1 Generatoren für Proxy-Arten

1.1 Simple-Proxy-Generator

Benötigt: $sourceTyp$ und $targetTyp$ mit $sourceTyp \Rightarrow_{exact} targetTyp \vee sourceTyp \Rightarrow_{gen} targetTyp$

```
function genSimpleProxy(sourceTyp, targetTyp){
        // Name des targetTyps ermitteln
        targetTypeName = nameOfTyp(targetTyp)
        return "simpleproxy for " + targetTypeName
}
```

Beispiel 1: Sei $sourceTyp =$ `ExtFire` und $targetTyp =$ `Fire`.

Es gilt: `ExtFire` $\Rightarrow_{gen}$ `Fire`.
Die Funktion `genSimpleProxy(Fire, Fire)` erzeugt folgenden Proxy:

```
simpleproxy for Fire
```

Beispiel 2: Sei $sourceTyp =$ `String` und $targetTyp =$ `String`.

Es gilt: `String` $\Rightarrow_{exact}$ `String`.
Die Funktion `genSimpleProxy(String, String)` erzeugt folgenden Proxy:

```
simpleproxy for String
```

Beispiel 3: Sei $sourceTyp =$ `boolean` und $targetTyp =$ `boolean`.

Es gilt: `boolean` $\Rightarrow_{exact}$ `boolean`.
Die Funktion `genSimpleProxy(boolean, boolean)` erzeugt folgenden Proxy:

```
simpleproxy for boolean
```

Beispiel 4: Sei $sourceTyp =$ `void` und $targetTyp =$ `void`.

Es gilt: `void` $\Rightarrow_{exact}$ `void`.
Die Funktion `genSimpleProxy(void, void)` erzeugt folgenden Proxy:

```
simpleproxy for void
```

1.2 Sub-Proxy-Generator

Benötigt: $sourceTyp$ und $targetTyp$ mit $sourceTyp \Rightarrow_{spec} targetTyp$

```
function genSubProxy(sourceTyp, targetTyp){
        head = genSubProxyHead(sourceTyp, targetTyp)
        body = genSubProxyBody(sourceTyp, targetTyp)
        return head + body
}

function genSubProxyHead(sourceTyp, targetTyp){
        sourceTypeName = nameOfTyp(sourceTyp)
        targetTypeName = nameOfTyp(targetTyp)
        return "subproxy for " + sourceTypeName + " with " +
            targetTypeName
}

function genSubProxyBody(sourceType, targetTyp){
        mDels = []
        for{pair : findMethodPairsNominal(sourceType, targetTyp){
        // erstes Element ist die Methode aus dem sourceType
        // zweites Element ist die Methode aus dem targetTyp
                mDel = pair[0] + " → " + pair[1]
                mDels.add(mDel)
        }
        // alle Elemente aus mDels mit Zeilenumbruch konkatenieren
        mDelsString = mDels.joining("\n")
        return "{\n" + mDelsString + "\n}"
}

function findMethodPairsNominal(sourceTyp, targetTyp){
        pairs = []
        for(sMethod : methodsOfType(sourceTyp)){
                sMethodName = nameOfMethod(sMethod)
                for(tMethod : methodsOfType(targetTyp){
                        tMethodName= nameOfMethod(tMethod)
        // wenn die Methodennamen gleich sind, dann werden die
            Methoden als Liste an die bestehende pairs-Liste gehaengt
                        if( sMethodName == tMethodName ){
                                pair = [sMethod, tMethod]
                                pairs.add(pair)
                        }
                }
        }
        return pairs
}
```

Beispiel: Sei $sourceTyp =$ `Partient` und $targetTyp =$ `Injured`.

Es gilt: `Partien` $\Rightarrow_{spec}$ `Injured`.
Die Funktion `generateSimpleProxy(Fire, Fire)` erzeugt folgenden Proxy:

```
        subproxy for Patient with Injured {
                heal(Medicine):void → heal(Medicine):void
        }
```

1.3 Nominal-Proxy-Generator

Der Generator für den Nominal-Proxy dient der Fallunterscheidung zwischen dem Simple- und dem Sub-Proxy.

