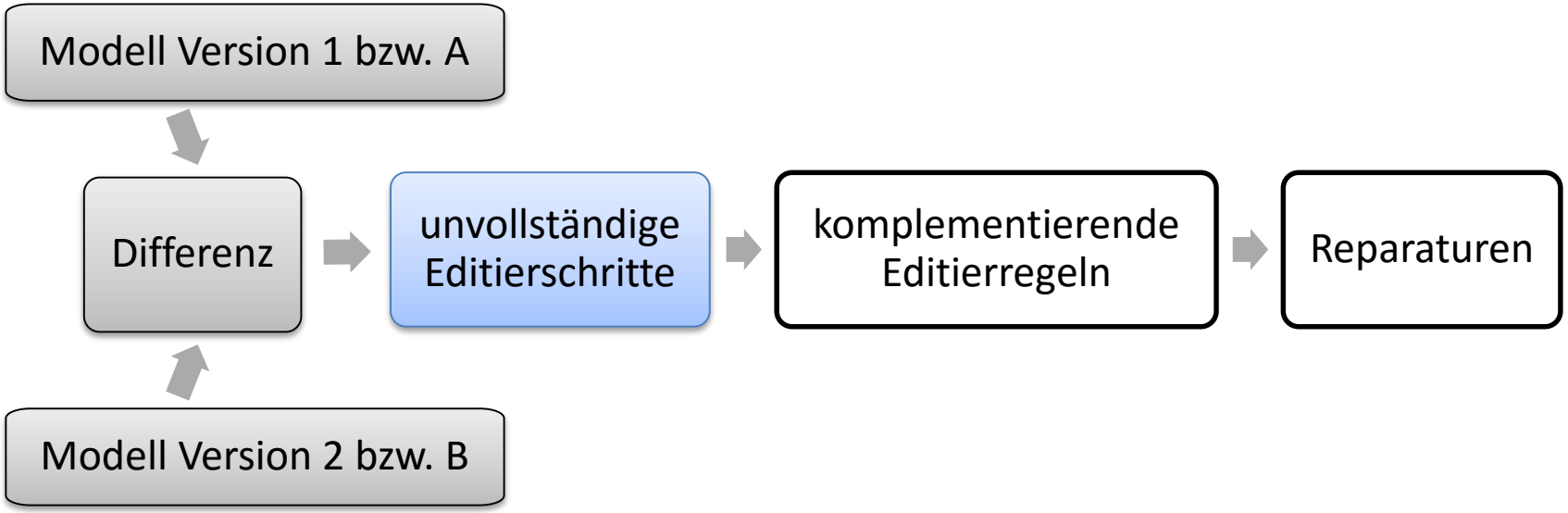
Berechnung historienbasierter Reparaturen

- **Differenzberechnung**
  - 1. Suche nach gemeinsamen Modellelementen.
  - 2. Leite daraus die Differenz zwischen den Modellen ab.



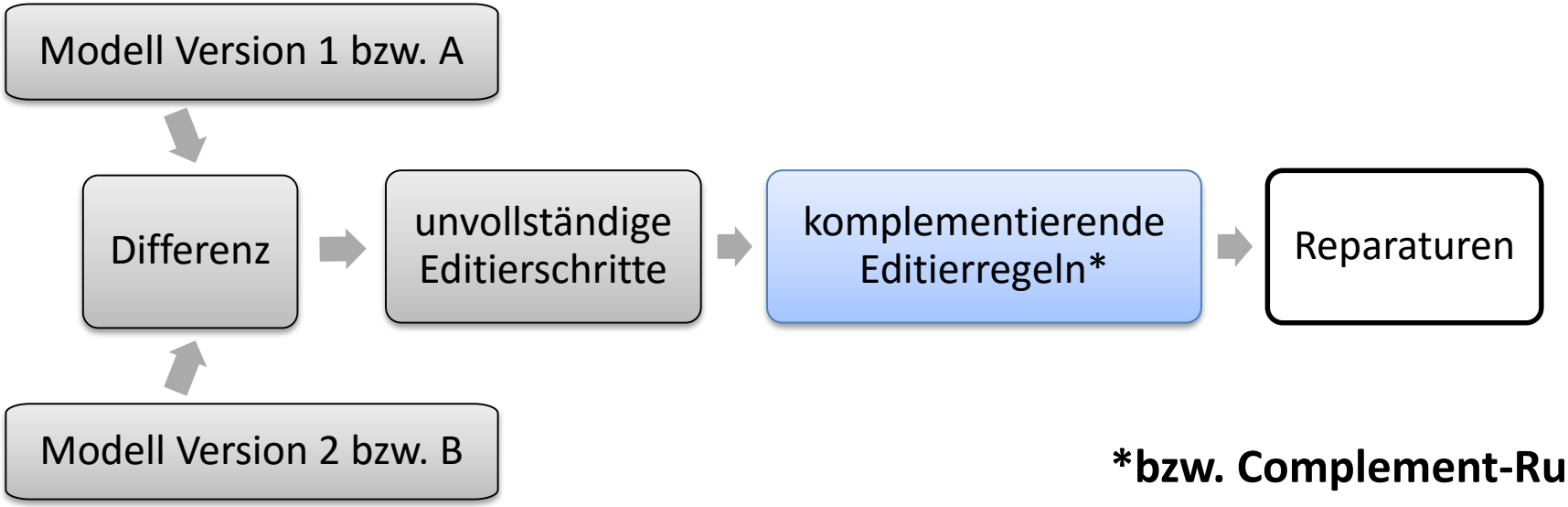
Berechnung historienbasierter Reparaturen

- **Erkennung unvollständiger Editierschritte**
  - **Eingabe:** Ein Menge an möglichen vollständigen Editierregeln.
  - **Ausgabe:** Alle (möglicherweise) unvollständig oder inkorrekt ausgeführten Editierschritte.



Berechnung historienbasierter Reparaturen

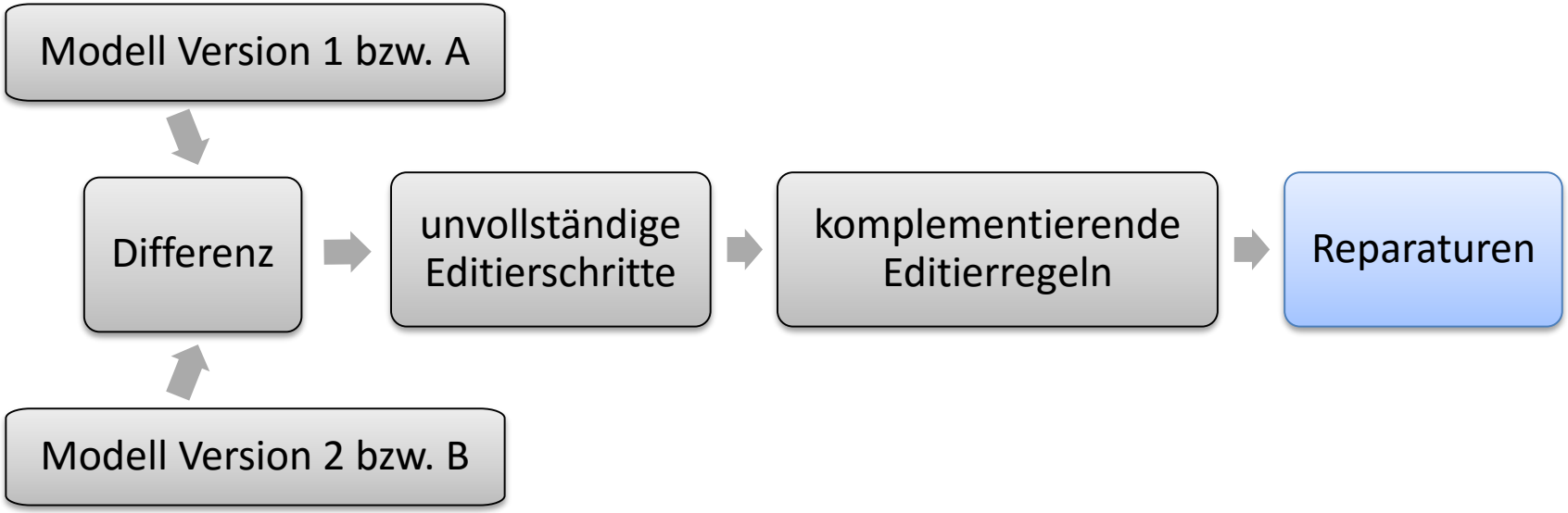
- **Ableitung einer komplementierenden Editierregel***
  - **Eingabe:** Editierregel (ER_{source}) und einen unvollständigen Editierschritt, welcher sich auf einen Teil (ER_{sub}) von ER_{source} abbilden lässt.
  - **Ausgabe:** $ER_{complement} = ER_{source} \,/\, ER_{sub}\ wobei\ ER_{sub} \subset ER_{source}$
    - (Muss für alle unvollständigen Editierschritte durchgeführt werden.)

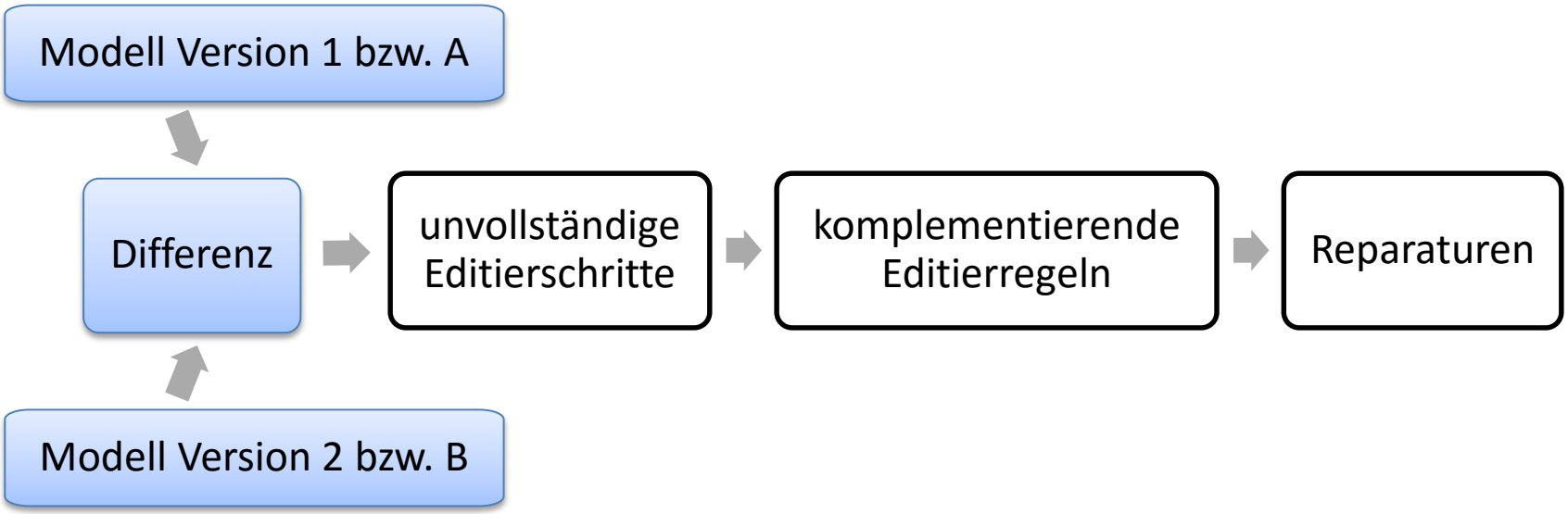


***bzw. Complement-Rule**

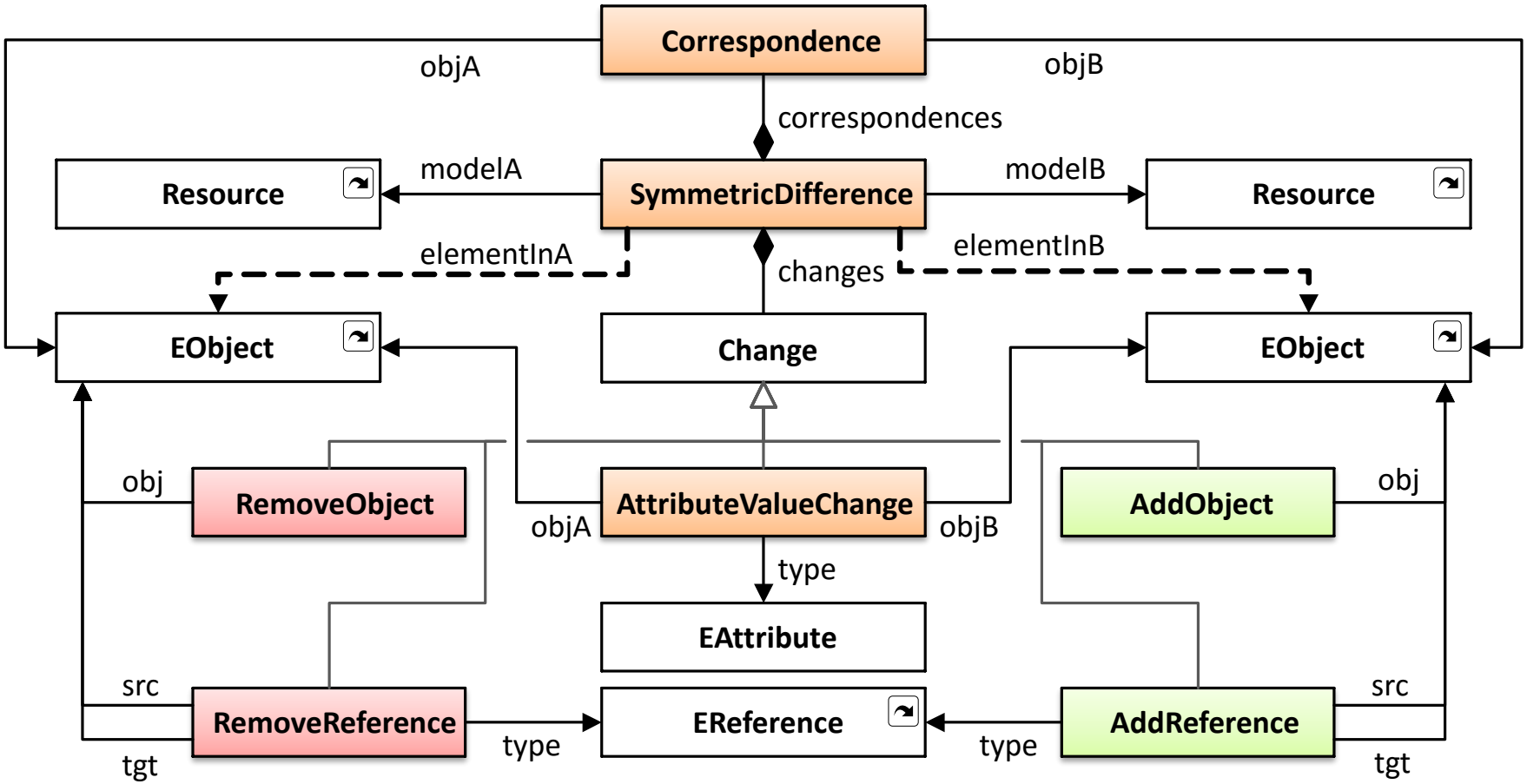
Berechnung historienbasierter Reparaturen

- **Berechnung der konkreten Reparaturen**
  - **Reparatur:** Suche alle konkreten vervollständigenden Editierschritte bzgl. der komplementierenden Editierregel.
    - Parameterbelegung für noch nicht ausgeführte Änderungen
    - (relative) Bewertung der Reparatur durchführen




Modell Version 1 bzw. A
Differenz
Modell Version 2 bzw. B
unvollständige Editierschritte
komplementierende Editierregeln
Reparaturen

Differenzdarstellung

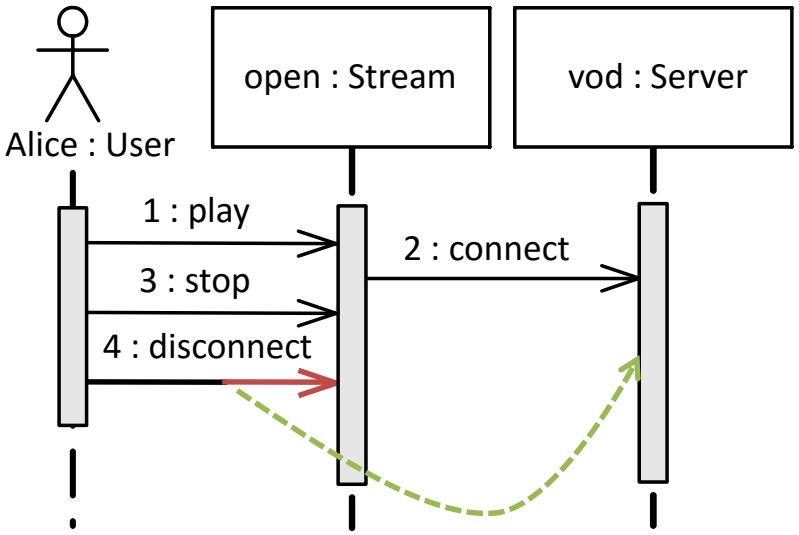


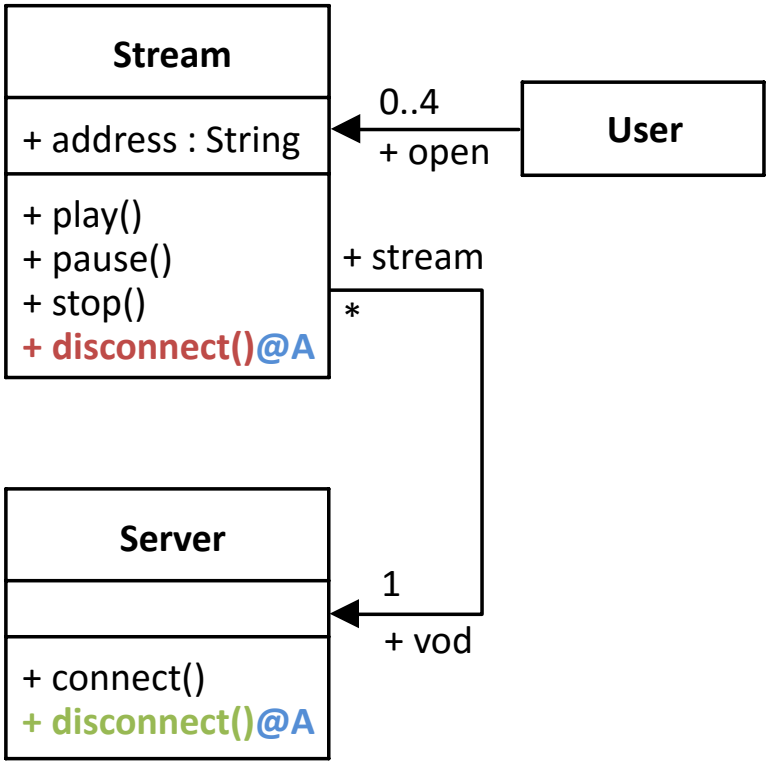
Editierregeln für Modellierungssprachen

- Eine Modellierungssprache wird durch ein Metamodell und zusätzliche **Constraints** beschrieben.
  - Hierdurch wird die Menge aller syntaktisch validen Modellinstanzen hinreichend beschrieben.
- Korrekte Editierschritte werden meist in imperativer Form in so genannten Commands codiert.
  - Blackbox bei der nur die resultierenden Änderungen (z.B. für Undo) beobachtet werden können.
- **Graphtransformationsregeln** bieten ein formales Konzept, um die Veränderung von Graphen zu beschreiben.
  - Editierschritte können deklarativ beschrieben werden.

Editierregeln für Modellierungssprachen

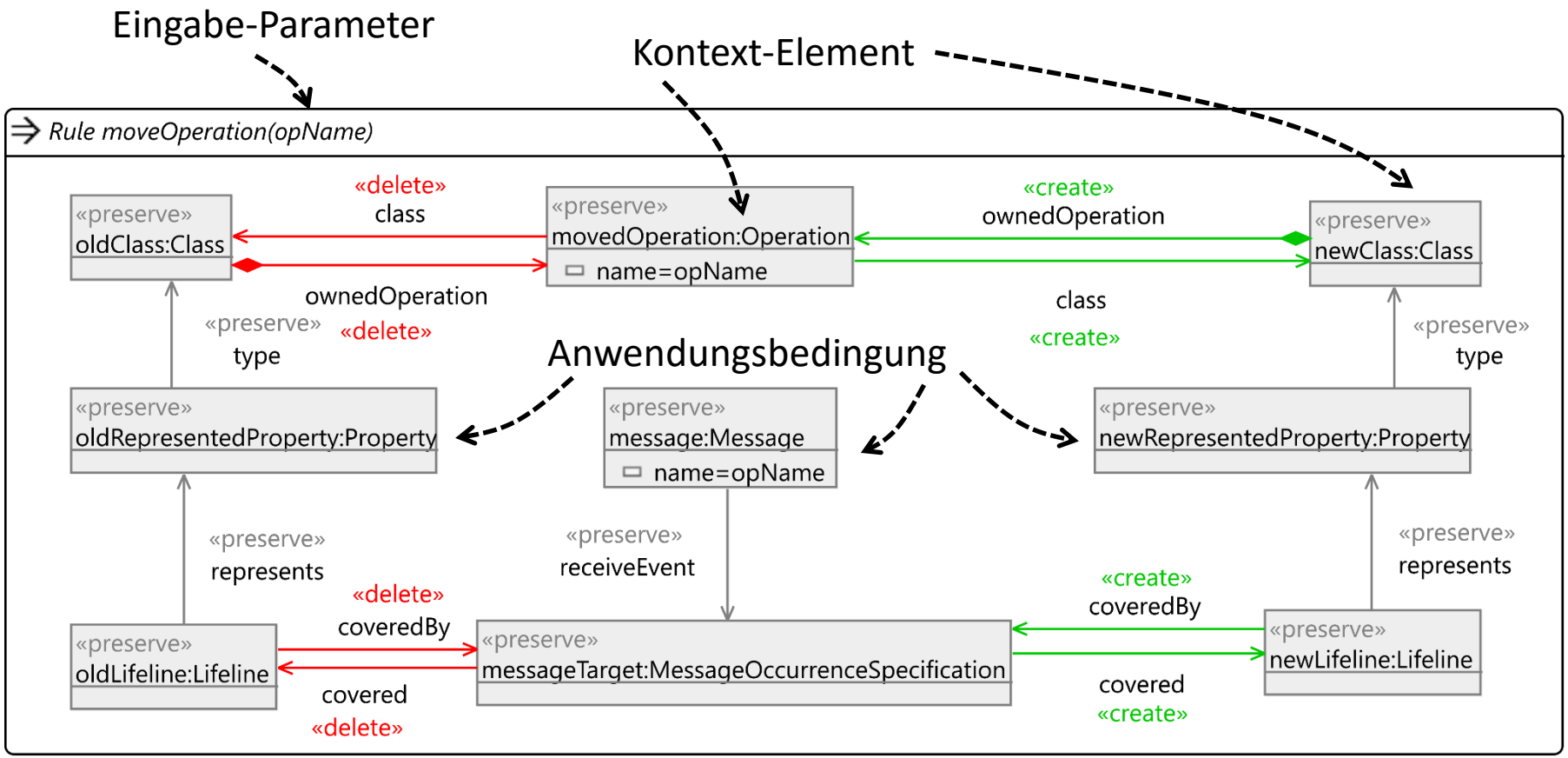
1. Verschiebe eine Operation.
2. Passe den Empfänger der zugehörigen Nachrichten an.





