Story Pattern

(Theorie und Verwendung)

Motivation:

- intuitivere Verständlichkeit durch bekannte UML-Notation
- Ausdrucksmächtigkeit durch OO Graphmodell
- problemnahe Modellierung
- formal definierte Semantik

Def. 4: Objektgraphen (erweiterte gakk-Graphen)

G := (SI, Ext) mit

SI := (NL, EL, A, IsAs, Assocs, Attrs) mit (Schema Info)

NL endliche Menge von Knotenmarkierungen / -typen / -labeln

EL endliche Menge von Kantenmarkierungen / -typen / -labeln

A endliche Menge von Attribut(nam)en

IsAs $\subseteq$ **Rel (desc** $\in$ **NL, anch** $\in$ **NL)** **mit Rel := "Relation über"**

Assocs $\subseteq$ **Rel (el** $\in$ **EL, srcNl** $\in$ **NL, srcCard** $\in$ **MultiInfo := {one, many}**
assocType $\in$ **P (AssocTypes := {qualified, aggregation, ordered})**
tgtNl $\in$ **NL, tgtCard** $\in$ **MultiInfo)** **mit Rel := "Relation über"**

Attrs $\subseteq$ **Func (A**$\rightarrow$ **{Boolean, Integer, String, Float, P(Integer), ...})**

Ext := (N, E, l, av)

N endliche Menge von Knoten(bezeichnern)

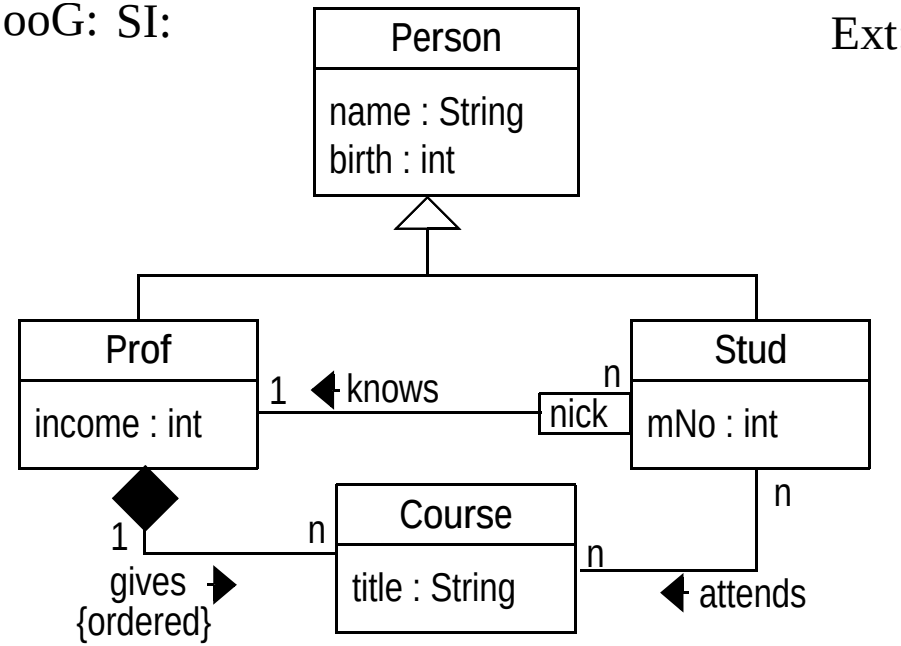
E Rel (src $\in$ N, el $\in$ EL, **i** $\in$ $\varepsilon \cup$ ***R*****, q** $\in$ $\varepsilon \cup$ **AttrValues**, tgt $\in$ N)

l Func ((N) $\rightarrow$ NL)

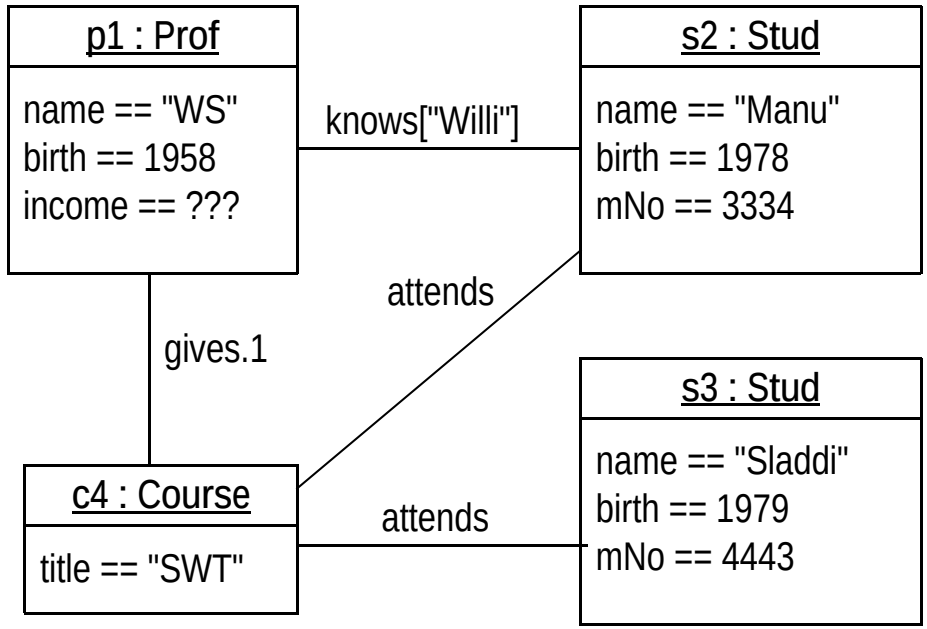
av Func ((N, A) $\rightarrow$ AttrValues := {true, false} $\cup$ *N* $\cup$ *R* $\cup$ *char** $\cup$ *N** $\cup$ ...)

