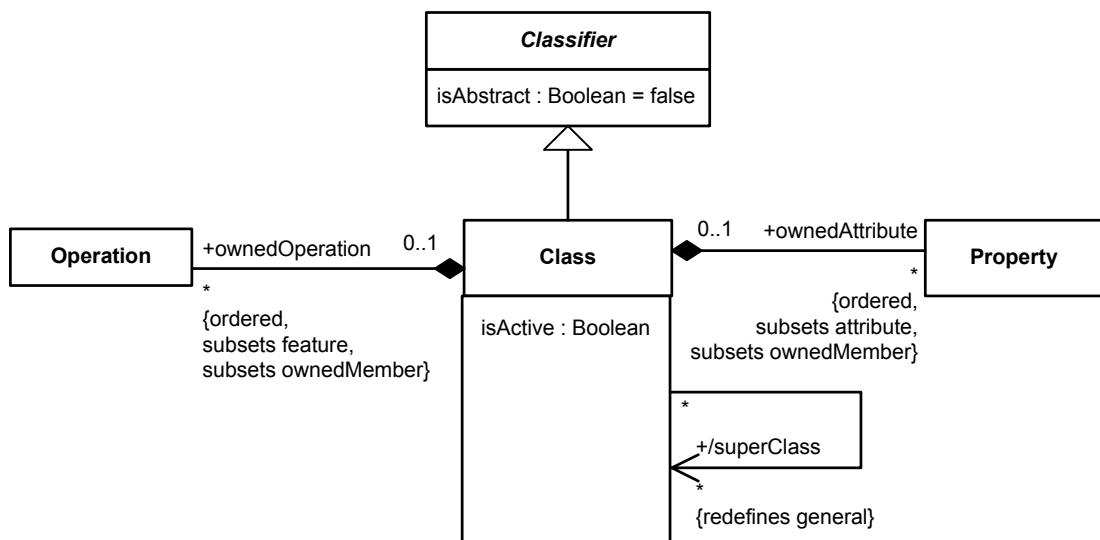


Die folgenden Abschnitte behandeln die Metamodelle zu Notationselementen aus Kapitel 6 (Klassendiagramm). Diese Abschnitte sind für die Zertifizierung zum „OMG Certified UML Professional (Fundamental)“ wichtig. Zum schnelleren Auffinden haben wir die Seitenzahlen des dazugehörigen Abschnitts in der 3. Auflage von „UML 2 glasklar“ hinzugefügt (in der 2. Auflage sind diese Abschnitte im Buch abgedruckt)

Klasse (S.108)