Legende: eingefügt @ Element-ID
gelöscht @ Element-ID

Editierregeln für Modellierungssprachen

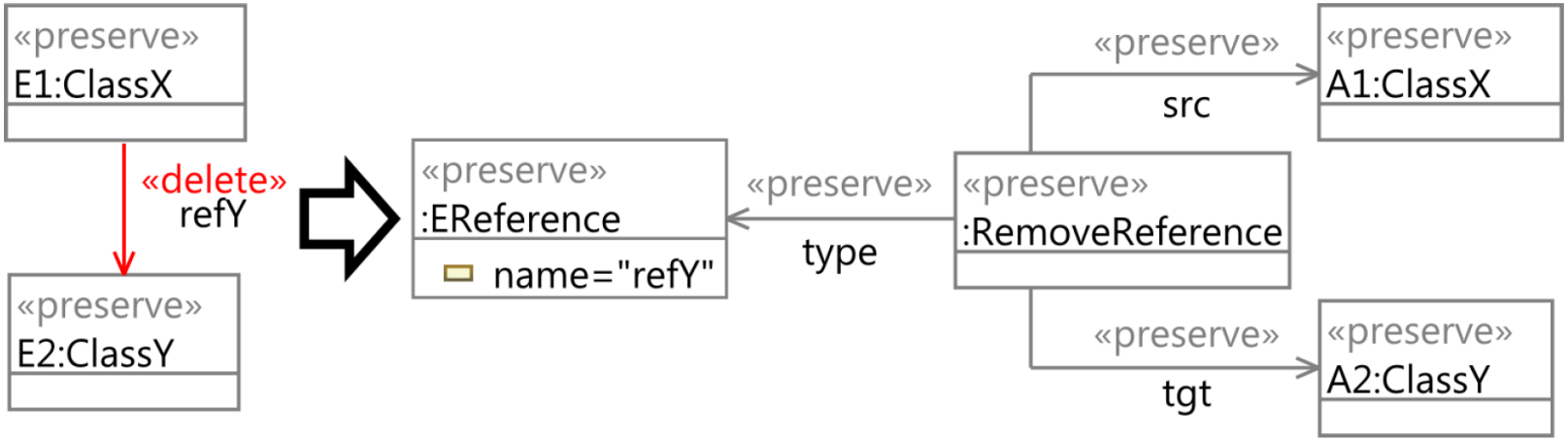


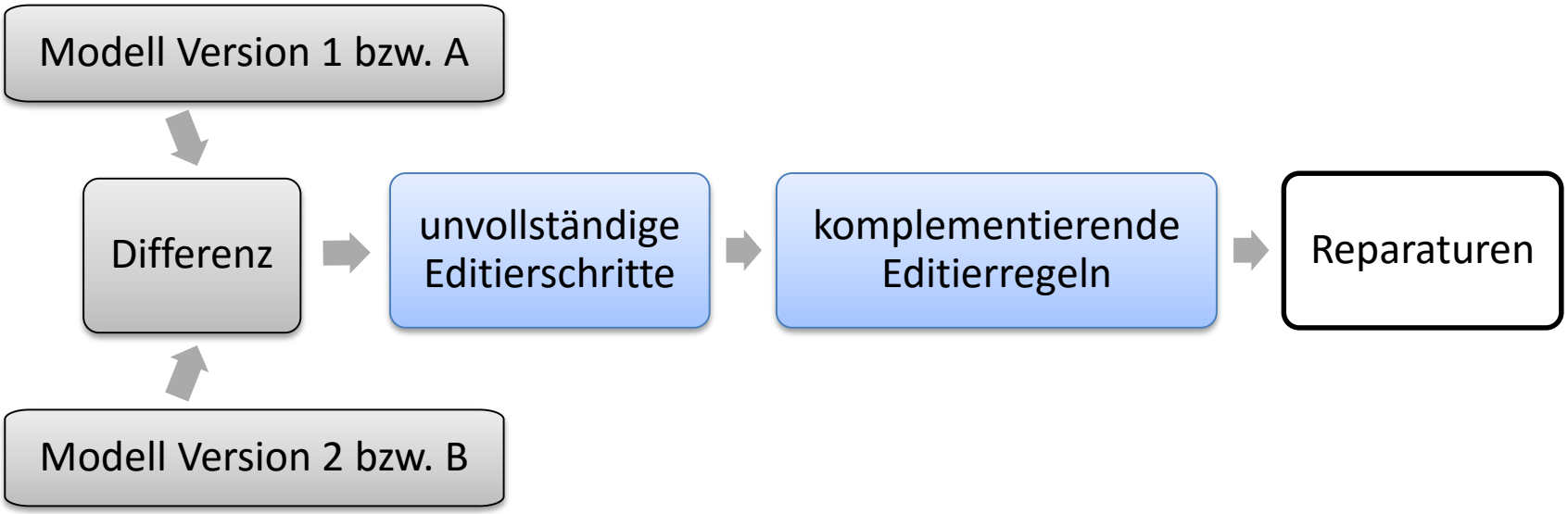
Transformation von Editier- zu Erkennungsregeln

- Jede Editierregel hinterlässt ein Änderungsmuster, in der Modellhistorie ($Modell\ A \rightarrow Modell\ B$).
- Sofern die Differenz zwischen den Modellen genau diese Änderungen wiedergibt, lässt sich die Anwendung der Editierregel aus der Differenz erkennen.
- Kehrer et al. [KKT11] beschreibt die Transformation von Editierregeln in das differenzbasierte Änderungsmuster dieser Regel.
- Dieses Muster wird im folgenden als **Erkennungsregel** RR_x der Editierregel ER_x bezeichnet.

Transformation von Editier- zu Erkennungsregeln

- **Eigenschaften einer Erkennungsregel:**
  - Die Regel erstreckt sich über 3 Modelle:
    - $Modell\ A \leftarrow Differenz \rightarrow Modell\ B$
  - enthält entsprechende Muster für Objekt/Referenz Löschung/Erzeugung und Attributwertänderungen




Modell Version 1 bzw. A
Differenz
Modell Version 2 bzw. B
unvollständige Editierschritte
komplementierende Editierregeln
Reparaturen

Erkennung unvollständiger Editierschritte [MoDELs]

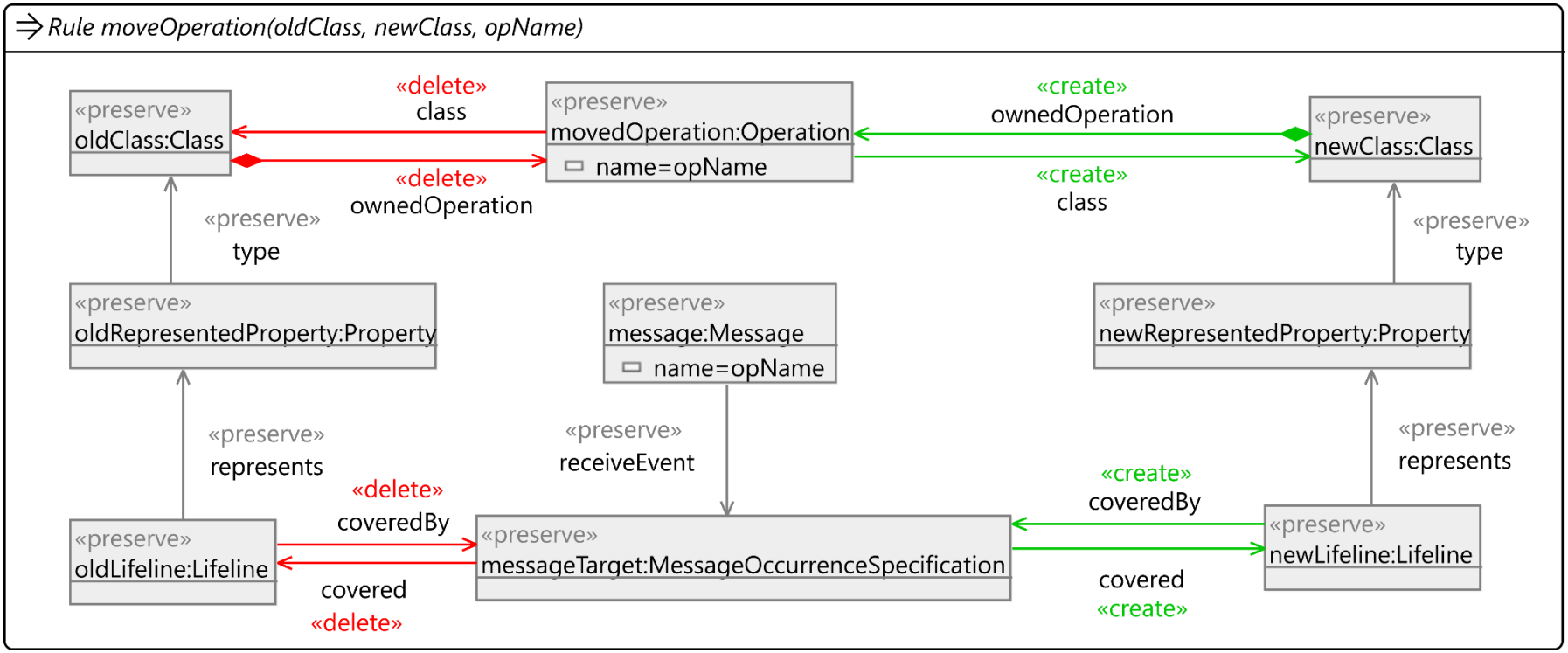
- **Editierregeln mit unterschiedlichem Konsistenzgrad**
  - $Rulebase_{source}$: Enthält alle Editierregeln die bei Anwendung die Konsistenz eines Modells bewahren.
  - $Rulebase_{sub}$: Enthält alle (unvollständigen) Editierregeln die (bei Anwendung) eine Inkonsistenz auslösen könnten.
- **Berechnung:**
  - Erkenne alle Editierschritte $ER_{sub_i/source_i}$ mit $Rulebase_{sub/source}$
  - Filtere alle Matches für ER_{source_i} und $ER_{sub/source} \subseteq ER_{sub/source}$
  - Suche alle Einbettungen $ER_{source_j} \supset ER_{sub_i}$ in $Rulebase_{source}$
  - **Komplement:** $ER_{complement_{j,i}} = ER_{source_j} \,/\, ER_{sub_i}$

Erkennung unvollständiger Editierschritte [FP]

- **Erkennung partieller Editierregeln**
  - $Rulebase_{source}$: Enthält komplexe Editierregeln, welche häufig unvollständig durchgeführt werden und dadurch wiederholt zu Inkonsistenzen führen.
  - $Rulebase_{sub}$ wird nicht benötigt!
- **Berechnung:**
  - Betrachte jeden potentiell unvollständigen Editierschritt $ER_{sub_i} \subset ER_{source_j}$
  - **Komplement:** $ER_{complement_{j,i}} = ER_{source_j} \,/\, ER_{sub_i}$

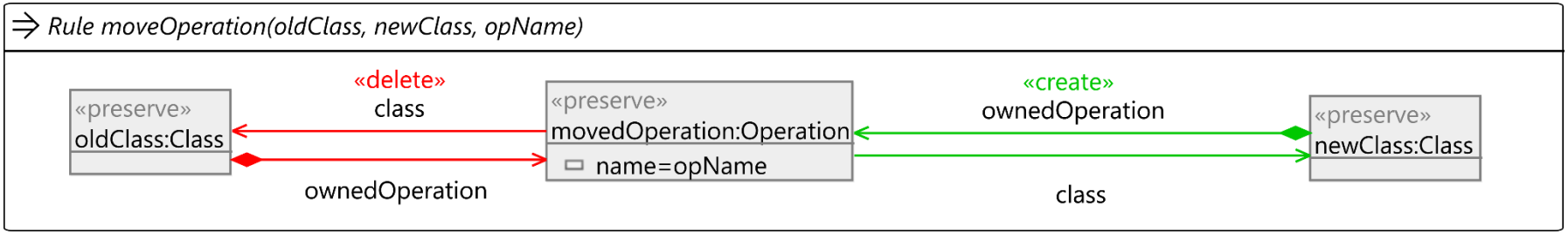
Erkennung unvollständiger Editierschritte

» **Editierregel:** verschiebe Operation, setze Nachrichten-Empfänger



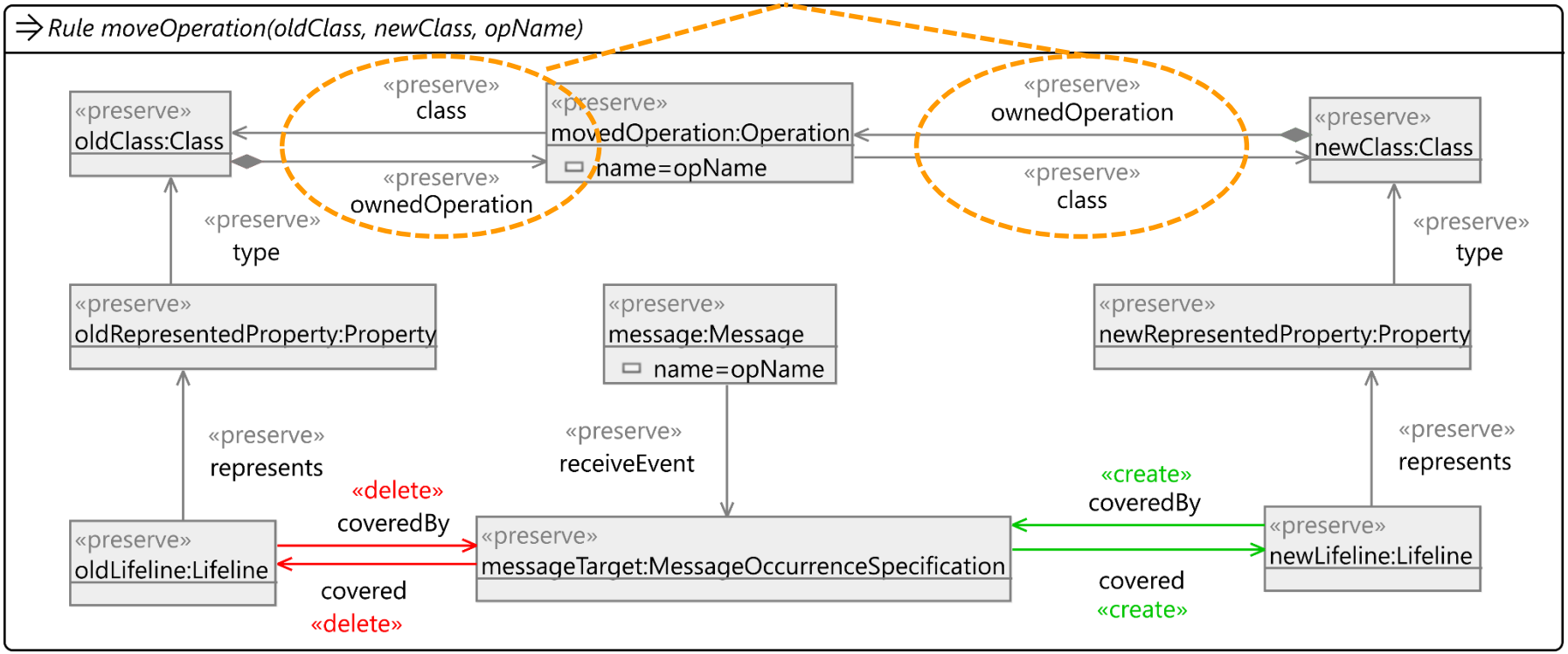
Erkennung unvollständiger Editierschritte

» **Sub-Editierregel:** verschiebe Operation



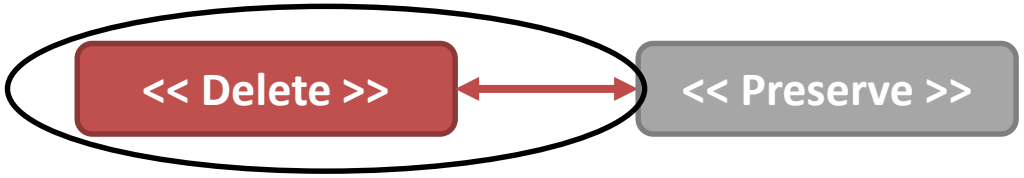
Ableiten der Complement-Rule

- **Complement-Editierregel:** setze Nachrichten-Empfänger
  - „entferne“ die eingebetteten Änderungen der Sub-Editierregel

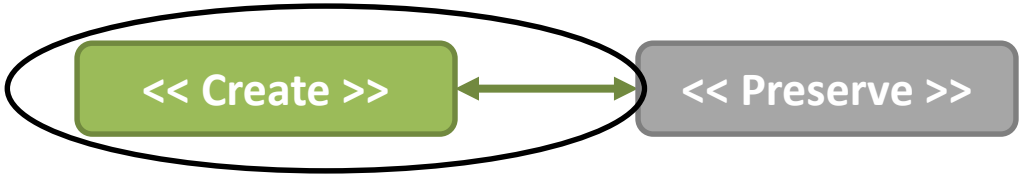


Ableiten der Complement-Rule

- $ER_{complement} = ER_{source} \,/\, ER_{sub}\ wobei\ ER_{sub} \subset ER_{source}$
  - Bereits ausgeführte löschende Änderungen werden aus der Editierregel entfernt.

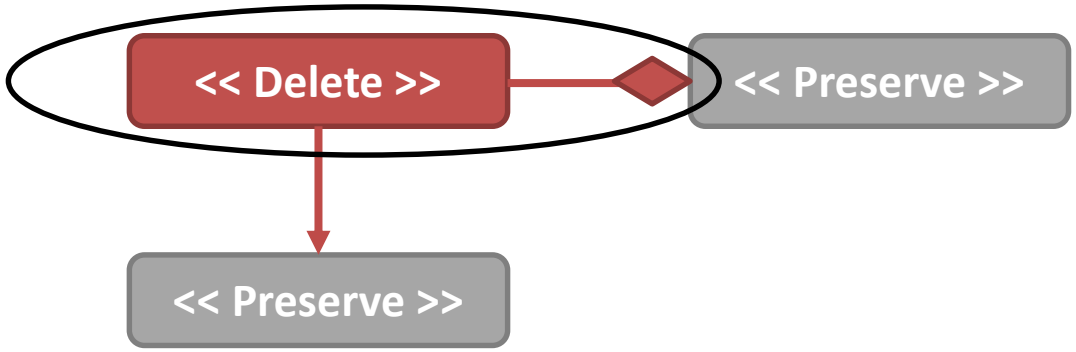


  - Bereits ausgeführte hinzufügende Änderungen werden in Kontext (<< Preserve >>) der Editierregel umgewandelt.

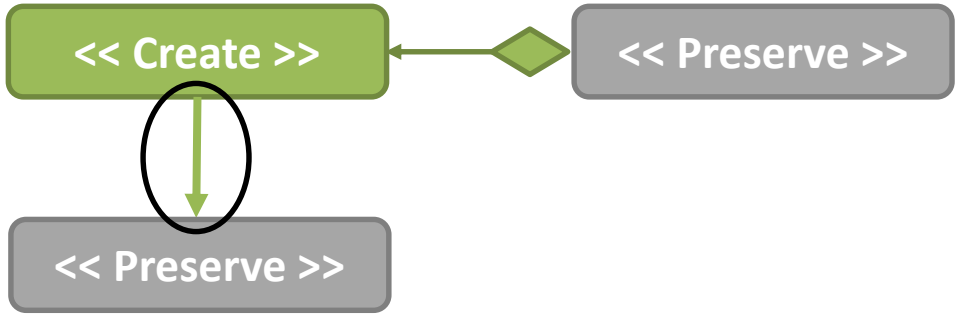


Ableiten der Complement-Rule (transiente Effekte)

» Hängende Kanten werden mit entfernt:

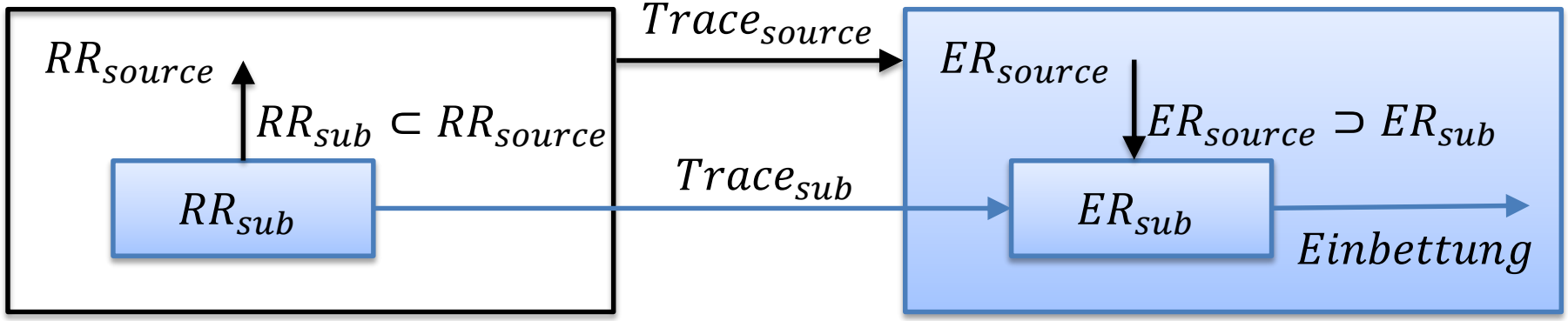


» Knoten inklusive Containment-Referenzen werden ggf. als bereits erzeugt betrachtet:



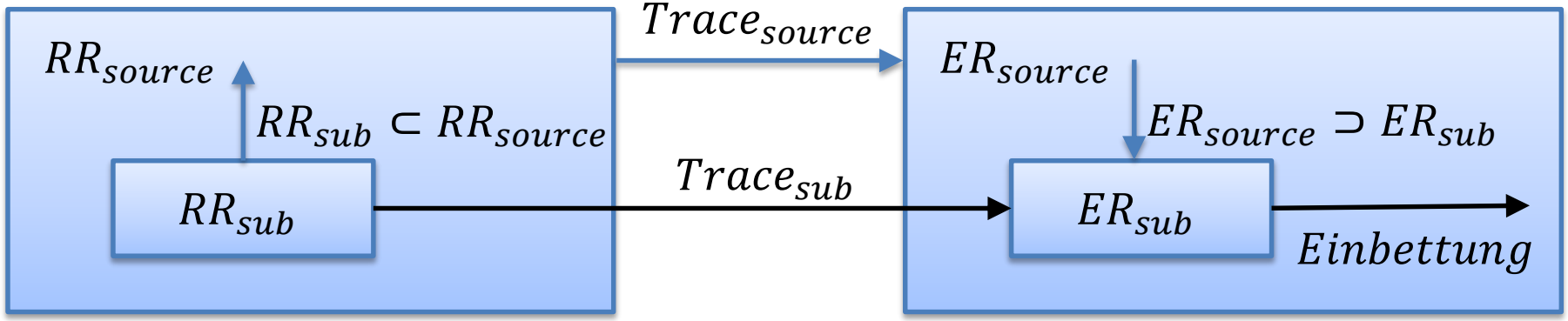
Ableiten der Complement-Rule

- Die Berechnung der Complement-Rule ist bei [MoDELs] und [FP] im wesentlichen identisch, nur die Abbildung von den Erkennungs- auf die Editierregeln unterscheidet sich.
  - **[MoDELs]: $RR_{sub} \rightarrow ER_{sub} \rightarrow Einbettung \rightarrow ER_{source}$**
  - [FP]: $(RR_{sub} \subset RR_{source}) \rightarrow (ER_{source} \supset ER_{sub})$



Ableiten der Complement-Rule

- Die Berechnung der Complement-Rule ist bei [MoDELs] und [FP] im wesentlichen identisch, nur die Abbildung von den Erkennungs- auf die Editierregeln unterscheidet sich.
  - [MoDELs]: $RR_{sub} \rightarrow ER_{sub} \rightarrow Einbettung \rightarrow ER_{source}$
  - **[FP]: $(RR_{sub} \subset RR_{source}) \rightarrow (ER_{source} \supset ER_{sub})$**