Bsp. 4: Klassendiagramm und Objektgraph

ooG: SI:



Ext:



ooG = (SI, Ext); SI = (NL, EL, A, IsAs, Assocs, Attrs); NL = {Person, Prof, Stud, Course};
EL = {knows, gives, attends}; A = {name, birth, income, title, mNo}
IsAs = {(Prof, Person), (Stud, Person), (Prof, Prof), (Stud, Stud), (Person,Person), (Course,Course)};
Assocs = { (attends, Stud,many, {}, Course, many), (knows, Stud, many, {qualified}, Prof, one),
(gives, Prof, one, {aggregation, ordered}, Course, many) }
Attrs = {(Person.name) → String, (Person.birth) → int, (Prof.income) → int, Stud.mNo → int, Course.title → String}

Ext = (N, E, l, av)
N = {p1, s2, s3, c4};
E = {(s2, attends, ε, ε, c4), (s3, attends, ε, ε, c4), (p1, gives, 1, ε, c4), (s2, knows, ε, "Willi", p1)};
nl = {(p1)→Prof, (s2)→Stud, (s3)→Stud, (c4)→Course}; av={(p1,Prof.name)→"WS", (p1,Prof.birth)→1958, ...}

Def. 5: Abkürzungen

- Mit GraphClass (SI) bezeichnen wir die Menge aller Graphen mit gleichem SchemaInfo SI
- Im folgenden sei SI gegeben und GC := GraphClass(SI)
- Ein Graph $G \in GC$ wird vereinfachend nur mit (N, E, l, av) angegeben
- Sei $G=(N,E,l,av) \in GC$, dann bezeichnet
  - N_G := N die Knotenmenge von G
  - E_G := E die Kantenmenge von G
  - l_G := l die Knotenmarkierungsfunktion von G
  - av_G := die Attributierungsfunktion von G

Def. 5: Basis-Story-Pattern

Ein Story Pattern ist ein Paar von Graphen (LG, RG) mit LG, RG $\in$ GC und LG und RG sind konsistent markiert,
d.h. $l_{LG}|_{N_{LG} \cap N_{RG}} = l_{RG}|_{N_{LG} \cap N_{RG}}$

