

1 Struktur für die Definition von Typen

Die Typen seien in einer Bibliothek L in folgender Form zusammengefasst:

Regel	Erläuterung
$L ::= TD^*$	Eine Bibliothek L besteht aus einer Menge von Typdefinitionen.
$TD ::= PD RD$	Eine Typdefinition kann entweder die Definition eines provided Typen (PD) oder eines required Typen (RD) sein.
$PD ::=$ $\text{provided } T \text{ extends } T'$ $\{FD^*MD^*\}$	Die Definition eines provided Typen besteht aus dem Namen des Typen T , dem Namen des Super-Typs T' von T sowie mehreren Feld- und Methodendeklarationen.
$RD ::= \text{required } T \{MD^*\}$	Die Definition eines required Typen besteht aus dem Namen des Typen T sowie mehreren Methodendeklarationen.
$FD ::= f:T$	Eine Felddeklaration besteht aus dem Namen des Feldes f und dem Namen seines Typs T .
$MD ::= m(T):T'$	Eine Methodendeklaration besteht aus dem Namen der Methode m , dem Namen des Parameter-Typs T und dem Namen des Rückgabe-Typs T' .

Tabelle 1: Struktur für die Definition einer Bibliothek von Typen

Weiterhin sei die Relation $<$ auf Typen durch folgende Regeln definiert:

$$\frac{\text{provided } T \text{ extends } T' \in L}{T < T'}$$

$$\frac{\text{provided } T \text{ extends } T'' \in L \wedge T'' < T'}{T < T'}$$

Darüber hinaus seien folgende Funktionen definiert:

$$\begin{aligned} felder(T) &:= \{ f : T' \mid f : T' \text{ ist Felddeklaration von } T \} \\ methoden(T) &:= \{ m(T') : T'' \mid m(T') : T'' \text{ ist Methodendeklaration von } T \} \\ feldTyp(f, T) &:= T' \mid f : T' \text{ ist Felddeklaration von } T \end{aligned}$$

Das Matching eines Typs A zu einem Typ B wird durch die asymmetrische Relation $A \Rightarrow B$ beschrieben. Dabei wird A auch als *Source-Typ* und B als *Target-Typ* bezeichnet.

2 Struktur für die Definition von Proxies

Ein Proxy wird auf der Basis einer Matchingrelation erzeugt. In Abhängigkeit von der zugrundeliegenden Matchingrelation zwischen dem *Source*- und dem *Target-Typen* werden unterschiedliche Arten von Proxies erzeugt:

- *Struktureller Proxy*
- *Simple-Proxy*
- *Sub-Proxy*
- *Container-Proxy*
- *Content-Proxy*

Der Typ des Proxies entspricht immer dem *Source-Typ* der zugrundeliegenden Matchingrelation. Die unterschiedlichen Proxies werden dabei durch die Regeln der Tabellen 2 und 3 beschrieben:

Regel	Erläuterung
$STRUCTPROXY ::= \text{structproxy for } R \{TARGET^*\}$	Ein <i>struktureller Proxy</i> wird für ein <i>required Interface</i> R mit einer Mengen von Targets erzeugt.
$TARGET ::= P \{MDEL^*\}$	Ein Target besteht aus dem Typ P des Targets (ein <i>provided Typ</i>) und einer Mengen von Methodendelegationen.
$MDEL ::= CALLM \rightarrow DELM$	Eine Methodendelegation besteht aus einer aufgerufenen Methode und aus einem Delegationsziel.
$CALLM ::= m(SP) : STPROXY$	Eine aufgerufene Methode besteht aus dem Namen der Methode m , dem Parametertyp SP und einem Single-Target-Proxy zur Konvertierung des Rückgabetyps des Delegationsziels.
$DELM ::= n(STPROXY) : R$	Ein Delegationsziel besteht aus dem Namen der Methode n , dem Rückgabetyptyp TR und einem Single-Target-Proxy zur Konvertierung des Parametertyps der aufgerufenen Methode.
$STPROXY ::= NPX$	Ein Nominal-Proxy ist ein Single-Target-Proxy.