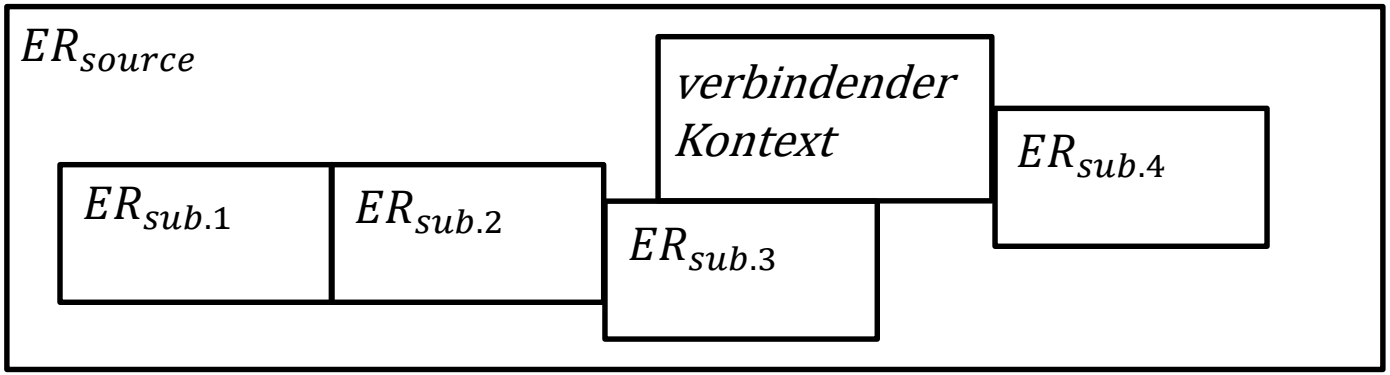
Erkennung unvollständiger Editierschritte

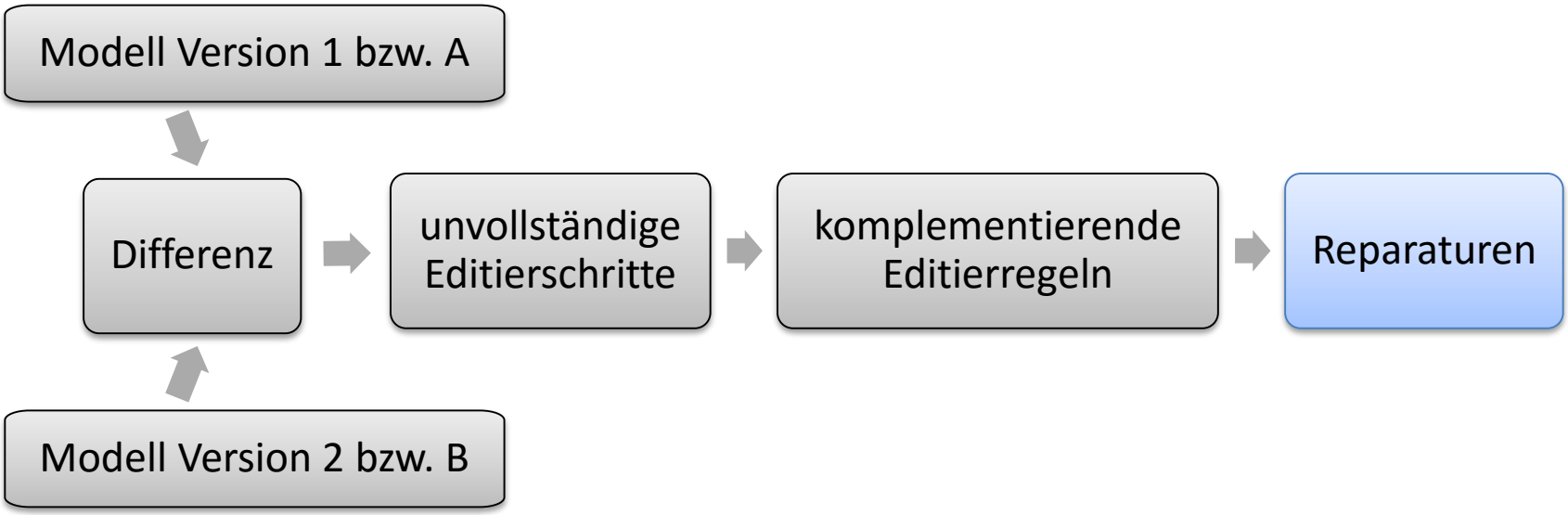
- **Vergleich der beiden Ansätze**
  - **Ansatz [MoDELs]** gibt die Zerlegung von ER_{source} vor:
    - höherer Konfigurationsaufwand
    - zusätzliche Berechnung / Verwaltung der Einbettungen zwischen $ER_{sub_i} \subset ER_{source_j}$
    - kleinerer Lösungsraum (abhängig $Rulebase_{sub}$ Granularität)
      $\Longrightarrow$ z.B. nur Reparaturen zwischen je 2 Sichten
  - **Ansatz [FP]** erzeugt die Zerlegung nach Bedarf:
    - wähle immer die <u>maximal</u> mögliche $ER_{sub_i} \subset ER_{source_j}$
    - Generalisierung des [MoDELs]-Ansatzes
      - $Rulebase_{sub}$ entspricht den low-level Änderungen: einfügen / löschen von Objekten / Referenzen
    - größerer Lösungsraum und im allg. komplexere Berechnung

Erkennung unvollständiger Editierschritte

- **Vergleich der beiden Ansätze**
  - **[MoDELs]:** Falls mehrere $ER_{sub.i}$ in eine ER_{source} eingebettet werden können, so muss für alle enthaltenen Sub-Regeln $\{ER_{sub.i}, \ldots, ER_{sub.n}\}$, $(2^n - n)$ weitere Sub-Regeln existieren.
  - Potenzmenge der Sub-Regeln + ggf. verbindenden Kontext

  ⇒ Würde sonst ggf. zu einer Reparatur pro $ER_{sub.i}$ führen.




Modell Version 1 bzw. A
Differenz
Modell Version 2 bzw. B
unvollständige Editierschritte
komplementierende Editierregeln
Reparaturen

Berechnung der Reparaturen

- **Eingabe:**
  - Eine komplementierende Editierregel.
  - Einen eindeutigen Match der ER_{sub}, für den Kontext der historischen bzw. bereits ausgeführten Änderungen.
- **Gesucht:**
  - Alle vollständigen LHS Matches der komplementierenden ER.

Berechnung der Reparaturen

- **[MoDELs]-Ansatz:**
  - Die Erkennung einer EO_{sub} impliziert eine Inkonsistenz.
  - ⇒ Jede komplementierende Editierregel, welche auf Modell B anwendbar ist, stellt eine Reparatur dar.
- **[FP]-Ansatz:**
  - Wird der Algorithmus zur Erkennung partieller Editieroperationen auf eine Differenz angewendet, so ergeben sich alle möglichen komplementierenden Editieroperation.
  - Ab wann führt ein Editierschritt, der als unvollständig bzgl. einer bestimmten Editierregel erkannt wurde, zu einer Inkonsistenz?
  - *Intuition:* Eine Reparatur muss eine Änderung durchführen, so dass sich die Validierung der Konsistenzregel partiell verbessert.

Editierregel-Filter

- **potential consistency improving:**
  - Die Editierregel enthält Änderungen, welche Eigenschaften (Referenzen / Attribute bzgl. des Metamodells) des Modells verändern, an denen die Konsistenzregel scheitert.
- **consistency improving:**
  - Mindestens eine Änderungen der Editierregel hat einen positiven Effekt auf die Validierung der Konsistenzregel.
- **consistency restoring:**
  - Eine Reparatur stellt die Konsistenz, bzgl. einer Inkonsistenz, vollständig wieder her.

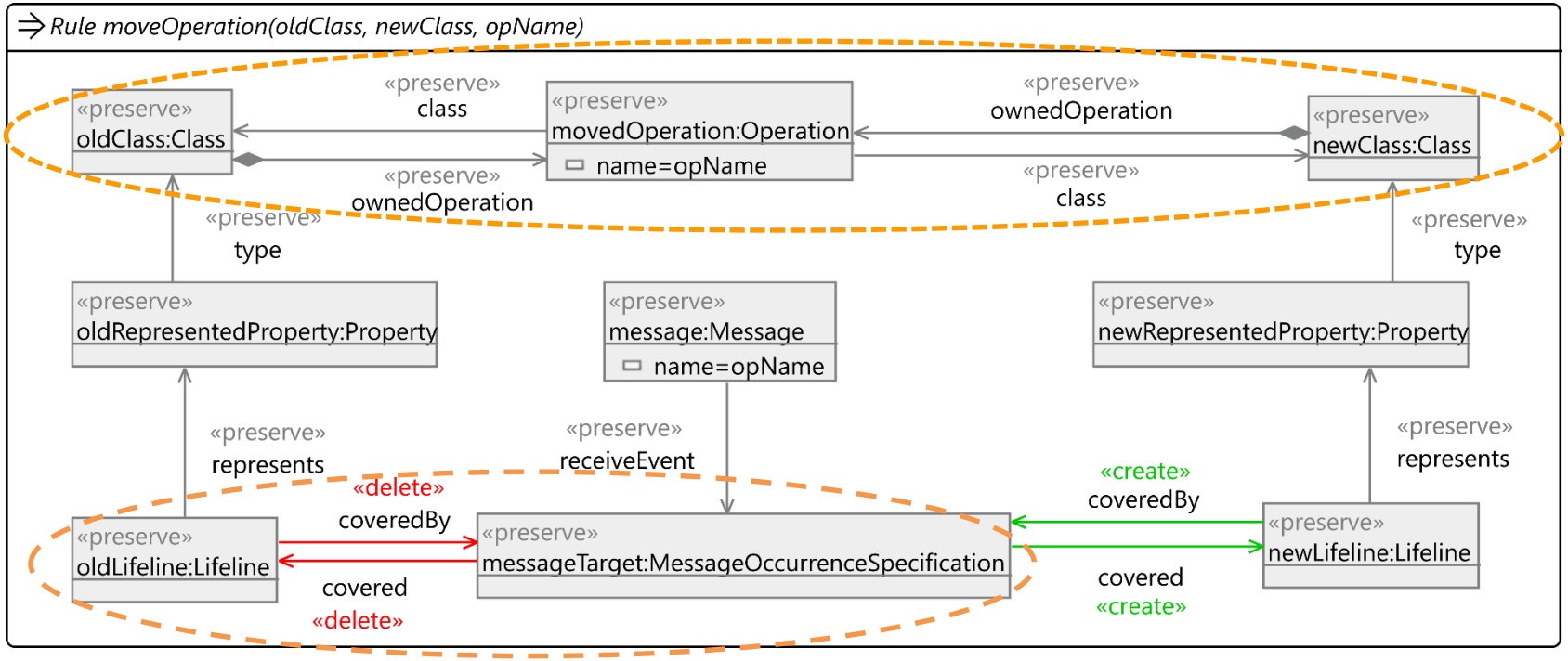
Editierregel-Filter

- Zunächst werden alle **abstrakten Reparaturen** bzgl. einer oder mehrerer Inkonsistenzen berechnet.
- Jede Änderung einer abstrakten Reparatur (R) oder einer Editierregel (ER) lässt sich identifizieren bzw. zuordnen:
  - $\ddot{A}nderung_{ER/R} < Add/Delete/Modify,$ ***Kontext-Element***$, Feature\text{-}Typ >$
  - $\ddot{A}nderungstype_{ER/R} < Add/Delete/Modify,$ ***Kontext-Type***$, Feature\text{-}Typ >$

- **Level 1:** Betrachte nur Editierregeln welche ***potential consistency improving*** sind.
  $\Rightarrow \{Menge\ aller\ ER\ \ddot{A}nderungstypen\} \cap \{Menge\ aller\ R\ \ddot{A}nderungstypen\} \neq \emptyset$
- **Level 2:** analog zu Level 1 bzgl. komplementierender ER
- **Level 3:** Nur komplementierende ER, welche bei Anwendung ***consistency improving*** sind.
  $\Rightarrow \{Menge\ aller\ ER\ \ddot{A}nderungen\} \cap \{Menge\ aller\ R\ \ddot{A}nderungen\} \neq \emptyset$

Berechnung der Reparaturen

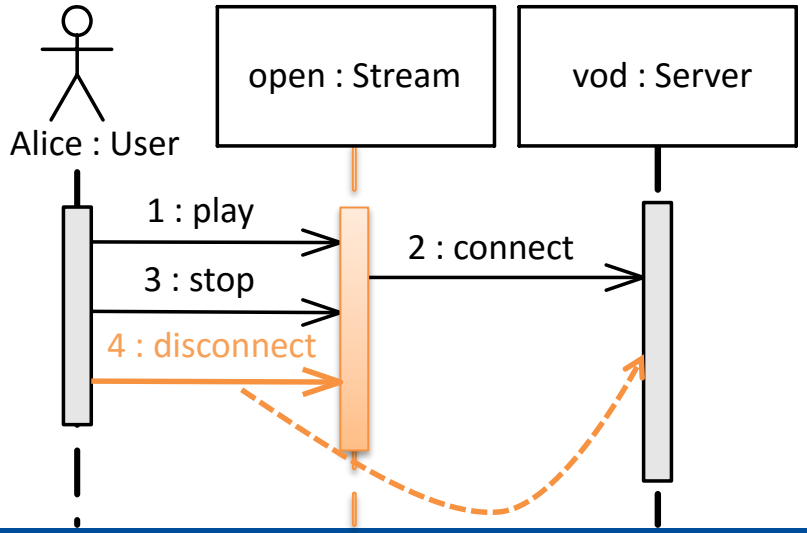
1. Setzte den Sub-E.R.-Match als Pre-Match für die Complement-Rule.
2. **Reparatur:** Berechne alle vollständigen Matches.
3. **Filter:** Suche nach Überschneidung mit den abstrakten Reparaturen.

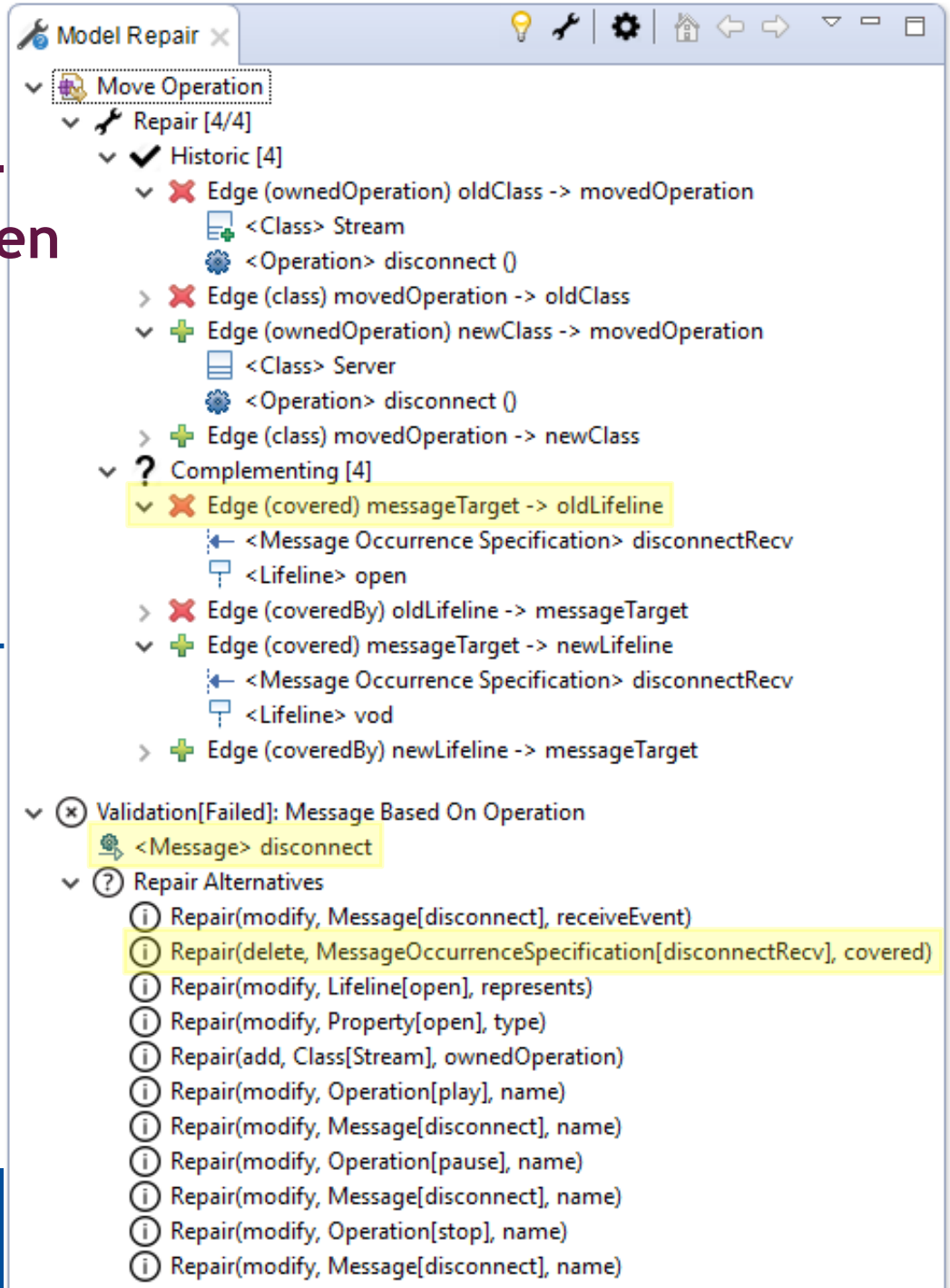


Berechnung der Reparaturen

$Message\ m:\ \forall l \in m.receiveEvent.covered:$
$\exists o \in l.represents.type.ownedOperation:$
$o.name = m.name$ [RE12]

$\Rightarrow z.B.\ lasse\ die\ Nachricht\ von$
$'vod:Server'\ empfangen$





Bewertung der Reparaturen

- Verwandte zustandsbasierten Verfahren suchen meist nach einer Reparatur, welche das Modell minimal modifiziert, um die Konsistenz wieder herzustellen.
  - Ggf. werden noch weitere Kriterien mit einbezogen.
- **historienbasiert Bewertung:** Verhältnis zwischen der Anzahl der historischen und der Anzahl der komplementierenden Änderungen. z.B.:
  - EO-1: historisch (3) ÷ komplementierend (2) = 1,5
  - EO-2: historisch (6) ÷ komplementierend (5) = 1,2
  - EO-3: historisch (3) ÷ komplementierend (3) = 1
  - EO-4: historisch (4) ÷ komplementierend (6) = 0,7

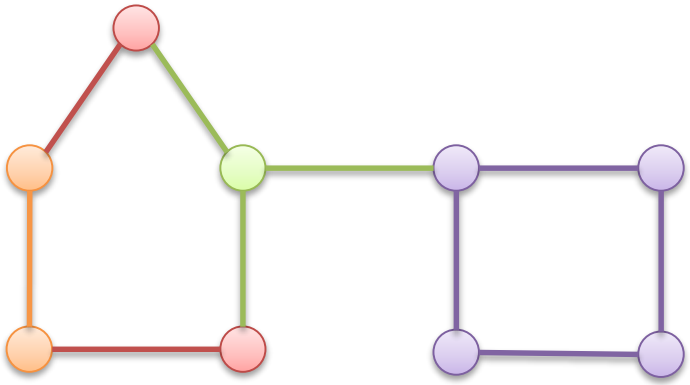
Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

ERKENNUNG UNVOLLSTÄNDIGER EDITIERSCHRITTE

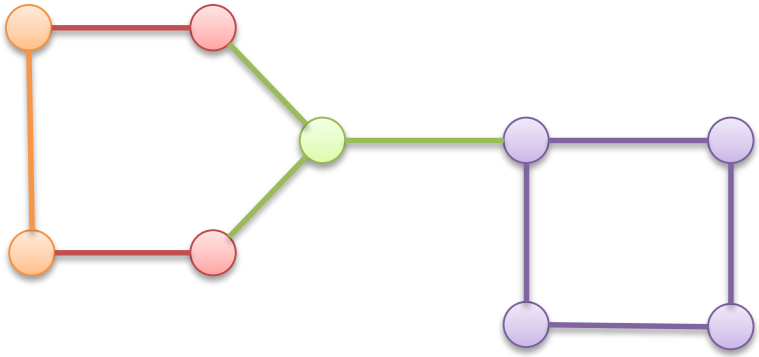
Graphentheorie

- Abbildung zwischen zwei Graphen P und W
  - Graph-Isomorphie: 1 zu 1 Zuordnung zwischen Knoten und Kanten

P:



W:

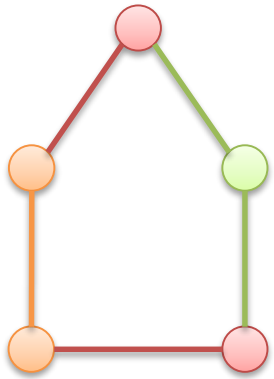


[Vismara, Finding Maximum Common Connected Subgraphs Using Clique Detection or Constraint Satisfaction Algorithms, 2008]

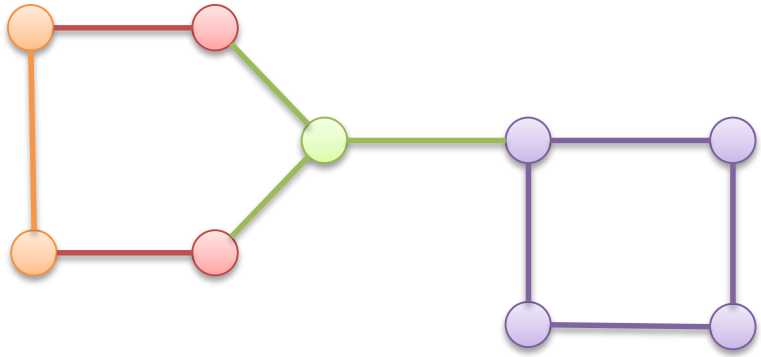
Graphentheorie

- Abbildung zwischen zwei Graphen P und W
  - Graph-Isomorphie: 1 zu 1 Zuordnung zwischen Knoten und Kanten
  - Sub-Graph-Isomorphie: Finde Graph P als Teil-Graph in W

P:



W:

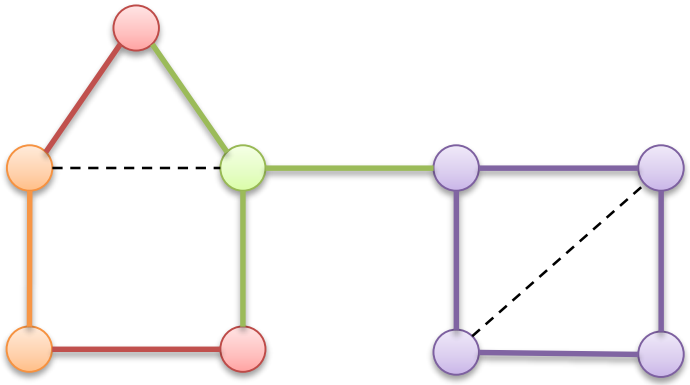


[Vismara, Finding Maximum Common Connected Subgraphs Using Clique Detection or Constraint Satisfaction Algorithms, 2008]

