Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

Modellgetriebene Softwareentwicklung und Graphtransformations-systeme

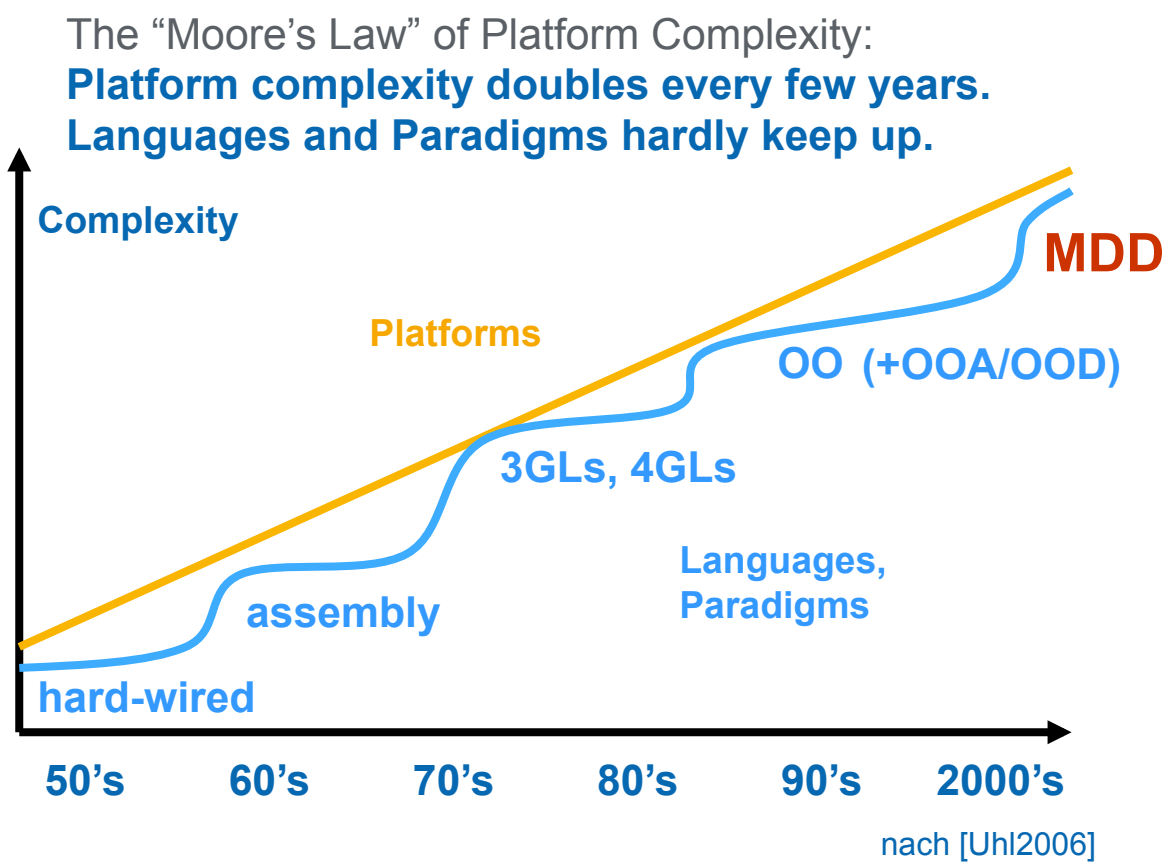
TU Berlin, 31.05.2010

Holger Giese
System Analysis & Modeling Group, Hasso Plattner Institute for Software Systems Engineering at the University of Potsdam, Germany
holger.giese@hpi.uni-potsdam.de

I Motivation für MDE

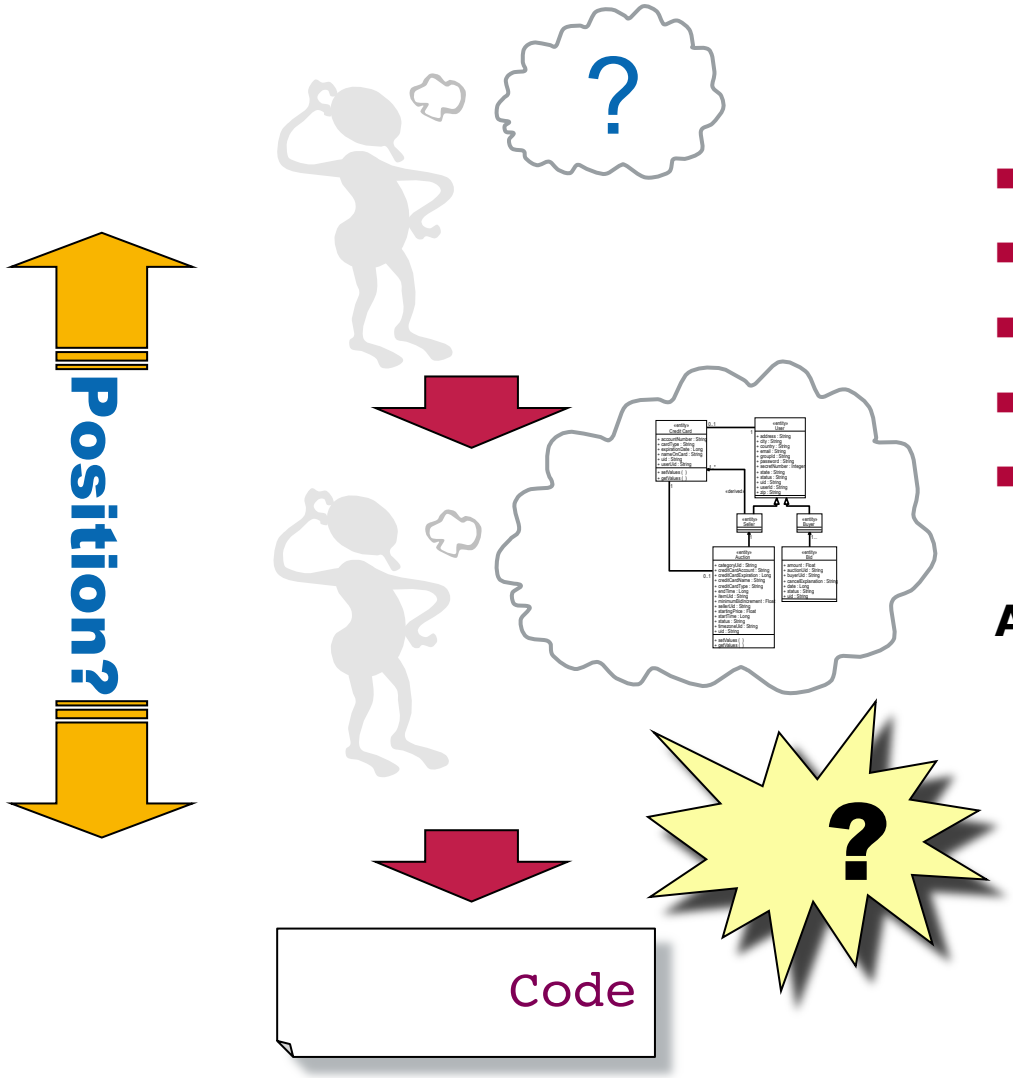
Anforderungen:

- Höhere Komplexität
- Schnellere Entwicklung
- Höhere Qualität



nach [Uhl2006]

Weitere Automatisierung durch MDE



- Anfang: Idee
- Anforderungen: Modelle
- Grobentwurf: Modelle
- Entwurf: Modelle
- Implementierung: Code

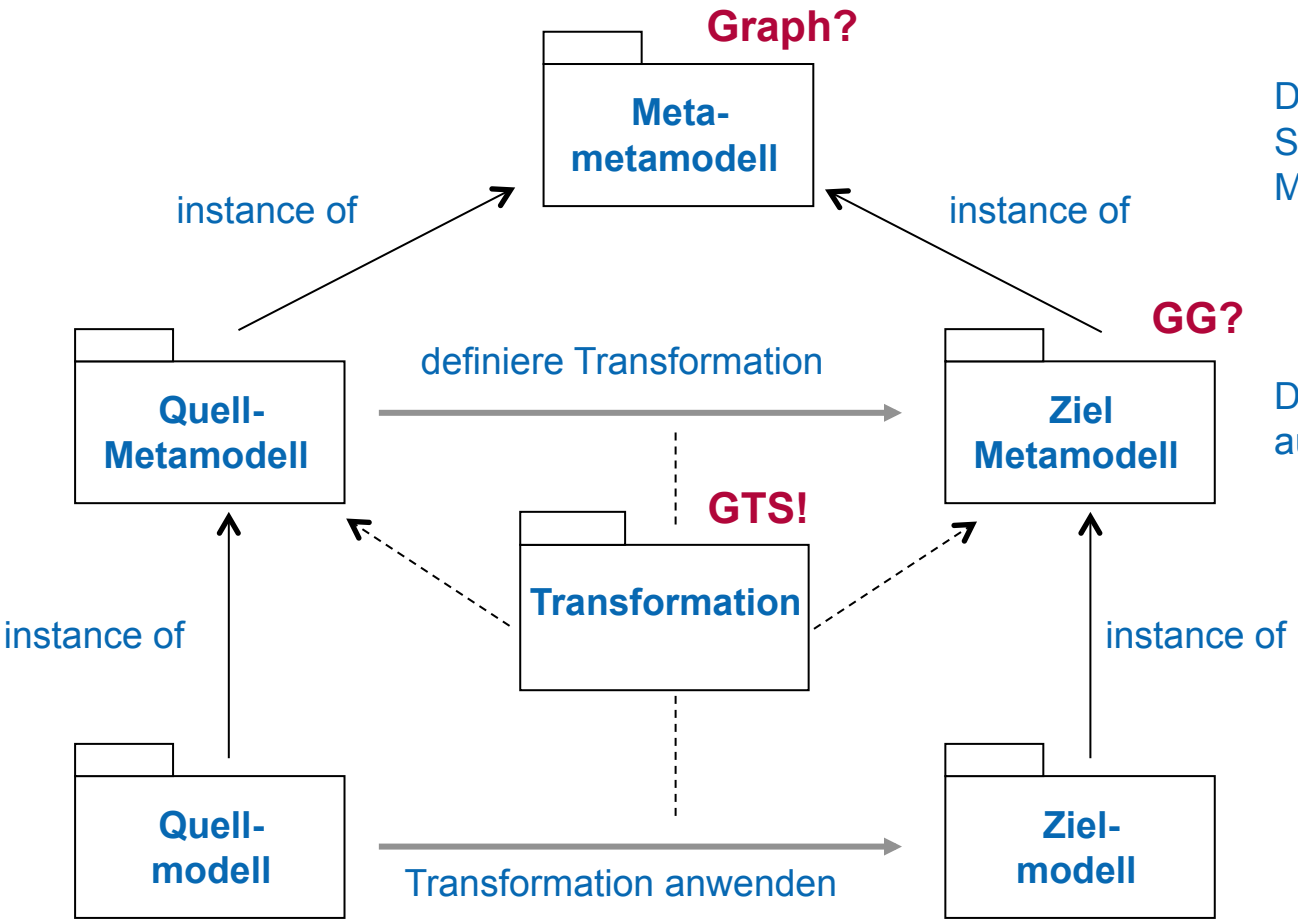
Abstraktion

Aber wie kommt man von

- **Abstrakten high-level Ideen/Modellen** zur
- **konkreten low-level Implementierung**?

[Kanshi2005]

Modelltransformationen und Metamodellierung



Definition der Transformations-Sprache auf Basis des **einen** Metametamodells

Definition der Transformation auf Basis der Metamodelle

Überblick

II MDE: GTS zur Definition, Manipulation und Charakterisierung von Eigenschaften der Modellierungssprachen

- Wohlverstandene Theorie ermöglicht effiziente Lösungen
- Formale Grundlage & Verifikation

III MDE: GTS zur Modellierung und Analyse hochgradig dynamischer Systeme

- Erzeugung und Löschung von Komponenten
- Veränderliche Strukturen

IV MDE & Self-Managed Systeme: Kombination beider vorherigen Fälle

- Manipulation der Software zur Laufzeit erfolgt über Modelle

II MDE: GTS ermöglicht effiziente Lösungen

II MDE: GTS zur Definition, Manipulation und Charakterisierung von Eigenschaften der Modellierungssprachen

- Wohlverstandene Theorie ermöglicht effiziente Lösungen
- Formale Grundlage & Verifikation

III MDE: GTS zur Modellierung und Analyse hochgradig dynamischer Systeme

- Erzeugung und Löschung von Komponenten
- Veränderliche Strukturen

IV MDE & Self-Managed Systeme: Kombination beider vorherigen Fälle

- Manipulation der Software zur Laufzeit erfolgt über Modelle

Beispiel: Tripel Graph Grammatiken

[Schürr1994]

- Deklarative Spezifikation
- Verknüpfung von drei Graphgrammatiken in einer TGG-Regel
  - Quell-, Zielgrammatik
  - Korrespondenzgrammatik zur Verknüpfung korrespondierender Elemente in Quell-, Zielgrammatik
- Grundidee:
  - Paralleler Aufbau der drei Graphen
  - Startgraph: Axiom
  - Weitere konsistente Paare werden über Regeln aufgebaut

GTS ermöglicht es bidirektionale Transformation zu spezifizieren!

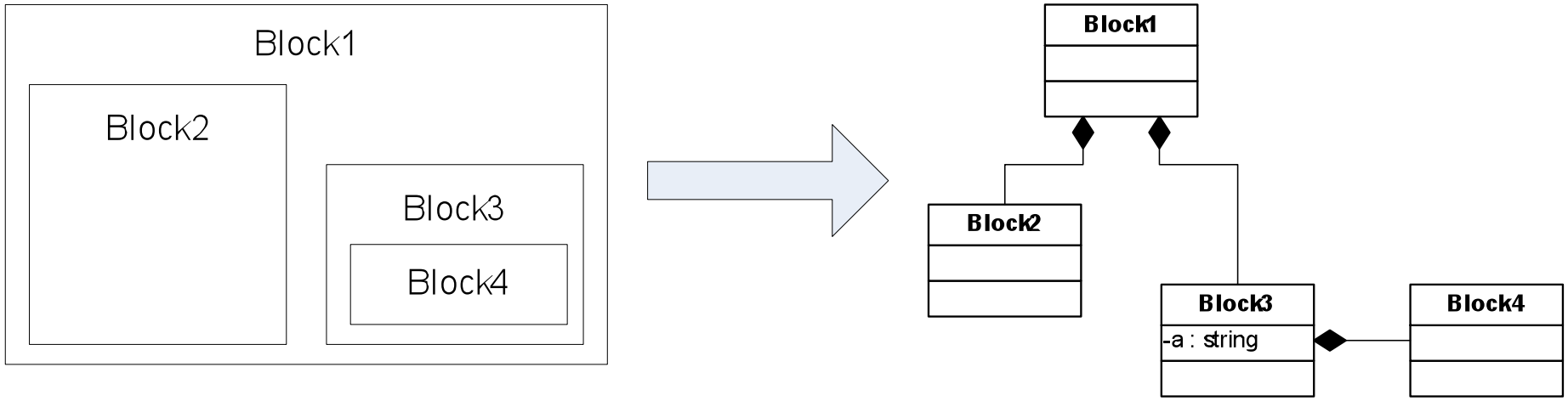
Modellsynchronisation mit TGGs

Idee:

- Modellsynchronisation auf Basis der TGG-Regeln
- Fälle:
  - Keine Veränderung ⇨ nur Überprüfen
  - Veränderung im Quellmodell ⇨ Änderung Propagieren (Überschreiben)
  - Veränderung im Zielmodell ⇨ Änderung Propagieren (Überschreiben)
  - Veränderung im Quell- und Zielmodell ⇨ Konflikt!

