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.
$TD ::= PD \vert RD$	Eine Typdefinition kann entweder die Definition eines provided Typen (PD) oder eines required Typen (RD) sein.
PD ::= provided T 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 ::= 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 folgenden Regel definiert:

$T < T' :=$ provided T extends $T' \in L \vee$ (provided T extends $T'' \in L \wedge T'' < T'$)

Darüber hinaus seien folgende Funktionen definiert:

$$felder(T) := \left\{ \; f : T' \;\middle|\; f : T' \text{ ist Felddeklaration von } T \; \right\}$$

$$methoden(T) := \left\{ \; m(T') : T'' \;\middle|\; m(T'):T'' \text{ ist Methodendeklaration von } T \; \right\}\}$$

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 folgende Struktur beschrieben :

Regel	**Erläuterung**
$STRUCTPROXY ::=$ `structproxy for` R $\{TARGET*\}$	Ein *struktureller Proxy* wird für ein *required Interface* R mit einer Mengen von Targets erzeugt.
$TARGET ::=$ $P\ \{MDEL*\}$	Ein Target besteht aus dem Typ P des Targets (ein *provided Typ*) 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 demdem Namen der Methode n, dem Rückgabetyp 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: Struktur für die Definition eines Proxies

Regel	Erläuterung
$STPROXY$ ::= `contentproxy for` P `with` P' $\{CEMDEL*\}$	Ein *Content-Proxy* ist ein Single-Target-Proxy, der für ein *provided Typ* P mit einem *provided Typ* P' als Target-Typ sowie einer Mengen von Content-Proxy-Methodendelegationen erzeugt wird.
$STPROXY$ ::= `containerproxy for` P `with` P' $\{f = NPX\}$	Ein *Container-Proxy* ist ein Single-Target-Proxy, der für ein *provided Typ* P mit einem *provided Typ* P' als Target-Typ sowie der Zuweisung eines Nominal-Proxies für den Target-Typ zu einem Feld f erzeugt wird.
NPX ::= `subproxy for` P `with` P' $\{NOMMDEL*\}$	Ein Sub-Proxy ist ein Nominal-Proxy, derfür ein *provided Typ* P mit einem *provided Typ* P' als Target-Typ sowie einer Mengen von Nominal-Proxy-Methodendelegationen erzeugt wird. Dabei gilt $P < P'$.
NPX ::= `simpleproxy for` P	Ein *Simple-Proxy* 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: Struktur für die Definition eines Proxies (Fortsetzung)

3 Beispiel-Bibliothek

```
provided Fire extends Object{}

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( Fire fire )
}
```