Tabelle 2: Grammatikregeln für die Definition eines Proxies

Regel	Erläuterung
$STPROXY ::=$ $\text{contentproxy for } P$ $\text{with } P' \{CEMDEL^*\}$	Ein <i>Content-Proxy</i> ist ein Single-Target-Proxy, der für ein <i>provided Typ</i> P mit einem <i>provided Typ</i> P' als Target-Typ sowie einer Mengen von Content-Proxy-Methodendelegationen erzeugt wird.
$STPROXY ::=$ $\text{containerproxy for } P$ $\text{with } P' \{f = NPX\}$	Ein <i>Container-Proxy</i> ist ein Single-Target-Proxy, der für ein <i>provided Typ</i> P mit einem <i>provided Typ</i> P' als Target-Typ sowie der Zuweisung eines Nominal-Proxies für den Target-Typ zu einem Feld f erzeugt wird.
$NPX ::=$ $\text{subproxy for } P$ $\text{with } P' \{NOMMDEL^*\}$	Ein Sub-Proxy ist ein Nominal-Proxy, der für ein <i>provided Typ</i> P mit einem <i>provided Typ</i> P' als Target-Typ sowie einer Mengen von Nominal-Proxy-Methodendelegationen erzeugt wird. Dabei gilt $P < P'$.
$NPX ::=$ $\text{simpleproxy for } P$	Ein <i>Simple-Proxy</i> ist ein Nominal-Proxy, der aus einem Typen P , für den der Proxy erzeugt wird, besteht. Der Target-Typ ist in diesem Fall ebenfalls P . Alle Methoden werden in diesem Fall an den Target-Typ delegiert.
$NOMMDEL ::=$ $m(SP) : SR \rightarrow m(TP) : TR$	Eine Nominal-Proxy-Methodendelegation besteht aus zwei Methoden mit demselben Namen m und den jeweiligen Parameter- und Rückgabetypen SP und SR bzw. TP und TR .
$CEMDEL ::= m(SP) : NPX \rightarrow$ $f.m(NPX) : TR$	Eine Content-Proxy-Methodendelegation besteht aus zwei Methoden mit demselben Namen m , wobei die delegierte Methode (rechte Seite) auf einem Feld f des Target-Typs aufgerufen wird. Dabei besteht die aufgerufene Methode aus dem Parametertyp SP und einem Nominal-Proxy für den Rückgabetyp. Ferner besteht die delegierte Methode aus dem jeweiligen Rückgabetyp TR und einem Nominal-Proxy für den Parametertyp.

Tabelle 3: Grammatikregeln für die Definition eines Proxies (Fortsetzung)

3 Beispiel-Bibliothek

```
provided Fire extends Object{}

provided ExtFire extends Fire{}

provided FireState extends Object{
    isActive : boolean
}

provided Medicine extends Object{
    String getDescription()
}

provided Injured extends Object{
    void heal(Medicine med)
}

provided Patient extends Injured{}

provided FireFighter extends Object{
    FireState extinguishFire(Fire fire)
}

provided Doctor extends Object{
    void heal( Patient pat, Medicine med )
}

provided MedCabinet extends Object{
    med : Medicine
}

required MedicalFireFighter {
    void heal( Injured injured, MedCabinet med )
    boolean extinguishFire( ExtFire fire )
}
```

Listing 1: Bibliothek *Example* von Typen

4 Beispiel-Proxy für MedicalFireFighter

```
structproxy for MedicalFireFither{
  FireFighter {
    extinguishFire(ExtFire):
      containerproxy for FireState with boolean {
        isActive = simpleproxy for boolean
      }
      → extinguishFire(simpleproxy for Fire):boolean
  }

  Doctor {
    heal(Injured, MedCabinet):simpleproxy for void
    → heal(subproxy for Patient with Injured{
      heal(Medicine): void
      → heal(Medicine):void

    }, contentproxy for Medicine with MedCabinet{
      getDescription(): simpleproxy for String
      → med.getDescription():String
    }):void
  }
}
```