Beispiel - Modelle

Transformation des Blockdiagramms in ein Klassendiagramm

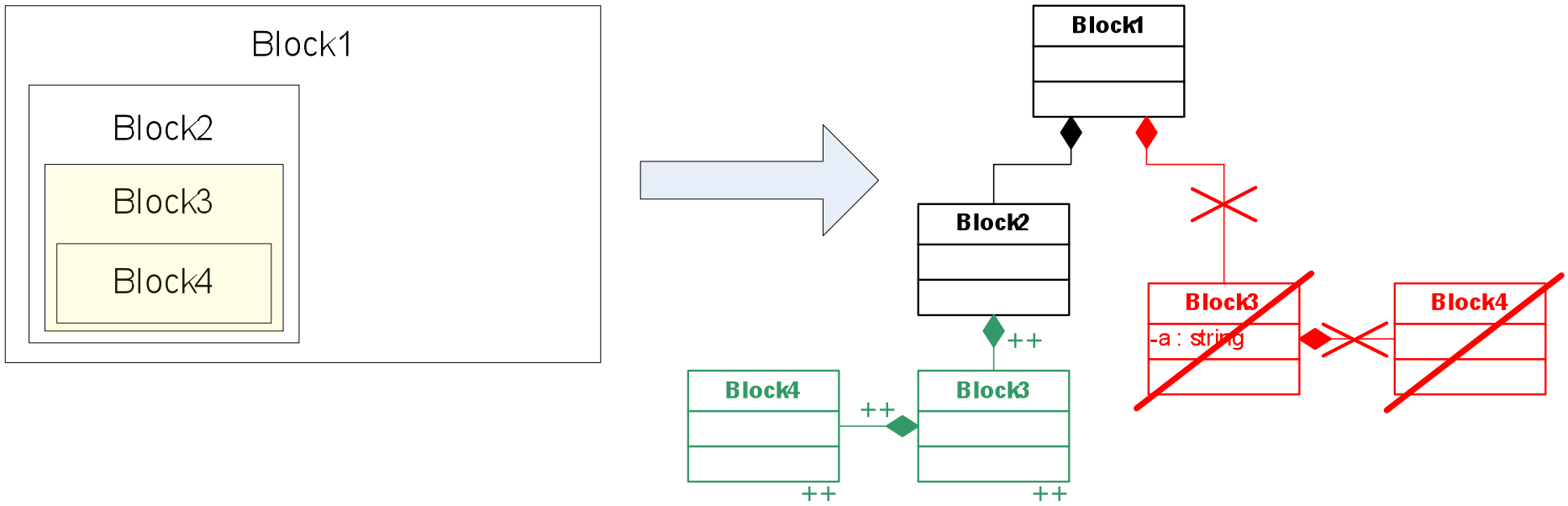


Blockdiagramm

Klassendiagramm

Beispiel - Neutransformation

Veränderung des Blockdiagramms und Synchronisation



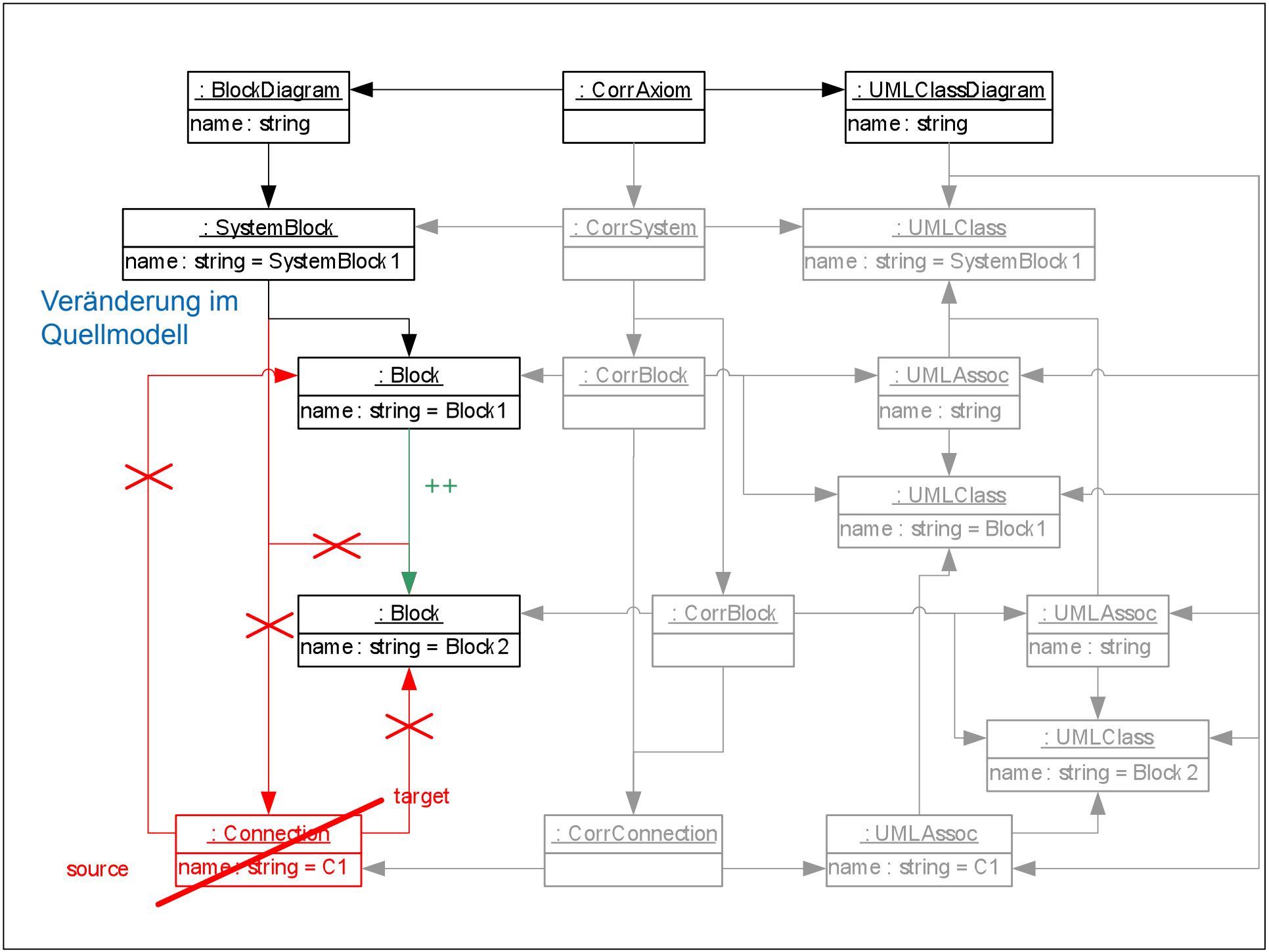
Blockdiagramm

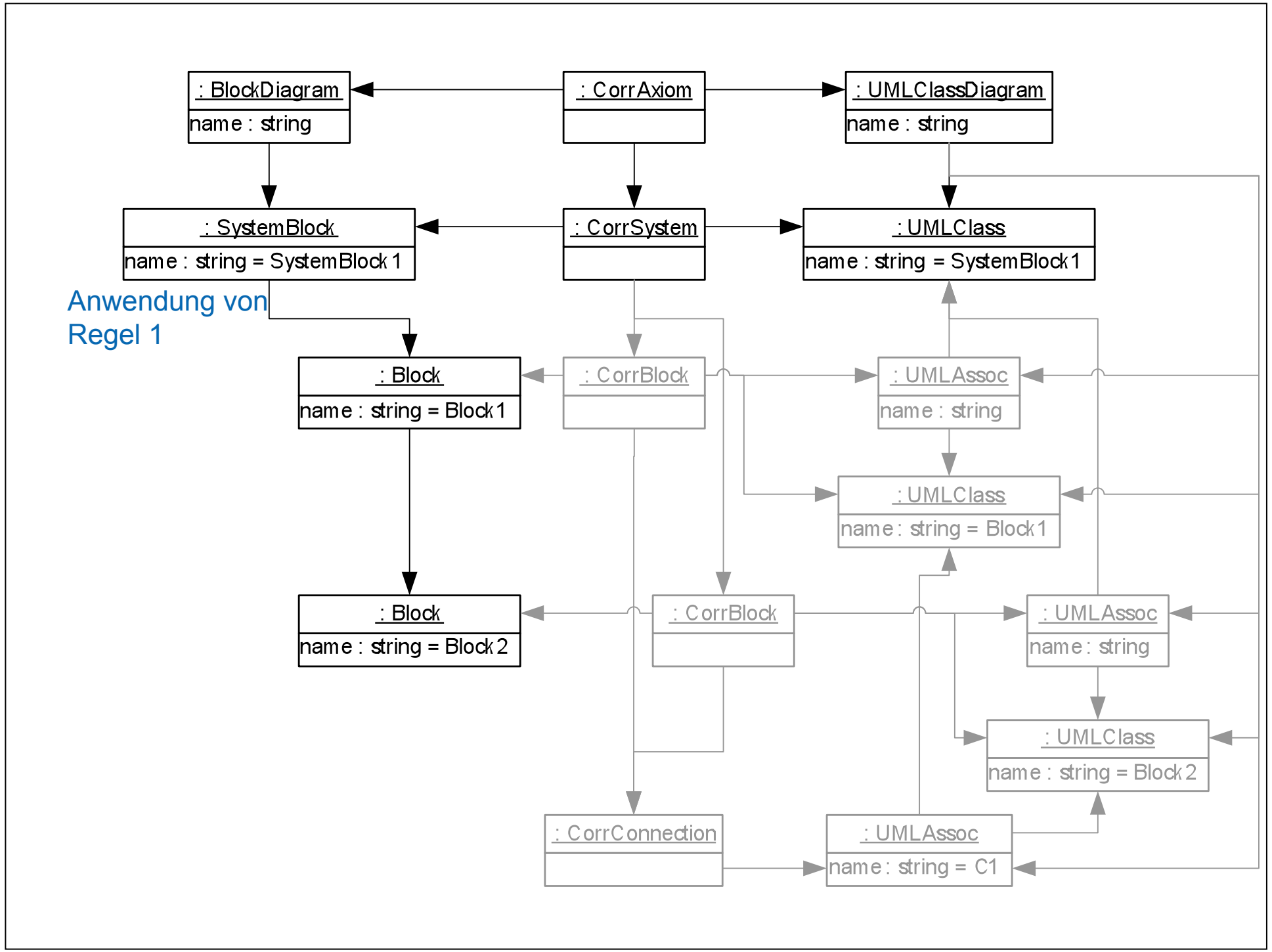
Klassendiagramm

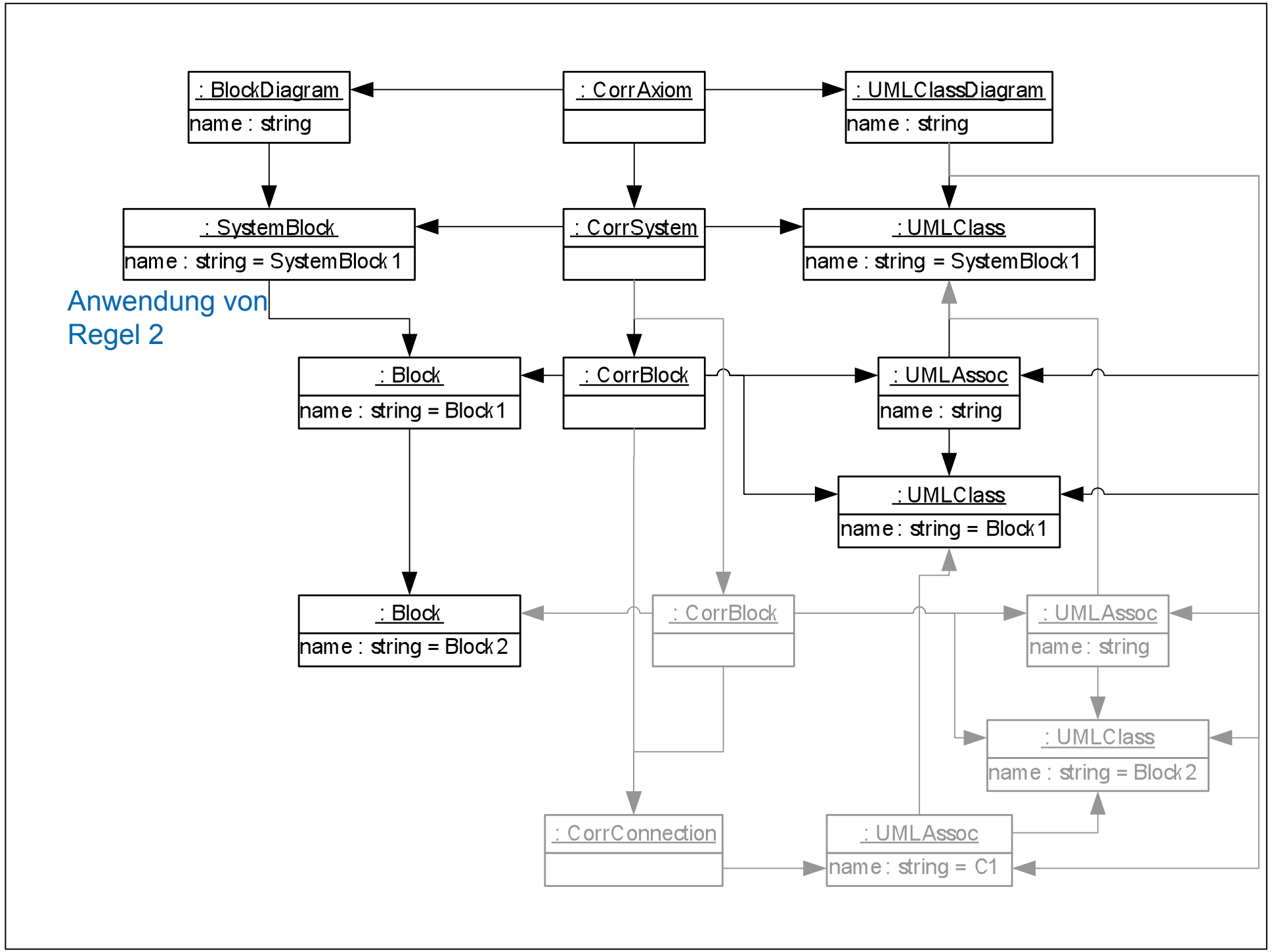
Synchronisations-algorithmus mittels TGGs

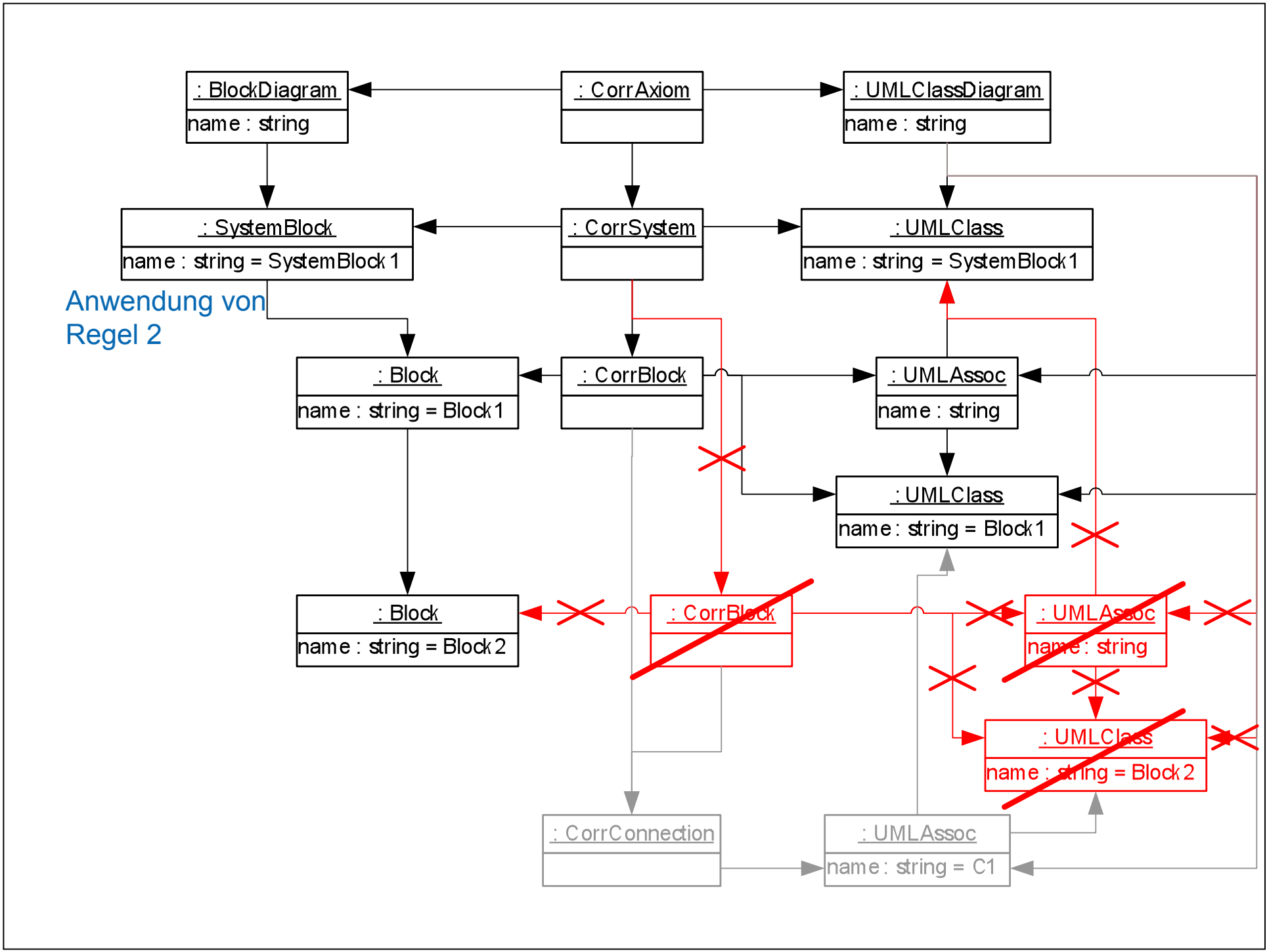
[Giese+2008]