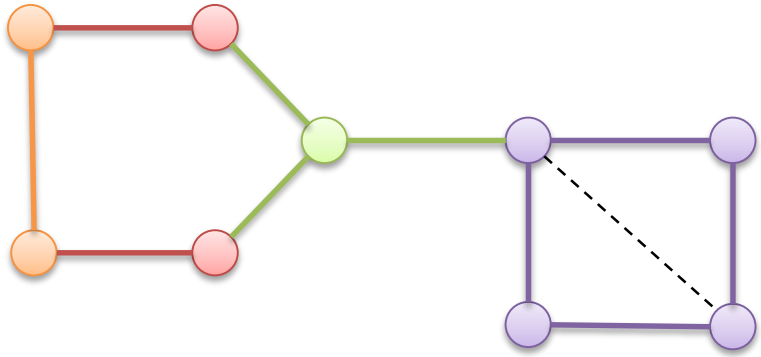
Graphentheorie

- Abbildung zwischen zwei Graphen P und W
  - Graph-Isomorphie: 1 zu 1 Zuordnung zwischen Knoten und Kanten
  - Sub-Graph-Isomorphie: Finde Graph P als Teil-Graph in W
  - **Maximum Common Subgraph (MCS):** Finde Teil-Graph von P in W
    - Eigenschaften: Connected Sub-Graph, Induced Sub-Graph

P:



W:

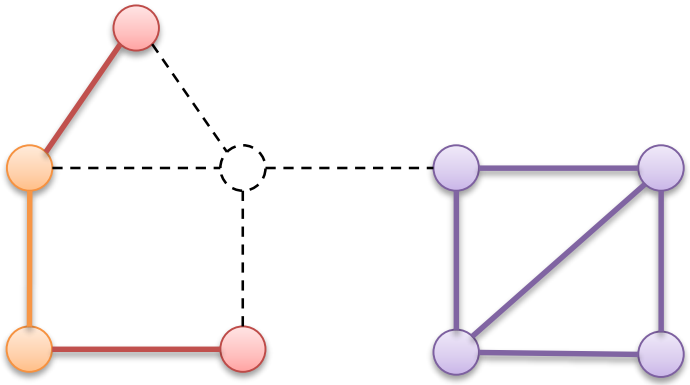


[Vismara, Finding Maximum Common Connected Subgraphs Using Clique Detection or Constraint Satisfaction Algorithms, 2008]

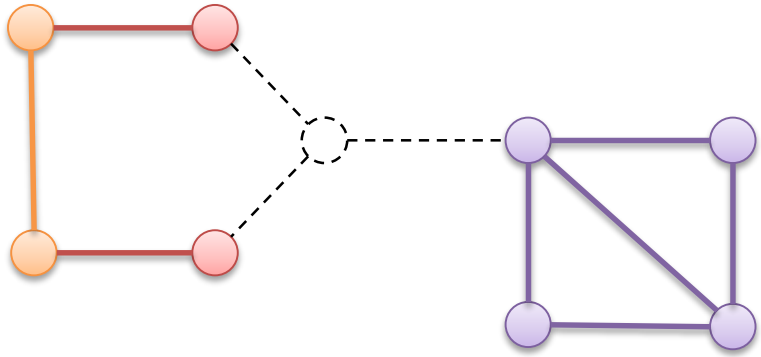
Graphentheorie

- Abbildung zwischen zwei Graphen P und W
  - Graph-Isomorphie: 1 zu 1 Zuordnung zwischen Knoten und Kanten
  - Sub-Graph-Isomorphie: Finde Graph P als Teil-Graph in W
  - **Maximum Common Subgraph (MCS):** Finde Teil-Graph von P in W
    - Eigenschaften: Connected Sub-Graph, Induced Sub-Graph

P:



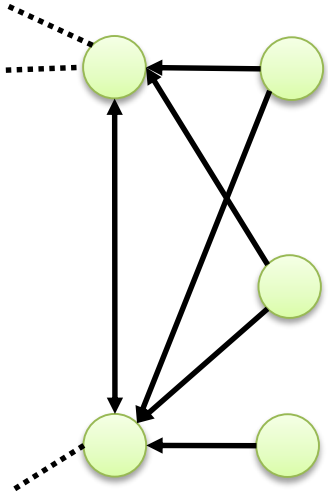
W:

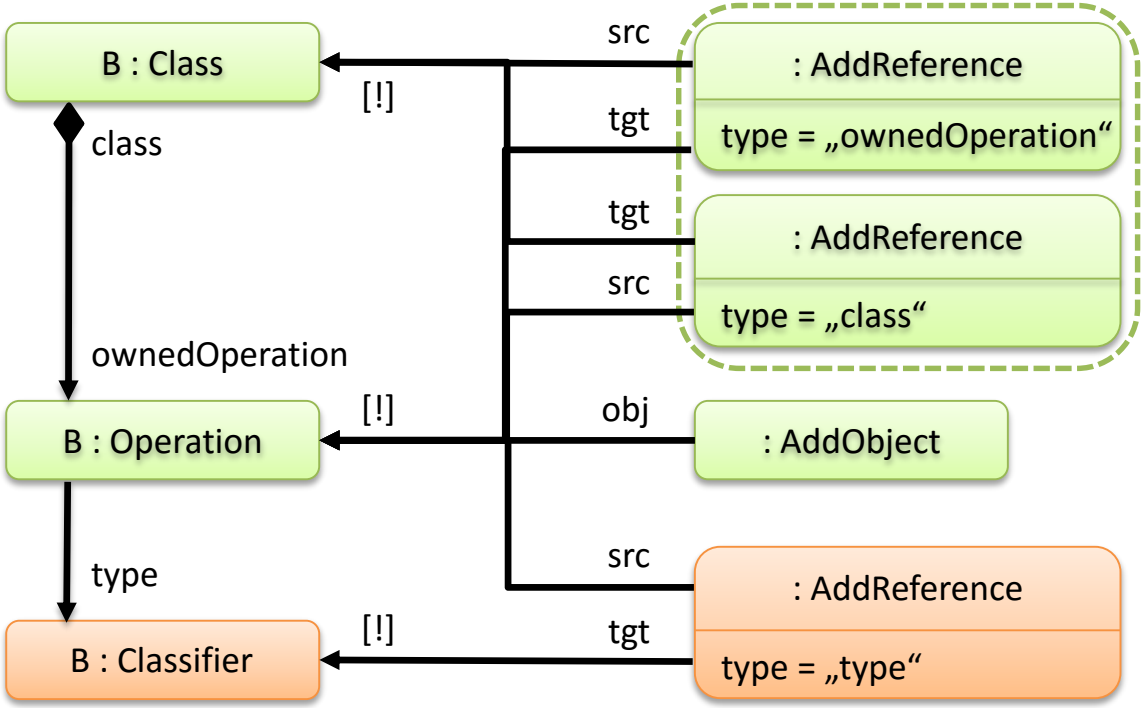


[Vismara, Finding Maximum Common Connected Subgraphs Using Clique Detection or Constraint Satisfaction Algorithms, 2008]

Partielles Pattern Matching - DSL

- **Arbeitsgraph W ↔ partieller Muster-Graph P' ↔ Muster-Graph P**
  - **Atomic-Patterns:** Knoten die nur gemeinsam gematcht werden dürfen.
  - **Mandatory-Edges:** Kanten [!] die nur zusammen mit dem Knoten aus dem Pattern entfernt werden dürfen.





Partielles Pattern Matching

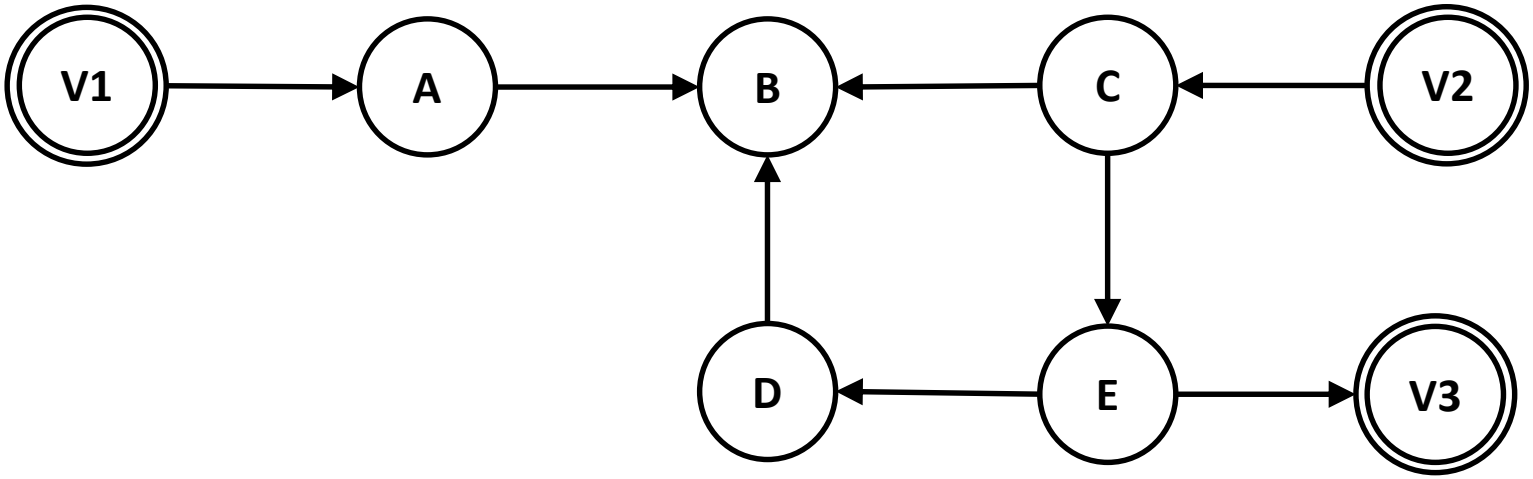
- **Gesucht:** alle maximalen Lösungen: $\mathbf{F} = \boldsymbol{v_1} \times \cdots \times \boldsymbol{v_n}$
- **Erkennungsregel / Änderungsmuster**
  - Variablen ≡ Änderungen ⇒ CS $= change_1 \times \cdots \times change_n$
  - verbindende Knoten ≡ Editierregel-Kontext/PAC
- **Muster-Graph** $\mathbf{G} = (\mathbf{V}, \mathbf{P}, \mathbf{E})$
  - **V-Knoten:** ein Knoten pro Variable in $\mathrm{F} = v_1 \times \cdots \times v_n$
  - **P-Knoten:** Knoten der V-Knoten verbindet
  - **Kanten** $\boldsymbol{E}$: $\{V, P\} \times \{V, P\}$
- **Arbeitsgraph W**
  - $\mathrm{W} = \{v_0 \rightarrow \{\mathbb{O}_{0.0}, \ldots, \mathbb{O}_{0.m}\}, \ldots, v_n \rightarrow \{\mathbb{O}_{n.0}, \ldots, \mathbb{O}_{n.m}\}, p_{n+1} \rightarrow \cdots\}$

Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

AUFBAUEN DES ARBEITSGRAPHEN

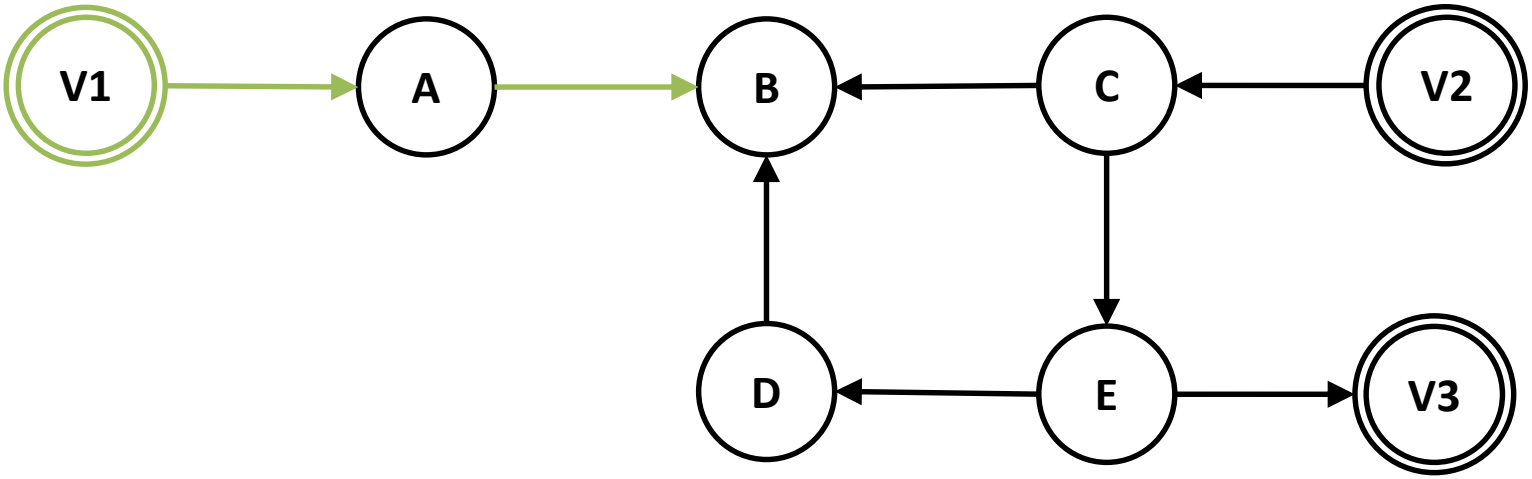
Aufbauen des Arbeitsgraphen

- **Arbeitsgraph W**
  - $W = \{v_0 \rightarrow \{@_{0.0}, \dots, @_{0.m}\}, \dots, v_n \rightarrow \{@_{n.0}, \dots, @_{n.m}\}, p_{n+1} \rightarrow \cdots\}$
- **Problem:**
  - Die Editierregel wird ausgehend von den Änderungen gesucht.
  - Modelle können nicht navigierbare Referenzen besitzen.
  - Man möchte vermeiden, das gesamte Modell abzusuchen, um bestimmte inzidente Referenzen eines Objekts zu finden.



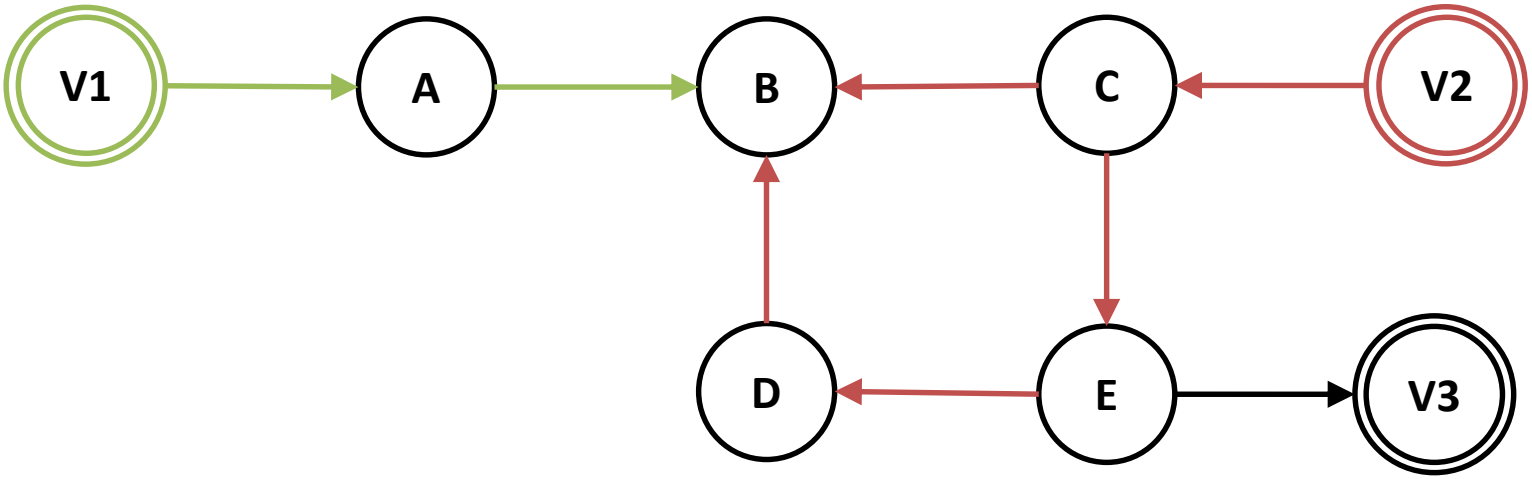
Muster-Optimierung

- **Gesucht:** Alle paarweisen Verbindungen zwischen V-Knoten.
- **Simulation:** Tiefensuche ausgehend von allen V-Knoten.
  - Pfade werden (falls möglich) aus beiden Richtungen beschritten.
  - Falls nötig, muss eine Kante „umgedreht“ werden.
  - Die Wahl einer inversen Kante ist nicht immer eindeutig.



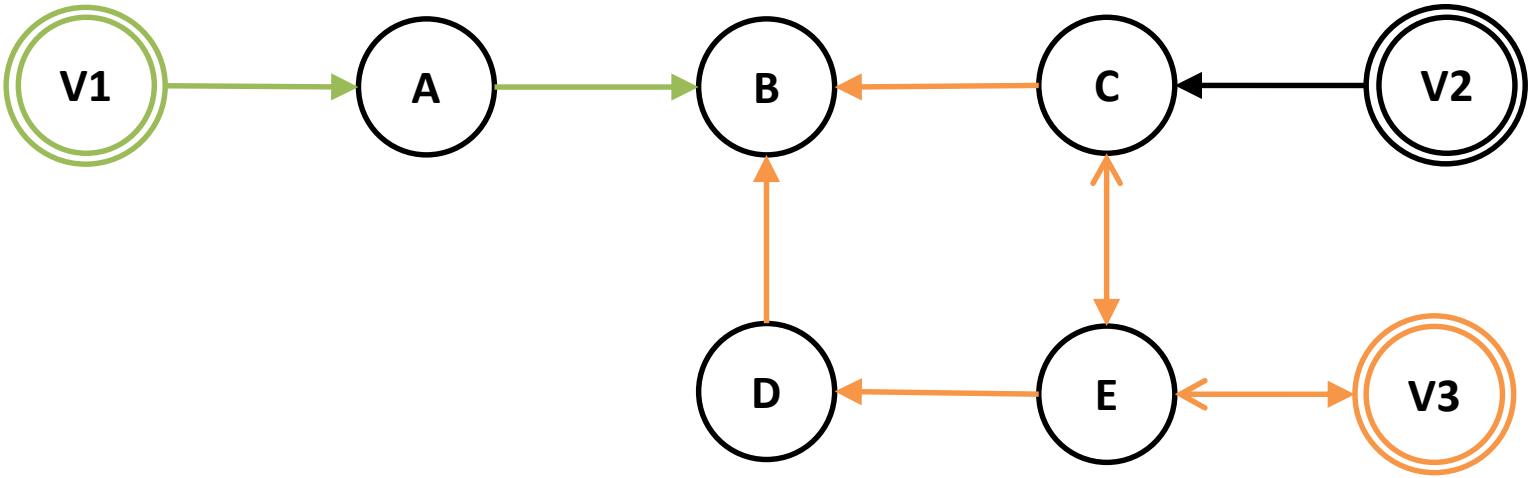
Muster-Optimierung

- **Gesucht:** Alle paarweisen Verbindungen zwischen V-Knoten.
- **Simulation:** Tiefensuche ausgehend von allen V-Knoten.
  - Pfade werden (falls möglich) aus beiden Richtungen beschritten.
  - Falls nötig, muss eine Kante „umgedreht“ werden.
  - Die Wahl einer inversen Kante ist nicht immer eindeutig.



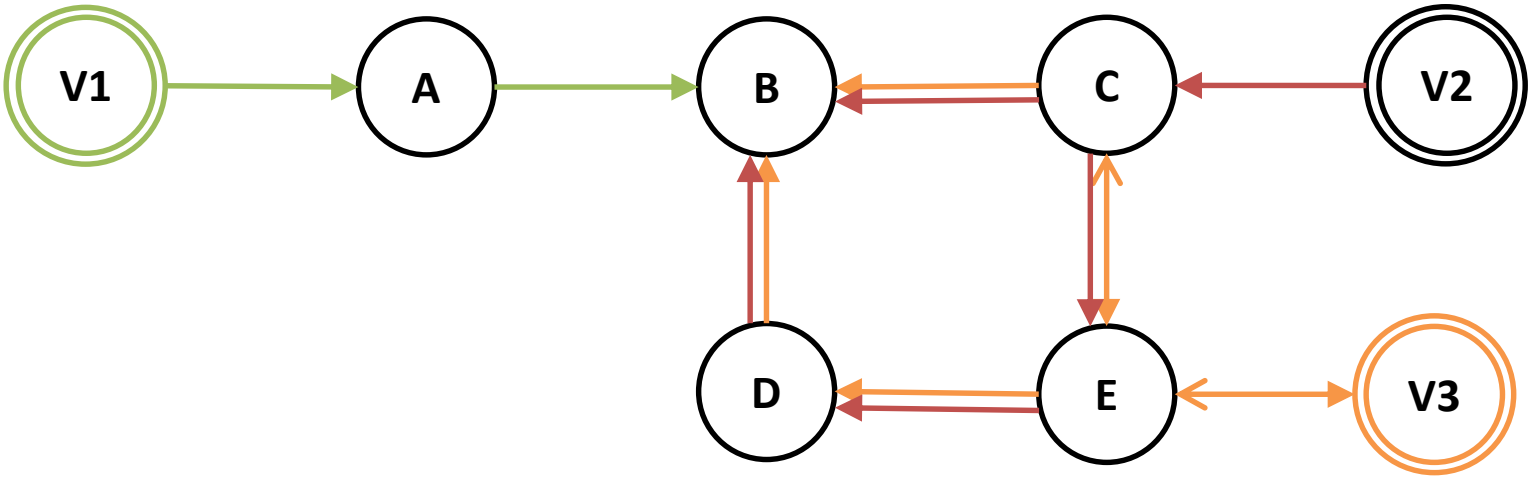
Muster-Optimierung

- **Gesucht:** Alle paarweisen Verbindungen zwischen V-Knoten.
- **Simulation:** Tiefensuche ausgehend von allen V-Knoten.
  - Pfade werden (falls möglich) aus beiden Richtungen beschritten.
  - Falls nötig, muss eine Kante „umgedreht“ werden.
  - Die Wahl einer inversen Kante ist nicht immer eindeutig.



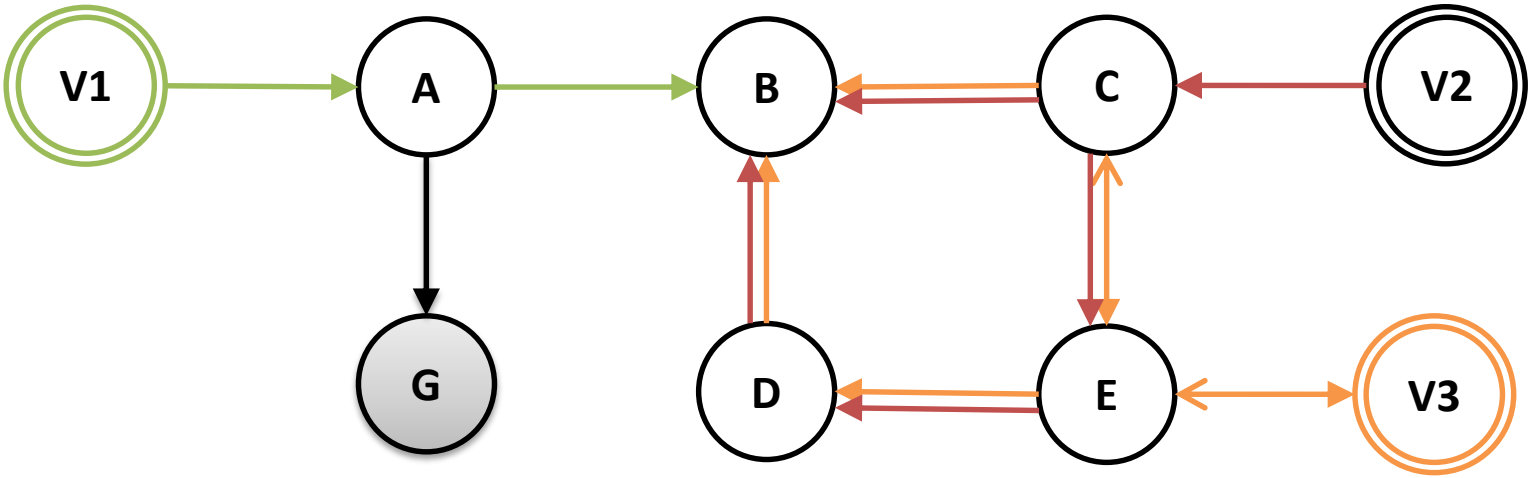
Aufbauen des Arbeitsgraphen

- **Arbeitsgraph W**
  - $W = \{v_0 \rightarrow \{\mathbb{O}_{0.0}, \dots, \mathbb{O}_{0.m}\}, \dots, v_n \rightarrow \{\mathbb{O}_{n.0}, \dots, \mathbb{O}_{n.m}\}, p_{n+1} \rightarrow \cdots\}$
- **Berechnung:** Ausgehend von jedem V-Knoten V_i werden alle Objekte auf Pfade des Teil-Graphen $M_i \subseteq M$ aufgesammelt.