Metamodell

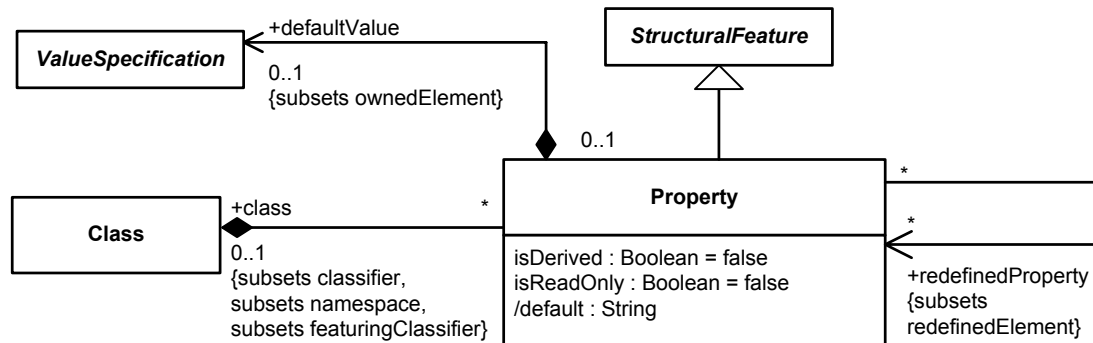


Metamodell Klasse

Das Metamodel zeigt das Element „Klasse“ als `Class` und seine Beziehungen im Metamodel. Wie die Generalisierungsbeziehung von `Class` zu `Classifier` zeigt, ist eine Klasse ein spezieller `Classifier` und erbt damit die meisten wesentlichen Eigenschaften wie Namensraum und Generalisierbarkeit (`+/superClass`). Nimmt das Attribut `isActive` einen `true`-Wert an, ist die Klasse aktiv; abstrakte Klassen werden durch das entsprechende `Classifier`-Attribut `isAbstract` gesteuert. Die Kompositionen von `Class` modellieren die Tatsache, dass eine Klasse jeweils beliebig viele (auch null) Attribute (`+ownedAttribute`) und Operationen (`+ownedOperation`) besitzen kann. Beachten Sie, dass Attribute durch die Klasse `Property` abgebildet werden. Dazu aber im nächsten Abschnitt mehr.

Attribut (S.111)

Metamodell



Metamodell Property

Das Metamodell zeigt die Einbettung des Konstrukts „Attribut“ als Klasse `Property` in das Metamodell der UML 2. Eine `Property` ist ein spezielles `StructuralFeature` und somit durch eine Spezialisierung überschreibbar (`+redefinedProperty`). Die enge Verbindung zur definierenden Klasse (`Class`) ist über eine Komposition (`+class`) modelliert. Für die Angabe des Vorgabewerts (`+defaultValue`) sieht die UML eine beliebige Wertspezifikation (`ValueSpecification`) vor.

Des Weiteren besitzt die Metaklasse `Property` noch verschiedene Attribute:

- `isDerived`: true, wenn das Attribut abgeleitet ist, der Vorgabewert hierfür ist false.
- `isReadOnly`: true, wenn das Attribut nur lesbar ist, der Vorgabewert hierfür ist false.
- `/default`: Dieses *abgeleitete* Metaattribut ermöglicht die Angabe eines Strings als Vorgabewert, der sich bereits aus der Beziehung (`+defaultValue`) zur `ValueSpecification` ergibt.

Die Klasse `Property` besitzt noch mehr Attribute und Assoziationen.

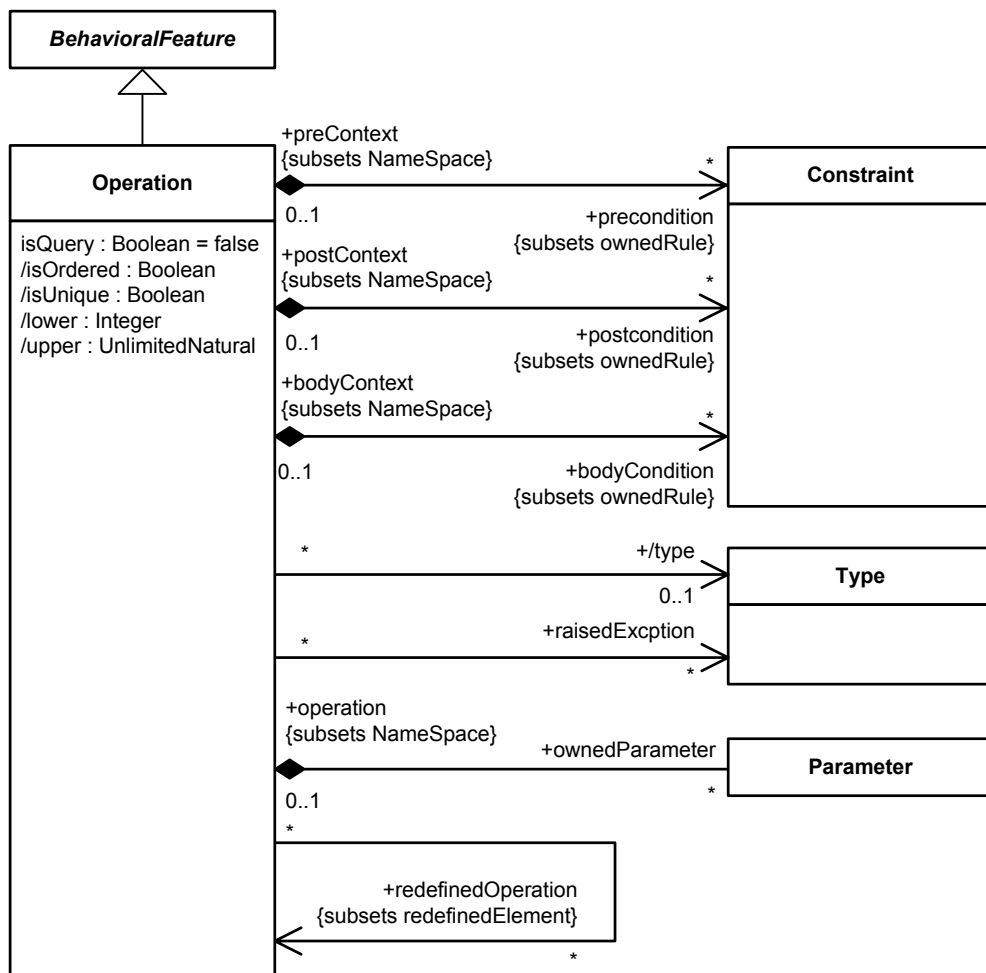
Operation (S.116)