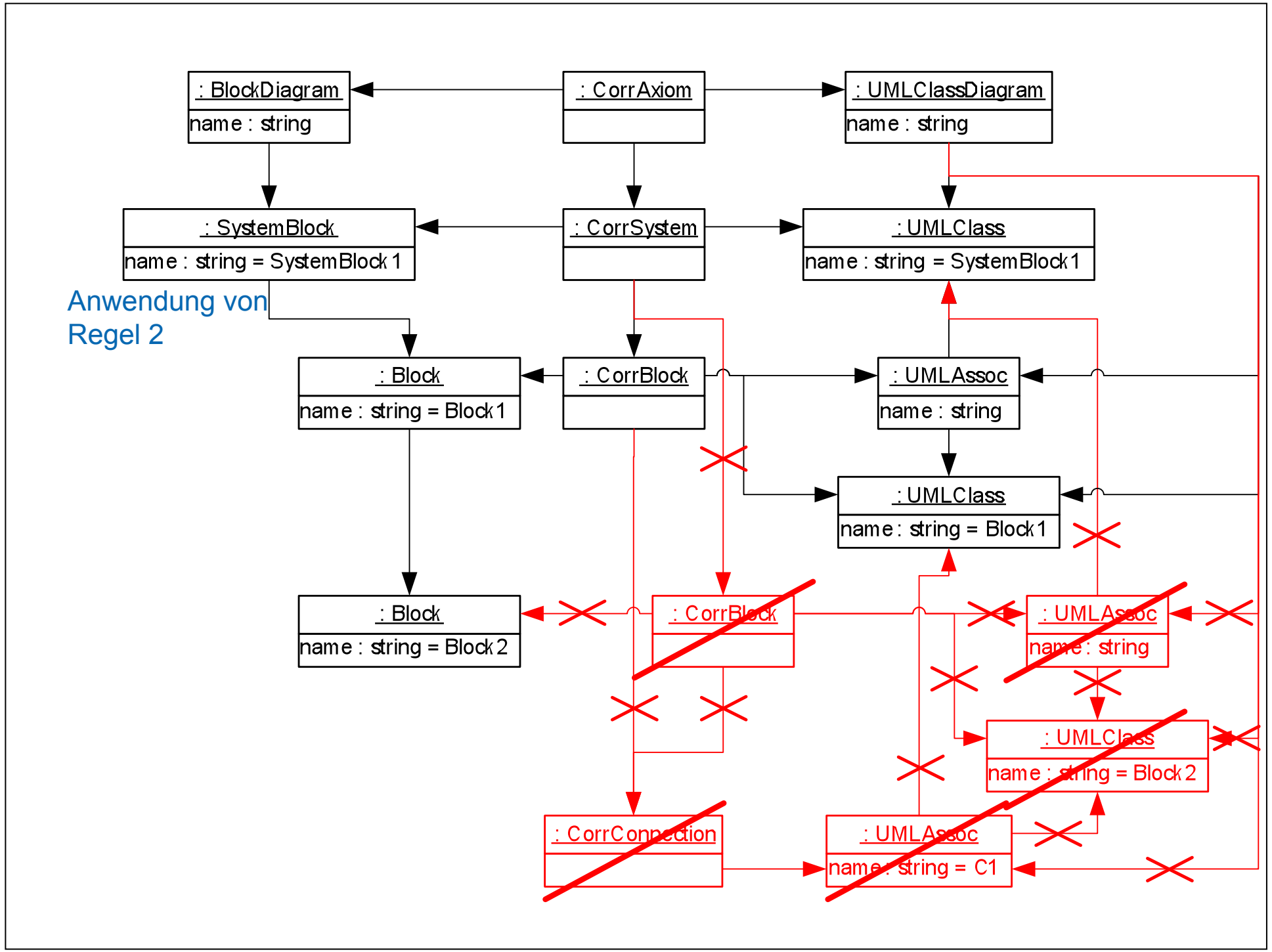
- Synchronisationsalgorithmus ähnlich Transformation
  - Durchlaufen des Korrespondenzmodells
  - Bei jedem Korrespondenzknoten
    - Prüfung auf Veränderungen
    - Transformieren hinzugefügter Elemente
    - Löschen und Neutransformieren veränderter Elemente
      - » Löschen von Elementen führt zum Löschen aller nachfolgenden Elemente
  - Lokale Synchronisation möglich
    - Start der Synchronisation bei beliebigem Korrespondenzknoten statt Wurzelknoten

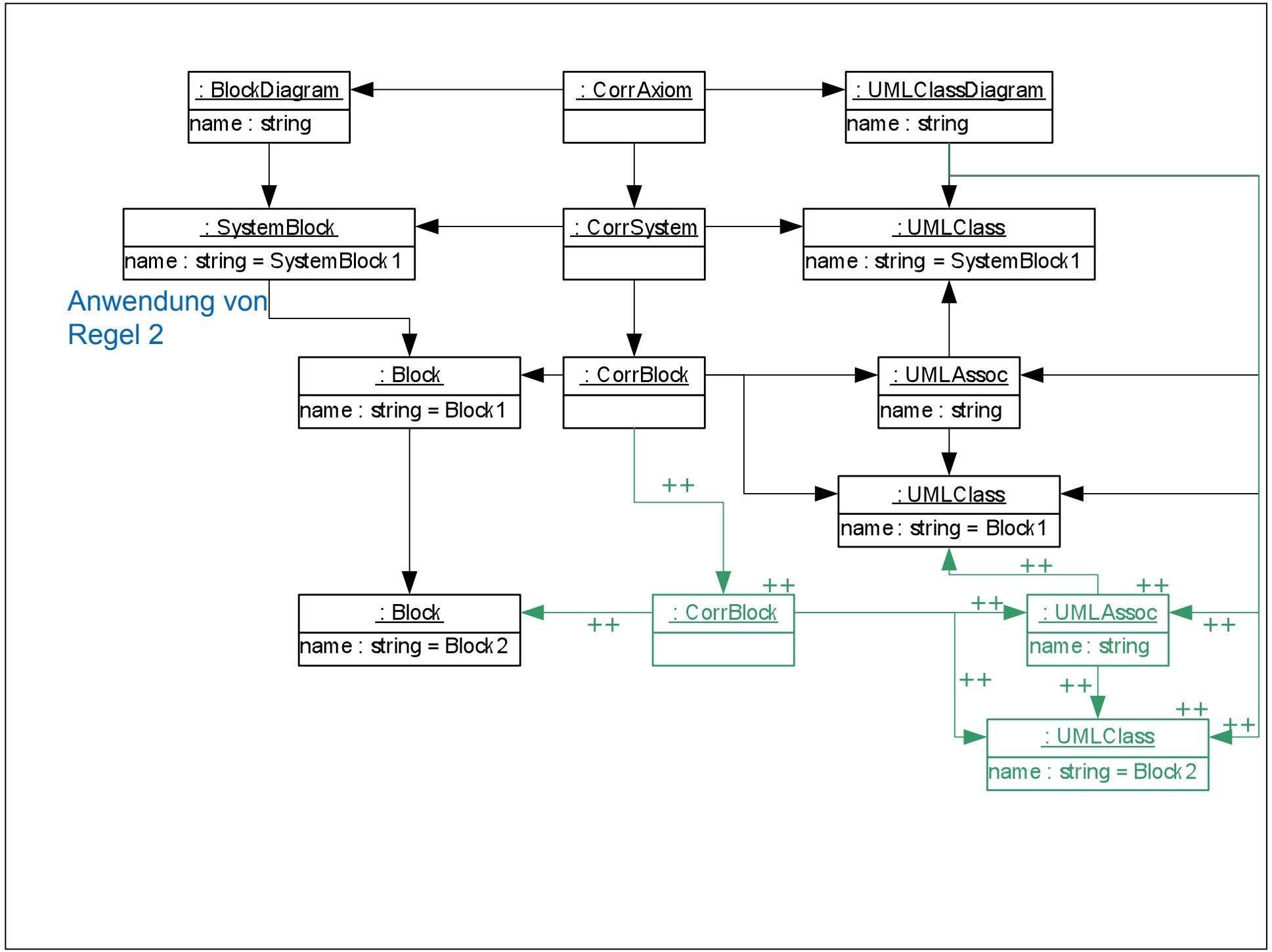

: BlockDiagram
name : string
: CorrAxiom
: UMLClassDiagram
name : string
: SystemBlock
name : string = SystemBlock1
: CorrSystem
: UMLClass
name : string = SystemBlock1
Veränderung im Quellmodell
: Block
name : string = Block1
: CorrBlock
: UMLAssoc
name : string
++
: UMLClass
name : string = Block1
: Block
name : string = Block2
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block2
target
: Connection
name : string = C1
source
: CorrConnection
: UMLAssoc
name : string = C1


: BlockDiagram
name : string
: CorrAxiom
: UMLClassDiagram
name : string
: SystemBlock
name : string = SystemBlock1
: CorrSystem
: UMLClass
name : string = SystemBlock1
Anwendung von
Regel 1
: Block
name : string = Block1
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block1
: Block
name : string = Block2
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block2
: CorrConnection
: UMLAssoc
name : string = C1

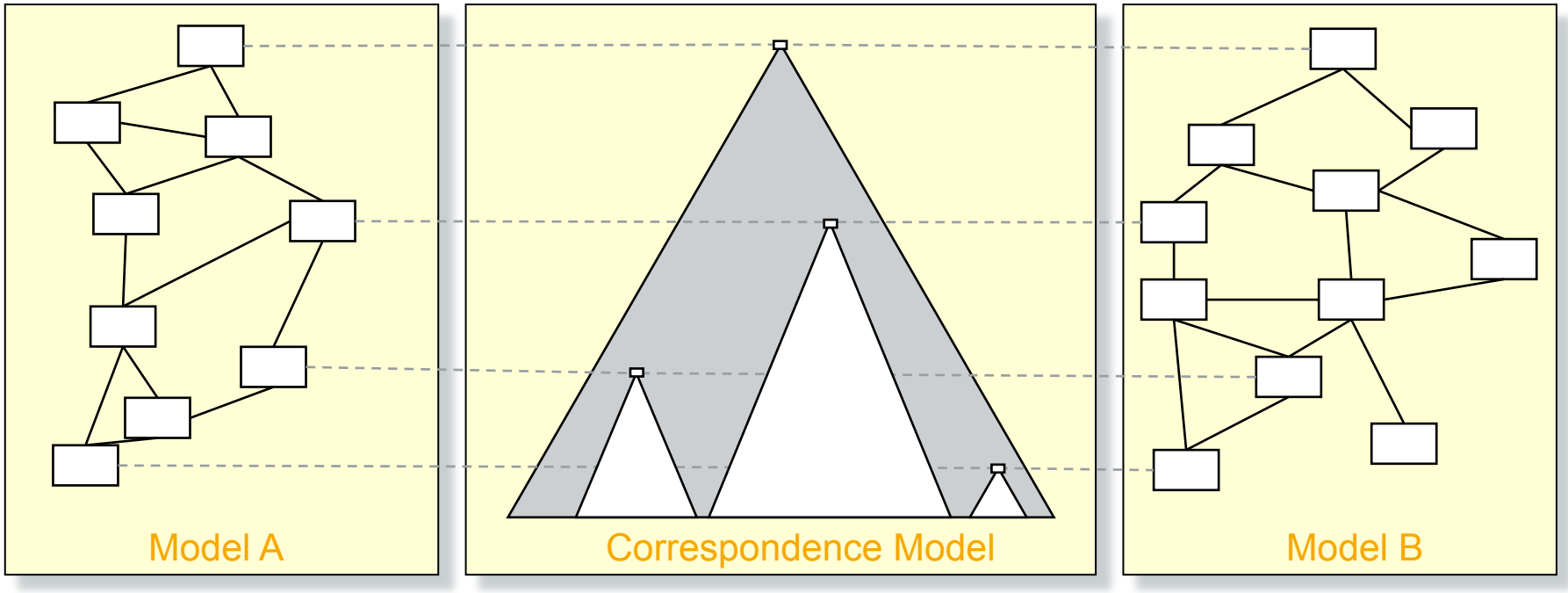

: BlockDiagram
name : string
: CorrAxiom
: UMLClassDiagram
name : string
: SystemBlock
name : string = SystemBlock1
: CorrSystem
: UMLClass
name : string = SystemBlock1
Anwendung von
Regel 2
: Block
name : string = Block1
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block1
: Block
name : string = Block2
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block2
: CorrConnection
: UMLAssoc
name : string = C1


: BlockDiagram
name : string
: CorrAxiom
: UMLClassDiagram
name : string
: SystemBlock
name : string = SystemBlock1
: CorrSystem
: UMLClass
name : string = SystemBlock1
Anwendung von
Regel 2
: Block
name : string = Block1
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block1
: Block
name : string = Block2
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block2
: CorrConnection
: UMLAssoc
name : string = C1


: BlockDiagram
name : string
: CorrAxiom
: UMLClassDiagram
name : string
: SystemBlock
name : string = SystemBlock1
: CorrSystem
: UMLClass
name : string = SystemBlock1
Anwendung von
Regel 2
: Block
name : string = Block1
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block1
: Block
name : string = Block2
: CorrBlock
: UMLAssoc
name : string
: UMLClass
name : string = Block2
: CorrConnection
: UMLAssoc
name : string = C1


: BlockDiagram
name : string
: CorrAxiom
: UMLClassDiagram
name : string
: SystemBlock
name : string = SystemBlock1
: CorrSystem
: UMLClass
name : string = SystemBlock1
Anwendung von
Regel 2
: Block
name : string = Block1
: CorrBlock
: UMLAssoc
name : string
++
: UMLClass
name : string = Block1
++
++
++
: Block
name : string = Block2
++
: CorrBlock
++
++
: UMLAssoc
name : string
++
++
++
++
: UMLClass
name : string = Block2
++

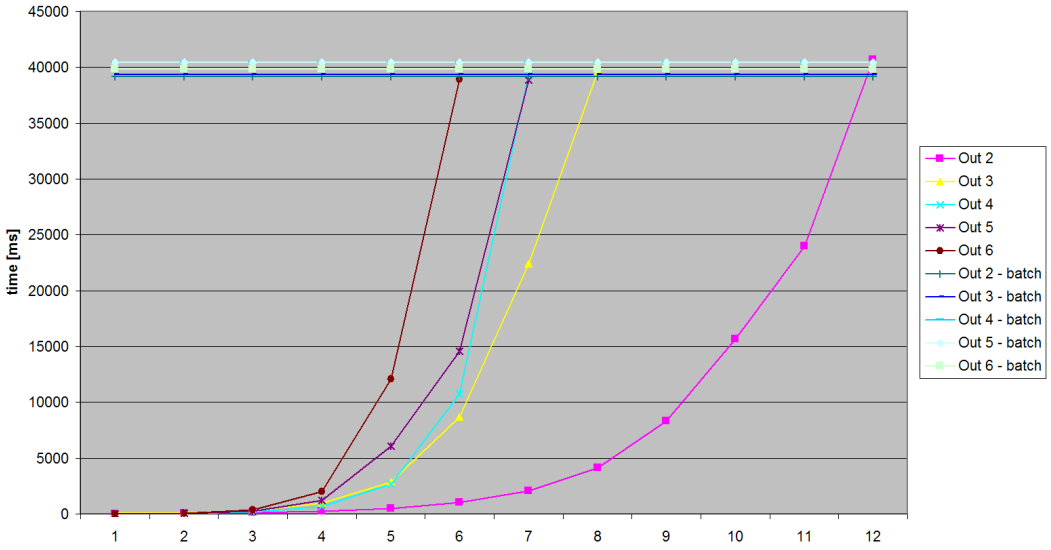
Inkrementelle Modellsynchronisation



- **Idee:** starte an den Änderungen
- Aufwand hängt nur von der Höhe im Korrespondenzgraph ab
- Im **Durchschnitt** hängt der Aufwand nur logarithmisch von der Modellgröße ab