Aufbauen des Arbeitsgraphen

- **Arbeitsgraph W**
  - $W = \{v_0 \rightarrow \{\mathbb{O}_{0.0}, \dots, \mathbb{O}_{0.m}\}, \dots, v_n \rightarrow \{\mathbb{O}_{n.0}, \dots, \mathbb{O}_{n.m}\}, p_{n+1} \rightarrow \dots\}$
- Zweige des Patterns, welche keine Pfade zwischen V-Knoten bilden, werden nicht in den Arbeitsgraphen aufgenommen.
  - Solche Knoten müssen ggf. nachträglich ermittelt werden.



Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

MATCHING-ALGORITHMUS

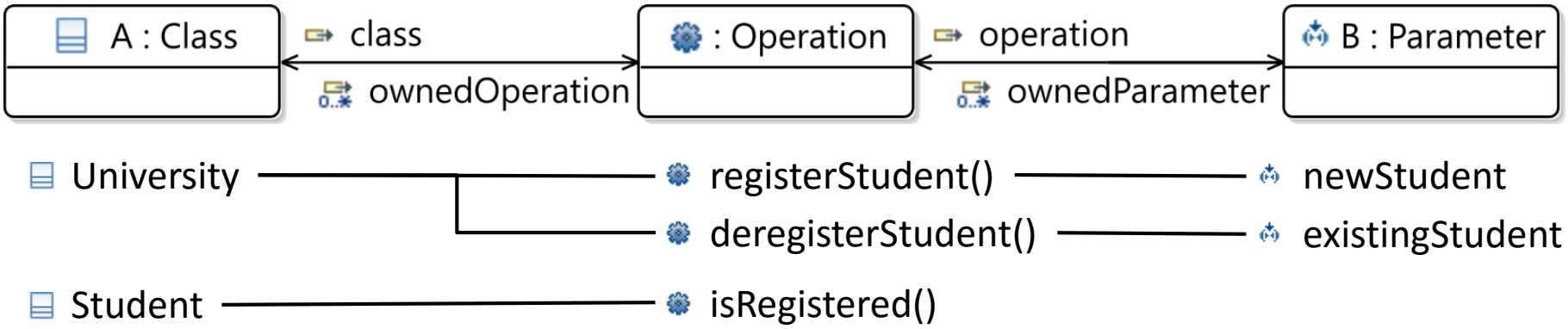
Matchingalgorithmus - Iteration des Lösungsraums

- Es wird nach allen maximalen (d.h. nicht mehr erweiterbaren) Variablen-Belegungen gesucht: $\mathbf{F} = \boldsymbol{v_1} \times \cdots \times \boldsymbol{v_n}$
- Die Lösungen werden ausgehend von jedem V-Knoten (v_{inital}) der Reihe nach betrachtet.
- Betrachte für v_{inital} nur noch nicht zugewiesene Objekte.
- ($v_{i<initial} = \emptyset$) Forderung verletzt $\Longrightarrow$ duplizierte Belegung

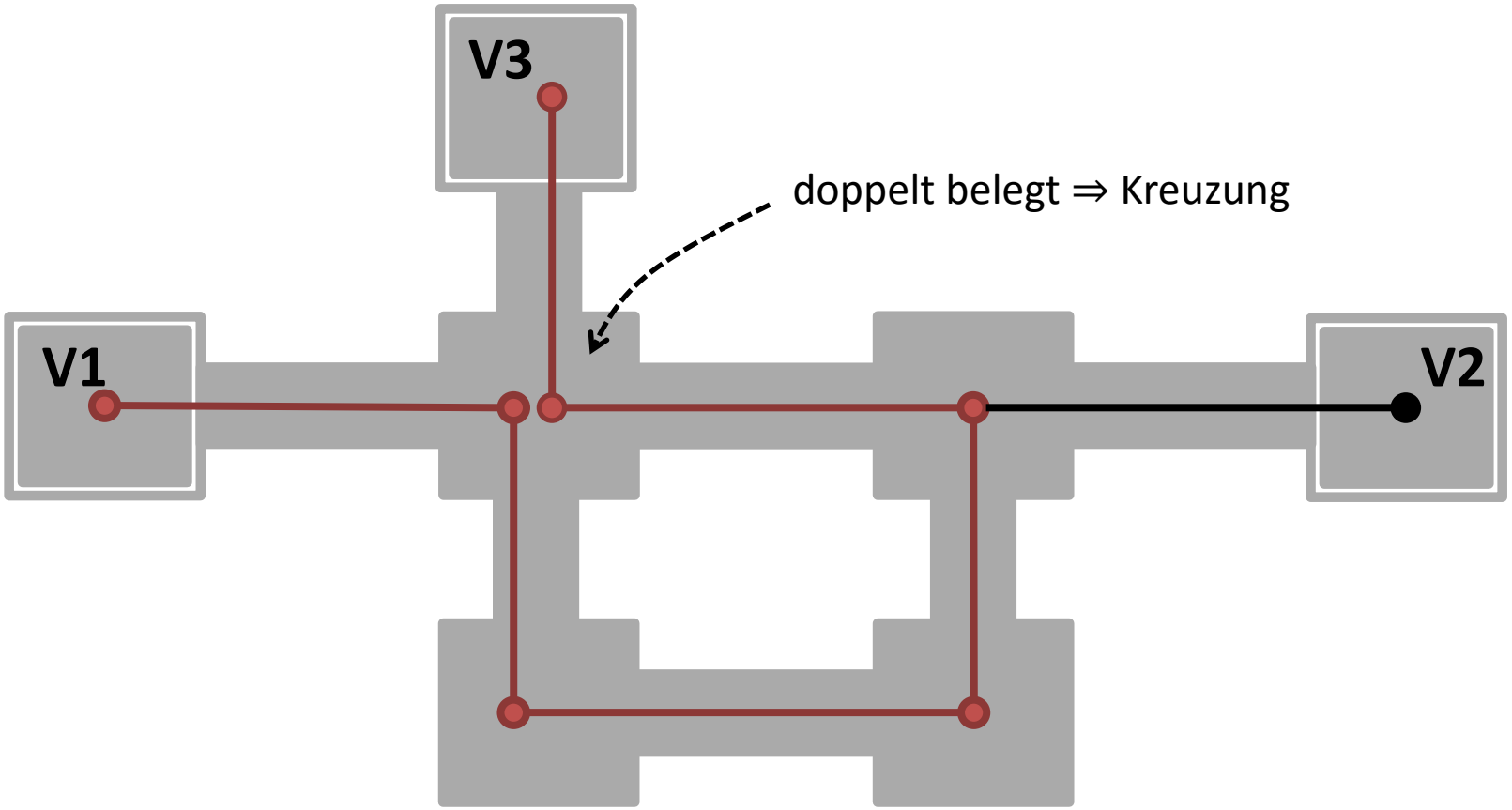
v_1	v_2	v_3	v_4	v_5	v_6	
$\mathbb{0}_{\mathbf{1.1}}$	$\mathbb{0}_{2.3}$	$\mathbb{0}_{3.2}$	$\mathbb{0}_{4.1}$			
$\mathbb{0}_{\mathbf{1.1}}$	$\mathbb{0}_{2.2}$			$\mathbb{0}_{5.1}$		
$\emptyset$	$\mathbb{0}_{\mathbf{2.1}}$	$\mathbb{0}_{3.1}$				
$\emptyset$	$\emptyset$	$\emptyset$	$\mathbb{0}_{\mathbf{4.2}}$	$\mathbb{0}_{5.1}$	$\mathbb{0}_{6.1}$	
$\emptyset$	$\emptyset$	$\emptyset$	$\mathbb{0}_{\mathbf{4.2}}$	$\mathbb{0}_{5.1}$		← Teilmengen
$\emptyset$	$\emptyset$	$\emptyset$	$\mathbb{0}_{\mathbf{4.1}}$	$\mathbb{0}_{5.1}$		← Überschneidungen

Matchingalgorithmus – Pfad-Matching

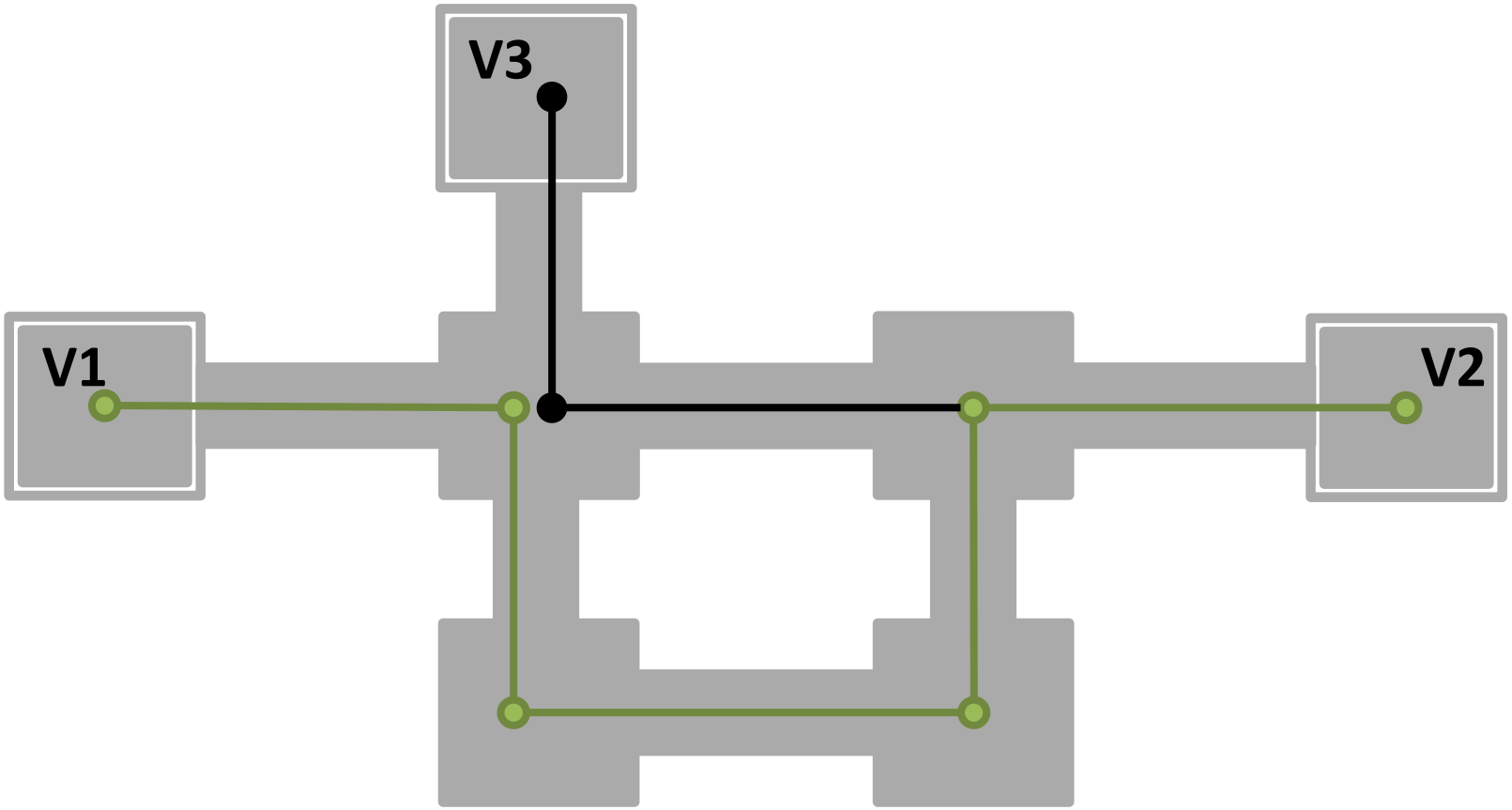
- **Isomorpher (Teil-)Pfad** zwischen Knoten A und B:
  - Ein Pfad darf geschlossen werden.
  - Ein Pfad darf sich nicht selbst kreuzen.



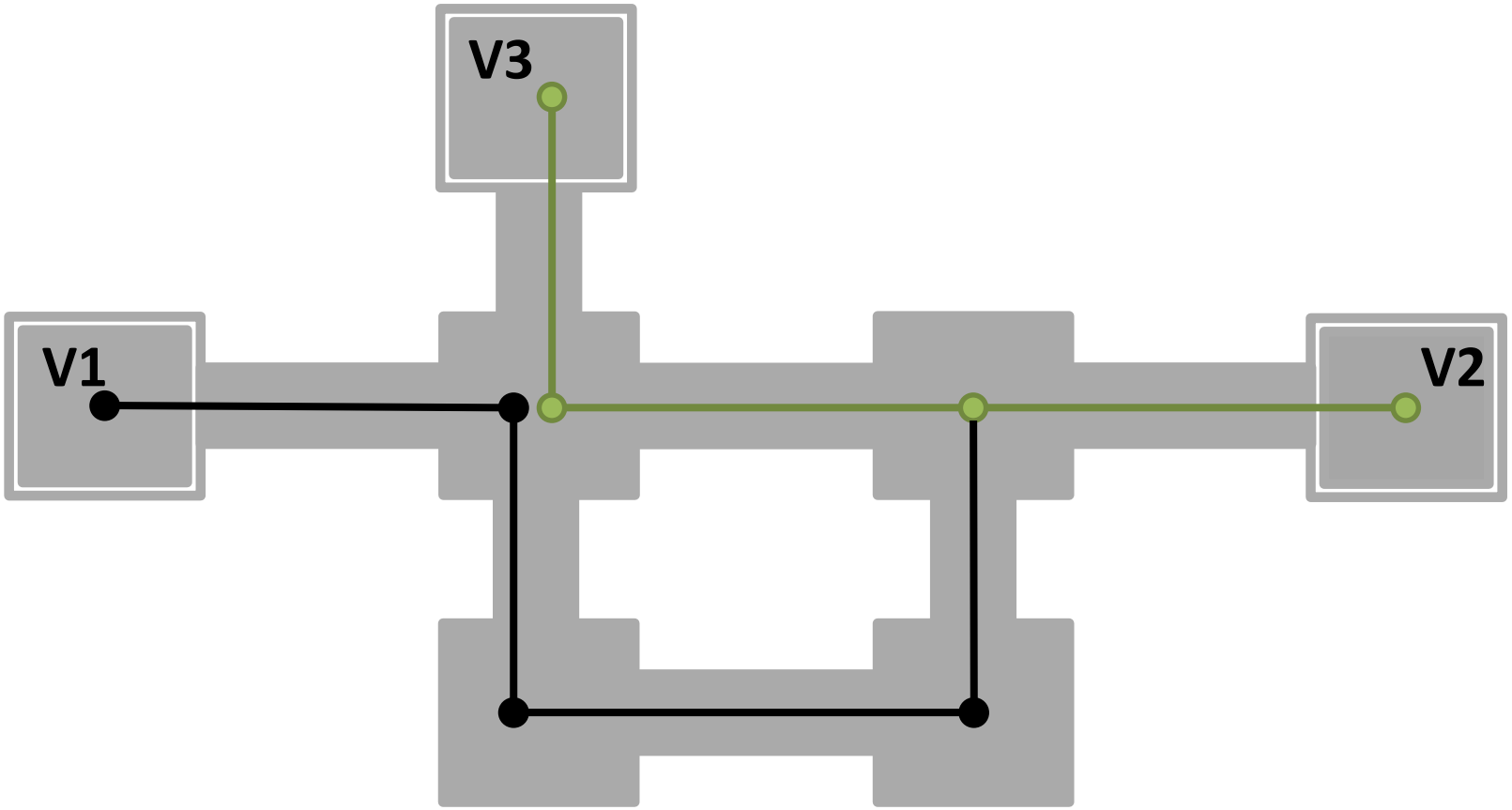
Matchingalgorithmus – Pfad-Matching



Matchingalgorithmus – Pfad-Matching



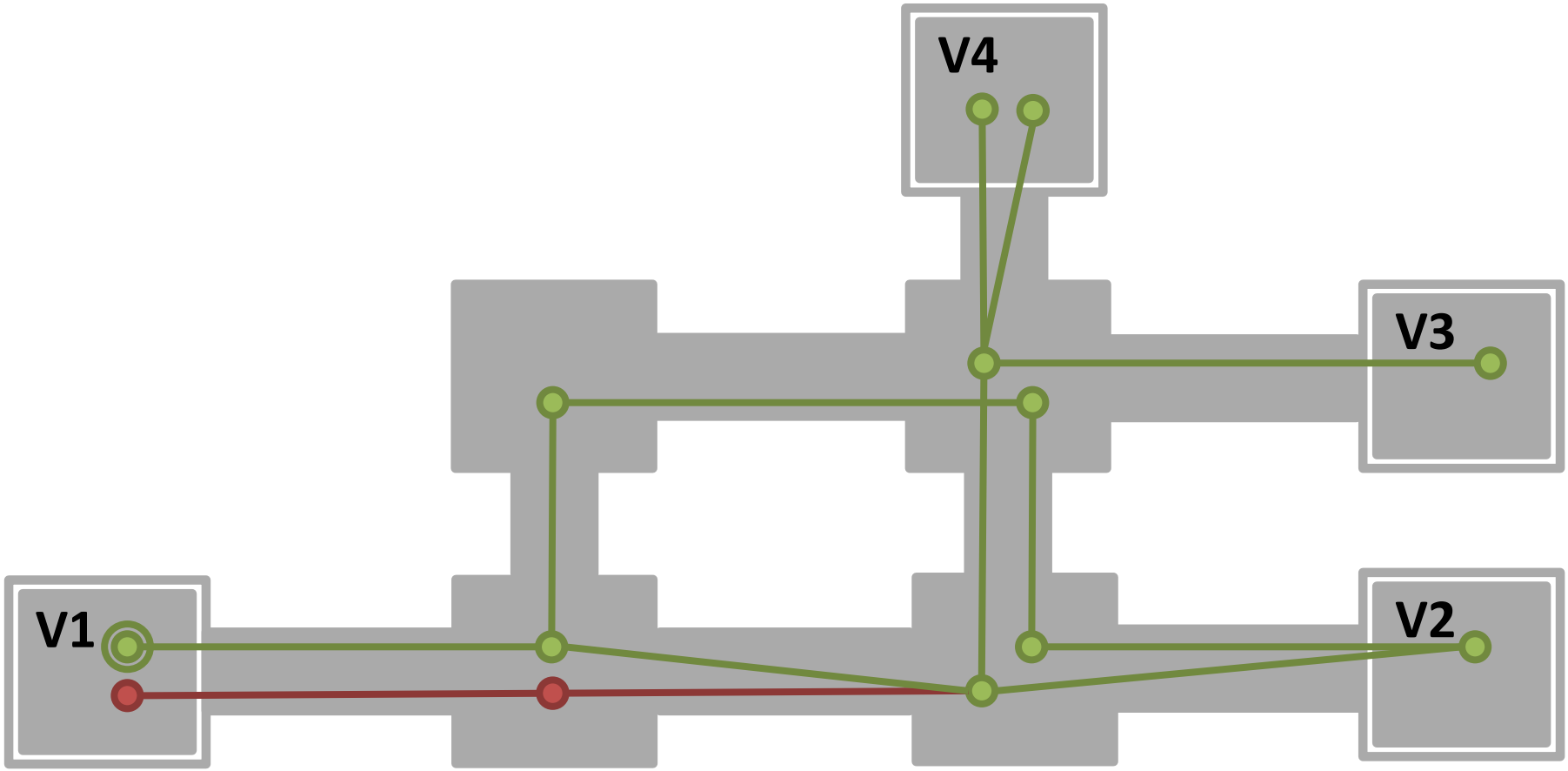
Matchingalgorithmus – Pfad-Matching



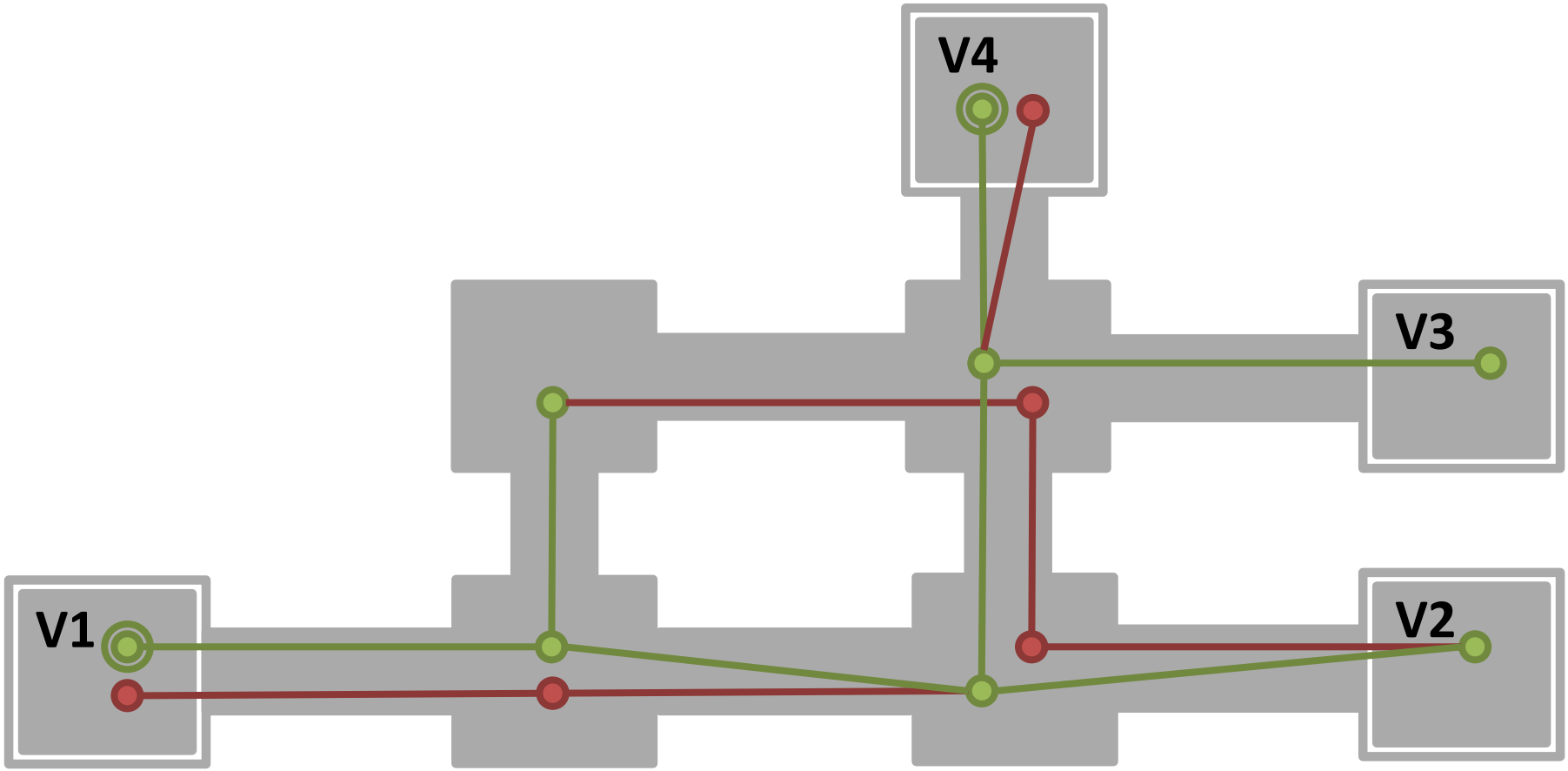
Matchingalgorithmus

- **Selection:** $S(v_i, \mathbb{O}_{i.x})$
  - Setzte Objekt $\mathbb{O}_{i.x}$ als Match für V-Knoten v_i
  - Wähle jeden *isomorphen Pfad* zwischen v_i und $\{V \,/\, v_i\}$ (ausgehend von Objekt $\mathbb{O}_{i.x}$) als aktuellen Arbeitsgraph G_i
  - Speichre die Restriktionen $R(v_i, n_j, \mathbb{O}_{i.p}, \dots, \mathbb{O}_{i.q})$, welche sich aus den isomorphen Pfaden ergeben, auf den Restriktionsstack R_{n_j} (für alle $n_j \in N$).
- **Restriction:** $R(v_i, n_j, \mathbb{O}_{i.p}, \dots, \mathbb{O}_{i.q})$
  - Setze Objekte $\mathbb{O}_{i.p}, \dots, \mathbb{O}_{i.q}$ für Knoten n_j im Arbeitsgraph.
  - Jeder Knoten n_j besitzt einen *Restriktionsstack* R_{n_j}
  - Lege $v_i \rightarrow \{G_{i-1}(n_j)/\{\mathbb{O}_{i.p}, \dots, \mathbb{O}_{i.q}\}\}$ (oben) auf R_{n_j} ab,
    - wobei $G_{i-1}(n_j)$ alle zurzeit verfügbaren Matches für n_j enthält.
- **Permission:** $R^{-1}(v_i, n_j)$
  - Annahme: $v_i \rightarrow \{\mathbb{O}_{i.p}, \dots, \mathbb{O}_{i.q}\}$ liegt oben auf R_{n_j}
  - dann setzte $G_i(n_j) = G_{i+1}(n_j) \cup \{\mathbb{O}_{i.p}, \dots, \mathbb{O}_{i.q}\}$ für Knoten n_j im Arbeitsgraph.