Listing 2: Proxy für MedicalFireFighter

5 Generierung der Proxies auf Basis von Matchern

Die Matcher beinhalten zum Einen die Definition der jeweiligen Matchingrelation (\Rightarrow) sowie die Regeln zur Erzeugung eines Proxies, die auf jener Matchingrelation basieren. Alle Arten von Proxies, die durch die folgenden Matcher erzeugt werden, können am Beispiel aus Abschnitt 4 nachvollzogen werden.

5.1 Matcher

5.1.1 ExactTypeMatcher

Der *ExactTypeMatcher* stellt ein Matching von einem Typ T zu demselben Typ T her. Die dazugehörige Matchingrelation \Rightarrow_{exact} wird durch folgende Regel beschrieben:

$$\overline{T \Rightarrow_{exact} T}$$

5.1.2 GenTypeMatcher

Der *GenTypeMatcher* stellt ein Matching von einem Typ T zu einem Typ T' mit $T > T'$ her. Die dazugehörige Matchingrelation \Rightarrow_{gen} wird durch folgende Regel beschrieben:

$$\frac{T > T'}{T \Rightarrow_{gen} T'}$$

5.1.3 SpecTypeMatcher

Der *SpecTypeMatcher* stellt im Verhältnis zum *GenTypeMatcher* das Matching in der entgegengesetzten Richtung dar. Die dazugehörige Matchingrelation \Rightarrow_{spec} wird durch folgende Regel beschrieben:

$$\frac{T < T'}{T \Rightarrow_{spec} T'}$$

Die oben genannten Matchingrelationen werden für die Definition weiterer Matcher zusammengefasst, wodurch sich die Matchingrelation $\Rightarrow_{internCont}$ ergibt:

$$\frac{T \Rightarrow_{exact} T' \vee T \Rightarrow_{gen} T' \vee T \Rightarrow_{spec} T'}{T \Rightarrow_{internCont} T'}$$

5.1.4 ContentTypeMatcher

Der *ContentTypeMatcher* matcht einen Typ T auf einen Typ T' , wobei T' ein Feld enthält, auf dessen Typ T'' der Typ T über die Matchingrelation $\Rightarrow_{internCont}$ gematcht werden kann. So kann bspw. der Typ `FireState` aus Listing 3 auf den Typ `boolean` gematcht werden.

Die dazugehörige Matchingrelation $\Rightarrow_{content}$ wird durch folgende Regel beschrieben:

$$\frac{\exists f : T'' \in felder(T').T \Rightarrow_{internCont} T''}{T \Rightarrow_{content} T'}$$

5.1.5 ContainerTypeMatcher

Der *ContainerTypeMatcher* stellt im Verhältnis zum *ContentTypeMatcher* das Matching in der entgegengesetzten Richtung dar. So kann bspw. auch der Typ `boolean` auf den Typ `FireState` aus Listing 3 gematcht werden.

Die dazugehörige Matchingrelation $\Rightarrow_{container}$ wird durch folgende Regel beschrieben:

$$\frac{\exists f : T'' \in felder(T).T'' \Rightarrow_{internCont} T'}{T \Rightarrow_{container} T'}$$

Zur Definition des letzten Matchers werden die Matchingrelationen der oben genannten Matcher noch einmal zusammengefasst. Dabei entsteht die Matchingrelation $\Rightarrow_{internStruct}$, welche durch folgende Regel beschrieben wird:

$$\frac{T \Rightarrow_{exact} T' \vee T \Rightarrow_{spec} T' \vee T \Rightarrow_{gen} T' \vee T \Rightarrow_{container} T' \vee T \Rightarrow_{content} T'}{T \Rightarrow_{internStruct} T'}$$

5.1.6 StructuralTypeMatcher

Der *StructuralTypeMatcher* matcht einen *required Typ* R auf einen *provided Typ* P auf der Basis struktureller Eigenschaften der Methoden, die in den Typen deklariert sind.