Metamodell

Die Klasse `Operation` ist eine Unterklasse der abstrakten Klasse `BehavioralFeature`. Dadurch ist die `Operation` auch spezialisier- und überschreibbar (`+redefinedOperation`).

Das Attribut `isQuery` macht deutlich, ob bei der Ausführung der Operation Datenveränderungen auftreten können (`isQuery=false`) oder das System unverändert bleibt (`isQuery=true`). Das abgeleitete Attribut `isOrdered` spezifiziert mit true, dass die Rückgabeparameter der Operation geordnet sind. `isUnique` steuert das Vorhandensein von Duplikaten für die Rückgabeparameter der Operation, bei true sind alle Rückgabeparameter verschieden. Mit `lower` und `upper` werden die Unter- bzw. Obergrenze der Multiplizität eines Rückgabeparameters definiert.

Eine Operation kann beliebig viele Parameter (*ownedParameter*) besitzen. Daneben ist die Möglichkeit der Modellierung beliebig vieler Vor- und Nachbedingungen (*+precondition*, *+postcondition*) bzw. von Bedingungen an die Methode (Implementierung) der Operation (*+bodyCondition*) aus den Kompositionsbeziehungen zwischen *Constraint* und *Operation* ersichtlich. Die Assoziationen zur Metamodellklasse *Type* spezifiziert den Typ des Rückgabewerts (*+/type*) und/oder den Typ von möglichen auftretenden Ausnahmen bei der Ausführung einer Operation (*raisedException*).