Matchingalgorithmus



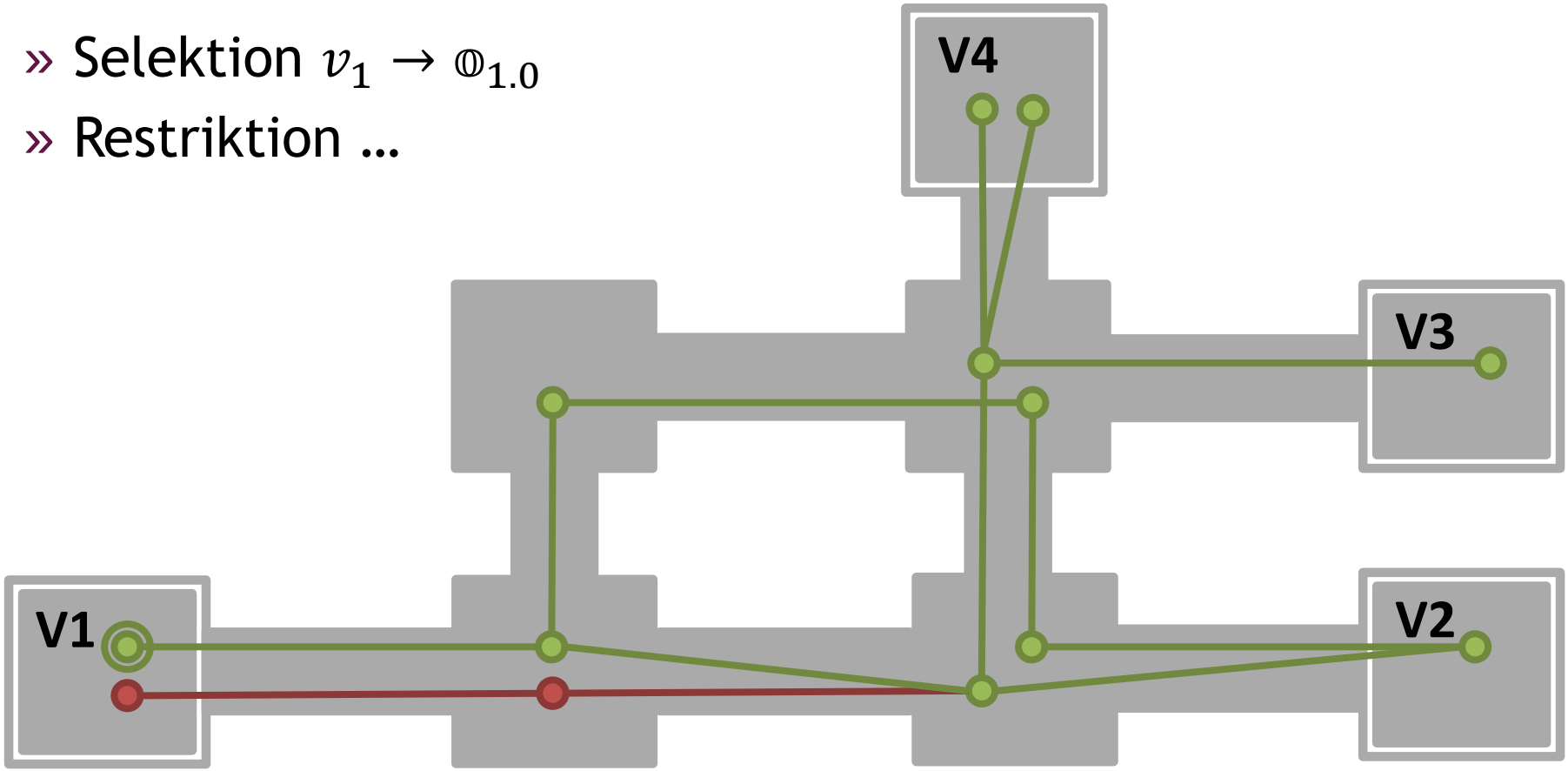
Matchingalgorithmus



- **Initialisierung bzgl. $V = \{v_n, \dots, v_0\}$**
  - Lege eine Liste mit allen noch nicht zugewiesenen Objekten der V-Knoten an:
    - $Z = \{v_0 \rightarrow \{\mathbb{O}_{0.0}, \dots, \mathbb{O}_{0.m}\}, \dots, v_n \rightarrow \{\mathbb{O}_{n.0}, \dots, \mathbb{O}_{n.m}\}\}$
  - Speichere den aktuellen Iterationszustand jeder Variablen:
    - $I = \{v_0 \rightarrow \mathbb{O}_{0.x}, \dots, v_n \rightarrow \mathbb{O}_{n.x}\}$ (initial alle $v_i \rightarrow \emptyset$)
  - Starte das Matching mit $M(v_0, \dots, v_n)$.
- **Matching: $M(v_k, \dots, v_l)$**
  - Falls v_k kein Folgeobjekte $\mathbb{O}_{k.x}$ mit $(x < m)$ bzgl. der Iteration (I) besitzt,
    - (re)initialisiere die Iteration (I) mit allen noch nicht zugewiesenen Objekten (Z).
  - Selektiere $S(v_i, \mathbb{O}_{i.x})$ für alle $v_{\{k,\dots,l\}}$ und entferne $\mathbb{O}_{i.x}$ aus Z,
    - wobei die Iteration $v_i \rightarrow \mathbb{O}_{i.x}$ von $v_{\{k+1,\dots,l\}}$ jeweils mit den (aktuell) nicht eingeschränkten Objekten des Arbeitsgraphen $G_{i-1}(v_0)$ (re)initialisiert wird.
- **Backtracking: $B() : v_s$**
  - Hebe die Restriktionen $R^{-1}(v_i, n_j)$ schrittweise (rückwärts) $\{v_n, \dots, v_0\}$ auf bis
    - entweder ein Knoten v_k erreicht wird, der ein Folgeobjekte $\mathbb{O}_{initial.x}$ besitzt
    - oder Objekte eines Knoten v_u $(k < u < l)$ freigegeben werden, welche noch nicht zugewiesen wurde. Vertausche ggf. die Knoten v_k und v_u (bzgl. $V = \{v_n, \dots, v_0\}$).
  - Starte einen neues (Sub-)Matching $M(v_k, \dots, v_l)$. (wird ggf. zu v_0)