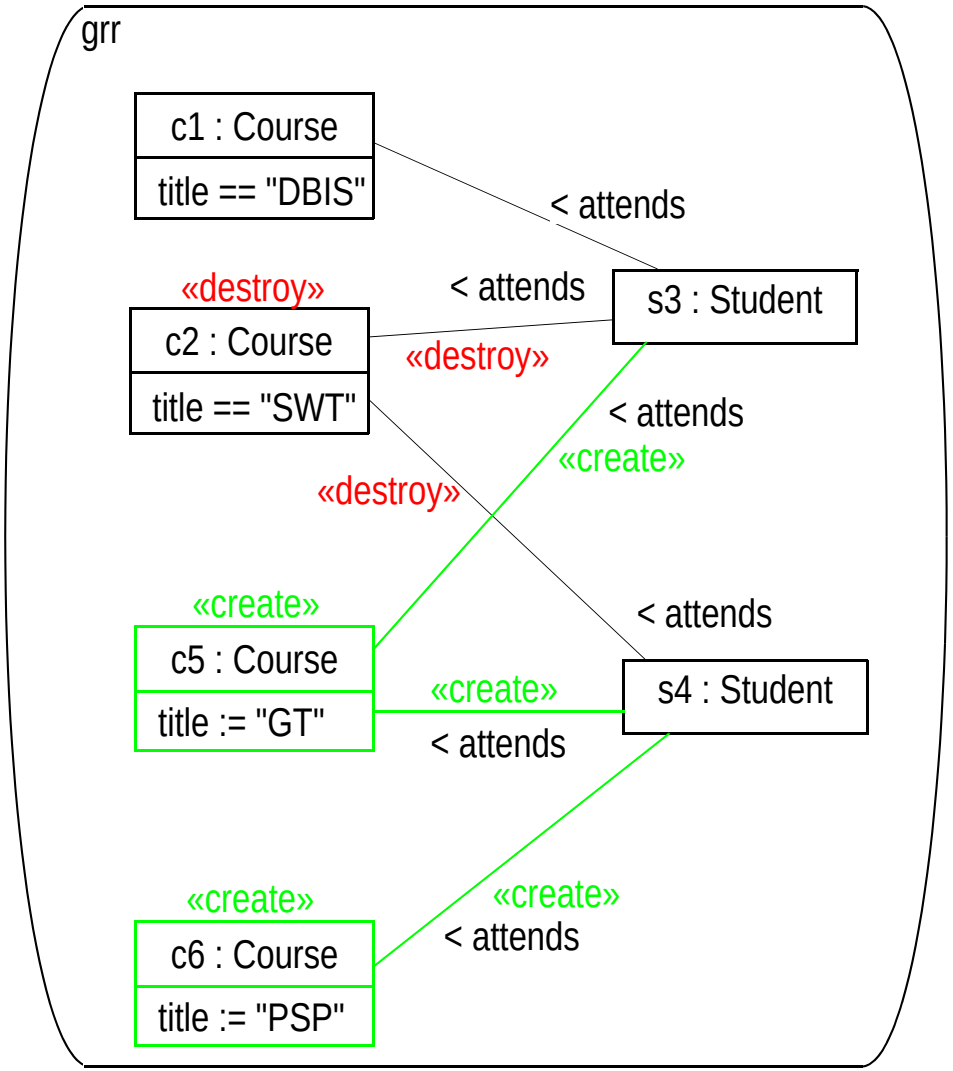
Sei grr := (LG, RG) gegeben, dann bezeichnen wir mit:

- $DelN_{grr} := N_{LG} - N_{RG}$ die Menge der von grr zu löschenden Knoten
- $DelE_{grr} := E_{LG} - E_{RG}$ die Menge der von grr zu löschenden Kanten

- $CoreN_{grr} := N_{LG} \cap N_{RG}$ die Kern- oder Kontextknoten von grr

- $AddN_{grr} := N_{RG} - N_{LG}$ die Menge der von grr neu zu erzeugenden Knoten
- **$AddE_{grr} := E_{RG} - E_{LG}$**

Bsp. 5: Basisgraphersetzungsregel:



N_{LG} = { c1, c2, s3, s4 }

E_{LG} = { (s3, attends, ε, ε, c1), (s3, attends, ε, ε, c2), (s4, attends, ε, ε, c2) }

l_{LG} = { c1 -> Course, c2 -> Course, s3 -> Student, s4 -> Student }

av_{LG} = { (c1, Title) -> "DBIS", (c2, Title) -> "SWT" }

N_{RG} = { c1, s3, s4, c5, c6 }

E_{RG} = { (s3, attends, ε, ε, c1), (s3, attends, ε, ε, c5), (s4, attends, ε, ε, c5), (s4, attends, ε, ε, c6) }

l_{RG} = { c1 -> Course, s3 -> Student, s4 -> Student, c5 -> Course, c6 -> Course }

av_{RG} = { (c1, Title) -> "DBIS", (c5, Title) -> "GT", (c6, Title) -> "PSP" }

$DelN_{grr}$ = { c2 }
$DelE_{grr}$ = { (s3, attends, ε, ε, c2), (s4, attends, ε, ε, c2) }

$CoreN_{grr}$ = { c1, s3, s4 }

$AddN_{grr}$ = { c5, c6 }

$AddE_{grr}$ = { (s3, attends, ε, ε, c5), (s4, attends, ε, ε, c5), (s4, attends, ε, ε, c6) }

Ausführung von Story Pattern

1. Anwendungsstelle suchen
2. Löschungen
3. Neu erzeugen

1. Suchen:

Suche Teilgraph der zu linker Regelseite passt

Def. 6: Teilgraph

Seien TG, G $\in$ GC gegeben.

TG ist Teilgraph von G (in Zeichen TG <= G)

:<=>

$N_{TG} \subseteq N_G$

unQual (E_{TG}) $\subseteq$ (unQual (E_G) $\cap$ ($N_{TG} \times EL \times N_{TG}$))

$\forall$ e := (sn, el, i, q, tn) $\in$ E_{TG} gilt $\exists$ e' := (sn, el, i', q', tn) $\in$ E_G so daß

i != ε => i == i' und
q != ε => q == q'