Somit soll bspw. der Typ `MedicalFireFighter` auf den Typ `FireFighter` (siehe Listing 3) gematcht werden. Ein weiteres Beispiel bezogen auf die Typen aus Listing 3 ist das Matching des Typs `MedicalFireFighter` auf den

Typ `Doctor` angebracht werden.

Damit ein Typ R auf einen Typ P über den *StrukturalTypeMatcher* gematcht werden kann, muss mindestens eine Methode aus R zu einer Methode aus P gematcht werden. Die Menge der gematchten Methoden aus R in P wird wie folgt beschrieben:

$$structM(R, P) := \left\{ m(T) : T' \in methoden(R) \left| \begin{array}{l} \exists n(S) : S' \in methoden(P). \\ S \Rightarrow_{internStruct} T \wedge \\ T' \Rightarrow_{internStruct} S' \end{array} \right. \right\}$$

Da die Notation es nicht hergibt, ist zusätzlich zu erwähnen, dass, sofern in m und n mehrere Parameter verwendet werden, deren Reihenfolge irrelevant ist.

Die Matchingrelation für die *StrukturalTypeMatcher* wird durch folgende Regel beschrieben:

$$\frac{structM(R, P) \neq \emptyset}{R \Rightarrow_{struct} P}$$

5.2 Generierung der Proxies

Zum besseren Verständnis werden die Regeln zur Generierung der Proxies auf das Beispiel aus Abschnitt 4 angewendet. Da die Proxies verschachtelt sind, erfolgt die beispielhafte Generierung in den folgenden Abschnitten schrittweise parallel zur Beschreibung der Generatoren.

5.2.1 Struktureller Proxy

Der *strukturelle Proxy* bildet immer den Einstiegspunkt für die Generierung eines Proxies für einen *required Typ* über das Nichtterminal *STRUCTPROXY* aus der Grammatik für die Definition von Proxies (Abschnitt 2). Als Basis des Generators für einen *strukturellen Proxy* fungiert der *StrukturalTypeMatcher*. Daher wird durch diesen Generator ein *struktureller Proxy* für einen *required Typ* R generiert.

Für die Generierung eines solchen Proxies vom Typ R muss sichergestellt werden, dass alle in R enthaltenen Methoden durch ein oder mehrere *provided Typen* innerhalb der gesamten Bibliothek L gematcht werden. Die folgende Funktion *cover* beschreibt daher eine Menge von Mengen von *provided Typen*, die für die Erzeugung eines *strukturellen Proxies* für R verwendet

werden können.

$$cover(R, L) := \left\{ \{P_1, \dots, P_n\} \mid \begin{array}{l} P_1 \in L \wedge \dots \wedge P_n \in L \wedge \\ methoden(R) = structM(R, P_1) \cup \\ \dots \cup structM(R, P_n) \wedge \\ structM(R, P_1) \neq \emptyset \wedge \\ \dots \wedge structM(R, P_n) \neq \emptyset \end{array} \right\}$$

Ausgehend von einer Bibliothek L kann ein *struktureller Proxy* zum *require Typ* R aus einer Menge von *provided Typen* P mit $P \in cover(R, L)$ generiert werden. Dazu werden die Grammatikregeln aus Abschnitt 2, die für den *strukturellen Proxy* relevant sind, um Nebenbedingungen erweitert, die Tabelle 4 zu entnehmen sind.