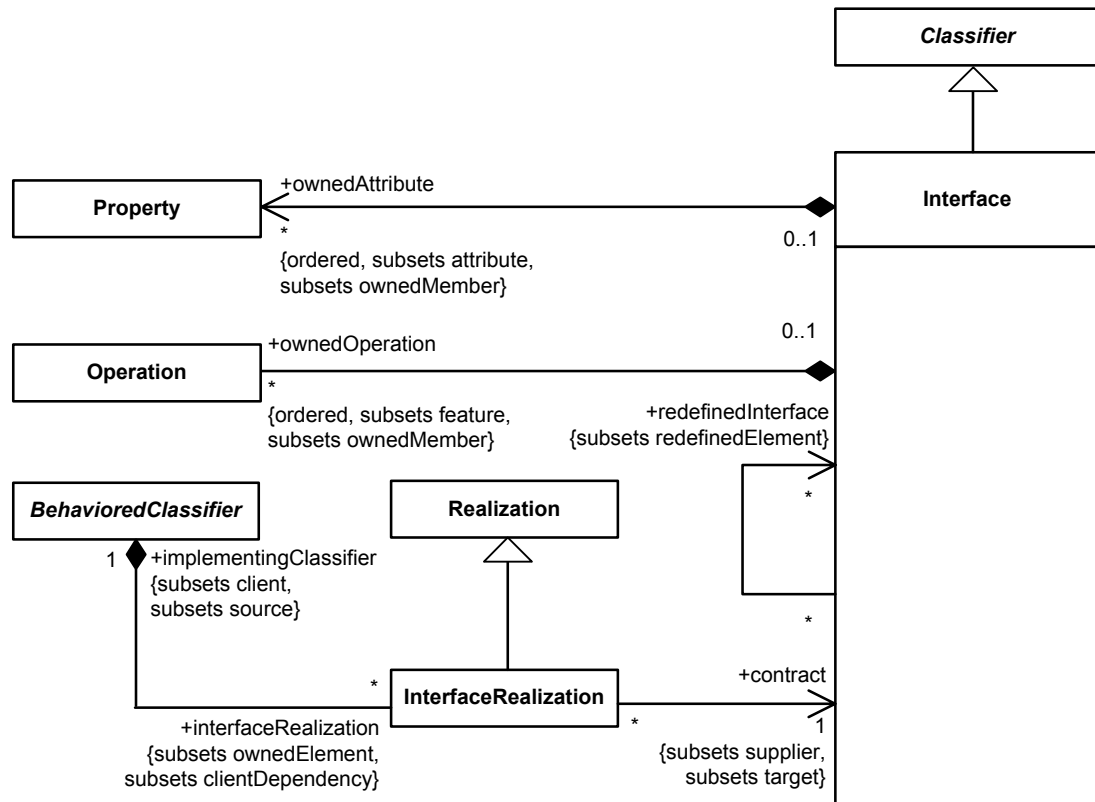
Metamodell Operation

Schnittstelle (S.122)

Metamodell

Die Metaklasse *Interface* ist eine Spezialisierung von *Classifier* und damit unter anderem spezialisier- und überschreibbar (*+redefinedInterface*). Sie repräsentiert eine Schnittstelle. Die Attribute (*+ownedAttribute*) und Operationen (*+ownedOperation*) sind wie bei der Klasse ins Metamodell eingebunden. Die Sichtbarkeit aller Attribute und Operationen einer Schnittstelle muss *public* sein.

Die Klasse `InterfaceRealization` repräsentiert das Element der Implementierungsbeziehung im Metamodell. Diese stellt eine spezielle Realisierungsbeziehung (`Realization`) dar und ist mit der Klasse `BehavoredClassifier` (die den die Schnittstelle implementierenden Classifier repräsentiert) über eine Kompositionsbeziehung (`+implementingClassifier`) verbunden. Da eine Schnittstelle von mehreren Classifiern realisiert werden kann, ist die Menge der Implementierungsbeziehungen aus Sicht der Schnittstelle unbegrenzt.

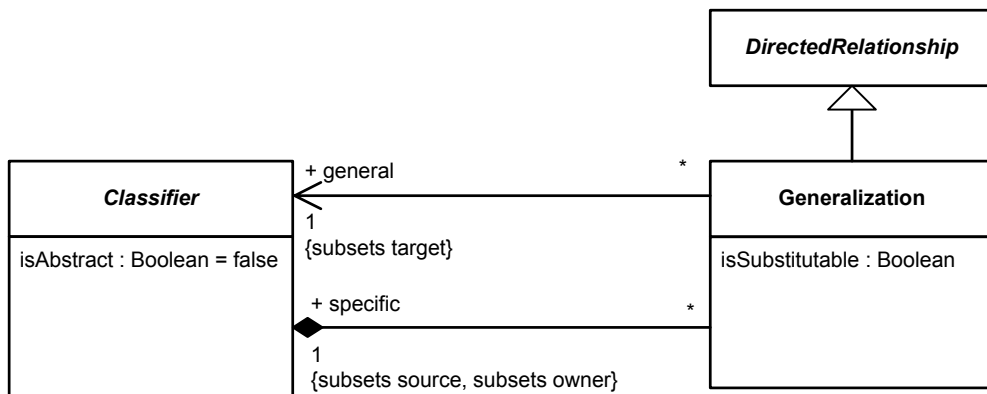


Metamodell `Interface`

Generalisierung (S. 128)