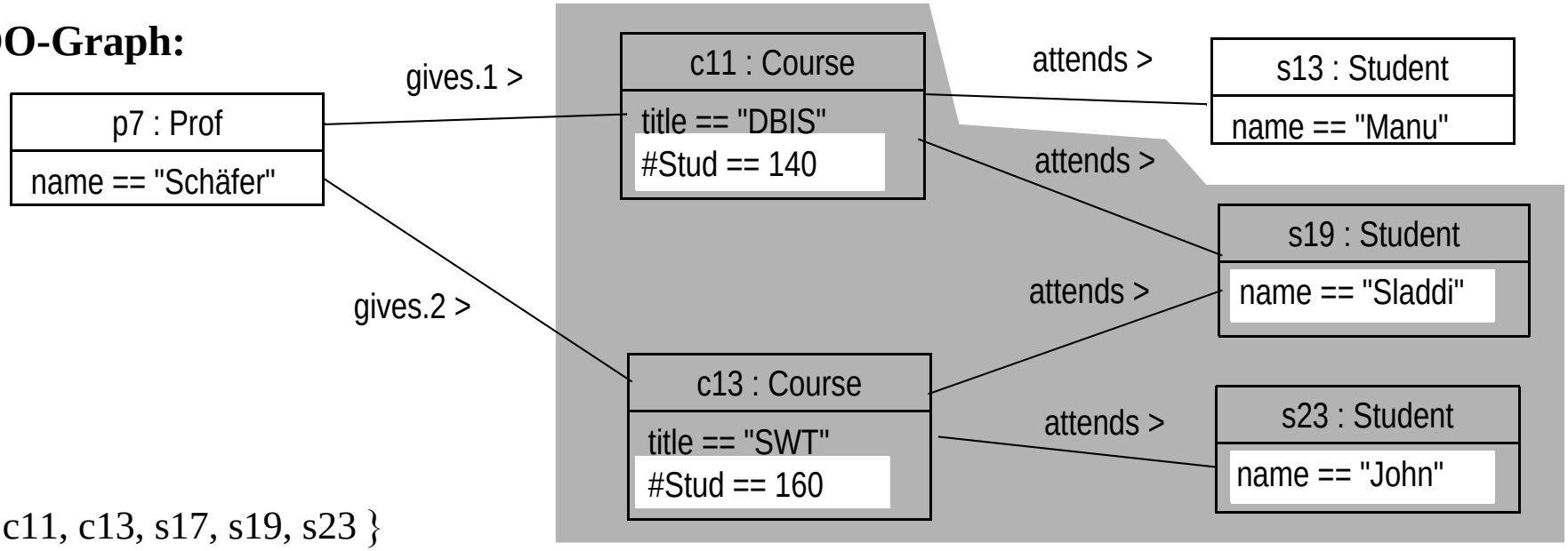
wobei

unQual (E) := { (sn, el, tn) | (sn, el, i, q, tn) $\in$ E}

$l_{TG} = l_G|_{N_{TG}}$ Knotenmarkierungen stimmen überein

$\forall n \in N_{TG}, a \in A$ mit $av_{TG}(n, a) = X$ gilt $av_G(n, a) = X$

Bsp. 1:OO-Graph:



N_G = { p7, c11, c13, s17, s19, s23 }

E_G = { (p7, gives, **1,** ε, c11), (p7, gives, **2,** ε, c13), (s17, attends, ε, ε, c11), (s19, attends, ε, ε, c11), (s19, attends, ε, ε, c13), (s23, attends, ε, ε, c13) }

l_G = { p7 -> Prof, c11 -> Course, c13 -> Course, s17 -> Student, s19 -> Student, s23 -> Student }

av_G = { (p7, name) -> "Schäfer", (c11, title) -> "DBIS", (c11, #Stud) -> 140, (c13, title) -> "SWT", (c13, #Stud) -> 160, (s17, name) -> "Manu", (s19, Name) -> "Sladdi", (s23, Name) -> "John" }

N_{TG} = { c11, c13, s19, s23 } E_{TG} = { (s19, attends, ε, ε, c11), (s19, attends, ε, ε, c13), (s23, attends, ε, ε, c13) }

l_{TG} = { c11 -> Course, c13 -> Course, s19 -> Student, s23 -> Student }

av_{TG} = { (c11, title) -> "DBIS", (c13, title) -> "SWT" }

Def. 7 Graphisomorphismen (Abbildung zwischen Strukturgleichen Graphen):

Seien G1, G2 $\in$ GC gegeben.