Benötigt: $sourceTyp$ und $targetTyp$ mit $sourceTyp \Rightarrow_{internCont} targetTyp$

```
function genNominalProxy(sourceTyp, targetTyp){
        if(sourceTyp ⇒spec targetTyp){
                return genSubProxy(sourceTyp, targetTyp)
        }
        return genSimpleProxy(sourceTyp, targetTyp)
}
```

1.4 Container-Proxy-Generator

Der Generator für den Container-Proxy liefert eine Liste von Proxies.

Benötigt: $sourceTyp$ und $targetTyp$ mit $sourceTyp \Rightarrow_{container} targetTyp$

```
function genContainerProxies(sourceTyp, targetTyp){
        proxies = []
        matchingFields = findMatchingFields(sourceTyp, targetTyp)
        // fuer jedes Feld, dessen Typ auf den targetTyp matcht,
            wird ein Proxy erzeugt
        for(field : matchingFields){
                proxy = genContProxy(sourceType, targetType, field)
                proxies.add(proxy)
        }
        return proxies
}

function findMatchingFields(typWithField, matchingTyp){
        matchingFields = []
        // alle Felddeklarationen des typWithField durchlaufen
        for(field : fieldsOfType(typWithField)){
                // den Typ aus der Felddeklaraion ermitteln
                fieldTyp = typOfField(field)
                if(fieldTyp ⇒internCont matchingTyp){
                        matchingFields.add(field)
                }
        }
        return matchingFields
}

function genContainerProxy(sourceTyp, targetTyp, field){
        head = genContProxyHead(sourceTyp, targetTyp)
        body = genContProxyBody(field, targetTyp)
        return head + body
}

function genContainerProxyHead(sourceTyp, targetTyp){
        sourceTypeName = nameOfTyp(sourceTyp)
        targetTypeName = nameOfTyp(targetTyp)
```

```
        return "containerproxy for " sourceTypeName + " with " +
            targetTypeName
}

function genContainerProxyBody(field, targetTyp){
        // den Namen des Feldes ermitteln
        fieldName = nameOfField(field)
        // den Typ des Feldes ermitteln
        fieldTyp = typOfField(field)
        // Nominal-Proxy fuer den fieldTyp und den targetTyp
            erzeugen
        fieldProxy = genNominalProxy(fieldTyp, targetTyp)
        return "{\n"+ field + "=" + fieldProxy  +"\n}"
}
```

Beispiel: Sei $sourceTyp =$ `FireState` und $targetTyp =$ `boolean`.

Es gilt: `FireState` $\Rightarrow_{container}$ `boolean`.
Da `FireState` nur eine Felddeklaration enthält, kann die Funktion `genContainerProxies(FireSt` eine Liste mit maximal einem Proxy erzeugen. Da der Typ der Felddeklaration in `FireState` gleich dem $targetTyp$ ist, wird in der Funktion `genContainerProxyBody` die Funktion `genNominalProxy(boolean, boolean)` aufgerufen. Welches Ergebnis dieser Aufruf zufolgen hat, wurde bereits in Abschnitt zum Simple-Proxy-Generator (Beispiel 3) gezeigt. Im Folgenden ist das Element aus der Ergebnisliste des Funktionsaufrufes `genContainerProxies(FireState, boolean)` aufgeführt.

```
        containerproxy for FireState with boolean {
                isActive = simpleproxy for boolean
        }
```