Metamodell

Folgendes Metamodell zeigt, dass eine Generalisierungsbeziehung (`Generalization`) eine spezielle gerichtete Beziehung (`DirectedRelationship`) ist. Sie besteht zwischen *zwei Classifiern*, einem verallgemeinerten an der Spitze (`+general`) und einem spezialisierten (`+specific`) Classifier am Fuß des Generalisierungspfeils. Sie sehen zudem, dass ein Classifier mehrere Generalisierungsbeziehungen (`*`) besitzen darf. Somit lässt die UML die Mehrfachgeneralisierung und damit Mehrfachvererbung zu. Die Generalisierung wird immer vom Subtyp (z.B. von einer Unterklasse) besessen und mit ihm zerstört (Komposition), der Supertyp kennt die Generalisierung nicht, er wird „nur ergänzt“. Das Attribut `isSubstitutable` definiert mit einem `true`-Wert, dass alle Subtypen der Generalisierungsbeziehung anstelle des Supertypen eingesetzt werden können.

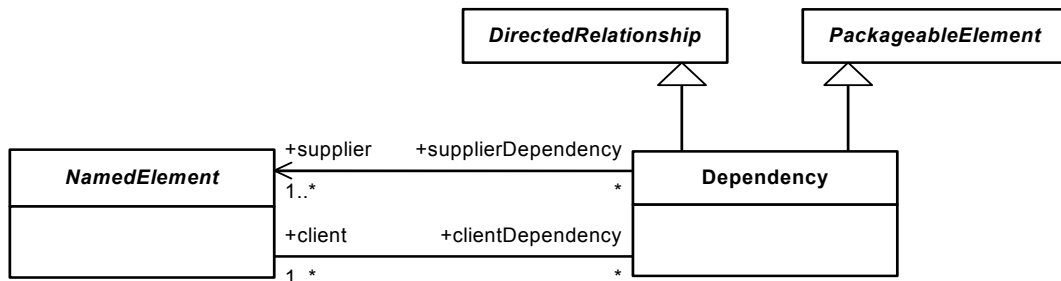


Metamodell Generalization

Abhängigkeitsbeziehung (S.153)

Metamodell

Folgende Abbildung zeigt die Stellung der Abhängigkeitsbeziehung im Metamodell. Die entsprechende Klasse `Dependency` ist eine Unterklasse der abstrakten Klassen `DirectedRelationship` und `PackageableElement`.

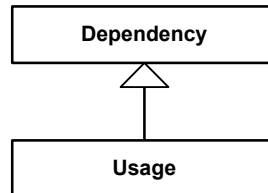


Metamodell Abhängigkeitsbeziehung

Daneben bestehen zwei Assoziationen zur abstrakten Klasse `NamedElement`. Diese sagen aus, dass jeweils mindestens ein bis beliebig *viele benannte Elemente* in ihrer Rolle als `+supplier` bzw. `+client` in einer Abhängigkeit existieren und ein benanntes Element seinerseits keine oder *beliebig viele* Abhängigkeiten besitzen kann. Daraus folgt, dass nicht nur Klassen, sondern beliebige benannte Elemente voneinander abhängig sein können. Interessanterweise „kennen“ die benannten Elemente die Abhängigkeitsbeziehung, da es sich hier um keine gerichteten Beziehungen handelt. Dadurch ist die Navigierbarkeit zwischen Client- und Supplier-Elementen jederzeit möglich.