Eine **bijektive** Funktion match: N_{G1}-> N_{G2} heißt Graphisomorphismus

:<=>

match (N_{G1}) = N_{G2} wobei match(N) := { match(n) | n $\in$ N }

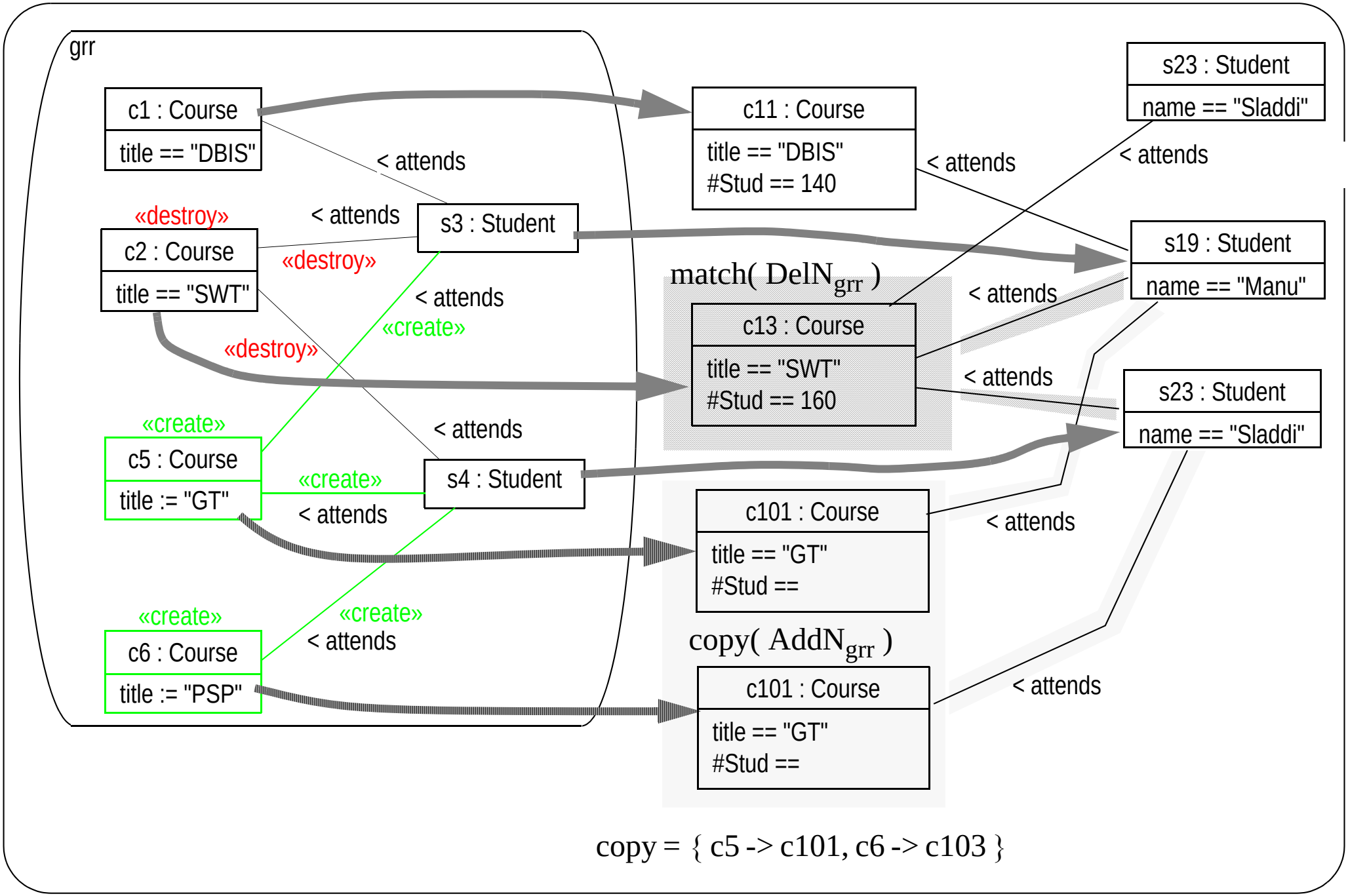
match (E_{G1}) = E_{G2} wobei match(E) := { (match(sn), el, i, q, match(tn)) | (sn, el, i, q, tn) $\in$ E }

$\forall$ n $\in$ N_{G1} gilt l_{G1}(n) $\in$ **IsAs** (l_{G2} (match (n)))die Markierungen sind Zuweisungskompatibel

$\forall$ n $\in$ N_{G1}, a $\in$ A mit av_{G1}(n, a) = X gilt av_{G2}(match (n), a) = X

Mit ISOs(G1, G2) bezeichnen wir die Menge aller Graphisomorphismen von N_{G1} zu N_{G2}

match = { c1 -> c11, c2 -> c13, s3 -> s19, s4 -> s23 }



copy = { c5 -> c101, c6 -> c103 }

Def. 8 Semantik von Story Pattern:

$Sem(grr) \subseteq GC \times GC$ oder auch $Sem[grr] : GC \rightarrow P(GC)$

mit $(G1, G2) \in Sem(grr)$ oder auch $G2 \in Sem[grr] (G1)$

:<=>

1. (Suchen)

   $\exists\, TG \in GC$ mit $TG <= G1$ und $\exists$ match $\in ISOs(TG, LR)$ (TG heißt Anwendungsstelle)

2. (Löschen)