Regel	Nebenbedingungen
$STRUCTPROXY ::=$ $structproxy \text{ for } R$ $\{TARGET_1 \dots$ $TARGET_n\}$	$typ(STRUCTPROXY) = R$ $methoden(STRUCTPROXY) =$ $cmethoden(TARGET_1) \cup \dots \cup cmethoden(TARGET_n)$ $methoden(R) = methoden(STRUCTPROXY)$
$TARGET ::=$ $P \{MDEL_1 \dots$ $MDEL_n\}$	$typ(TARGET) = P$ $cmethoden(TARGET) =$ $cmethode(MDEL_1) \cup \dots \cup cmethode(MDEL_n)$ $dmethoden(TARGET) =$ $dmethode(MDEL_1) \cup \dots \cup dmethode(MDEL_n)$ $dmethoden(TARGET) \subseteq methoden(P)$
$MDEL ::=$ $CALLM \rightarrow DELM$	$cmethode(MDEL) = methode(CALLM)$ $dmethode(MDEL) = methode(DELM)$ $paramTargetTyp(DELM) = paramTyp(CALLM)$ $returnTargetTyp(CALLM) = returnTyp(DELM)$
$CALLM ::=$ $m(SP) : STPROXY$	$SR = typ(STPROXY)$ $methode(CALLM) = m(SP) : SR$ $paramTyp(CALLM) = SP$ $targetTyp(STPROXY) = returnTargetTyp(CALLM)$
$DELM ::=$ $n(STPROXY) : R$	$DP = typ(STPROXY)$ $methode(DELM) = n(DP) : R$ $returnTyp(DELM) = R$ $targetTyp(STPROXY) = paramTargetTyp(DELM)$

Tabelle 4: Grammatikregeln und Nebenbedingungen für die Definition eines strukturellen Proxies

Beispiel: Bezogen auf das Beispiel aus Abschnitt 4 soll ein *struktureller Proxy* für den *required Typ* `MedicalFireFighter` aus den *provided Typen* `FireFighter` und `Doctor` generiert werden. Dazu gelte folgende Bedingung:

$$\{\text{FireFighter}, \text{Doctor}\} \in \text{cover}(\text{MedicalFireFighter}, \text{Example})$$

Die Generierung der *strukturellen Proxies* erfolgt dann in folgenden Schritten, beginnend mit dem Nichtterminal *STRUCTPROXY*. Bei jedem Schritt werden ein oder mehrere Nichtterminal ersetzt und die Attribute aus den Nebenbedingungen soweit wie möglich zugewiesen.

$$\text{STRUCTPROXY} \mid \text{typ}(\text{STRUCTPROXY}) = \text{MedicalFireFighter}$$

$$\begin{array}{l} \text{structproxy for MedicalFireFighter} \\ \{ \text{TARGET}_1 \\ \text{TARGET}_2 \} \end{array} \mid \begin{array}{l} \text{typ}(\text{TARGET}_1) = \text{FireFighter} \\ \text{typ}(\text{TARGET}_2) = \text{Doctor} \end{array}$$

$$\begin{array}{l} \text{structproxy for MedicalFireFighter} \\ \{ \text{FireFighter}\{ \\ \text{MDEL}_1\} \\ \text{Doctor}\{ \\ \text{MDEL}_2\} \} \end{array}$$

$$\begin{array}{l} \text{structproxy for MedicalFireFighter} \\ \{ \text{FireFighter}\{ \\ \text{CALLM}_1 \rightarrow \text{DELM}_1\} \\ \text{Doctor}\{ \\ \text{CALLM}_2 \rightarrow \text{DELM}_2\} \} \end{array}$$

5.2.2 Container-Proxy

5.2.3 Content-Proxy

5.2.4 Sub-Proxy

5.2.5 Simple-Proxy

Ein *Simple-Proxy* kann sowohl auf der Basis des *ExactTypeMatchers* als auch auf Basis des *GenTypMatchers* generiert werden. Die für den *Simple-Proxy* relevanten Grammatikregeln aus Abschnitt 2 werden für die Generierung noch um Nebenbedingungen erweitert. Die vollständige Liste der Regeln und Nebenbedingungen für den *Simple-Proxy* ist Tabelle 5 zu entnehmen.