[Giese+2008]

Messungen zur Modellsynchronisation

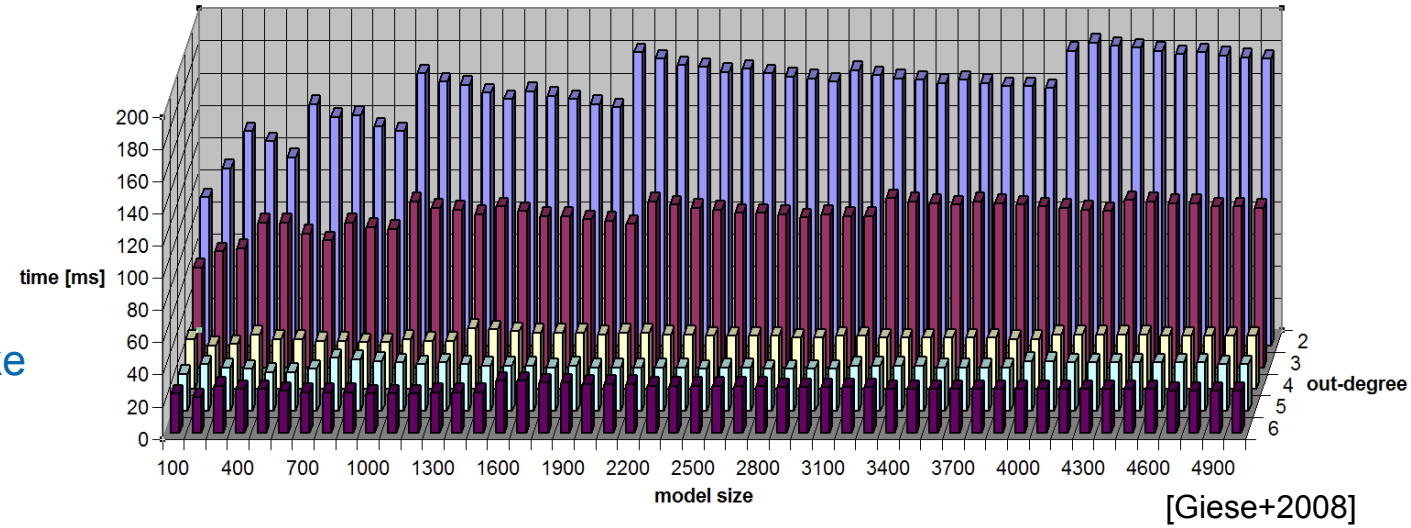


Aufwand pro Höhe:

- Ausgangsgrad: Anzahl der unter-geordneten Blöcke
- Feste Modellgröße: 5000 Blöcke

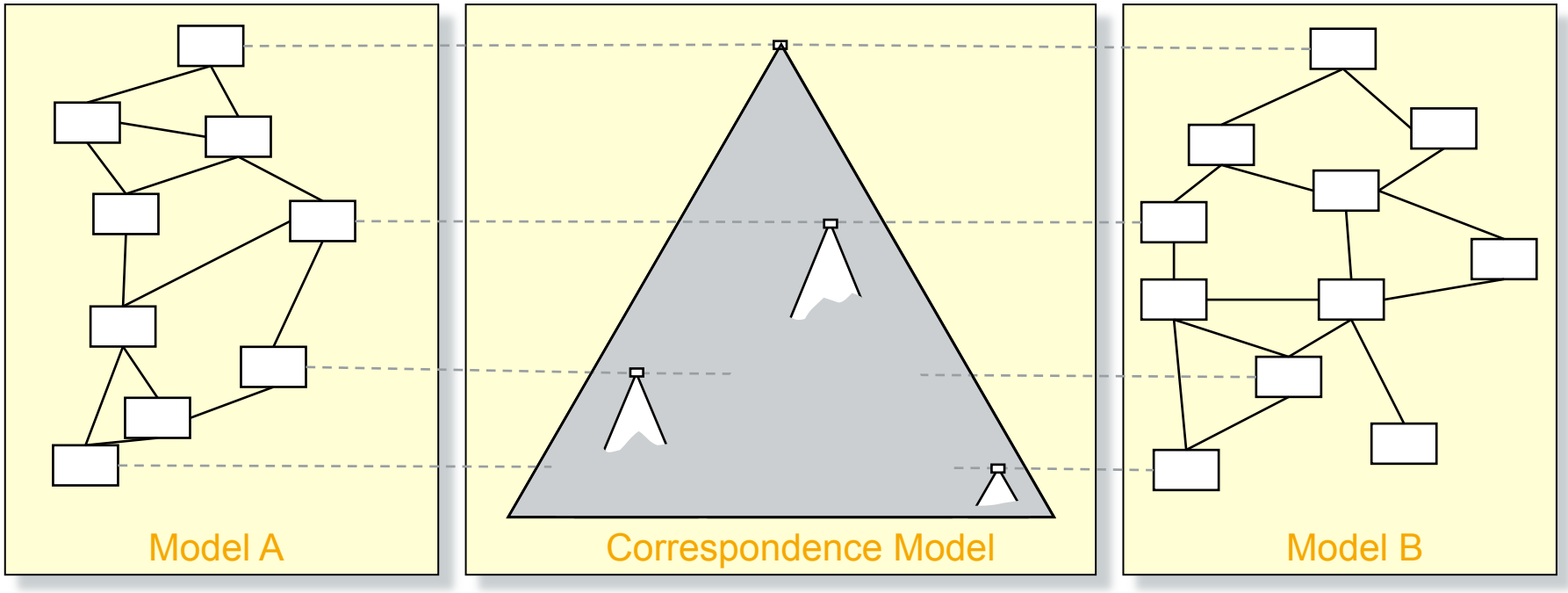
Durchschnittliche Aufwand:

- Ausgangsgrad: Anzahl der Unterblöcke
- Modellgröße: Anzahl der Blöcke



[Giese+2008]

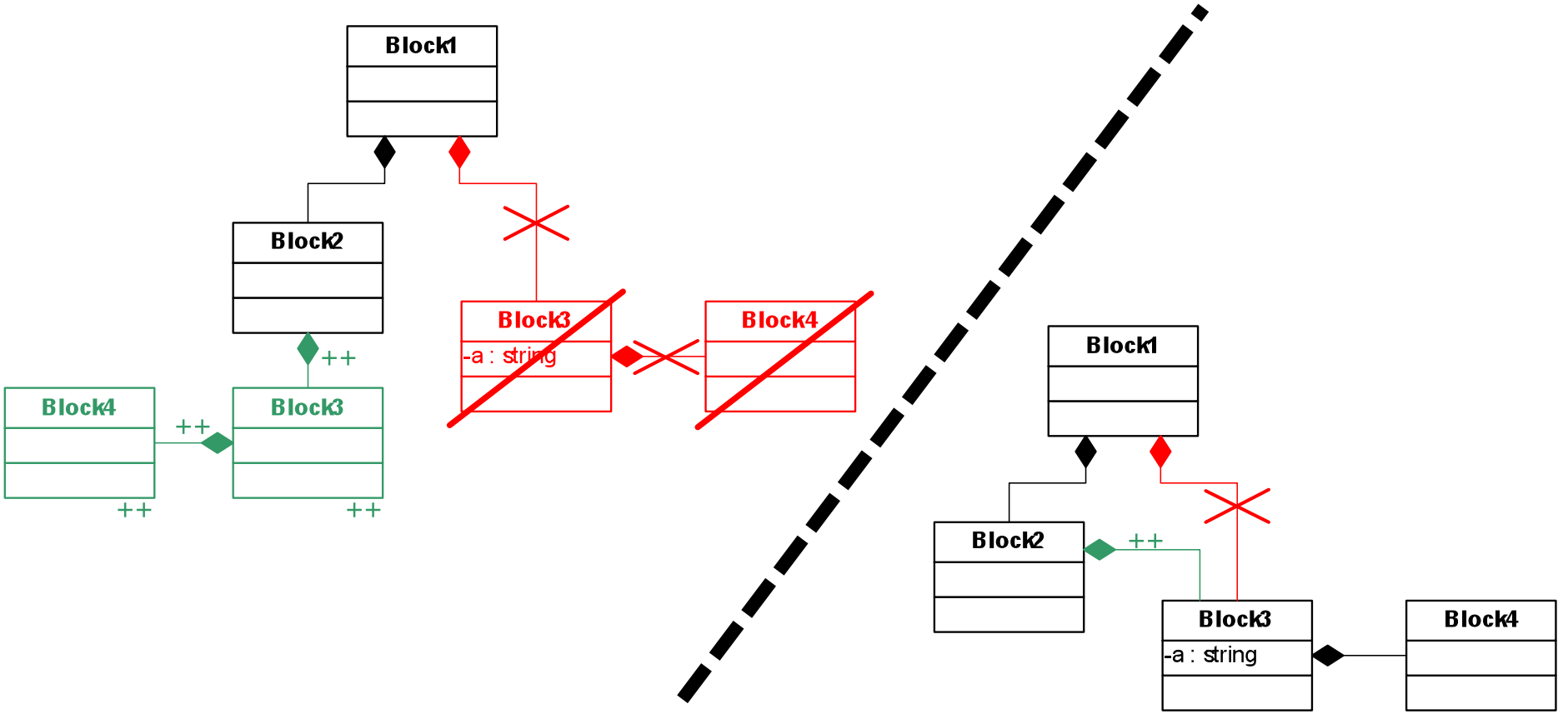
Optimierter Synchronisa-tionsalgorithmus



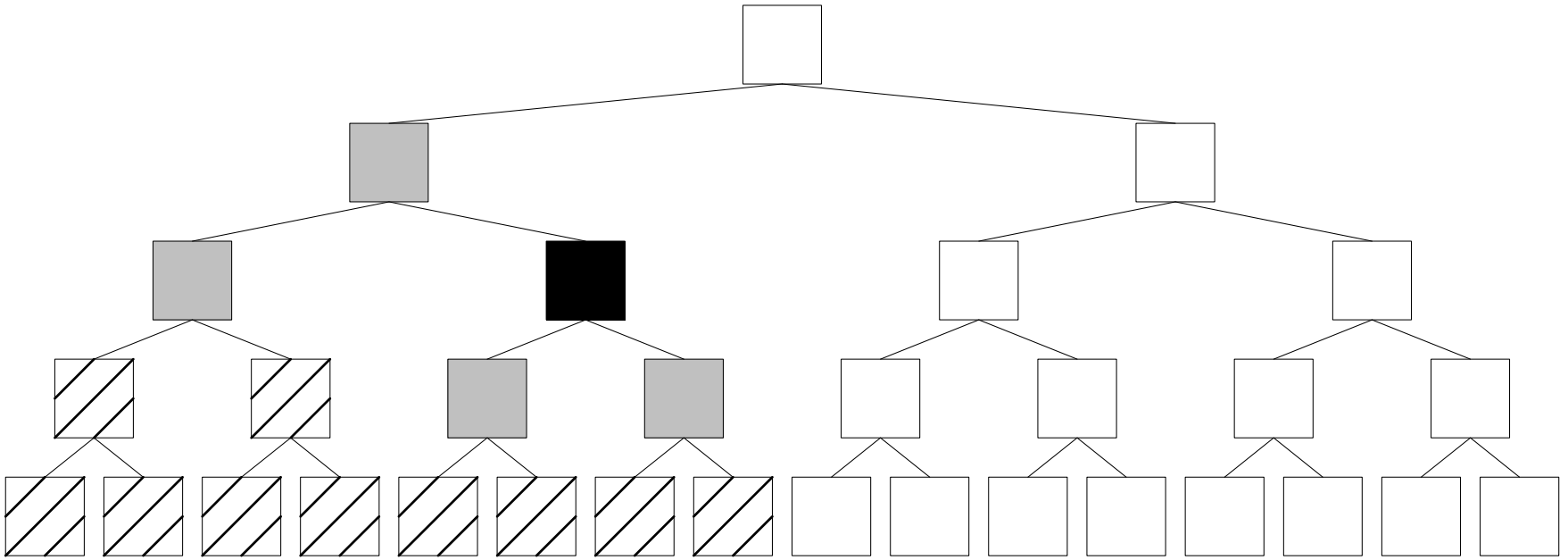
- **Ideen:**
  - Verwende Teile wieder (wo möglich)
  - Versuche zusätzlich Abbruch der Synchronisation (wo möglich)
- Aufwand hängt nicht mehr von der Höhe ab
- Im **Worst Case** bleibt der Aufwand häufig unabhängig von der Modellgröße

Beispiel – benötigte Synchronisation

Eigentlich wäre nur nötig:
Anpassung der Verknüpfungen im Ziel- und Korrespondenzmodell ausreichend