   $\exists\, IG <= G1$ mit

   $N_{IG} = N_{G1}$ - $\mathbf{parts_{G1}^*}$ (match $(DelN_{grr})$)

   where $\mathbf{parts_{G1}}$ (N) := { n' | $\exists$ (sn, el, i, q, n) $\in \mathbf{E_{G1}}$ mit sn $\in$ N

   and aggregation $\in \mathbf{assocType_{G1}(el)}$ }

   $E_{IG} = (E_{G1}$ - match ($DelE_{grr}$)) $\cap$ $(N_{IG} \times EL \times R \times AttrValues \times N_{IG})$

   $l_{IG} = l_{G1}|_{N_{IG}}$

   $av_{IG} = av_{G1}|_{N_{IG} \times A}$

3. (Erzeugen)

   (neue Knoten)

   $\exists$ N_{new} mit $N_{new} \cap N_{G1} = \varnothing$ und $\exists$ bijektive Funktion copy mit copy ($AddN_{grr}$) = N_{new}

   wobei copy durch den match eindeutig festgelegt seien soll, d.h.

   match = match' => copy = copy' und

   $N_{G2} = N_{IG} \cup N_{new}$

   $E_{G2} = (E_{IG}$ - replace (E_{IG}, E_{new})) $\cup$ E_{new}
   where E_{new} := { (copyMatch (sn), el, i, q, copyMatch (tn)) |
   $\exists$ (sn, el, j, q, tn) $\in$ $AddE_{grr}$ and
   i == 1 + (max k in { (sn, el, k, q, tn") $\in$ E_{IG} }) falls j == ε
   i == j sonst } // bis jetzt nur hinten anfügen
   and where replace (E_{IG}, E_{new}) := // zu-eins Assocs
   { (sn, el, i, q, tn) $\in$ E_{IG} | tgtCard (el) == one and (sn, el, i, q, tn') $\in$ E_{new}
   or srcCard (el) == one and (sn', el, i, q, tn) $\in$ E_{new} }

   $l_{G2}(n) = l_{GR}(nr)$ falls n = copy(nr) und = $l_{IG}(n)$ sonst

   $av_{G2}(n, a) = av_{GR}(nr, a)$ falls n = copy(nr) oder n = match(nr) und = $av_{IG}(n, a)$ sonst

Story-Diagramme

(Theorie und Verwendung)

Graphische Kontrollstrukturen für Story Patterns auf Basis von Activity Diagrammen

Definition: Story Specifikation

OOSpec := {story_diags := (name, activities, transitions, start)}

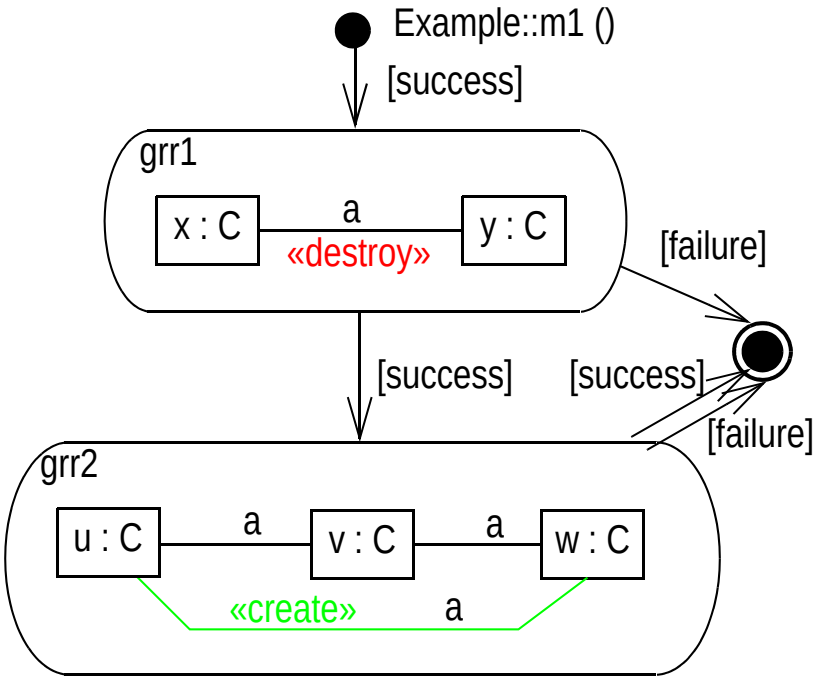
name name des Story-Diagramms

activities $\subseteq$ {Story Patterns} $\cup$ name (story_diags) $\cup$ {"stop"}

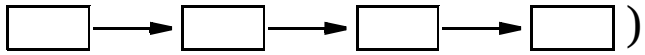
transitions $\subseteq$ activities $\times$ {success, failure} $\times$ activities

start $\in$ activites die Startaktivität

Beispiel:


Example::m1 ()
[success]
grr1
x : C
a
«destroy»
y : C
[failure]
[success]
[success]
[failure]
grr2
u : C
a
v : C
a
w : C
«create»
a

Semantik von Example.m1 ()

G1 := ()

G2 := ()

Ist (G1, G2) in Sem [Example.m1()] ?