Regel	Nebenbedingungen
$STPROXY ::= NPX$	$typ(STPROXY) = typ(NPX)$ $targetTyp(STPROXY) = targetTyp(NPX)$
$NPX ::=$ simpleproxy for P	$targetTyp(NPX) \Rightarrow_{exact} P$ $typ(NPX) = P$ $methoden(NPX) = methoden(P)$
$NPX ::=$ simpleproxy for P	$targetTyp(NPX) \Rightarrow_{gen} P$ $typ(NPX) = P$ $methoden(NPX) = methoden(P)$

Tabelle 5: Regeln und Nebenbedingungen für Simple-Proxies

Als Beispiel wird von dem Matching ausgegangen, welches auch bei der Beschreibung des *ExactTypeMatchers* (Abschnitt 5.1.1) angeführt wurde:

$$\mathbf{Fire} \Rightarrow_{exact} \mathbf{Fire}$$

Der *Simple-Proxy* basiert auf zwei Matchern, von denen Die Basis für den *Simple-Proxy* sind sowohl der •

6 alter kram

6.1 StructuralTypeMatcher

Ein struktureller Proxy für ein *required Interface* R aus einer Menge von *provided Typen* P wird durch folgende Regeln und Nebenbedingungen beschrieben:

Regel	Nebenbedingungen
$STRUCTPROXY ::=$ $\text{structproxy for } R$ $\{TARGET_1 \dots$ $TARGET_n\}$	$typ(STRUCTPROXY) = R$ $methoden(STRUCTPROXY) =$ $cmethoden(TARGET_1) \cup \dots \cup cmethoden(TARGET_n)$ $methoden(R) = methoden(STRUCTPROXY)$
$TARGET ::=$ $P \{MDEL_1 \dots$ $MDEL_n\}$	$typ(TARGET) = P$ $cmethoden(TARGET) =$ $cmethode(MDEL_1) \cup \dots \cup cmethode(MDEL_n)$ $dmethoden(TARGET) =$ $dmethode(MDEL_1) \cup \dots \cup dmethode(MDEL_n)$ $dmethoden(TARGET) \subseteq methoden(P)$
$MDEL ::=$ $CALLM \rightarrow DELM$	$cmethode(MDEL) = methode(CALLM)$ $dmethode(MDEL) = methode(DELM)$ $paramTargetTyp(DELM) = paramTyp(CALLM)$ $returnTargetTyp(CALLM) = returnTyp(DELM)$
$CALLM ::=$ $m(SP) : STPROXY$	$SR = typ(STPROXY)$ $methode(CALLM) = m(SP) : SR$ $paramTyp(CALLM) = SP$ $targetTyp(STPROXY) = returnTargetTyp(CALLM)$
$DELM ::=$ $n(STPROXY) : R$	$DP = typ(STPROXY)$ $methode(DELM) = n(DP) : R$ $returnTyp(DELM) = R$ $targetTyp(STPROXY) = paramTargetTyp(DELM)$

Tabelle 6: Grammatik für die Definition eines Proxies

Regeln für das Nonterminal *STPROXY* unterliegen Nebenbedingungen, die teilweise erst unter Zuhilfenahme der folgenden Matcher erfüllt werden können.