Lokale Synchronisation und Synchronisationsabbruch



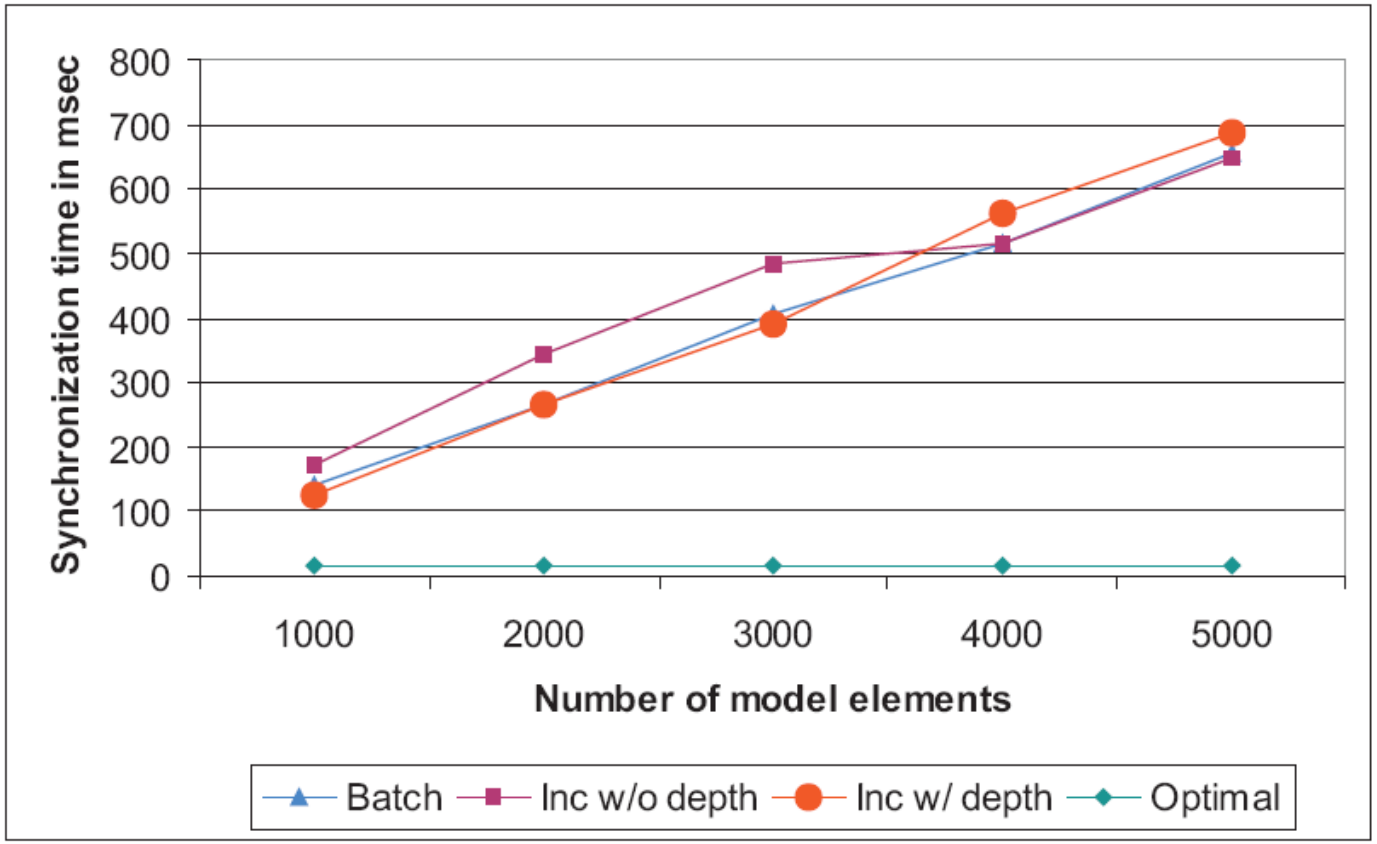
Korrespondenzknoten der Veränderung

Von altem und neuem Algorithmus durchlaufener Korrespondenzknoten

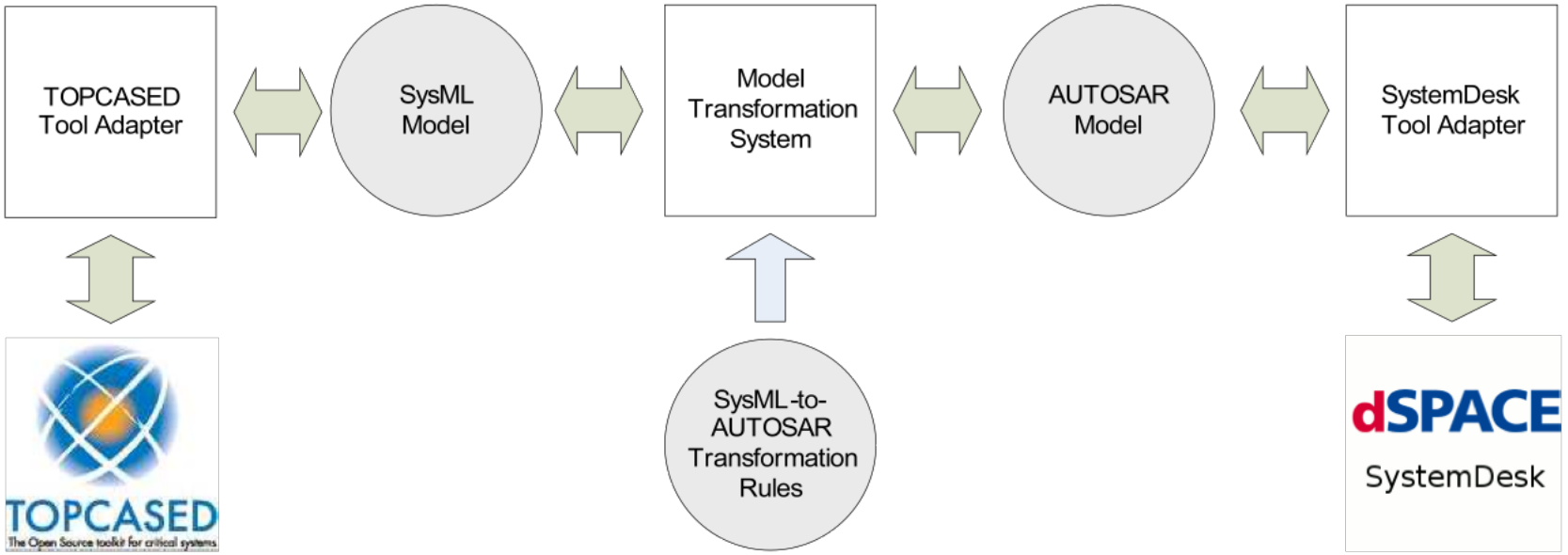
Von altem Algorithmus durchlaufener Korrespondenzknoten

Vergleich

- Optimierter TGG Algorithmus führt faktisch zur völliger Entkoppelung von der Modellgröße (für den Worst Case)



Beispielanwendung



Anwendungsbeispiel aus dem Bereich Automotive Systeme:

- SysML für Systemmodelle (TOPCASED)
- AUTOSAR für Modelle der Softwarearchitektur (SystemDesk)

Synchronisation mit TGGs - Zusammenfassung

- Es gibt ein prototypisches Werkzeug für Fujaba und EMF Modelle.

Vorteile:

- Nutz **deklaratives Modell** für bidirektionale Synchronisation
- Unterstützt **alle denkbaren Veränderungen** auf der Quell- oder Zielseite
- Ermöglicht extrem effiziente Synchronisation, die häufig von der Modellgröße unabhängig ist

Beschränkung:

- Umgang mit Konflikten durch parallele Änderungen?

II.2 Formale Grundlage & Verifikation

II MDE: GTS zur Definition, Manipulation und Charakterisierung von Eigenschaften der Modellierungssprachen

- Wohlverstandene Theorie ermöglicht effiziente Lösungen
- Formale Grundlage & Verifikation

III MDE: GTS zur Modellierung und Analyse hochgradig dynamischer Systeme

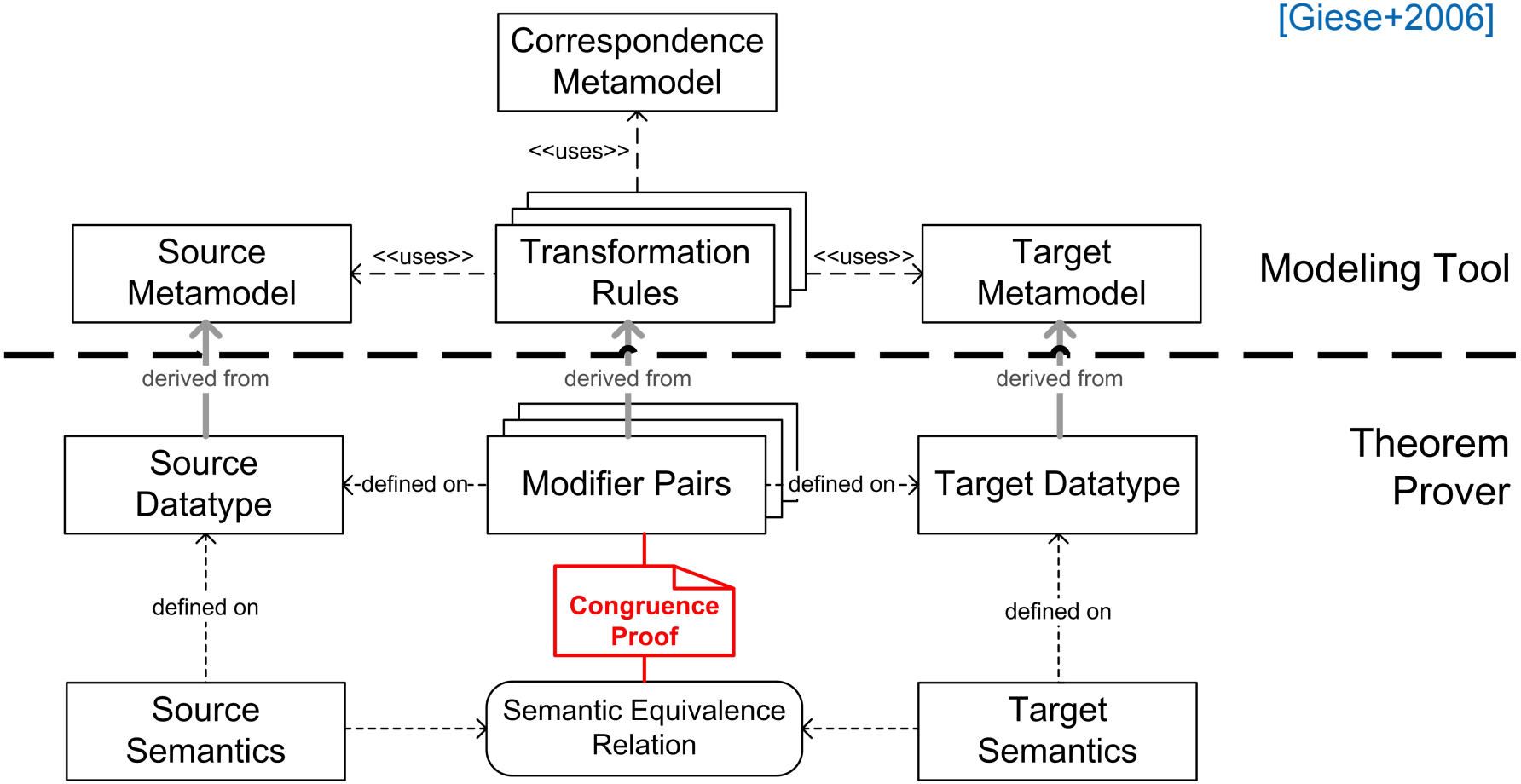
- Erzeugung und Löschung von Komponenten
- Veränderliche Strukturen

IV MDE & Self-Managed Systeme: Kombination beider vorherigen Fälle

- Manipulation der Software zur Laufzeit erfolgt über Modelle

Verifikation der Transformation: TGGs

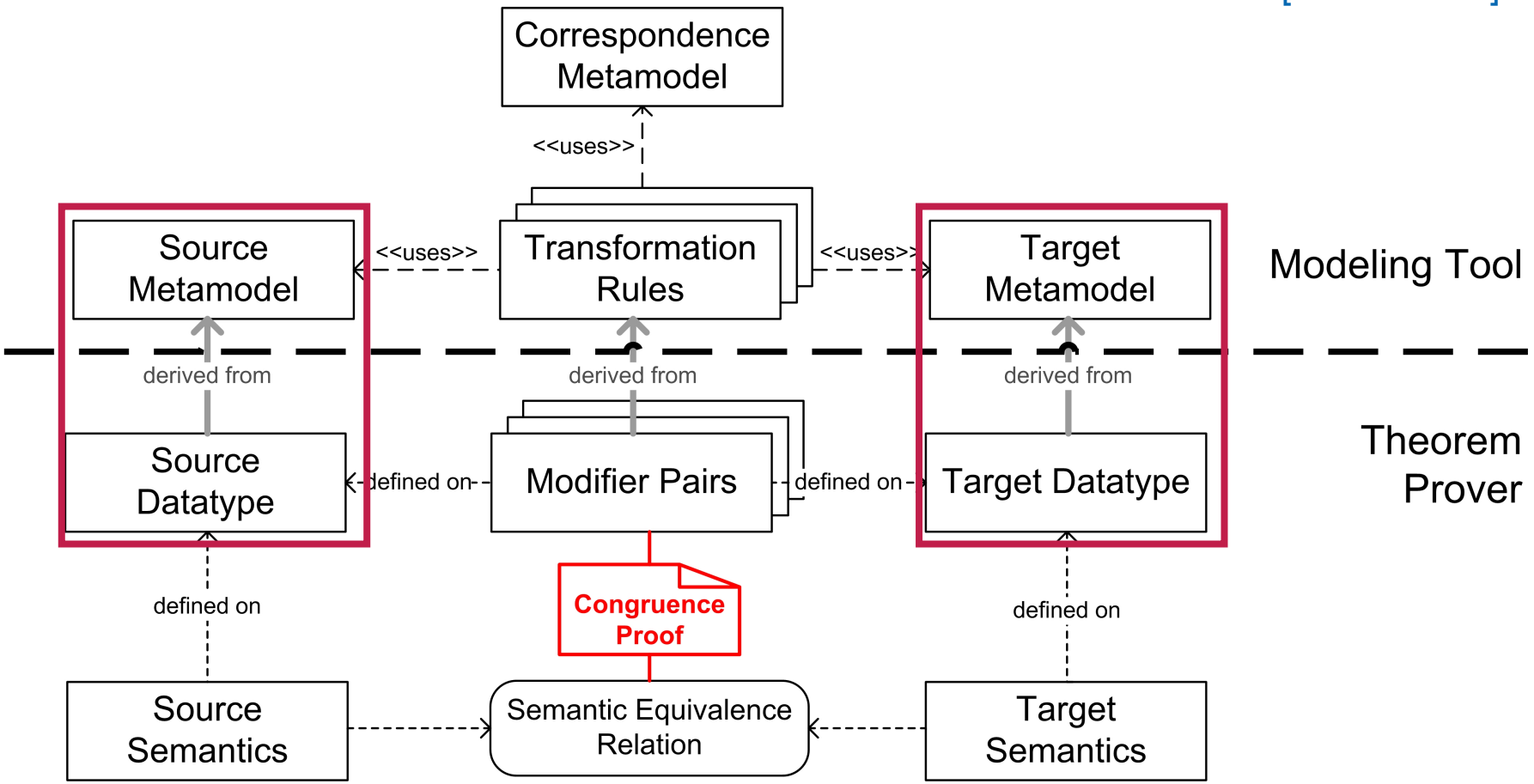
[Giese+2006]



- **Nur:** I/O-Automata nach PLC Code (mit Isabelle/HOL)

Verifikation der Transformation: TGGs

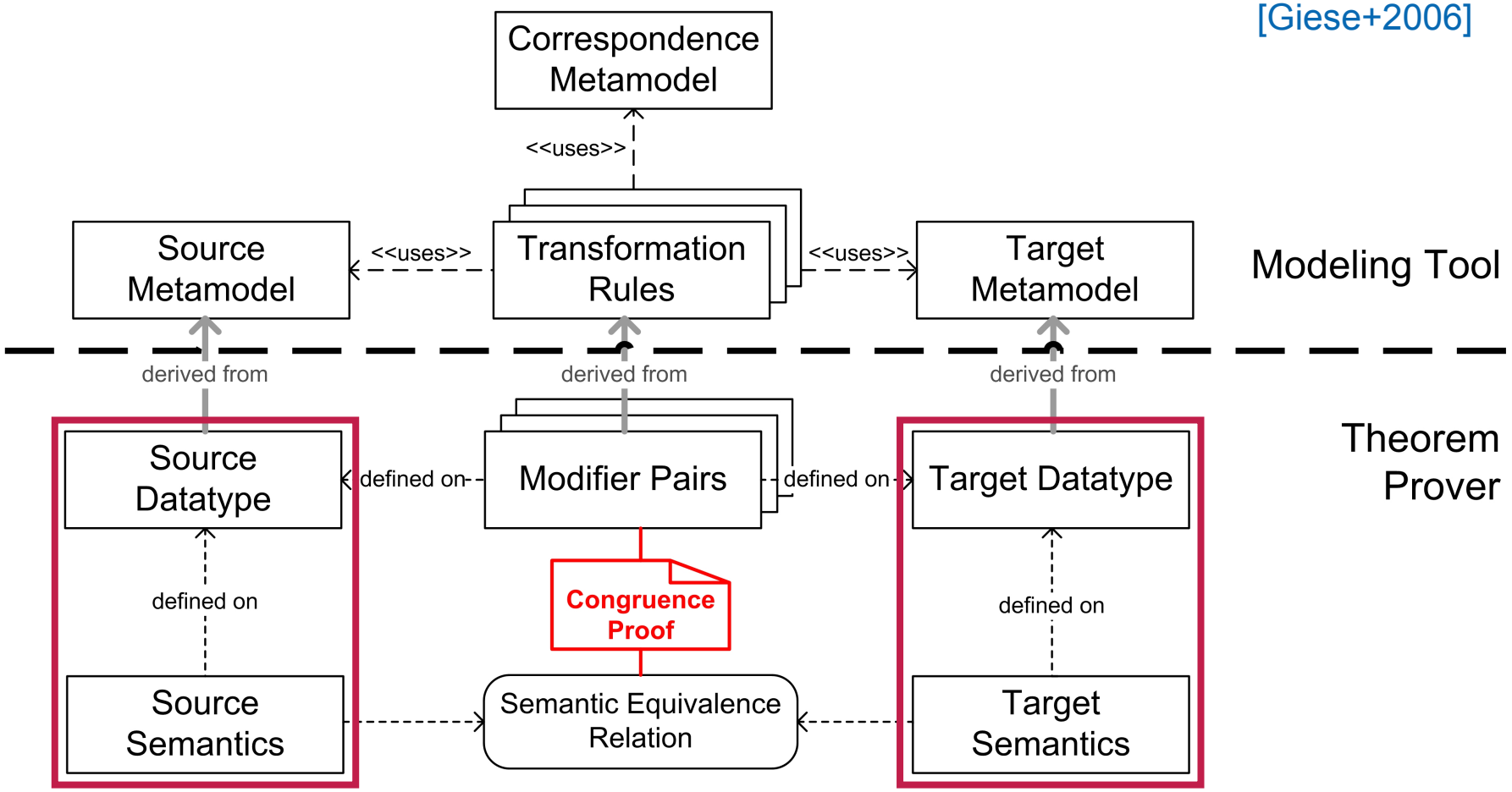
[Giese+2006]