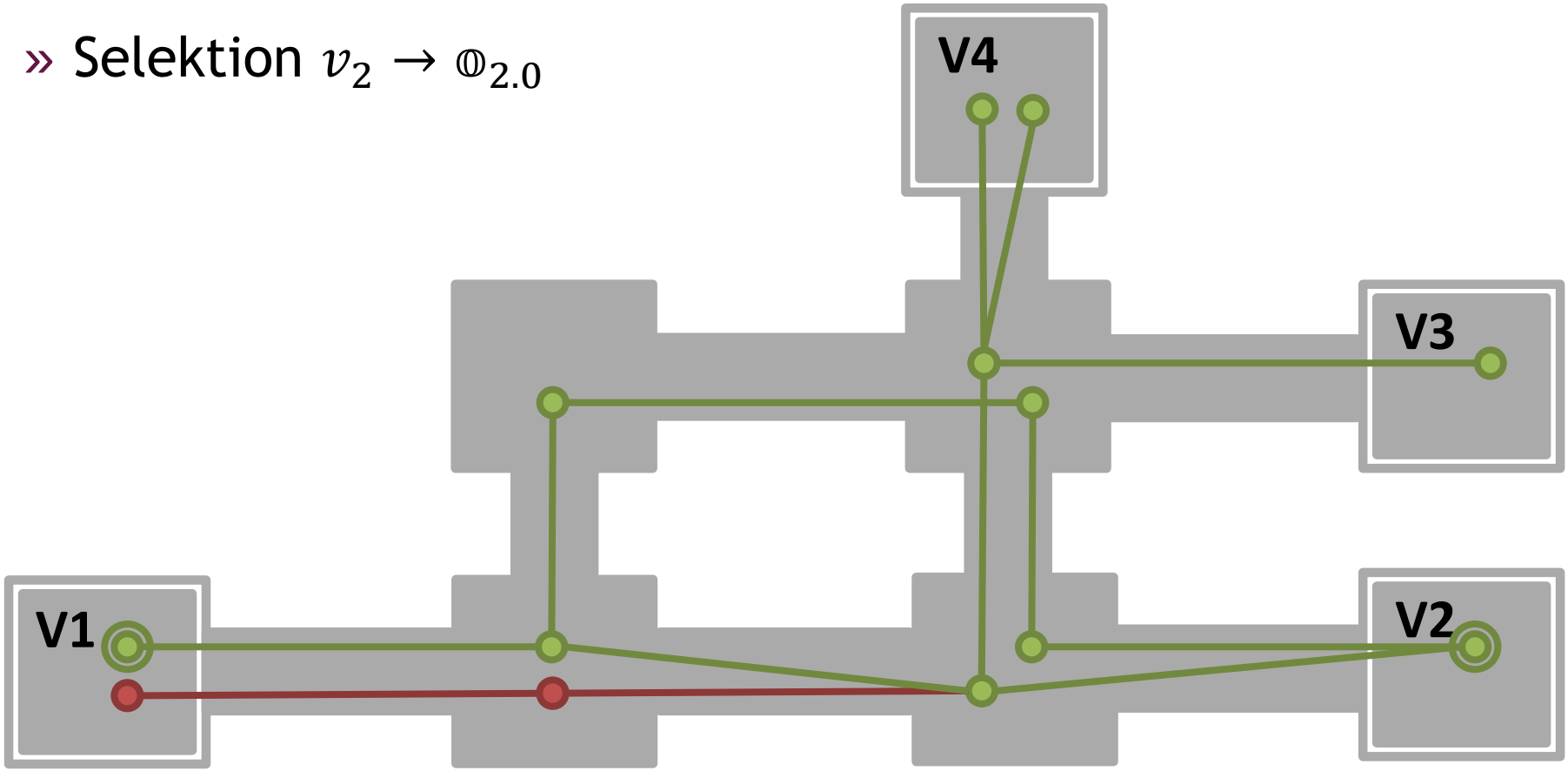
Matchingalgorithmus

- Selektion $v_1 \rightarrow \mathbb{O}_{1.0}$
- Restriktion …



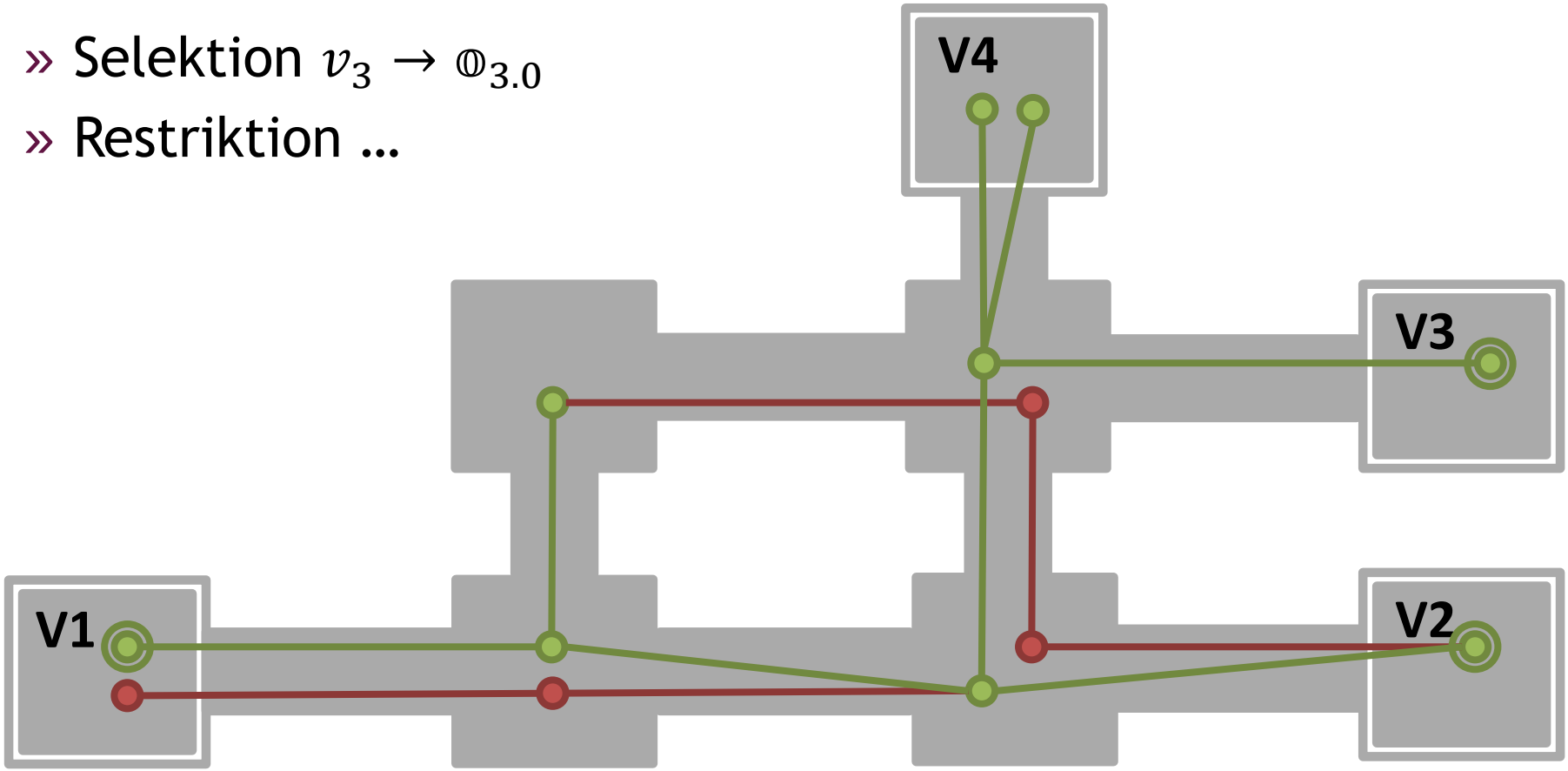
Matchingalgorithmus

» Selektion $v_2 \rightarrow \mathbb{O}_{2.0}$



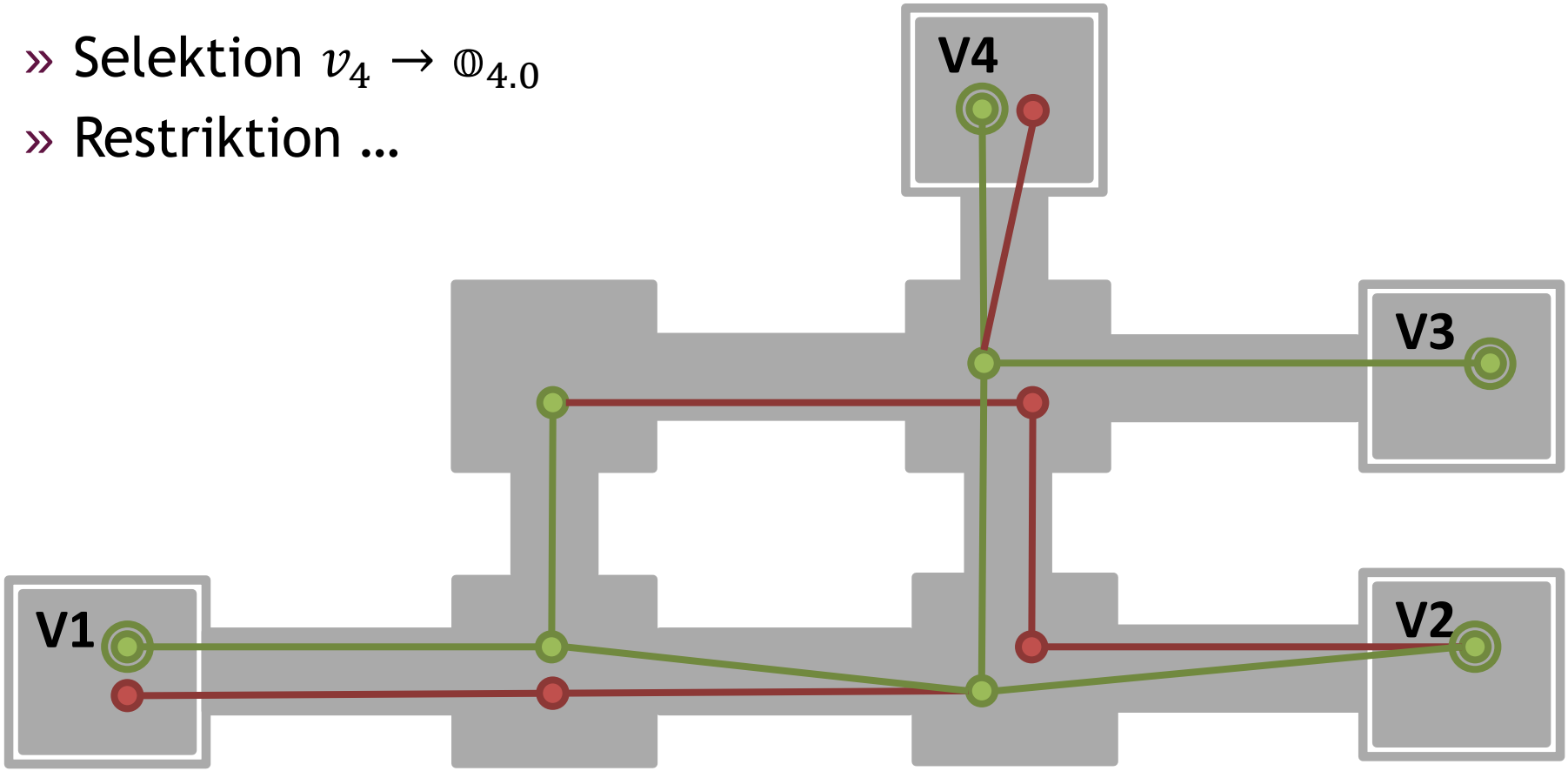
Matchingalgorithmus

- Selektion $v_3 \rightarrow \mathbb{O}_{3.0}$
- Restriktion …



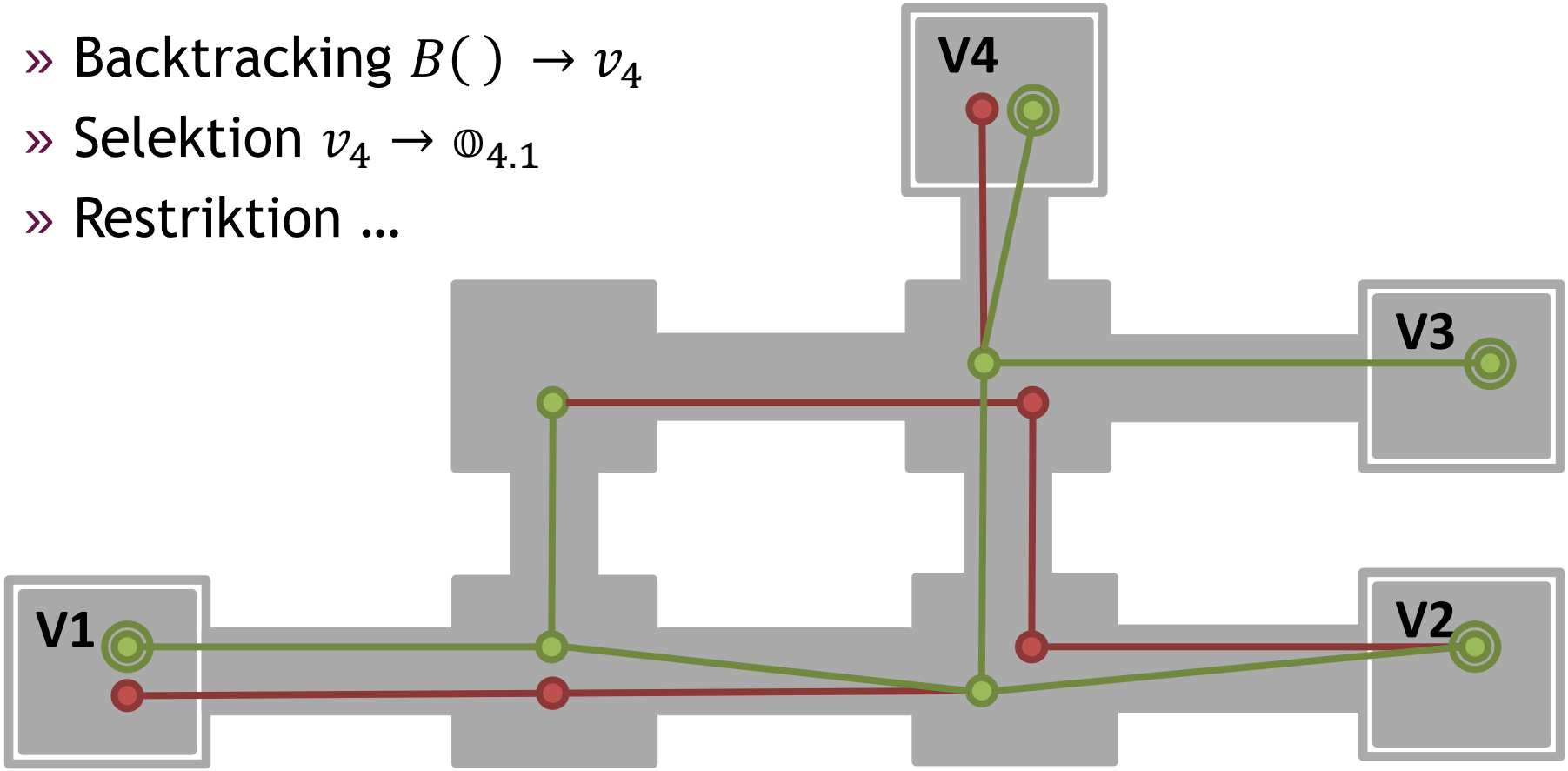
Matchingalgorithmus

- Selektion $v_4 \rightarrow \mathbb{O}_{4.0}$
- Restriktion …



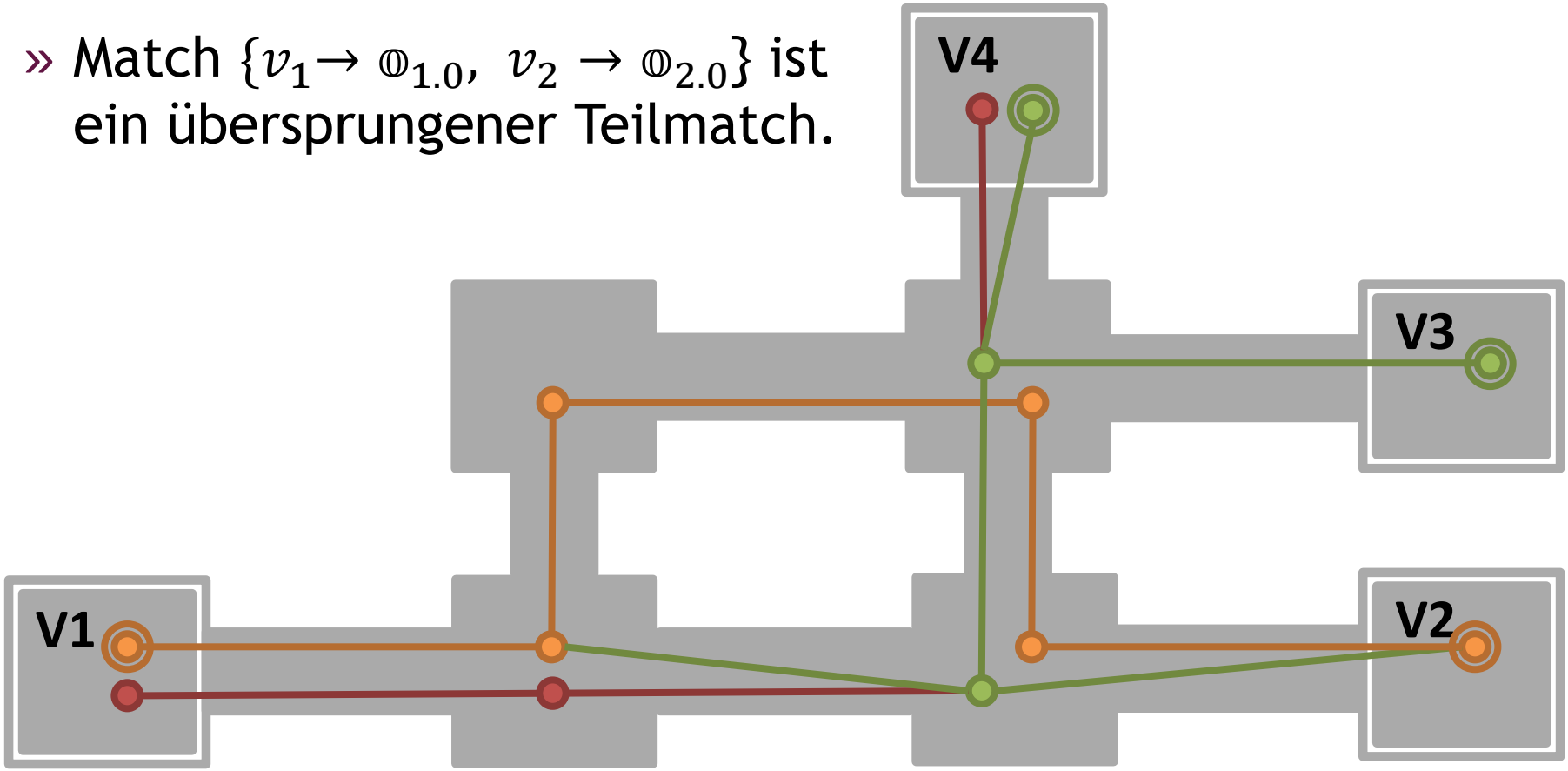
Matchingalgorithmus

- Backtracking $B(\,) \rightarrow v_4$
- Selektion $v_4 \rightarrow \mathbb{O}_{4.1}$
- Restriktion …



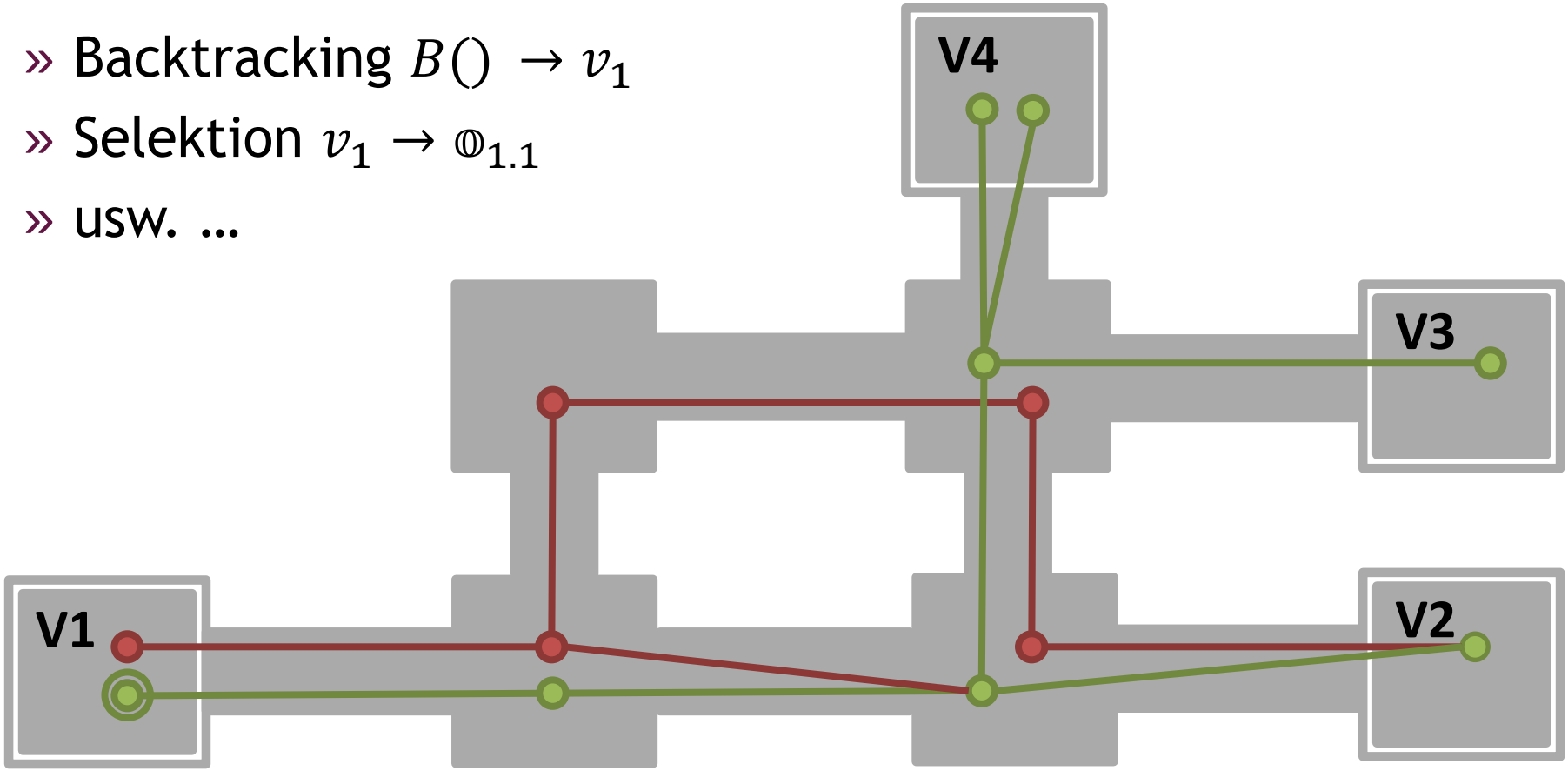
Matchingalgorithmus

» Match $\{v_1 \rightarrow \mathbb{O}_{1.0},\ v_2 \rightarrow \mathbb{O}_{2.0}\}$ ist ein übersprungener Teilmatch.



Matchingalgorithmus

- Backtracking $B() \rightarrow v_1$
- Selektion $v_1 \rightarrow \mathbb{0}_{1.1}$
- usw. …

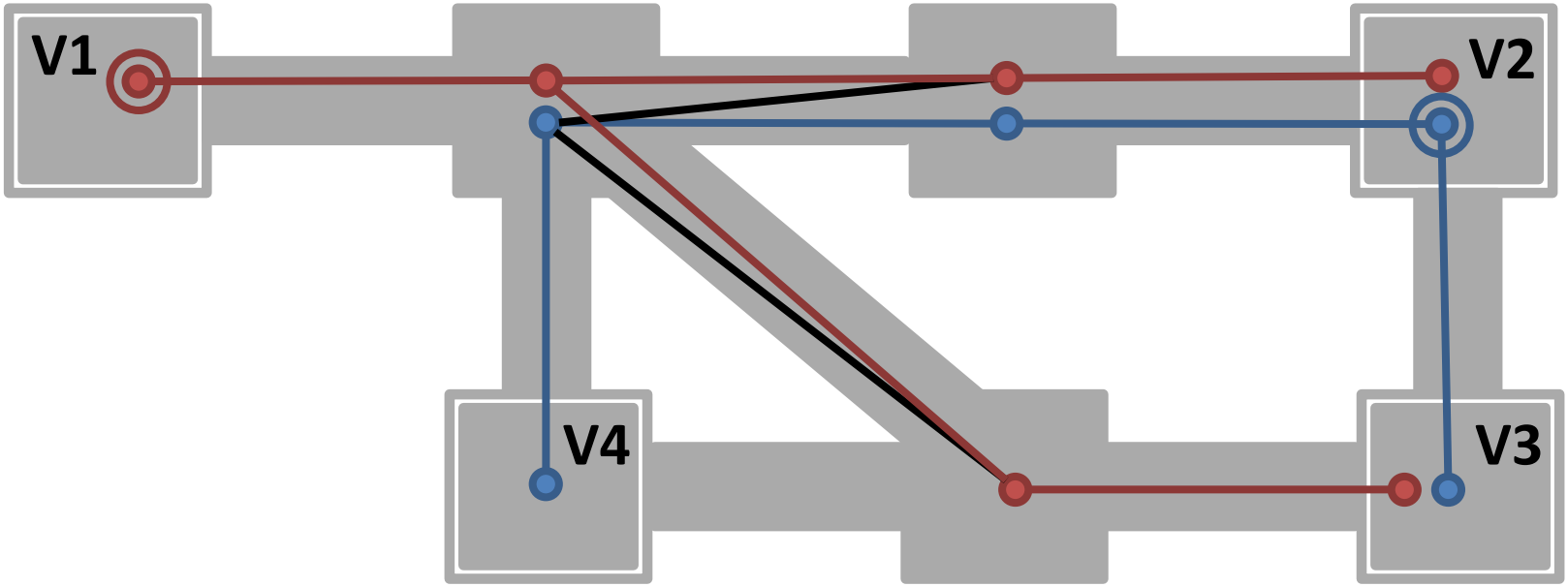


Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

MATCHING-ALGORITHMUS (ERWEITERUNG)

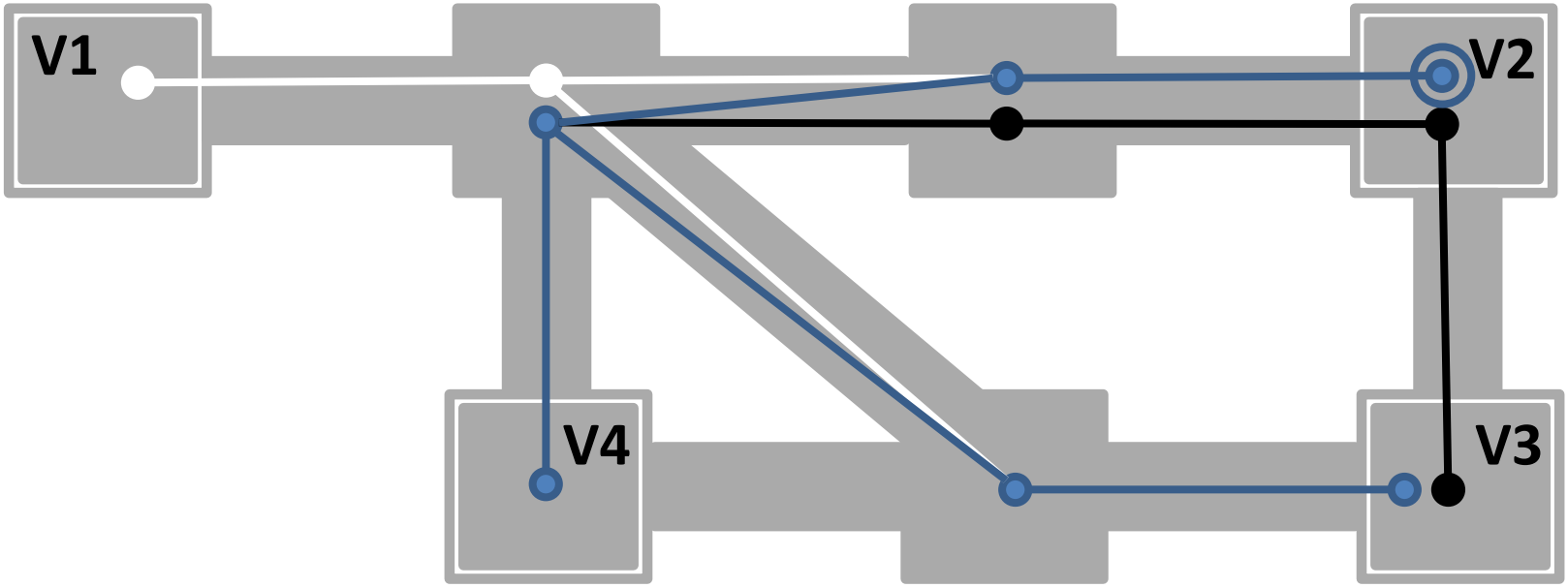
Matchingalgorithmus – Zerlegung des Lösungsraums

- Um einen fehlenden/neuen Match zu erzeugen muss es mindestens ein Pfad geben, der in keinem anderen Matches enthalten ist.
- Entferne ausgehend von $v_{initial}$ alle Objekte aus dem Basisarbeitsgraphen, die vollständig in dessen Selektion enthalten sind.



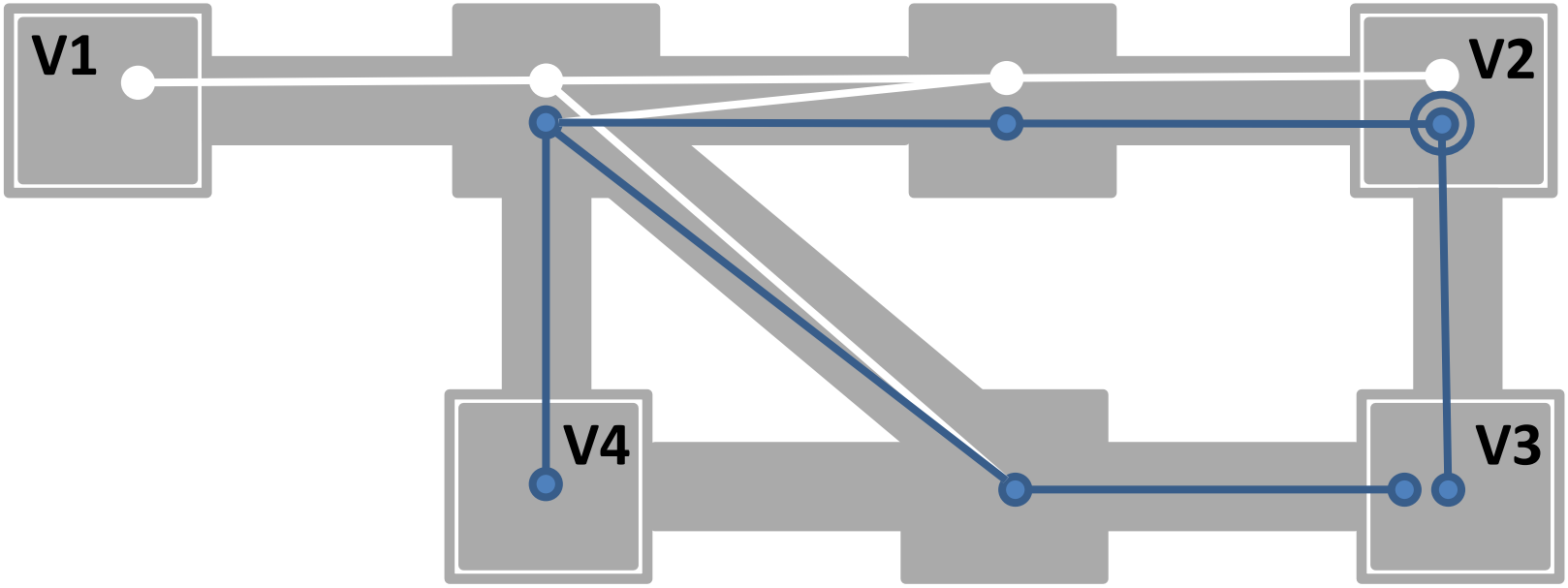
Matchingalgorithmus – Zerlegung des Lösungsraums

- Um einen fehlenden/neuen Match zu erzeugen muss es mindestens ein Pfad geben, der in keinem anderen Matches enthalten ist.
- Entferne ausgehend von $v_{initial}$ alle Objekte aus dem Basisarbeitsgraphen, die vollständig in dessen Selektion enthalten sind.



Matchingalgorithmus – Zerlegung des Lösungsraums

- Um einen fehlenden/neuen Match zu erzeugen muss es mindestens ein Pfad geben, der in keinem anderen Matches enthalten ist.
- Entferne ausgehend von $v_{initial}$ alle Objekte aus dem Basisarbeitsgraphen, die vollständig in dessen Selektion enthalten sind.