Progress hat Backtring-Semantik =>

In Progress: (G1, G2) nicht in Sem [grr1 & grr2]

Backtracking Semantik ist ziemlich teuer

In Story Diagrammen kein Backtracking =>

Regel ausführen, Erfolg merken, niemals rückgängig machen

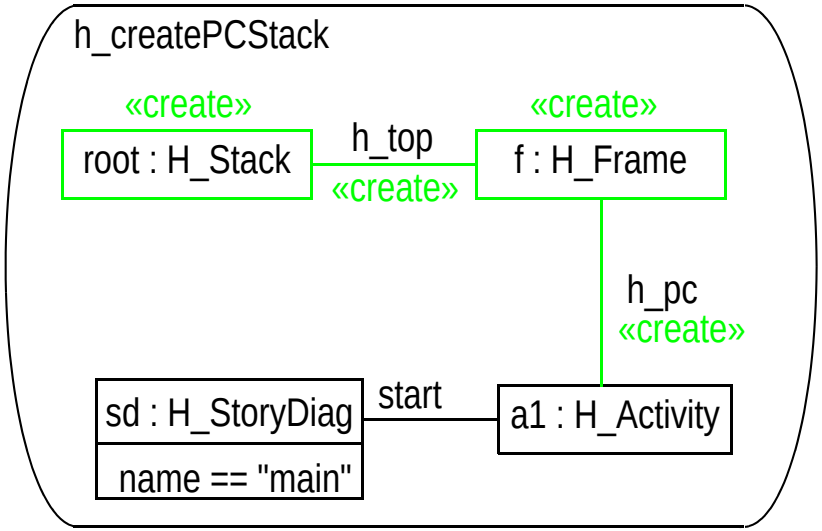
Sem [Example.m1()] := Sem [choose grr1 & sdm := success else sdm := failure end
& choose def (sdm == success) & grr2 & sdm := success else sdm := failure end]

Ausführung von Story-Diagrammen

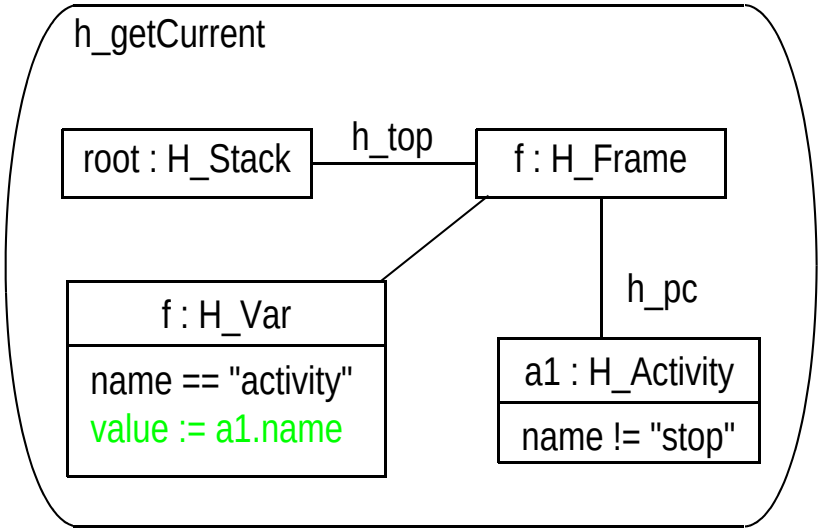
- Alternative 1: Rückführung auf Progress Kontrollstrukturen
  - geht nur für wohlgeformte Kontrollflüsse (später)
  - auf Dauer soll das ohne Progress gehen
  -
- Alternative 2: Kontrollflußinterpreter
  - implizite Einführung eines Prozedurkellers (wie schon bei den Regeln mit Parametern)
  - zusätzliche Program Counter Kante (pc) zeigt auf akutelle Aktivitiät
  - pro Arbeitsschritt:
    - aktuelle Aktivität ausführen
    - pc Kante gemäß Erfolg oder Misserfolg entlang ausgehender Transition weiter setzen