1.5 Content-Proxy-Generator

Benötigt: $sourceTyp$ und $targetTyp$ mit $sourceTyp \Rightarrow_{content} targetTyp$

```
function genContentProxies(sourceTyp, targetTyp){
        proxies = []
        matchingFields = findMatchingFields(targetTyp, sourceTyp)
        // fuer jedes Feld des targetTyps, dessen Typ auf den
            sourceTyp matcht, wird ein Proxy erzeugt
        for(field : matchingFields){
                proxy = genContentProxy(sourceType, targetType,
                    field)
                proxies.add(proxy)
        }
        return proxies
}

function genContentProxy(sourceTyp, targetTyp, field){
        head = genContentProxyHead(sourceTyp, targetTyp)
        body = genContentProxyBody(sourceTyp, field)
```

```
        return head + body
}

function genContentProxyHead(sourceTyp, targetTyp){
        sourceTypeName = nameOfTyp(sourceTyp)
        targetTypeName = nameOfTyp(targetTyp)
        return "contentproxy for " sourceTypeName + " with " +
            targetTypeName
}

function genContentProxyBody(sourceTyp, field){
        mDels = []
        // den Namen des Feldes ermitteln
        fieldName = nameOfField(field)
        // den Typ des Feldes ermitteln
        fieldTyp = typOfField(field)
        for(pair : findMethodPairsNominal(sourceTyp, fieldTyp)){
                sMethod = pair[0]
                tMethod = pair[1]
                modSMethod = genContentSourceMethod(sMethod,
                    tMethod)
                modTMethod = genContentTargetMethod(sMethod,
                    tMethod)
                mDel = modSMethod + " → " + fieldName + "." +
                    modTMethod
                mDels.add(mDel)
        }
        return "{\n" + mDels.joining("\n") + "\n}"
}

function genContentSourceMethod(sMethod, tMethod){
        sMethodName = nameOfMethod(sMethod)
        sMethodReturnTyp = returnTypOfMethod(sMethod)
        // Parameterttypen der Methoden ermitteln
        sMethodParamTypes = paramTypesOfMethod(sMethod)
        tMethodParamTypes = paramTypesOfMethod(tMethod)
        // Proxies fuer Parametertypen erzeugen
        modParamTypen = genNominalParamProxies(sMethodParamTypes,
            tMethodParamTypes)
        return genMethodDecl(sMethodName, modParamTypen,
            sMethodReturnTyp)
}

function genContentTargetMethod(sMethod, tMethod){
        tMethodName = nameOfMethod(tMethod)
        tMethodParamTypes = paramTypesOfMethod(tMethod)
        // Rueckgabetypen der Methoden ermitteln
        sMethodReturnTyp = returnTypOfMethod(sMethod)
        tMethodReturnTyp = returnTypOfMethod(tMethod)
        // Nominalproxy fuer den Rueckgabetyp generieren
        modReturnType =  genNomialProxy(tMethodReturnTyp,
            sMethodReturnTyp)
        return genMethodDecl(tMethodName, tMethodParamTypes,
            modReturnType)
}

function genMethodDecl(name, paramTypes, retrunTyp){
        return name + "(" + paramTypes.joining(",") + "):" +
            retrunTyp
```

```
}

function genNominalParamProxies(sourceParams, targetParams){
        proxies = []
        for(i = 0; i < length(sourceProxies); i++){
                targetParam = targetParams[i]
                sourceParam = sourceParams[i]
                proxy = genNominalProxy(targetParam, sourceParam)
                proxies.add(proxy)
        }
        return proxies
}
```

Beispiel: Sei $sourceTyp =$ `Medicine` und $targetTyp =$ `MedCabinet`.

Es gilt: `Medicine` $\Rightarrow_{content}$ `MedCabinet`.
Da `MedCabinet` nur eine Felddeklaration enthält, kann die Funktion `genContentProxies(Medicin` eine Liste mit maximal einem Proxy erzeugen. Dieser Proxy ist so geartet, dass er alle Aufrufe der Methoden, die in `Medicine` deklariert sind, an das Feld `med` aus `MedCabinet` delegiert.