- **Schritt 1:** Formalisierung des Metamodells in Isabelle/HOL

Verifikation der Transformation: TGGs

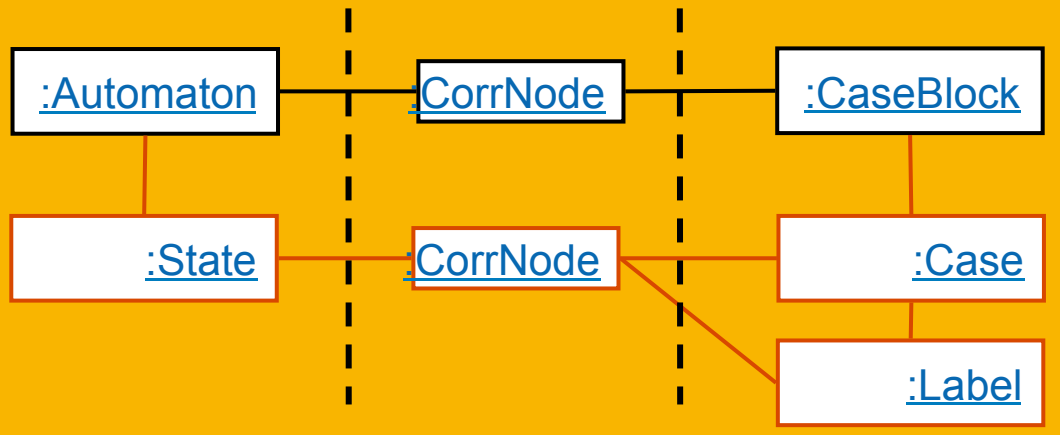
[Giese+2006]

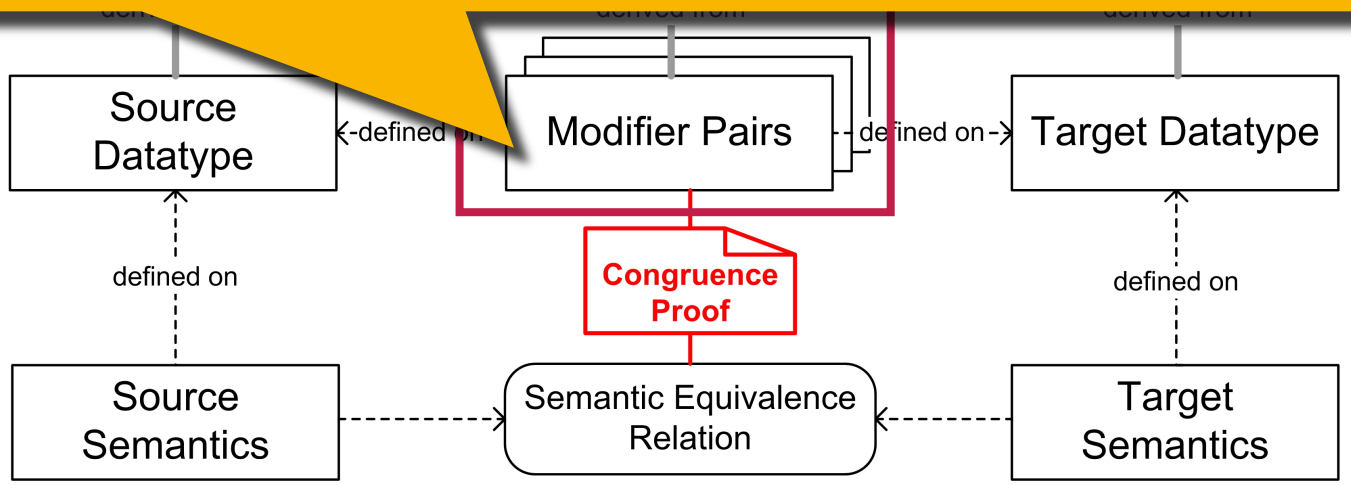


- **Schritt 2:** Definition der Semantik (rekursive Funktion)

Verifikation der Transformation: TGGs

Definition der Modifiers analog zu den TGG-Regeln:
- Erzeugen von Zuständen, Transitionen, Aktionen, …
- Erzeugen von Case-Blöcken, If-Anweisungen, Zuweisungen, …





Theorem Prover

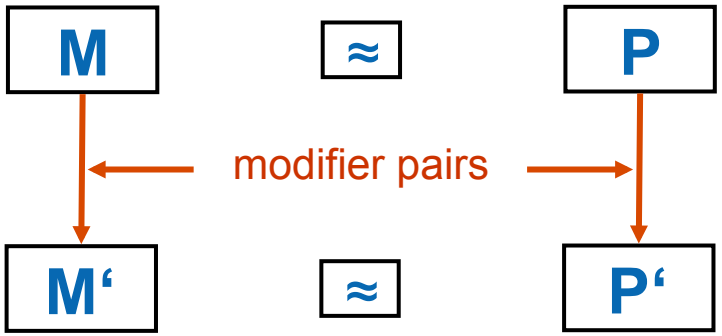
[Giese+2006]

Verifikation der Transformation: TGGs

[Giese+2006]

- **Axiom:** Leerer Automat und leeres PLC-Programm sind äquivalent.

- **Induktionsschritt:** Gleichzeitige Erweiterungen der Modelle durch die TGG erhalten die Äquivalenz.



- **Ergebnis:** Für jedes Modell M_n und das entsprechende Programm P_n = trans(M_n), dass durch die TGG erzeugt wurde gilt $M_n \approx P_n$

- Für die Verifikation einer relativ einfachen TGG wurden ca. **1500 Lopc** benötigt!

III MDE & hochgradig dynamische Systeme

II MDE: GTS zur Definition, Manipulation und Charakterisierung von Eigenschaften der Modellierungssprachen

- Wohlverstandene Theorie ermöglicht effiziente Lösungen
- Formale Grundlage & Verifikation

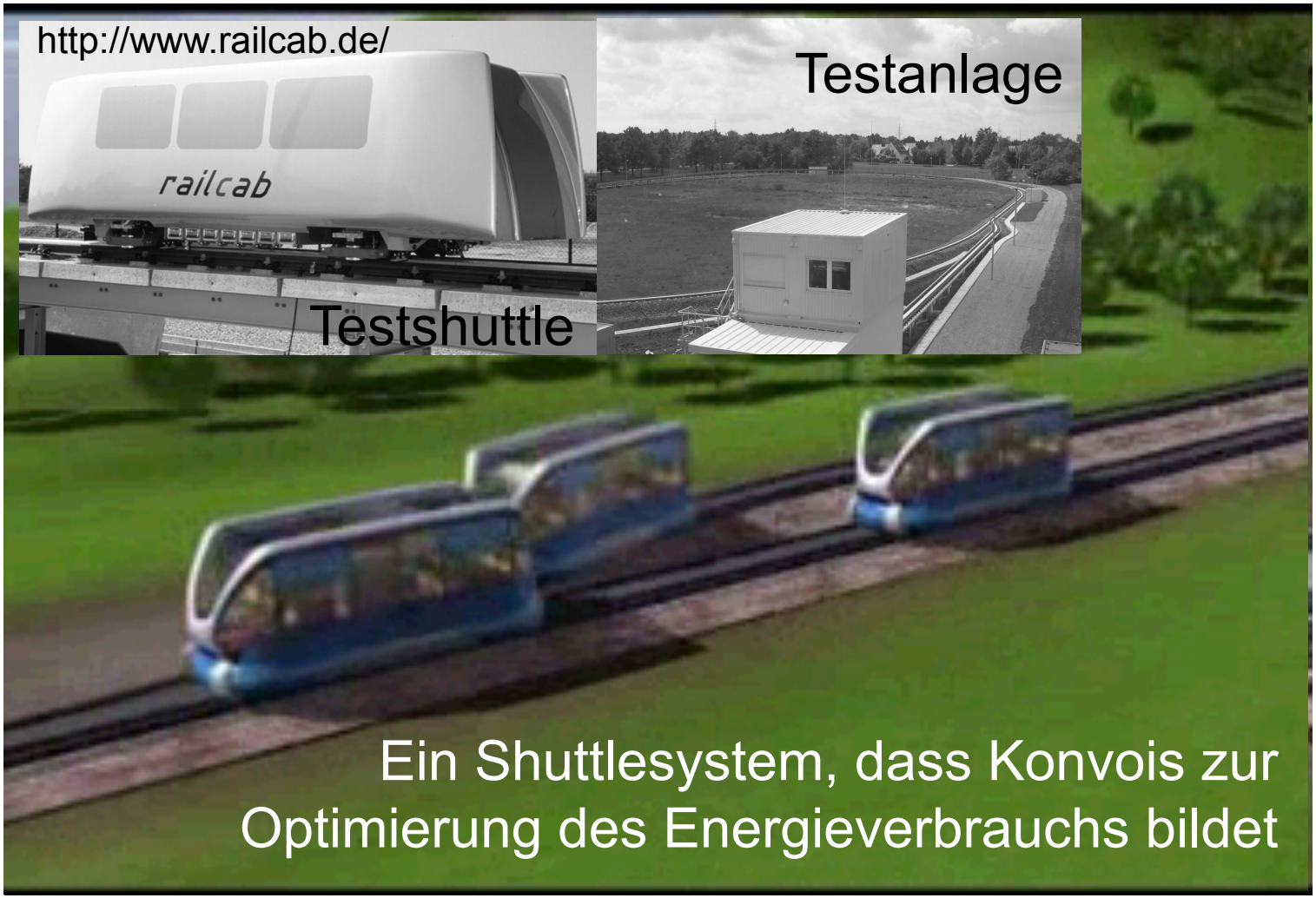
III MDE: GTS zur Modellierung und Analyse hochgradig dynamischer Systeme

- Erzeugung und Löschung von Komponenten
- Veränderliche Strukturen

IV MDE & Self-Managed Systeme: Kombination beider vorherigen Fälle

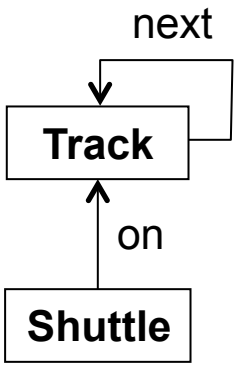
- Manipulation der Software zur Laufzeit erfolgt über Modelle

Beispiel für hochgradig dynamische Systeme

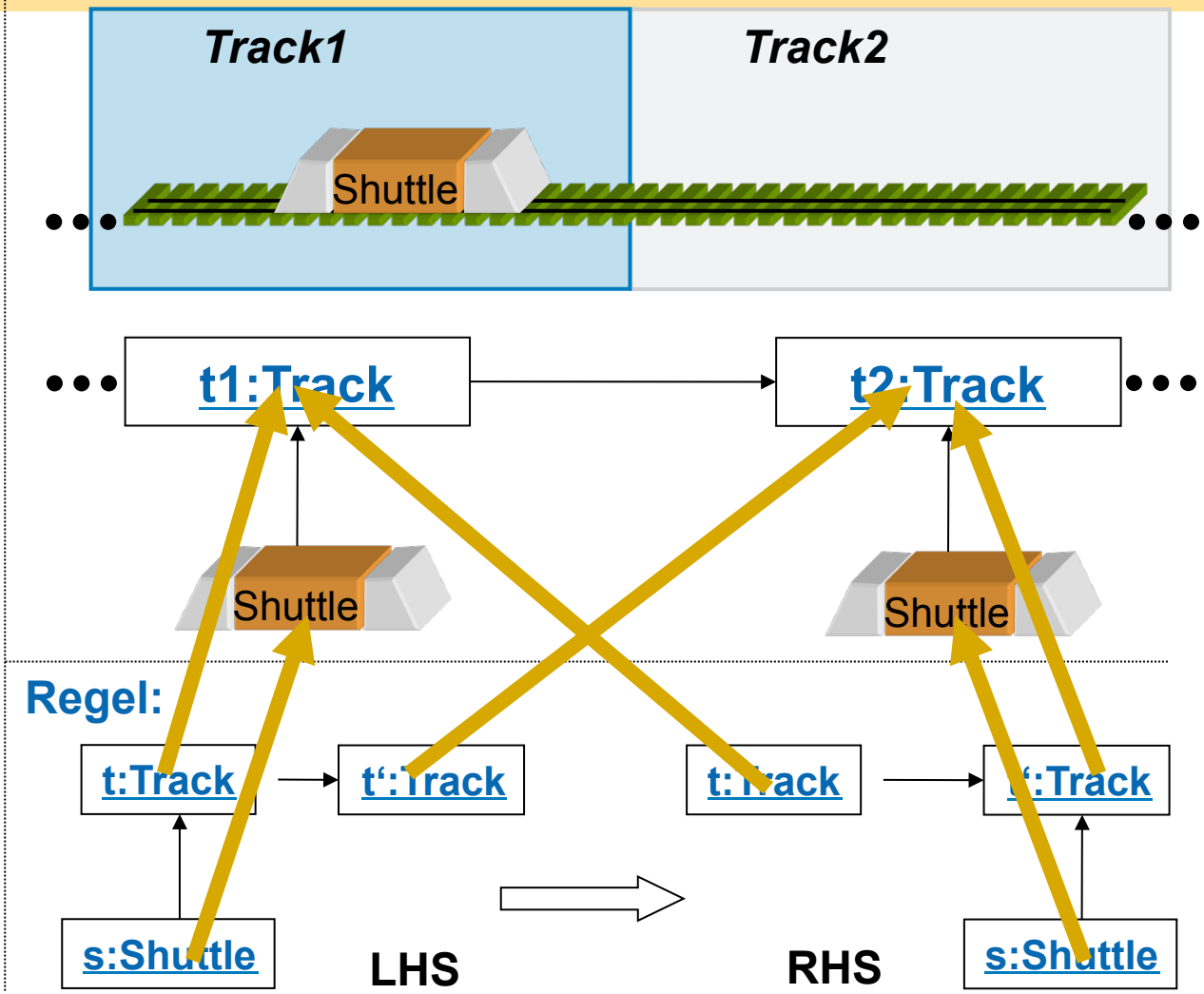


Struktur und Verhalten

- Abbildung der Gleisabschnitte
- Abbildung der Shuttles



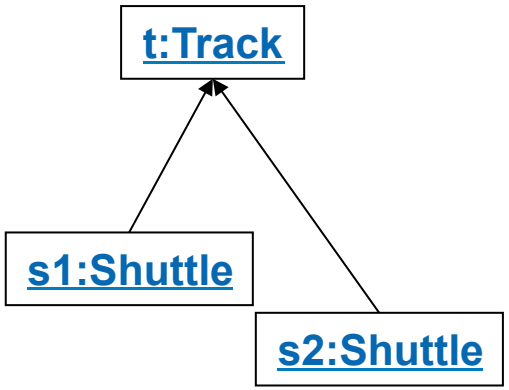
- Abbildung der Bewegung und Entscheidung-en auf Regeln

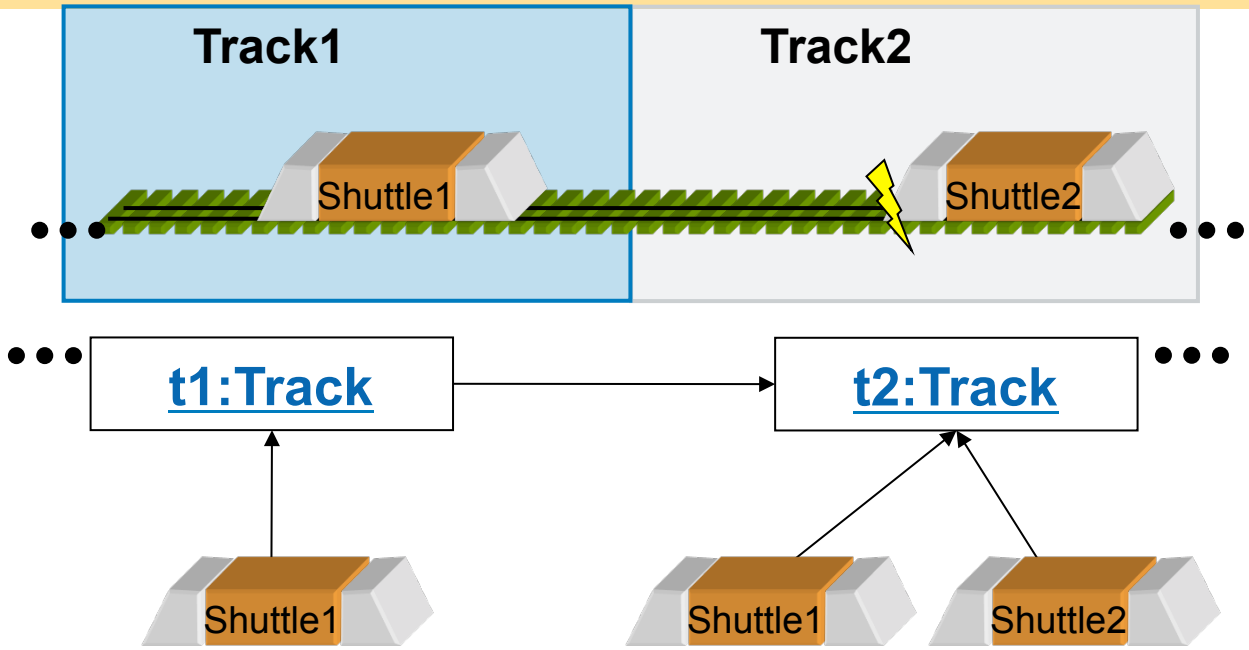


[Becker+2006]

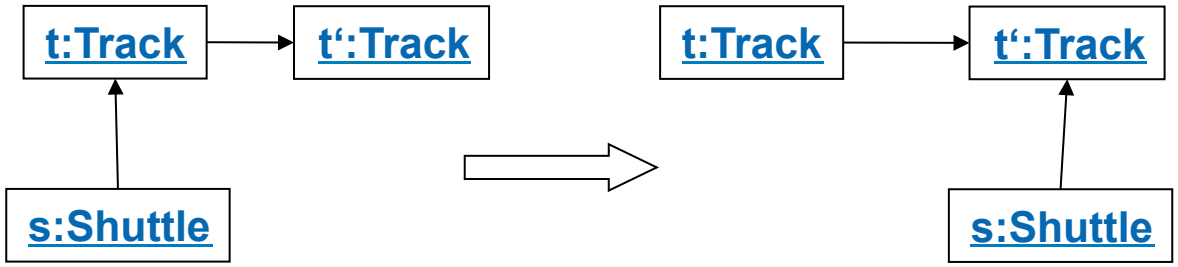
Ein unsicherer Fall ...