Sem [OOSpec] := Sem [h_createPCStack & loop h_step end]

mit



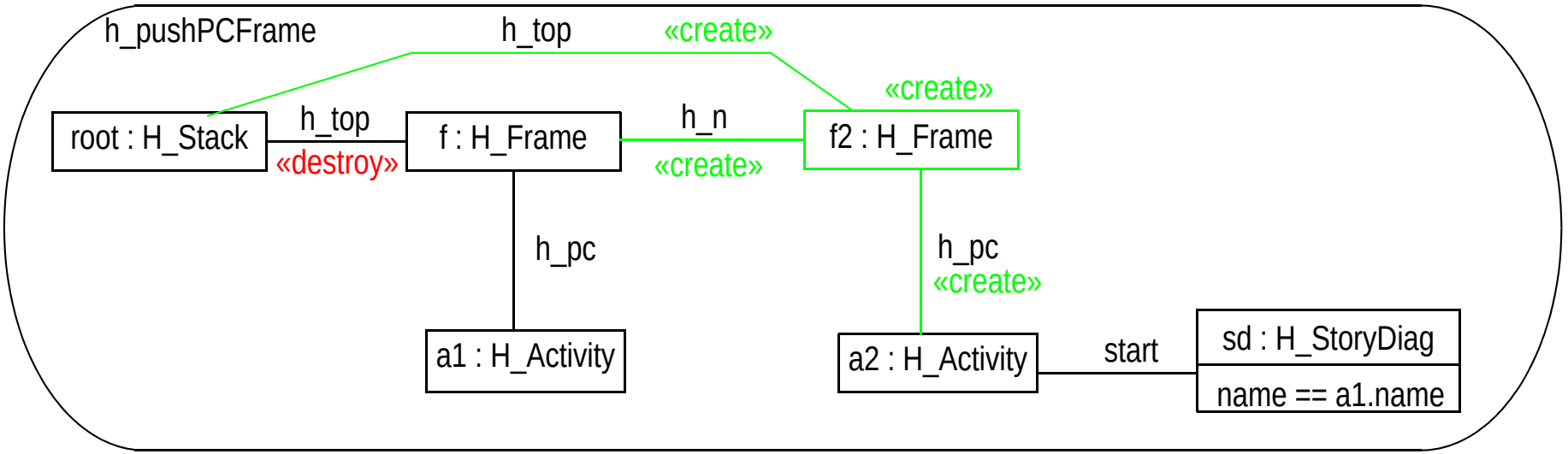

```
transaction h_step =
        ...
    &   h_getCurrent (out activity)
    &   h_exec (activity)
    &   h_advance ()
end
```

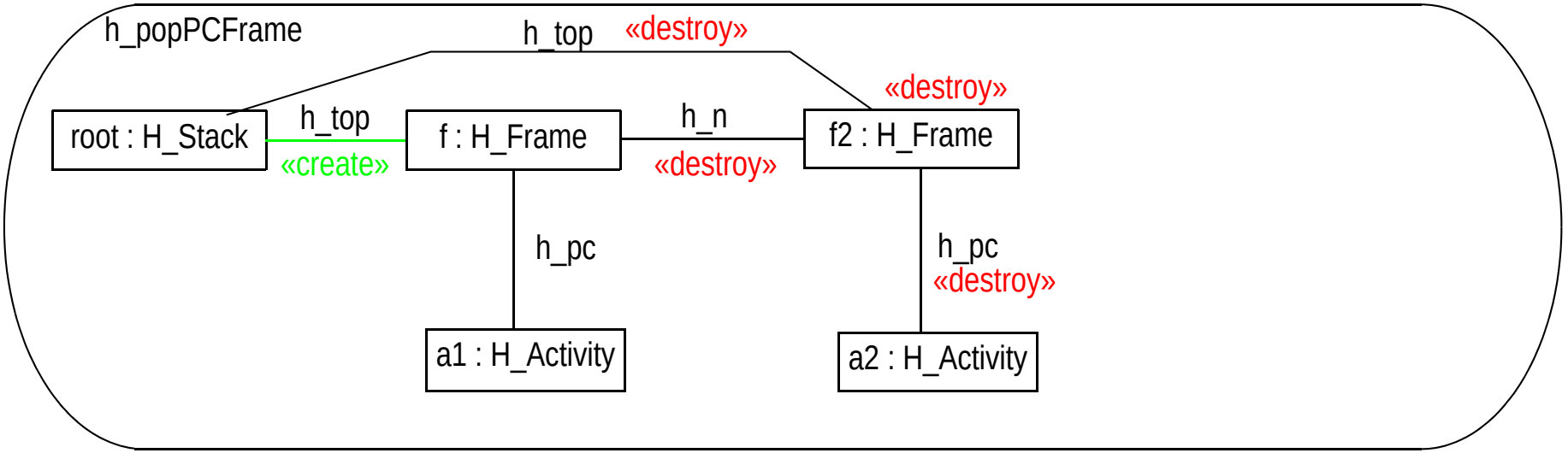


Sem [h_exec (activity)] :=

Sem [choose activity & h_sdm := "success" else h_sdm := "failure" end]
falls activity ein Story Pattern / eine Graphersetzungsregel ist

Sem [h_pushPCFrame & loop h_step end & h_popPCFrame]
sonst





Plus Übergabe von Parametern wie schon vorgeführt :)

transaction h_advance () = h_advanceSuccess or h_advanceFailure end