Listing 1: Bibliothek von Typen

4 Beispiel-Proxy für MedicalFireFighter

```
structproxy for MedicalFireFither{
        FireFighter {
         extinguishFire(Fire):
                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 Matcher

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

5.1 StructuralTypeMatcher

Das strukturelle Matching zwischen einem *required Interface* R und einem *provided Typ* P ist gegeben, sofern eine Methode aus R zu einer Methode aus P gematcht werden kann. Die Menge der aus R in P gematchten Methoden wird wie folgt beschrieben:

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

Da die Notation es nicht hergibt, ist zusätzlich zu erwähnen, dass die Reihenfolge der Parameter in m und n irrelevant ist.

Die Relation $\Rightarrow_{egsc}$ wird durch die übrigen Matcher in folgender Form beschrieben:

$$\frac{\begin{array}{r} A \Rightarrow_{exact} B \vee A \Rightarrow_{spec} B \vee A \Rightarrow_{gen} B \\ \vee A \Rightarrow_{container} B \vee A \Rightarrow_{content} B \end{array}}{A \Rightarrow_{internStruct} B}$$

Das strukturelle Matching von R und P wird dann durch folgende Regel beschrieben.

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

Für die Verwendung von R muss jedoch sichergestellt werden, dass alle darin enthaltenen Methoden durch ein oder mehrere *required Typen* innerhalb der gesamten Bibliothek L gematcht werden. Folgende Funktion beschreibt daher eine Menge von Mengen von*provided Typen*, die für die Erzeugung eines

strukturellen Proxies für R verwendet werden können.

$$cover(R, L) := \left\{ \{P_1, ..., P_n\} \;\middle|\; \begin{array}{l} P_1 \in L \wedge ... \wedge P_n \in L \wedge \\ methoden(R) = structM(R, P_1) \cup \\ \ldots \cup structM(R, P_n) \wedge \\ structM(R, P_1) \neq \emptyset \wedge \\ \ldots \wedge structM(R, P_n) \neq \emptyset \end{array} \right\}$$

Für R kann die Exploration abgebrochen werden, wenn $cover(R, L) = \emptyset$ gilt.

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

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

Tabelle 4: Grammatik für die Definition eines Proxies

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

5.2 ExactTypeMatcher

Die Matchingrelation für diesen Matcher wird durch folgende Regel beschrieben:

$$\frac{}{T \Rightarrow_{exact} T}$$

Ein Proxy für einen Typ T, der mit demselben Typ als Target-Typ erzeugt werden soll, ist ein *Simple-Proxy*. Die Regeln für den *Simple-Proxy*, sind im folgenden Abschnitt zum *GenTypeMatcher* beschrieben.

5.3 GenTypeMatcher

Die Matchingrelation für diesen Matcher wird durch folgende Regel beschrieben:

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

Ein Proxy für einen Typ T, der mit einem Typen-Typ T' mit $T \Rightarrow_{gen} T'$ erzeugt werden soll, ist ein *Simple-Proxy* und wird über die folgenden Regeln und Nebenbedingungen beschrieben:

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

Tabelle 5: Regeln und Nebenbedingungen für Simple-Proxies

5.4 SpecTypeMatcher

Die Matchingrelation für diesen Matcher wird durch folgende Regel beschrieben:

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

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
$NPX ::=$ `subproxy for` P `with` P' $\{NOMMDEL_1$ $\ldots$ $NOMMDEL_n\}$	$targetTyp(NPX) = P'$ $typ(NPX) = P$ $P \Rightarrow_{spec} P'$ $methoden(NPX) = cmethode(NOMMDEL_1) \cup$ $\ldots \cup cmethode(NOMMDEL_n)$ $methoden(NPX) \subseteq methoden(P)$ $methoden(P') \supseteq dmethode(NOMMDEL_1) \cup$ $\ldots \cup dmethode(NOMMDEL_n)$
$NOMMDEL ::=$ $m(SP) : SR \rightarrow$ $m(TP) : TR$	$SP >= TP$ $SR <= TR$ $cmethode(MOMMDEL) = m(SP) : SR$ $dmethode(MOMMDEL) = m(TP) : TR$

Tabelle 6: Regeln und Nebenbedingungen für Sub-Proxies

5.5 ContentTypeMatcher

Die Matchingrelation für diesen Matcher wird durch folgende Regel beschrieben:

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

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

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

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

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

Tabelle 7: Regeln und Nebenbedingungen für Contentproxies

5.6 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
$STPROXY$ ::= `containerproxy for` P `with` P' $\{f = NPX\}$	$targetTyp(STPROXY) = P'$ $typ(STPROXY) = P$ $P \Rightarrow_{container} P'$ $f : FT \in felder(P)$ $targetTyp(NPX) = P'$ $typ(NPX) = FT$

Tabelle 8: Regeln und Nebenbedingungen für Container-Proxies

6 Erweiterung um einen DVMatcher

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

Regel	Erläuterung
PD ::= `provided` T `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 ::= `factory` T $\{$ $FD*MD*\}$	Die Definition eines factory Typen besteht aus dem Namen des Typen T sowie mehreren Feld- und Methodendeklarationen.

Tabelle 9: Erweiterte Struktur für die Definition einer Bibliothek von Typen

Darüber hinaus wird folgende Funktion definiert:

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

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

Regel	Erläuterung
STPROXY ::= `dvproxy for` P `with` F `on m(`P'`):`P	Ein *DV-Proxy* ist ein Single-Target-Proxy, der für ein *provided Typ* 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 10: Erweiterung der Struktur für die Definition eines Proxies

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

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

Darüber hinaus müssen einige der oben beschriebenen Regeln angepasst werden, damit der *ContainerTypeMatcher*, der *ContentTypeMatcher* und der *StructuralTypeMatcher* mit dem *DVMatcher* arbeiten. Somit sind folgende Anpassungen von Nöten:

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

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

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

Tabelle 11: Regeln und Nebenbedingungen für DV-Proxies