Verbotener Graph





Regel:



[Becker+2006]

(1) Verifikation mittels Modelchecking

HPI Hasso Plattner Institut

FUJABA Real-Time Tool Suite

Modelchecking (MC):

- Backend: GTS Modelchecker GROOVE
- **Eine** feste Topologie mit 15 Tracks
- Bedingung: **Operationale Invariante**

Verifikationszeiten:

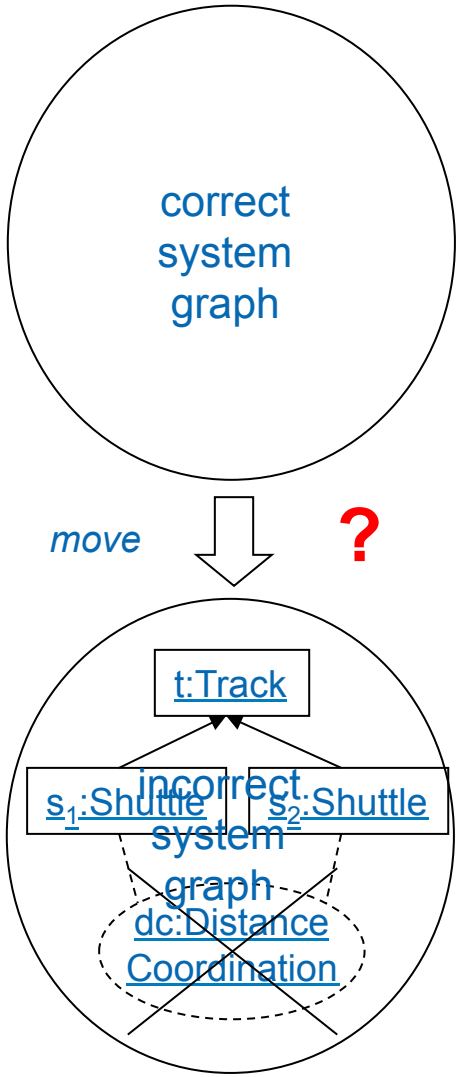
- 3 Shuttles ⇨ 2 min
- 4 Shuttles ⇨ 7 min
- 5 Shuttles ⇨ 55 min
- 6 Shuttles ???

[Becker+2006]

(2) Mittels Induktiver Invariante

- **Induktive Invariante:** jede Regelanwendung bzgl. Eines korrekten Graphen führt wieder zu einem korrekten Graphen (stärker als die Operationale Invariante ⇨ hinreichend aber nicht notwendig)
- **Verifikationsidee:**
  - ⇨ Repräsentiert unendlich viele Graphen nur durch den für die Regel relevanten Teil
  - ⇨ Gehe rückwärts vor

[Becker+2006]



Idee: etwas konkreter ...

- **Beobachtung:** jedes Gegenbeispiel muss den Schnitt aus der RHS einer Regel und eines verbotenen Graphen enthalten.

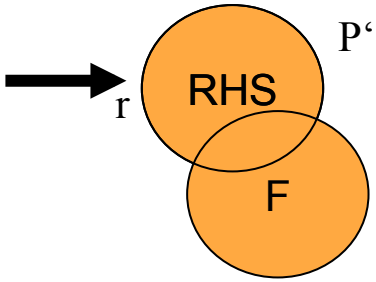
Ist (P,r) ein **Gegenbeispiel**, dann:

(1) Es existiert ein P', dass eine Kombination der RHS einer Regel $r \in R$ und eines verbotenen Graphen $F \in F$,

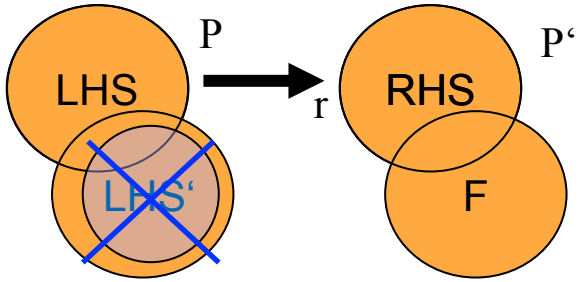
(2) mit $P \rightarrow_r P'$ (d.h. keine höher priorisierte Regel kann schalten) und

(3) Es existiert kein $F' \in F$, dass in P enthalten ist

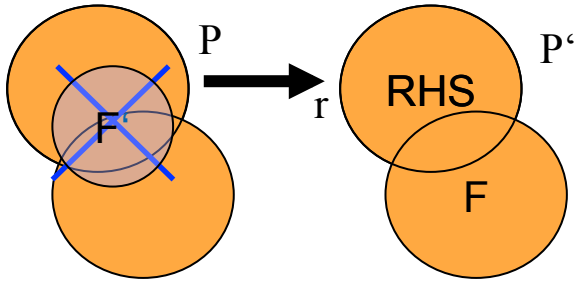
(1)



(2)



(3)

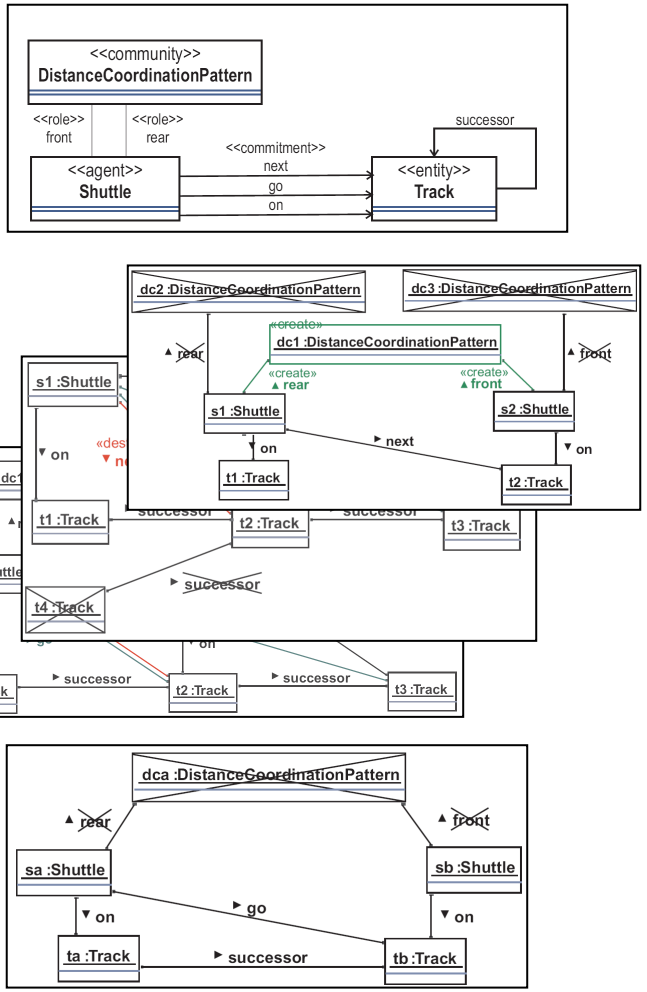


[Becker+2006]

Fallstudie

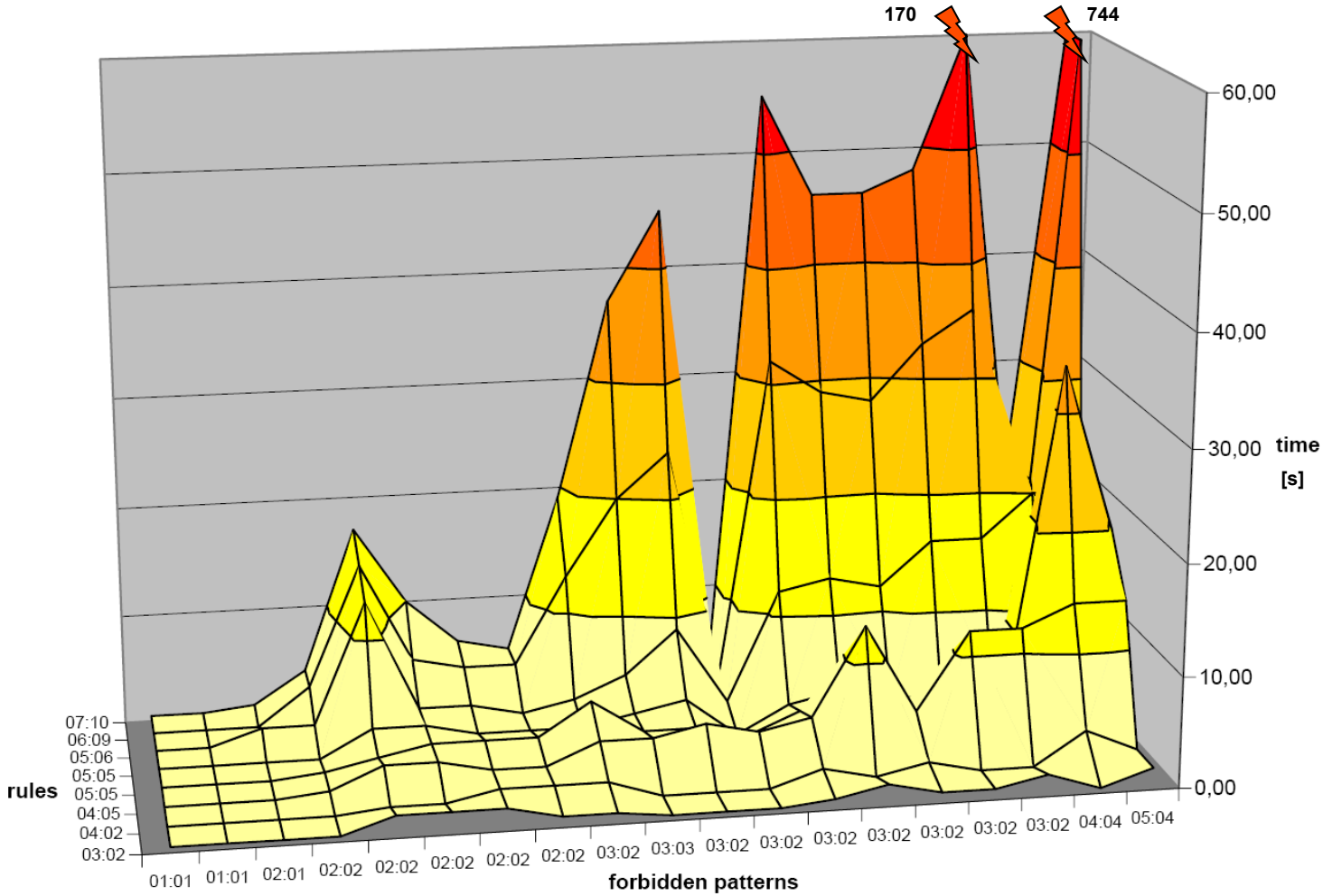
Charakteristika:

- 3 Klassen
- 6 Assoziationen
- 8 Regeln
  - Minimum: 3 Knoten and 2 Kanten
  - Maximum: 7 Knoten and 10 Kanten
- 19 verbotene Graphen
  - Minimum: 1 Knoten and 1 Kanten
  - Maximum: 5 Knoten and 4 Kanten



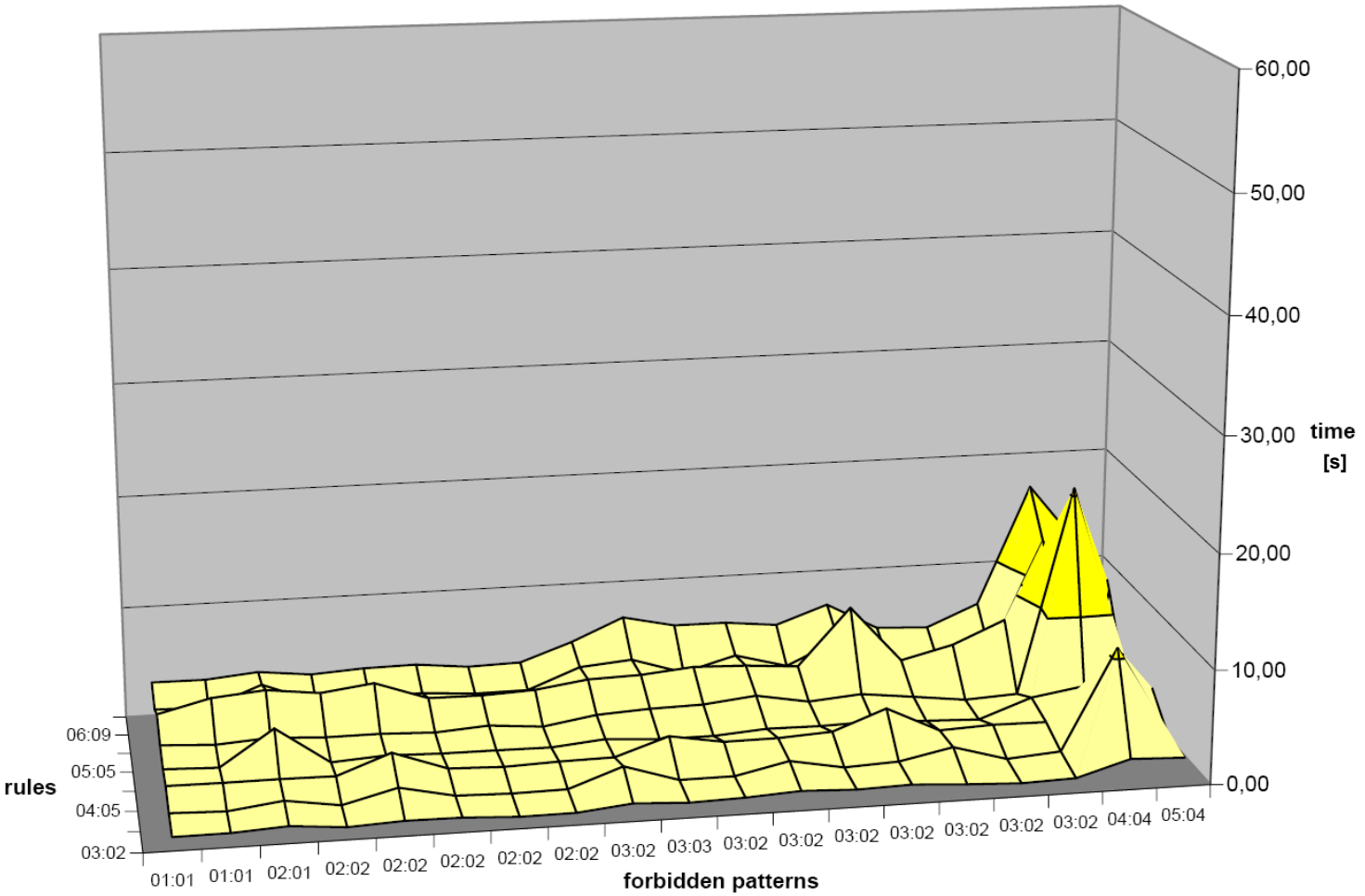
[Becker+2006]