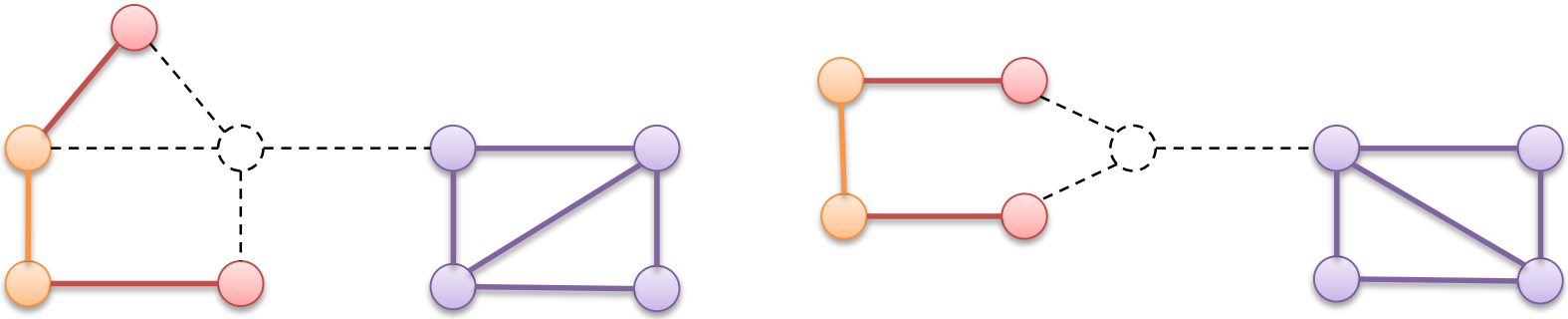


Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

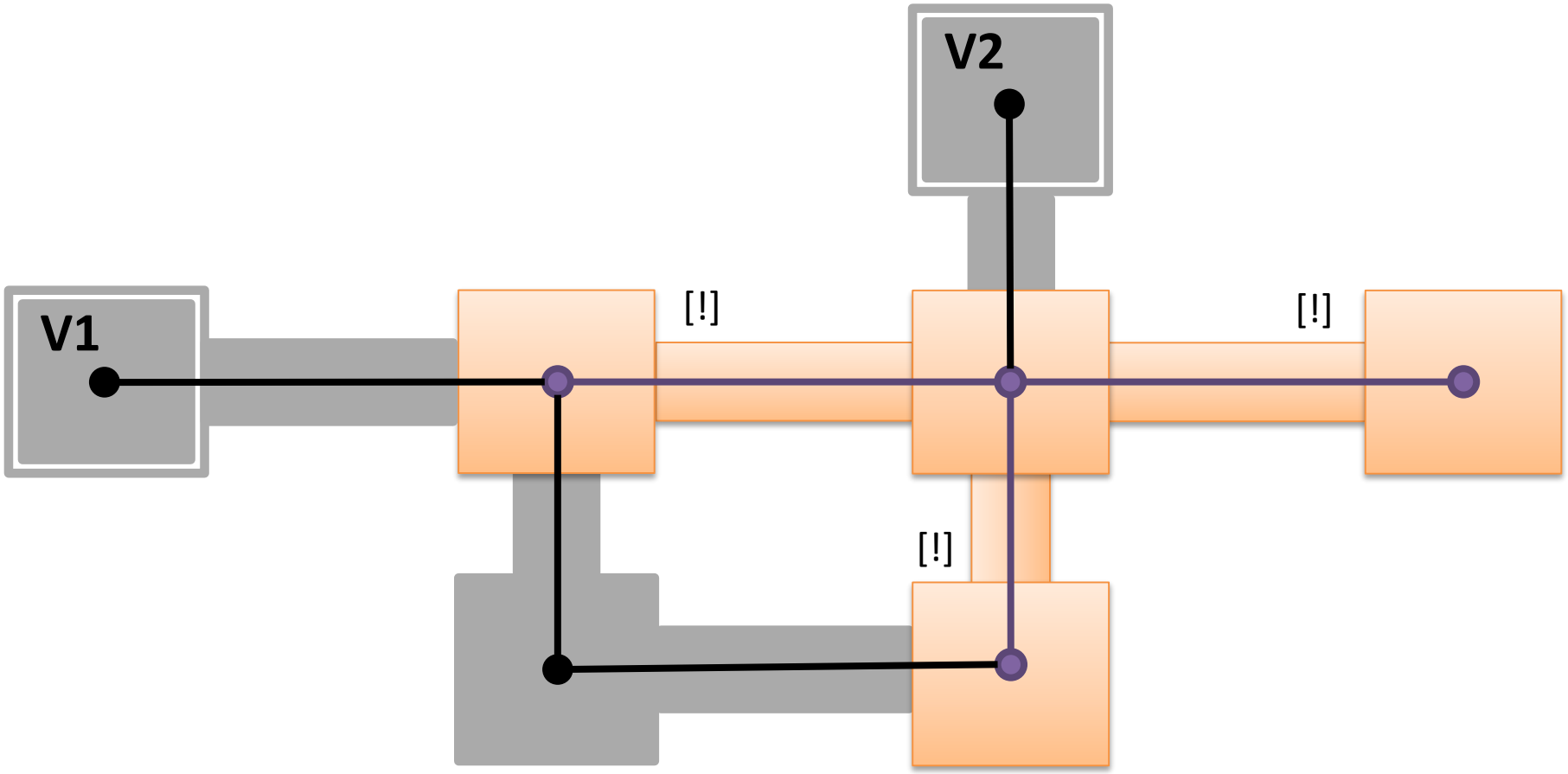
ATOMIC-PATTERNS

Atomic-Patterns

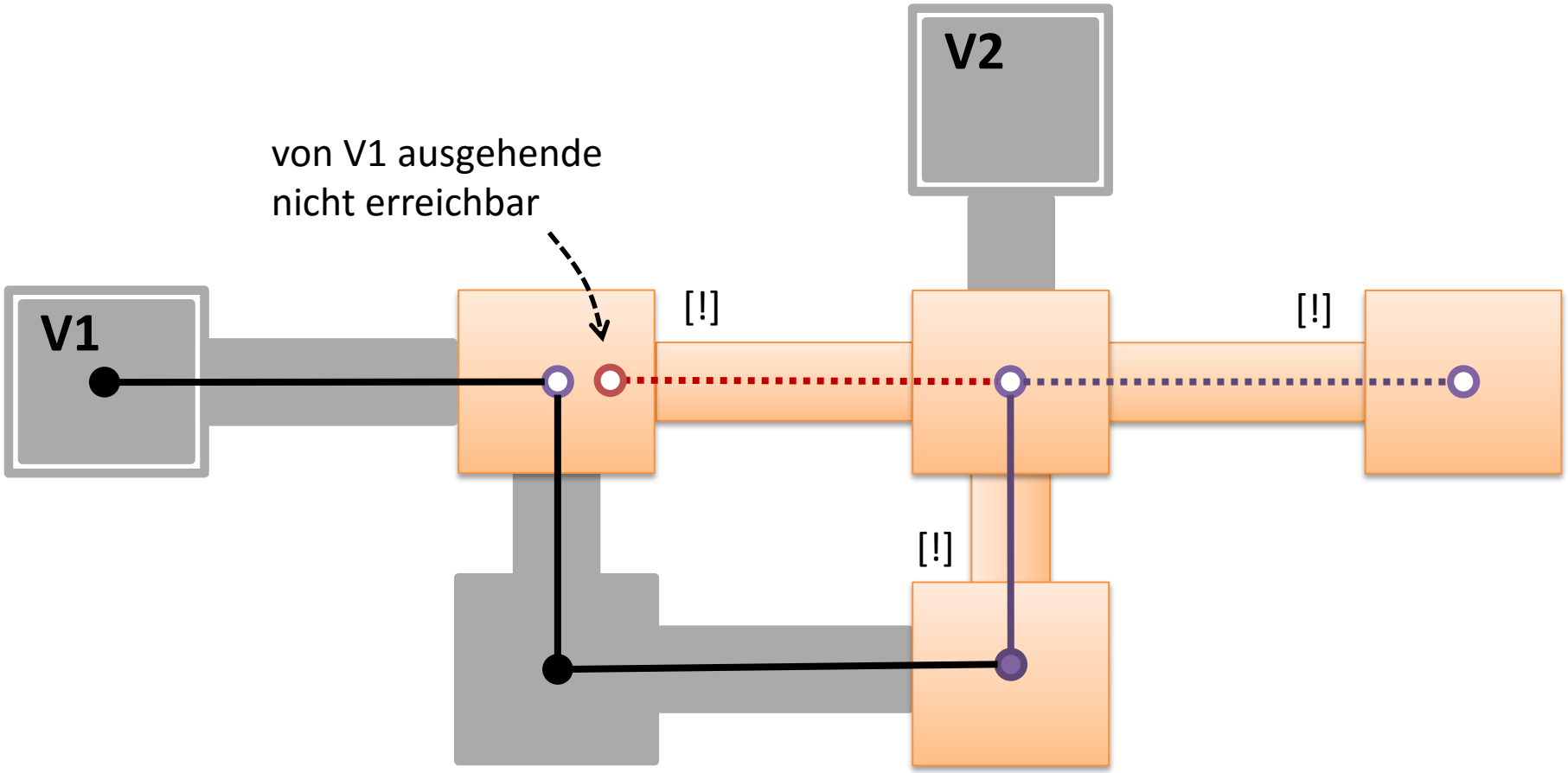
- **Atomic-Patterns**
  - Knoten die nur gemeinsam gematcht werden dürfen,
  - z.B. zusammenfassen von EOpposite-Änderungen.
- **Mandatory-Edges**
  - Kanten [!] die nur zusammen mit dem Knoten aus dem Pattern entfernt werden dürfen.
  - vgl. induzierte Kanten im MCS-Problem:



Atomic-Patterns – Mandatory-Edges

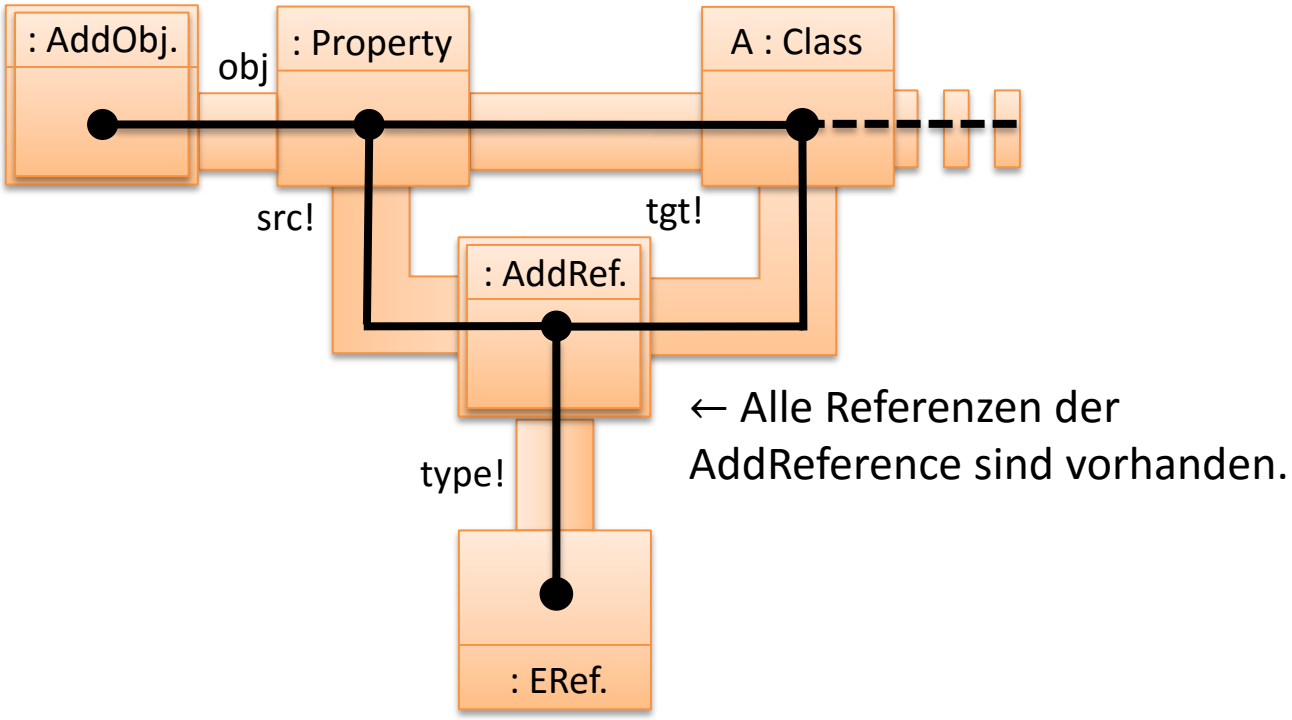


Atomic-Patterns – Mandatory-Edges



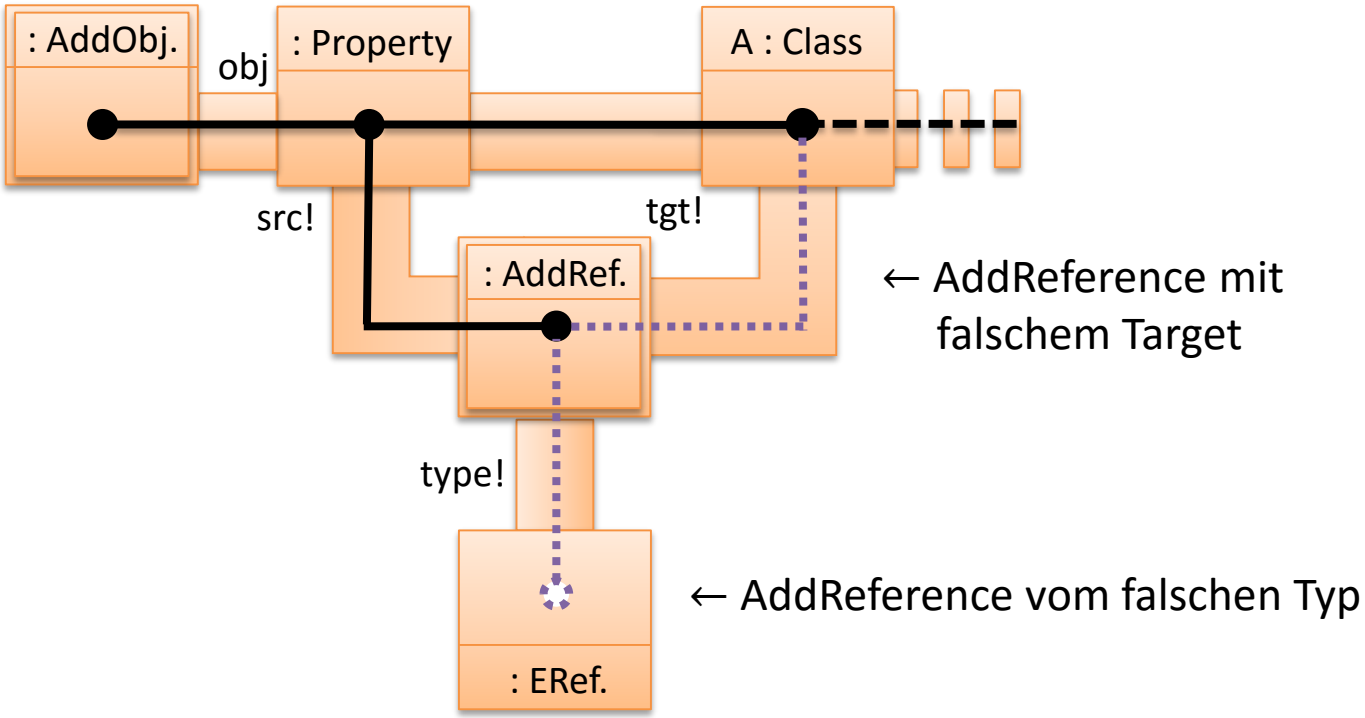
Atomic-Patterns – Mandatory-Edges

» Änderungen müssen induziert gematcht werden.
» Für Modellelemente ist dies nicht erforderlich.



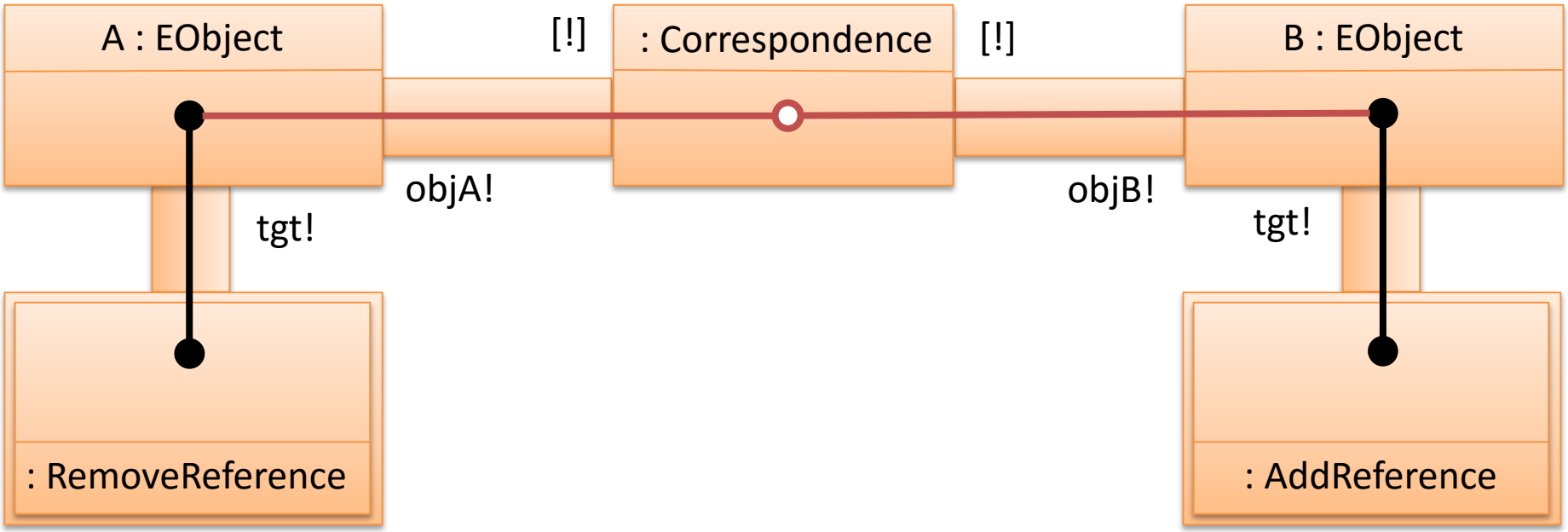
Atomic-Patterns – Mandatory-Edges

» Änderungen müssen induziert gematcht werden.
» Für Modellelemente ist dies nicht erforderlich.



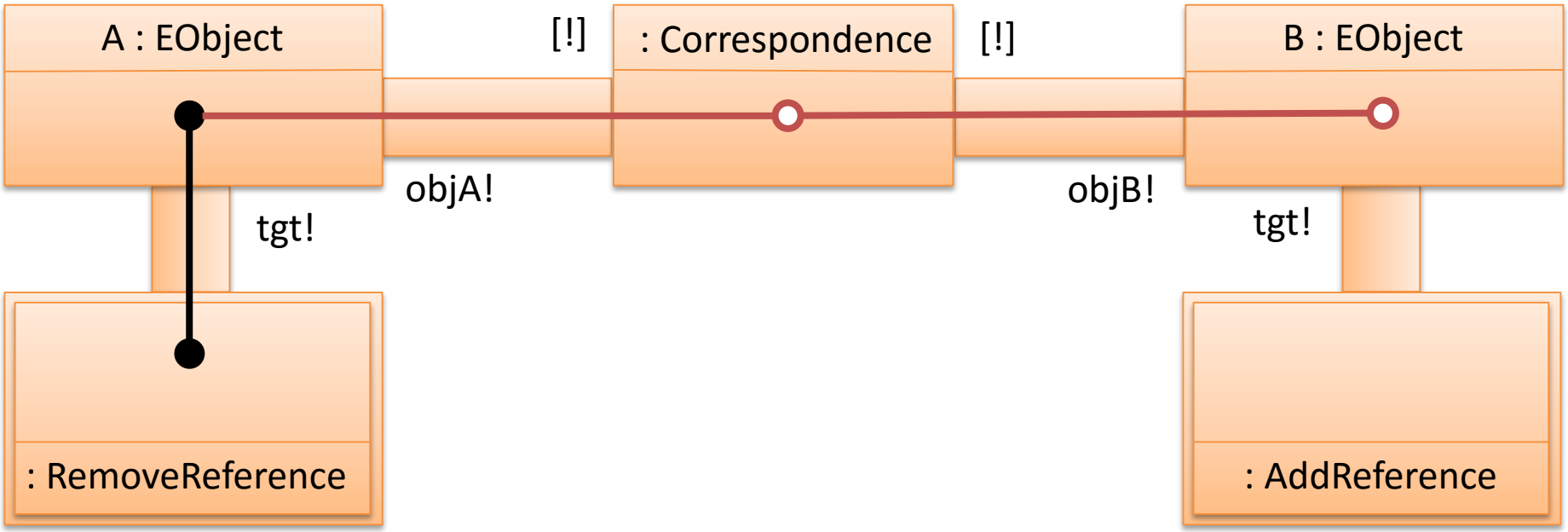
Atomic-Patterns – Mandatory-Edges

» Neue Korrespondenz zwischen [A] und [B] einfügen.
  » Ggf. alte Korrespondenzen von [A] und [B] löschen.



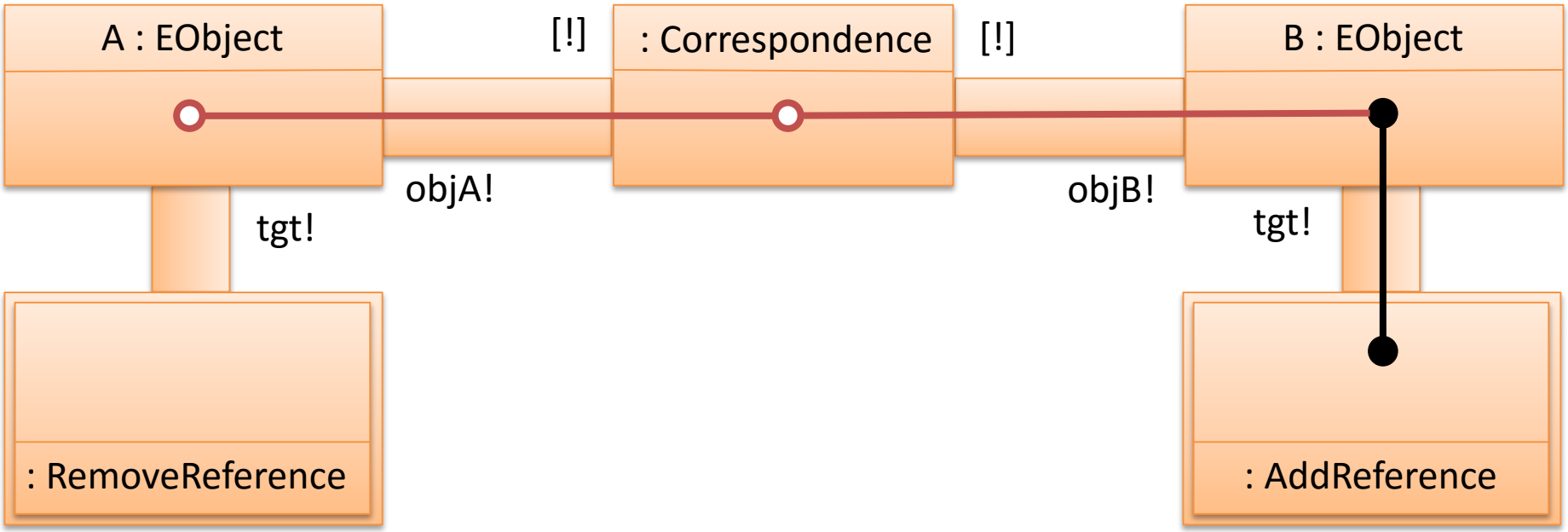
Atomic-Patterns – Mandatory-Edges

- Undo: löschen von [A]
- Einfügen einer Korrespondenz für [A].



Atomic-Patterns – Mandatory-Edges

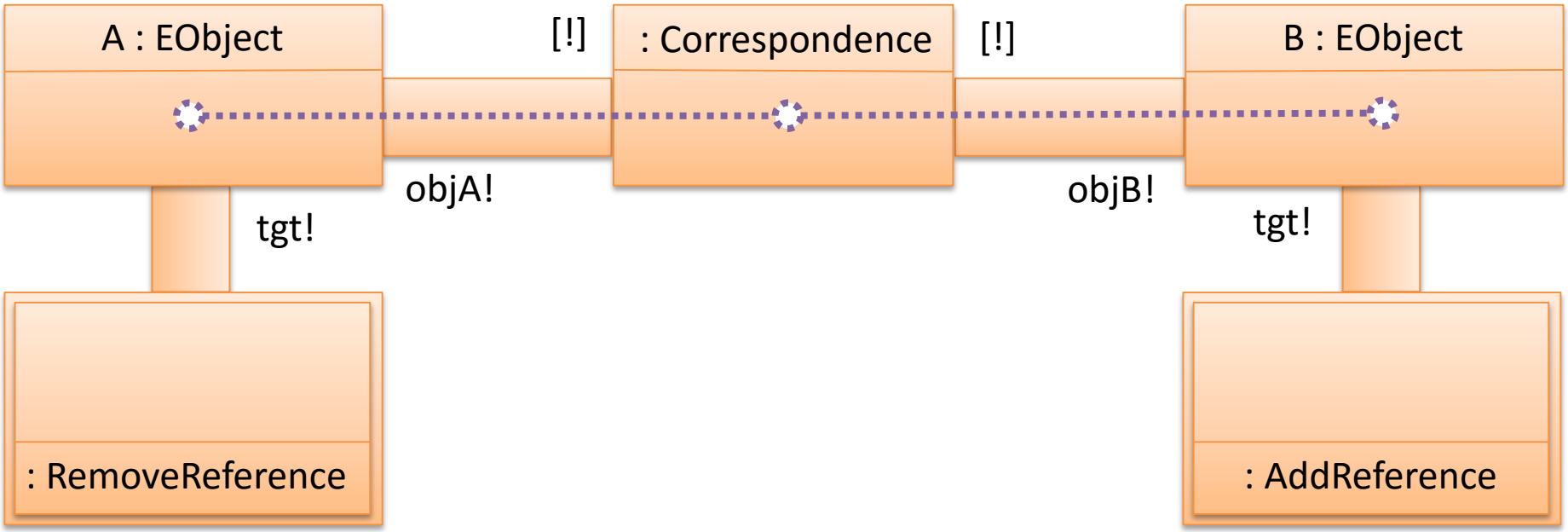
» Einfügen einer Korrespondenz für [B].



Atomic-Patterns – Mandatory-Edges

» partielles Matching von Korrespondenz-Mustern

⇒ ggf. interessant um Differenzen zu Optimieren



Historienbasierte Generierung von Reparaturen für domänenspezifische Modellierungssprachen

SCHLUSSFOLGERUNG

Future-Work

- **weitere Kriterien zur Bewertung der Reparaturen**
  - z.B. Anzahl der Änderungen, welche einen positiven /negativen Einfluss auf die Konsistenzregel haben.
- **Matching nicht anwendbarer Complement-Rules**
  - fehlenden Kontext / PACs in erzeugenden Anteil umwandeln
  - herstellen von negativen Anwendungsbedingungen (NACs)

  ⇒ State-Space-Exploration nach Editiersequenzen, welche sich aus mehreren Editieroperationen zusammensetzen.
- **Anbindung an ein Modell-Repository**
- **Erweiterung des Verfahrens auf Historienketten**
- **Evaluation: Beispiel-Katalog, Performance-Evaluierung**

Schlussfolgerung

- Das Verfahren ermittelt Reparaturen, welche die Änderungen der Modellhistorie fortsetzen.
- Der Prototyp zeigt, dass die Erkennung partieller Editierregeln grundsätzlich möglich ist.
  - Performance hängt im Wesentlichen von der Komplexität der Editierregel ab. Bisher gute Performance im (ms-Bereich).
- Einsatz als Ergänzung zu anderen Reparaturverfahren,
  - z.B. in Verbindung mit State-Space-Exploration.
- Verwandte Themenbereiche:
  - Autovervollständigung, Synchronisation von Modell-Sichten/Slices, Verfeinerungen (nicht Transformation) bzgl. Inter-Modellkonsistenz, Optimierung von Differenzen,…

Historienbasierte Generierung von Reparaturen
für domänenspezifische Modellierungssprachen

FRAGEN & DISKUSSION

Inhalt

- **Einführung**
  - Motivation & Beispiel
  - Konsistenz von Modellen
  - Analyse von Inkonsistenzen
- **Komplementierung inkonsistenter Editierschritte**
  - Differenzdarstellung
  - unvollständige Editierschritte
  - komplementierende Editierregel
  - Reparaturberechnung
- **Matching-Algorithmus**
  - aufbauen des Arbeitsgraphen
  - Matching-Algorithmus
  - Atomic-Patterns
- **Schlussfolgerung**