Faktor-IPS Tutorial

Teil 1: Modellierung und Produktkonfigurierung

Jan Ortmann, Gunnar Tacke
(Dokumentversion 39)

Einleitung

Faktor-IPS ist ein OpenSource-Werkzeug zur modellgetriebenen Entwicklung[1] versicherungsfachlicher Softwaresysteme mit Fokus auf der einheitlichen Abbildung des Produktwissens. Insbesondere können mit Faktor-IPS nicht nur die Modelle der Systeme bearbeitet, sondern auch die Produktinformationen selbst verwaltet werden. Neben reinen Produktdaten können einzelne Produktaspekte auch über eine Excel-ähnliche Formelsprache definiert werden. Darüber hinaus können Tabellen verwaltet und fachliche Testfälle definiert und ausgeführt werden.

Dieses Tutorial führt in die Konzepte von und die Arbeitsweise mit Faktor-IPS ein. Als durchgängiges Beispiel verwenden wir dazu eine stark vereinfachte Hausratversicherung. Die grundlegenden Konstruktions- und Modellierungsprinzipien lassen sich auch anhand dieses sehr einfachen fachlichen Modells darstellen. Insbesondere behandeln wir in dem Tutorial die Möglichkeiten zur Produktkonfiguration inklusive und wie Änderungen im Zeitablauf in einem fachlichen Modell abgebildet werden.

Das Tutorial ist in zwei Teile gegliedert:

1. Der erste Teil führt in die Arbeit mit dem Modellierungswerkzeug und dem generierten Sourcecode ein, und zeigt wie konkrete Produkte konfiguriert werden.
2. Der zweite Teil beschreibt die Verwendung von Tabellen, die Implementierung der Beitragsberechnung und zeigt, wie mit Hilfe von Formeln das Hausratmodell flexibel gestaltet werden kann.

Das Tutorial ist für Softwarearchitekten und Entwickler mit fundierten Kenntnissen über objektorientierte Modellierung mit der UML geschrieben. Erfahrungen mit der Entwicklung von Java-Anwendungen in Eclipse sind hilfreich, aber nicht unbedingt erforderlich.

Wenn Sie die einzelnen Schritte des Tutorials nicht selber durchführen wollen, können Sie das Endergebnis auch von www.faktorips.org herunterladen und installieren.

Ein weiteres Tutorial zeigt, wie man mit den hier erstellten Modellklassen in einer operativen Anwendung arbeitet (Tutorial Hausrat Angebotsystem).

Darüber hinaus gibt es ein Tutorial, das auf die Partitionierung großer Modelle eingeht. Insbesondere die Trennung in einzelne Sparten und die Trennung von spartenspezifischen und spartenübergreifenden Aspekten wird dort beispielhaft gezeigt.

Überblick über Teil 1 des Tutorials

Der erste Teil des Tutorials ist wie folgt gegliedert:

- Hello Faktor-IPS

1 Modell driven software development (MDSD). Eine sehr gute Beschreibung der zugrundeliegenden Konzepte findet sich in Stahl, Völter: Modellgetriebene Softwareentwicklung.

In diesem Kapitel wird ein erstes Faktor-IPS Projekt angelegt und eine erste Klasse definiert.

- Arbeiten mit Modell und Sourcecode
  Anhand eines Modells einer Hausratversicherung wird der Umgang mit dem Modellierungswerkzeug und dem generierten Sourcecode erläutert.
- Aufnahme von Produktaspekten ins Modell
  In dem Kapitel wird das Modell der Hausratversicherung um die Möglichkeiten zur Produktkonfiguration erweitert.
- Definition der Hausratprodukte
  Auf Basis des Modells werden nun zwei Hausratprodukte erfasst. Hierzu wird die Produktdefinitionsperspektive verwendet, die speziell für die Fachabteilung konzipiert ist.
- Zugriff auf Produktinformationen zur Laufzeit
  In dem Kapitel wird erläutert wie man zur Laufzeit, also in einer Anwendung oder einem Testfall auf Produktinformationen zugreift.

„Hello Faktor-IPS“

Im ersten Schritt dieses Tutorial legen wir ein Faktor-IPS Projekt an, definieren eine Modellklasse und generieren Java-Sourcecode zu dieser Modellklasse.

Falls Sie Faktor-IPS noch nicht installiert haben, tun Sie das jetzt. Die Software und die Installationsleitung finden Sie auf www.faktorips.org. In diesem Tutorial verwenden wir Eclipse 3.7 und Faktor-IPS 3.6.0 in Englisch. Starten Sie nun Eclipse. Am besten verwenden Sie für dieses Tutorial einen eigenen Workspace. Wenn Faktor-IPS korrekt installiert ist, sollten Sie bei geöffneter Java-Perspektive in der Toolbar folgende Symbole sehen[2]:

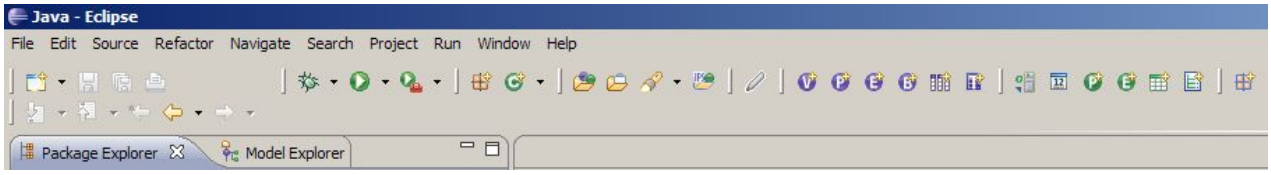


Abbildung 1: Faktor-IPS Icons

Faktor-IPS-Projekte sind normale Java-Projekte mit einer zusätzlich Faktor-IPS-Nature. Als erstes legen Sie also ein neues Java-Projekt mit dem Namen „Hausratmodell“ an. Hierzu klicken Sie im Menü auf File ► New ► Java Project. In dem Dialog brauchen Sie lediglich den Namen des Projektes angeben und klicken dann auf Finish.

In diesem Projekt werden wir die Modellklassen anlegen. Die Faktor-IPS-Nature fügen Sie dem Projekt hinzu, indem Sie es im Java Package-Explorer markieren und im Kontextmenü Faktor-IPS ►Add IPS-Nature... wählen.

2 In den Tutorials von Faktor-IPS wird von einer Installation ohne Faktor-IPS German Language-Pack ausgegangen. Wenn Sie das Language-Pack installiert haben, aber trotzdem ohne die Übersetzung arbeiten wollen, können sie einfach Eclipse mit einer anderen Locale starten, z.B. mit `eclipse -vmargs -Duser.language=en`

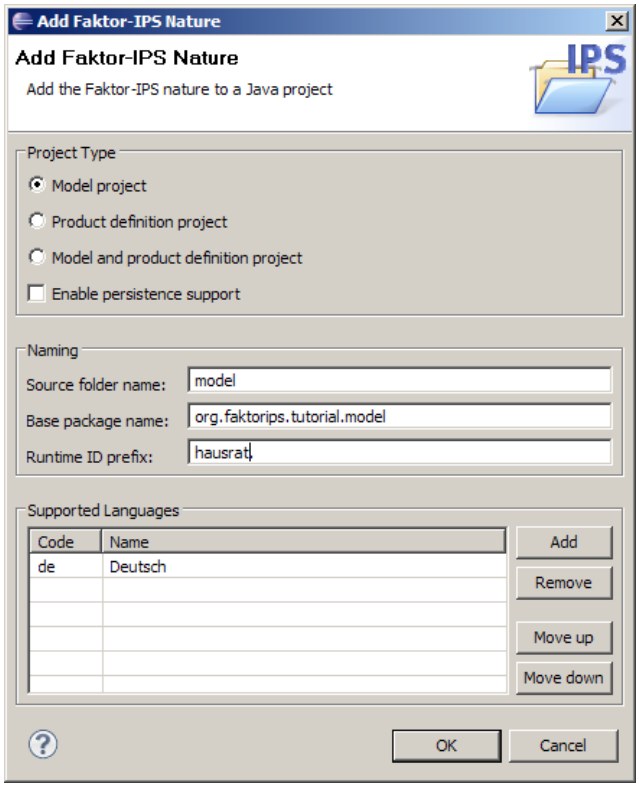


Abbildung 2: Hinzufügen der IPS-Nature

Als Sourceverzeichnis geben Sie „`model`“ ein, als Basis-Package für die generierten Java-Klassen "`org.faktorips.tutorial.model`", als Runtime-ID-prefix "`hausrat.`" und drücken OK. Dem Projekt werden die Laufzeitbibliotheken von Faktor-IPS hinzugefügt und das angegebene Sourceverzeichnis („model“) angelegt. In dem Sourceverzeichnis wird die Modelldefinition abgelegt. Unterhalb dieses Verzeichnisses kann die Modellbeschreibung wie in Java durch Packages (Pakete) strukturiert werden. Faktor-IPS verwendet wie Java qualifizierte Namen zur Identifikation der Klassen des Modells. Die Bedeutung des RuntimeID-Prefixes wird im Kapitel "Definition der Produkte" erläutert.

Darüber hinaus wurde dem Projekt ein neues Java Sourceverzeichnis mit dem Namen „derived“ hinzugefügt. In dieses Verzeichnis generiert Faktor-IPS Java Sourcefiles und kopiert XML-Dateien, die zu 100% generiert werden. Der Inhalt des Verzeichnisses kann also jederzeit gelöscht und neu erzeugt werden. Im Gegensatz hierzu enthält das ursprüngliche Java Sourceverzeichnis Dateien, die vom Entwickler bearbeitet werden können und die beim Generieren gemerged werden.

Bevor wir die erste Klasse *HausratVertrag* definieren, stellen Sie noch ein, dass der Workspace automatisch gebaut wird (im Menü: Project►Build automatically).

Wechseln Sie zunächst in den Modell-Explorer von Faktor-IPS direkt neben dem Package-Explorer.