Verwendungsbeziehung (S.154)

Metamodell



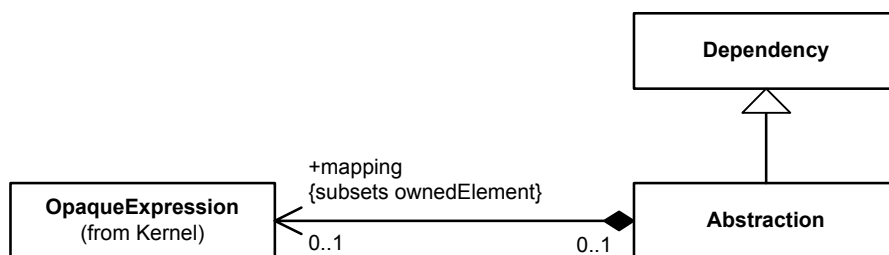
Metamodell Usage-Beziehung

Die Usage-Beziehung ist eine Spezialisierung der Abhängigkeitsbeziehung (Dependency). Die Klasse Usage besitzt keine weiteren Attribute oder Assoziationen.

Abstraktionsbeziehung (S.155)

Metamodell

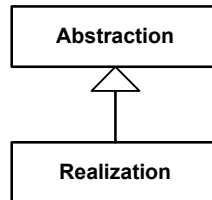
Wie aus dem Metamodell-Ausschnitt ersichtlich wird, ist eine Abstraktionsbeziehung (Klasse Abstraction) eine spezielle Abhängigkeitsbeziehung. Daneben kann eine Abstraktionsbeziehung eine OpaqueExpression, einen Ausdruck in einer bestimmten Sprache oder Notation, enthalten. Hier steht die OpaqueExpression für einen Ausdruck, der die Abstraktion formal beschreibt.



Metamodell Abstraction-Klasse

Realisierungsbeziehung (S.157)

Metamodell

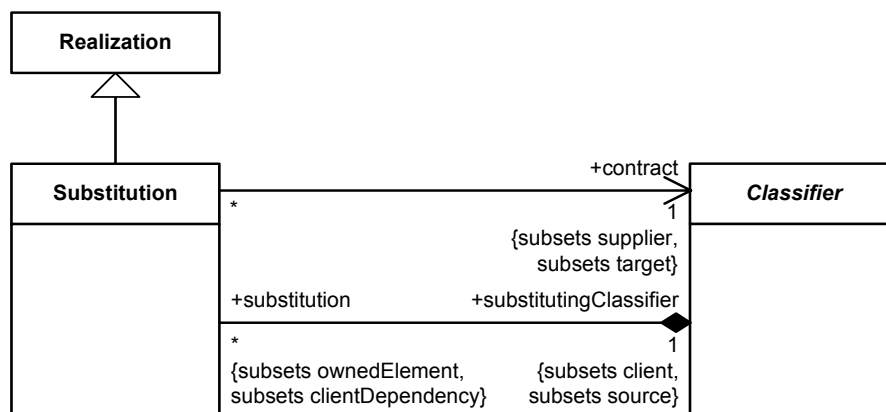


Metamodell Realization-Beziehung

Eine Realization-Beziehung ist eine Spezialisierung der Abstraktionsbeziehung (Abstraction), für die keine speziellen Attribute oder Assoziationen existieren.

Substitutionsbeziehung (S.158)

Metamodell



Metamodell Substitution-Beziehung

Eine Substitution-Beziehung ist eine spezielle Realisierungsbeziehung (Realization). Sie verbindet genau einen Supplier-Classifer (+contract) mit einem Client-Classifer (+substitutingClassifier, subsets client). Beachten Sie, dass der Client-Classifer der Besitzer der Substitutionsbeziehung ist (subsets ownedElement) und daher in einer Kompositionsbeziehung mit der Substitution steht.