Da die Methode `getDescription` aus *Medicine* keine Parameter verlangt, wird die Schleife in der Methode `genParamProxies` nicht durchlaufen. Demnach wird innerhalb des Content-Proxies nur ein Proxy für den Rückgabetyp der Methode `getDescription` erzeugt. Da der *sourceTyp* mit dem Typ des Feldes `med` in `MedCabinet` übereinstimmt, sind auch die Rückgabetypen der Methoden identisch (`String`). Daher wird für die Generierung des Proxies für den Rückgabetyp die Funktion `genNominalProxy(String, String)` aufgerufen, deren Ergebnis im Abschnitt zum Simple-Proxy (Beispiel 2) aufgezeigt wurde.

Im Folgenden ist das Element aus der Ergebnisliste des Funktionsaufrufes `genContainerProxies(Medicine, MedCabinet)` aufgeführt.

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

1.6 Single-Target-Proxy-Generator

Der Generator für den Single-Target-Proxy dient der Fallunterscheidung zwischen der Generierung eines Content-, eines Container- oder eines Nominal-Proxy.

Benötigt: $sourceTyp$ und $targetTyp$ mit $sourceTyp \Rightarrow_{internStruct} targetTyp$

```
function genSingleTargetProxies(sourceTyp, targetTyp){
        proxies = []
        if(sourceTyp ⇒internCont targetTyp){
                proxy = genNominalProxy(sourceTyp, targetTyp)
                proxies.add(proxy)
        }
        else if(sourceTyp ⇒content targetTyp){
                proxies = genContentProxy(sourceTyp, targetTyp)
        }
        else if(sourceTyp ⇒container targetTyp){
                proxies = genContainerProxy(sourceTyp, targetTyp)
        }
        return proxies
}
```