Invariant Check: Expliziter Algorithmus



[Becker+2006]

Invariant Check: Symbolischer Algorithmus



Selbe Skalierung

[Becker+2006]

IV MDE & Self-Managed Systeme

II MDE: GTS zur Definition, Manipulation und Charakterisierung von Eigenschaften der Modellierungssprachen

- Wohlverstandene Theorie ermöglicht effiziente Lösungen
- Formale Grundlage & Verifikation

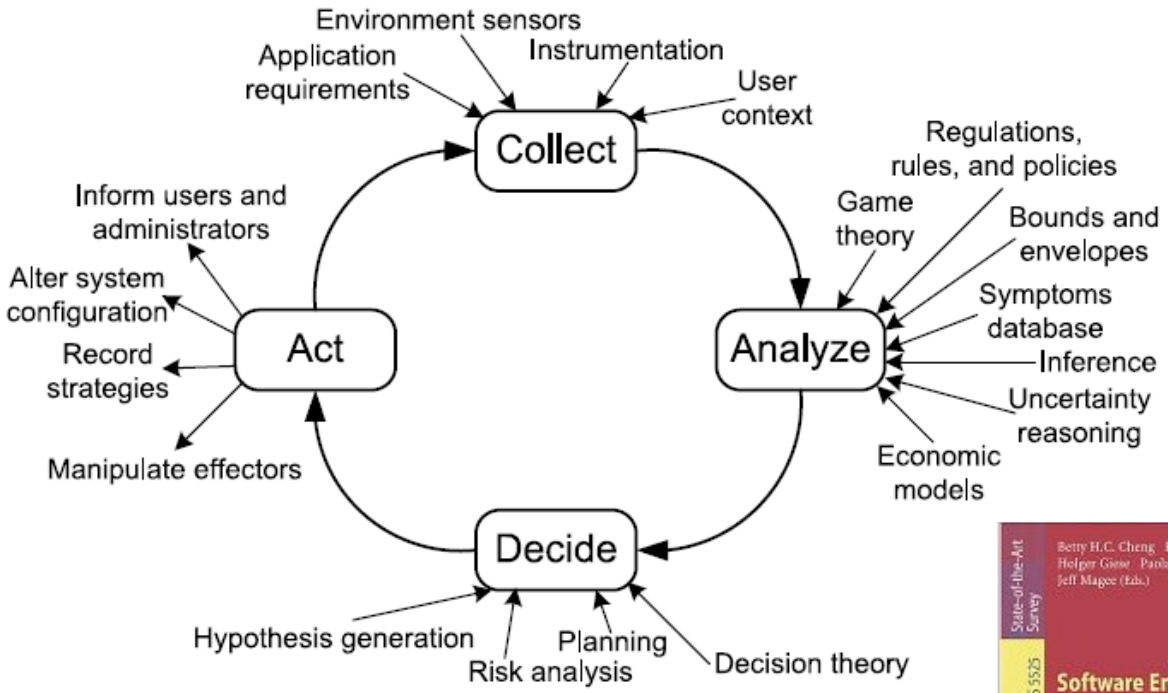
III MDE: GTS zur Modellierung und Analyse hochgradig dynamischer Systeme

- Erzeugung und Löschung von Komponenten
- Veränderliche Strukturen

IV MDE & Self-Managed Systeme: Kombination beider vorherigen Fälle

- Manipulation der Software zur Laufzeit erfolgt über Modelle

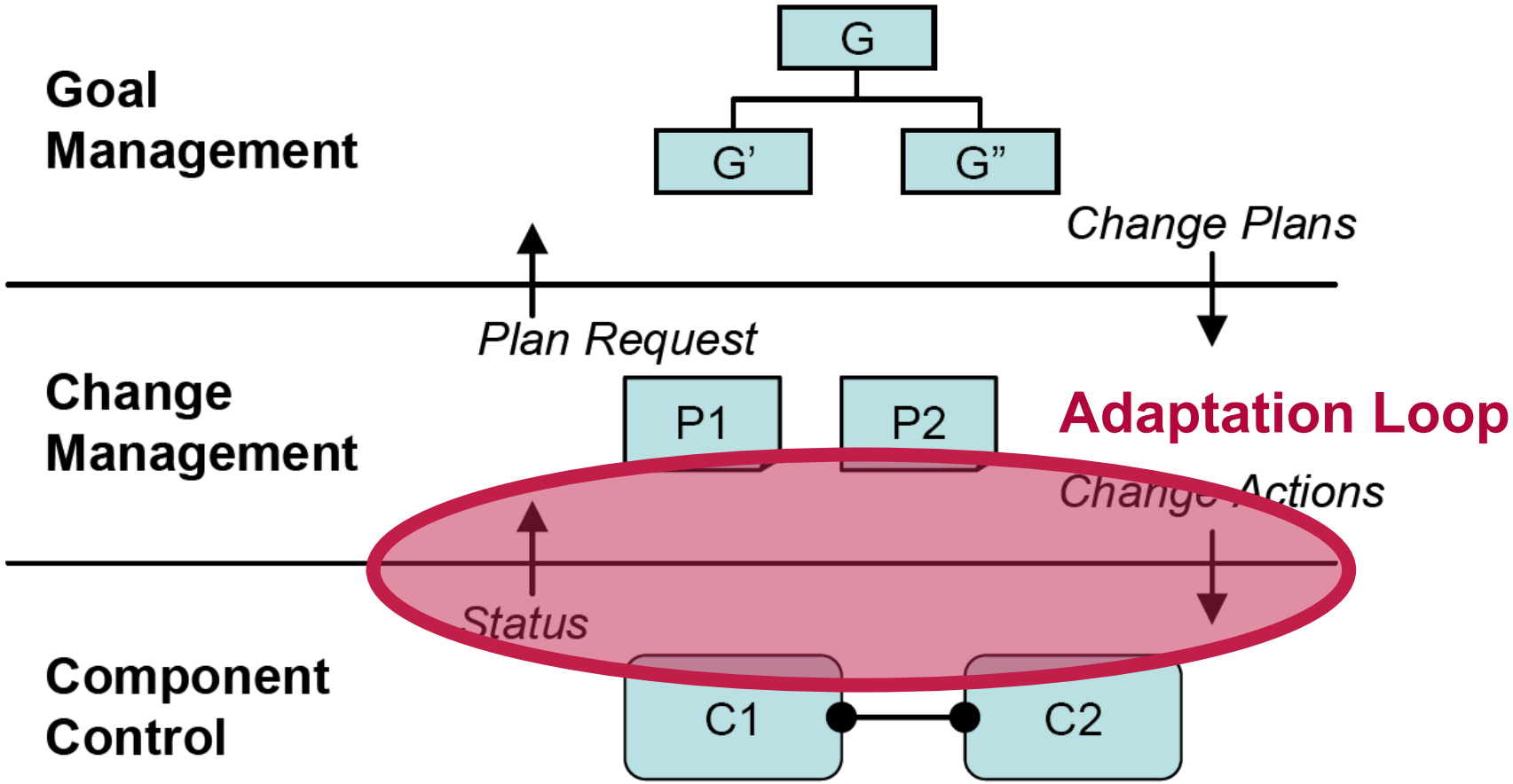
Adaptation Loop



Roadmap:

- Requirements
- Modeling
- **Feedback loops**
- Assurance

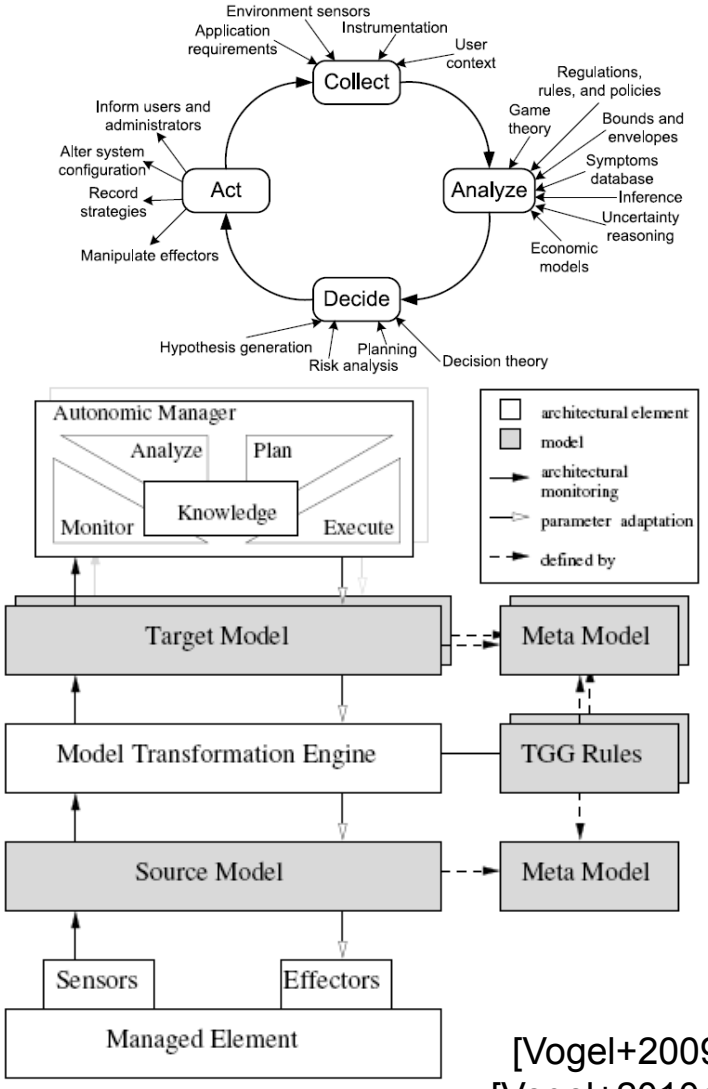
(Reference) Architecture Self-Managed Software



[Kramer&Magee2007]

Sensors + Effectors = Synchronisation

- Unterstützung der **Adaptation Loops** mit Modellen als Schnittstelle durch "Meta-Modelle" (EMF) und TGGs
- Extrahiert abstrakte Laufzeitmodelle für verschiedene Autonomic Manager (**collect**)
- Autonomic Manager können Laufzeitmodelle analysieren und ihr Vorgehen entscheiden (**analyzse** und **decide**)
- Passt die Managed Subsystem über die Laufzeitmodelle an (**act**)
- Synchronisiert Laufzeitmodelle **inkrementell**
- **Beispiel:** EJB Application Server



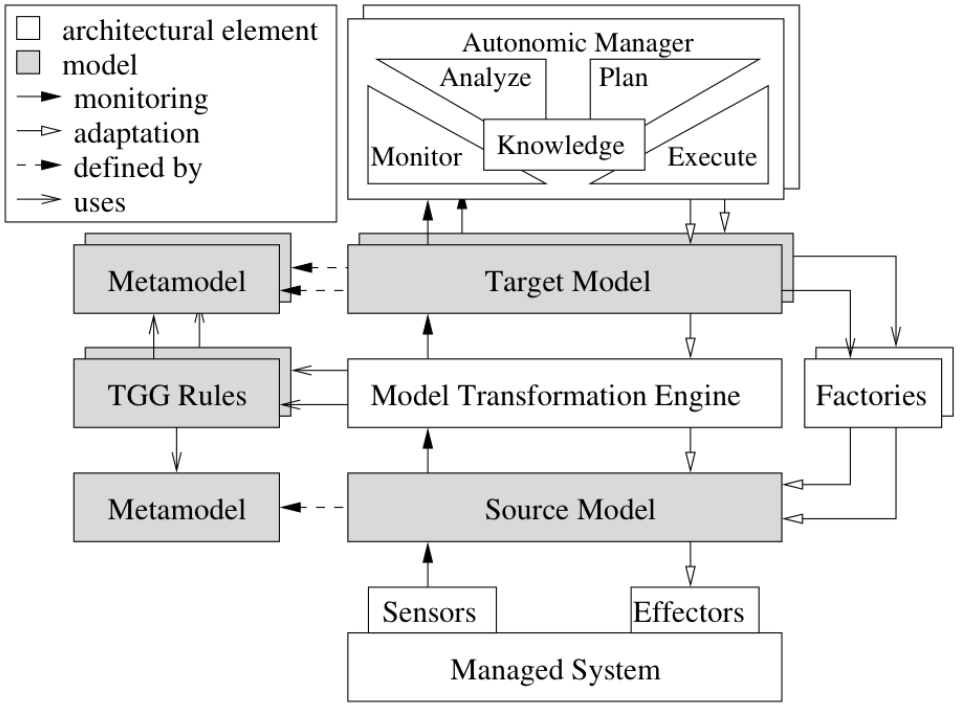
[Vogel+2009]
[Vogel+2010a]

Rückwärtssynchronization entgegen der Abstraktion

- **ABER:** Anpassung der Managed Subsysteme über die Laufzeitmodelle kann wegen Abstraktion unmöglich sein
- **IDEE:** Factories legen diese Anteile fest und werden beim Erzeugen neuer Anteile automatisch genutzt

Vorteile des Ansatzes:

- Höherwertige Sensoren und Effektoren
- Wiederverwendung der Autonomic Manager
- Management auf höherer Abstraktionsebene



[Vogel+2010b]

Zusammenfassung und Ausblick

- GTS ermöglichen effiziente Modelltransformation und –synchronisation und deren formale Verifikation für MDE.
- GTS können zur Modellierung und Analyse hochgradig dynamischer Systeme mit Erzeugung und Löschung von Komponenten sowie veränderlichen Strukturen genutzt werden.
- GTS ermöglichen Self-Managed Systeme bei denen die Manipulation der Software zur Laufzeit über Modelle und Modellsynchronisationstechniken erfolgt.

Ausblick:

- DFG Projekt KorMoran zu korrekten Transformationen (mit Sabine Glesner)
- Evolution der Modellierungssprachen? Story Diagramme können durch sich selbst modifiziert werden (Higher-Order Transformations)

Referenzen

[Becker+2006] BECKER, BASIL, DIRK BEYER, HOLGER GIESE, FLORIAN KLEIN und DANIELA SCHILLING: Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In: Proc. of the 28th International Conference on Software Engineering (ICSE), Shanghai, China. ACM Press, 2006.

[Giese+2006] GIESE, HOLGER, SABINE GLESNER, JOHANNES LEITNER, WILHELM SCHÄFER und ROBERT WAGNER: Towards Verified Model to Code Trans- formations. In: Proceedings of the Workshop MoDeV2a 2006: Perspectives on Integrating MDA and V&V, ACM/IEEE 9th International Conference on Model Driven Engineering Languages and Systems, Genua, Italy, 2006. CEA.

[Giese+2008] GIESE, HOLGER and ROBERT WAGNER: From model transformation to incremental bidirectional model synchronization. Software and Systems Modeling, 3 2008.

[Vogel+2009] Vogel, T., Neumann, S., Hildebrandt, S., Giese, H., Becker, B.: Model-Driven Architectural Monitoring and Adaptation for Autonomic Systems. In: Proc. of the 6th International Conference on Autonomic Computing and Communications (ICAC'09), Barcelona, Spain, ACM (15-19 June 2009) accepted paper.

[Vogel+2010b] Thomas Vogel and Stefan Neumann and Stephan Hildebrandt and Holger Giese and Basil Becker. Incremental Model Synchronization for Efficient Run-Time Monitoring. In Sudipto Ghosh, ed., Models in Software Engineering, Workshops and Symposia at MODELS 2009, Denver, CO, USA, October 4-9, 2009, Reports and Revised Selected Papers, vol. 6002 of Lecture Notes in Computer Science (LNCS), pages 124-139. Springer-Verlag, 2010.

[Vogel+2010] Thomas Vogel and Holger Giese. Adaptation and Abstract Runtime Models. In Proceedings of the 5th Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2010) at the 32nd IEEE/ACM International Conference on Software Engineering (ICSE 2010), Cape Town, South Africa, pages 39-48, 5 2010. ACM.