6.2 SpecTypeMatcher

Ein Proxy für einen Typ T , der mit einem Target-Typ T' mit $T \Rightarrow_{spec} T'$ erzeugt werden soll, ist ein *Sub-Proxy* und wird durch die folgenden Regeln und Nebenbedingungen beschrieben:

Regel	Nebenbedingungen
$NPX ::=$ $\text{subproxy for } P$ $\text{with } P' \{NOMMDEL_1$ $\dots NOMMDEL_n\}$	$\text{targetTyp}(NPX) = P'$ $\text{typ}(NPX) = P$ $P \Rightarrow_{\text{spec}} P'$ $\text{methoden}(NPX) = \text{cmethode}(NOMMDEL_1) \cup$ $\dots \cup \text{cmethode}(NOMMDEL_n)$ $\text{methoden}(NPX) \subseteq \text{methoden}(P)$ $\text{methoden}(P') \supseteq \text{dmethode}(NOMMDEL_1) \cup$ $\dots \cup \text{dmethode}(NOMMDEL_n)$
$NOMMDEL ::=$ $m(SP) : SR \rightarrow$ $m(TP) : TR$	$SP \geq TP$ $SR \leq TR$ $\text{cmethode}(MOMMDEL) = m(SP) : SR$ $\text{dmethode}(MOMMDEL) = m(TP) : TR$

Tabelle 7: Regeln und Nebenbedingungen für Sub-Proxies

6.3 ContentTypeMatcher

Die Matchingrelation für diesen Matcher wird durch folgende Regel beschrieben:

$$\frac{\exists f : T'' \in \text{felder}(T').T \Rightarrow_{\text{internCont}} T''}{T \Rightarrow_{\text{content}} T'}$$

Für die Relation $\Rightarrow_{\text{internCont}}$ gilt dabei:

$$\frac{T \Rightarrow_{\text{exact}} T' \vee T \Rightarrow_{\text{gen}} T' \vee T \Rightarrow_{\text{spec}} T'}{T \Rightarrow_{\text{internCont}} T'}$$

Ein Proxy für einen Typ P , der mit einem Target-Typ P' mit $P \Rightarrow_{\text{content}} P'$ erzeugt werden soll, ist ein *Content-Proxy* und wird durch die folgenden Regeln und Nebenbedingungen beschrieben:

Regel	Nebenbedingungen
$STPROXY ::=$ $\text{contentproxy for } P$ $\text{with } P' \{CEMDEL_1$ $\dots CEMDEL_n\}$	$typ(STPROXY) = P$ $targetTyp(STPROXY) = P'$ $P \Rightarrow_{content} P'$ $methoden(STPROXY) = cmethode(CEMDEL_1) \cup$ $\dots \cup cmethode(CEMDEL_n)$ $methoden(STPROXY) \subseteq methoden(P)$ $containerType(CEMDEL_1) = P'$ $\dots containerType(CEMDEL_n) = P'$
$CEMDEL ::=$ $CECALLM \rightarrow$ $f.CEDEL M$	$f : FT \in felder(containerType(CEMDEL))$ $methode(CEDEL M) \in methoden(FT)$ $paramTargetTyp(CEDEL M) = paramTyp(CECALLM)$ $returnTargetTyp(CECALLM) = returnTyp(CEDEL M)$
$CECALLM ::=$ $m(SP) : NPX$	$paramTyp(CECALLM) = SP$ $SR = typ(NPX)$ $targetTyp(NPX) = returnTargetTyp(CECALLM)$ $methode(CECALLM) = m(SP) : SR$
$CEDEL M ::=$ $m(NPX) : TR$	$returnTyp(CEDEL M) = TR$ $TP = typ(NPX)$ $targetTyp(NPX) = paramTargetTyp(CEDEL M)$ $methode(CEDEL M) = m(TP) : TR$

Tabelle 8: Regeln und Nebenbedingungen für Contentproxies

6.4 ContainerTypeMatcher

Die Matchingrelation für diesen Matcher wird durch folgende Regel beschrieben:

$$\frac{\exists f : T'' \in felder(T). T'' \Rightarrow_{internCont} T'}{T \Rightarrow_{container} T'}$$

Ein Proxy für einen Typ P , der mit einem Target-Typ P' mit $P \Rightarrow_{container} P'$ erzeugt werden soll, ist ein *Container-Proxy* und wird durch die folgenden Regeln und Nebenbedingungen beschrieben:

Regel	Nebenbedingungen
$STPROXY ::=$ $\text{containerproxy for } P$ $\text{with } P' \{f = NPX\}$	$\text{targetTyp}(STPROXY) = P'$ $\text{typ}(STPROXY) = P$ $P \Rightarrow_{\text{container}} P'$ $f : FT \in \text{felder}(P)$ $\text{targetTyp}(NPX) = P'$ $\text{typ}(NPX) = FT$

Tabelle 9: Regeln und Nebenbedingungen für Container-Proxies

7 Erweiterung um einen DVMatcher

Die o.g. Struktur für die Definition von Typen wird die Definition von *provided Typen* erweitert.

Regel	Erläuterung
$PD ::=$ $\text{provided } T \text{ extends } T'$ $\{FD^*MD^*FCD?\}$	Die Definition eines provided Typen besteht aus dem Namen des Typen T , dem Namen des Super-Typs T' von T sowie mehreren Feld- und Methodendeklarationen und einer optionalen Definition eines factory Typen.
$FCD ::= \text{factory } T \{$ $FD^*MD^*\}$	Die Definition eines factory Typen besteht aus dem Namen des Typen T sowie mehreren Feld- und Methodendeklarationen.

Tabelle 10: Erweiterte Struktur für die Definition einer Bibliothek von Typen

Darüber hinaus wird folgende Funktion definiert:

$$\text{fabriken}(T) := \{ F \mid F \text{ ist ein factory Typ, der in } T \text{ definiert wurde} \}$$

Weiterhin muss die Struktur für die Definition von Proxies um eine weitere Definition für einen *Single-Target-Proxy* erweitert werden.

Regel	Erläuterung
$STPROXY ::=$ $\text{dvproxy for } P$ $\text{with } F \text{ on } m(P') : P$	Ein <i>DV-Proxy</i> ist ein Single-Target-Proxy, der für ein <i>provided Typ</i> P erzeugt wird. Die Methodenaufrufe auf diesem Proxy werden an das Objekt delegiert, welches über die Methode m des Factory-Typen F aus dem Target-Typen P' erzeugt wird.

Tabelle 11: Erweiterung der Struktur für die Definition eines Proxies

Die Matchingrelation \Rightarrow_{dv} wird über folgende Regel beschrieben:

$$\frac{\exists F \in \text{fabriken}(T). \exists m(T') : T}{T \Rightarrow_{dv} T'}$$

Darüber hinaus müssen einige der oben beschriebenen Regeln angepasst werden, damit der *ContainerTypeMatcher*, der *ContentTypeMatcher* und der *StructuralTypeMatcher* den *DVMatcher* verwenden:

$$\frac{T \Rightarrow_{exact} T' \vee T \Rightarrow_{gen} T' \vee T \Rightarrow_{spec} T' \vee T \Rightarrow_{dv} T'}{T \Rightarrow_{internCont} T'}$$

$$\frac{T \Rightarrow_{internCont} T' \vee T \Rightarrow_{content} T' \vee T \Rightarrow_{container} T'}{T \Rightarrow_{internStruct} T'}$$

Ein Proxy für einen Typ P , der mit einem *Target-Typ* P' mit $P \Rightarrow_{dv} P'$ erzeugt werden soll, ist ein DV-Proxy und wird durch die folgenden Regeln und Nebenbedingungen beschrieben:

Regel	Nebenbedingungen
$STPROXY ::=$ $\text{dvproxy for } P$ $\text{with } F \text{ on } m(P') : P$	$\text{targetTyp}(STPROXY) = P'$ $\text{typ}(STPROXY) = P$ $P \Rightarrow_{dv} P'$ $F \in \text{fabriken}(P)$

Tabelle 12: Regeln und Nebenbedingungen für DV-Proxies