1.7 Struktureller Proxy

Benötigt: $sourceTyp$ und eine Menge von $targetTypen$ P aus einer Bibliothek L mit $P \in cover(sourceTyp, L)$.
Der Generator für die strukturellen Proxies liefert eine Liste von Proxies.

```
function genStructProxies(sourceTyp, targetTypen){
        proxies = []
        combis = combineMethodStructures(sourceTyp, targetTypen)
        for(combi : combis){
                proxiesByCombi = genStructProxiesByCombi(sourceTyp,
                    combi)
                proxies.addAll(proxiesByComni)
        }
        return proxies
}

function combineMethodStructures(sourceTyp, targetTypen){
        combis = []
        // eine Kombinationen ist eine Liste von Elementen mit
            folgender Struktur:
        // Index 0: targetTyp
        // Index 1: Liste von Elementen mit folgender Struktur:
                // Index 0: Methode aus sourceTyp
                // Index 1: Liste von Methoden aus targetTyp

        return combis
}

function genStructProxiesByCombi(sourceTyp, combi){
        proxies = []
        body = genStructProxyBody(sourceTyp, combi)
        return proxies
}

function genStructProxyBody(sourceTyp, combi){
        targets = []
        for(methodCombi : combi){
```

```
                targetHead = genTargetHead(methodCombi)

                sMethod = methodCombi[0]
                tMethod = methodcombi[2]

                modSMethods = genStructSourceMethods(sMethod,
                    tMethod)
                modTMethods = genStructTargetMethods(sMethod,
                    tMethod)

                combiMDels =
                    genCombinedMethodDelegations(modSMethods,
                    modTMethods)


        }
        return targetCombis
}

function genStructSourceMethods(sMethod, tMethod){
        souceMethods = []
        sMethodName = nameOfMethod(sMethod)
        sMethodReturnTyp = returnTypOfMethod(sMethod)
        // Parameterttypen der Methoden ermitteln
        sMethodParamTypes = paramTypesOfMethod(sMethod)
        tMethodParamTypes = paramTypesOfMethod(tMethod)
        // Proxies fuer Parametertypen erzeugen
        paramProxiesCombis =
            genParamProxiesWithPermutation(sMethodParamTypes,
            tMethodParamTypes)
        for(paramProxies : paramProxiesCombis){
                modSMethod = genMethodDecl(sMethodName,
                    paramProxies, sMethodReturnTyp)
                sourceMethods.add(modSMethod)
        }
        return sourceMethods;
}

function genParamProxiesWithPermutation(sMethodParamTypes,
    tMethodParamTypes){
        paramProxiesCombis = []
        // Es wird die Permutation der Parametertertypen der
            Methode des targetTypen erzeugt
        permTParamTypes = permute(tMethodParamTypes)
        for(i = 0; i < length(permTParamTypes); i++){
                tParamTypes = permTParamTypes[i]
                // Pruefen, ob die Parametertypen in dieser
                    Reihenfolge matchen
                if(!typesMatchesInternStruct(tParamTypes,
                    sMethodParamTypes)){
                        continue
                }
                paramTypesProxies =
                    genSingleTargetParamProxies(sMethodParamTypes,
                    tParamTypes)
                combinedParamTypeProxies =
                    flatCombine(paramTypesProxies)
                paramProxiesCombis.addAll(combinedParamTypeProxies)
        }
```

```
        return paramProxiesCombis
}

function genSingleTargetParamProxies(sParamTypes, tParamTypes){
        singleTargetProxies = []
        for(i = 0; i < length(sParamTypes); i++){
                paramTypProxies =
                    genSingleTargetProxies(tParamTypes[i],
                    sParamTypes[i])
                singleTargetProxies.add(paramTypProxies)
        }
        return singleTargetProxies
}

function typesMatchesInternStruct(sourceTypes, targetTypes){
        for( i = 0; i < length(sourceTypes); i++){
                if( sourceTypes[i] ⇒internStruct targetTypes[i]){
                        continue
                }
                return false
        }
        return true
}

function flatCombine(listOfLists){
        result = []
        for(headL : listOfLists){
                // urspruengliche Liste Kopieren
                tempList = copy(listOfLists)
                // das aktuelle Element in der Kopie entfernen
                tempList.remove(headL)
                // rekursiver Aufruf mit der Restliste
                tailCombis = flatCombine(tempList)
                for(head : headL){
                        if(length(tailCombis) == 0){
                                result.add([head])
                                continue
                        }
                        for(tail : tailCombis){
                                resultElem = [head]
                                resultElem.addAll(tail)
                                result.add(resultElem)
                        }
                }
        }
        return result
}


function genStructTargetMethods(sMethod, tMethod){
        targetMethods = []
        tMethodName = nameOfMethod(tMethod)
        tMethodParamTypes = paramTypesOfMethod(tMethod)
        // Rueckgabetypen der Methoden ermitteln
        sMethodReturnTyp = returnTypOfMethod(sMethod)
        tMethodReturnTyp = returnTypOfMethod(tMethod)
        // SingleTargetProxies fuer den Rueckgabetyp generieren
        returnTypProxies = genSingleTargetProxies(sMethodReturnTyp,
            tMethodReturnTyp)
```

```
        for(returnTypProxy : returnTypeProxies){
                modTMethod = genMethodDecl(tMethodName,
                    tMethodParamTypes, returnTypProxy)
                targetMethods.add(mTMethod)
        }
        return targetMethods
}

function genCombinedMethodDelegations(modSMethods, modTMethods){
        mDels = []
        for(sMethod : modSMethods){
                for(tMethod : modTMethods){
                        mDel = sMethod + " → " + tMethod
                        mDels.add(mDel)
                }
        }
        return mDels
}
```

2 Idee für neue Struktur

```
proxy for FireState with [boolean]{
        inject boolean in state
}

proxy for MedicalFireFighter with [FireFighter, Doctor]{
        extinguishFire(Fire):FireState →
            FireFighter.extinguishFire(Fire):boolean
        heal(Injured, MedCabinet):void → modiPos(1,0)
            Doctor.heal(Medicine, Patient):void
}

proxy for Medicine with [MedCabinet]{
        getDescription():String →
            MedCabinet.med.getDescription():String
}
```