Falls der Modell-Explorer nicht sichtbar ist, liegt das daran, dass Sie diesen Workspace bereits vor der Installation von Faktor-IPS verwendet haben. Rufen Sie in diesem Fall im Menü Window►Reset Perspective *auf.*

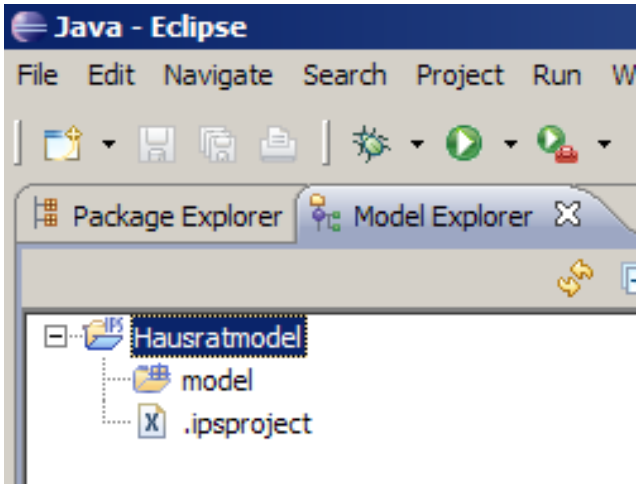


Abbildung 3: Ansicht der Projekte im Modell-Explorer

Im Modell-Explorer wird die Modelldefinition ohne die Java-Details dargestellt. In der Datei „.ipsproject“ sind die Eigenschaften des Faktor-IPS Projektes gespeichert. Hierzu gehören zum Beispiel die gerade im AddIpsNature-Dialog eingegebenen Informationen, Einstellungen für die Codegenerierung, die erlaubten Datentypen etc. Der Inhalt ist in XML abgelegt und ausführlich in der Datei dokumentiert.

Die Klassen werden wir in einem Package mit dem Namen `hausrat` ablegen. Zum Anlegen des IPS Packages markieren Sie zunächst das Sourceverzeichnis `model` und erzeugen über das Kontextmenü ein neues IPS Package `hausrat`.

Als nächstes wollen wir eine Klasse anlegen, die unseren Hausratvertrag repräsentiert. Markieren Sie dazu das neu angelegte Package im Package Explorer und drücken auf den Button in der Toolbar.

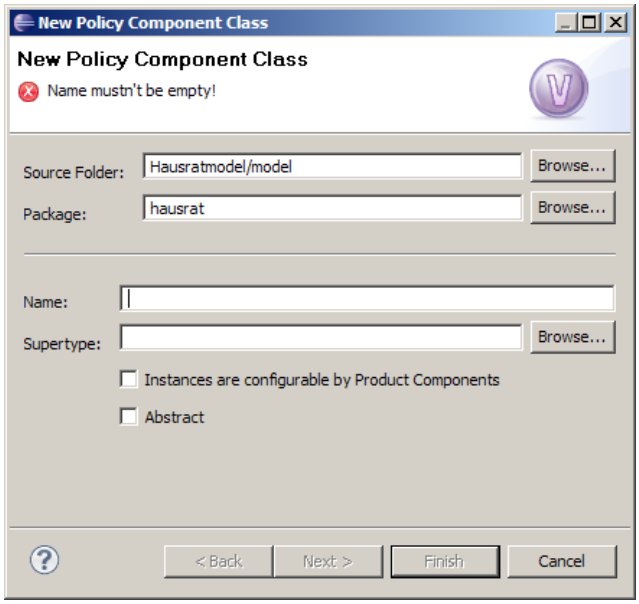


Abbildung 4: Anlegen einer neuen Vertragsklasse

In dem Dialog sind Sourceverzeichnis und Package bereits entsprechend vorbelegt und Sie geben noch den Namen der Klasse an, also *HausratVertrag* und klicken auf Finish. Faktor-IPS hat jetzt die neue Klasse angelegt und den Editor zur Bearbeitung geöffnet. Wechseln Sie zurück in den Package-Explorer. Sie sehen, dass die Klasse *HausratVertrag* in einer eigenen Datei mit dem Namen HausratVertrag.ipspolicycmpttype gespeichert ist.

Weiterhin hat der Codegenerator von Faktor-IPS bereits zwei Java-Sourcefiles erzeugt `org.faktorips.tutorial.modell.hausrat.IHausratVertrag` und `org.faktorips.tutorial.modell.internal.hausrat.HausratVertrag`.

Die erste Datei enthält das sogenannte published Interface der Modellklasse *HausratVertrag*. Es enthält alle Eigenschaften, die für Clients des Modells sichtbar und nutzbar sind. Da wir bisher noch keine Eigenschaften der Klasse *HausratVertrag* definiert haben, enthält dieses Interface erst einmal keine Methoden.

Die Klasse *HausratVertrag* ist die modellinterne Implementierung des published Interface. Ein kurzer Blick in den Sourcecode zeigt, dass hier schon einige Methoden generiert worden sind. Diese Methoden dienen unter anderem zur Konvertierung der Objekte in XML und zur Unterstützung von Prüfungen.

Durch die Trennung von published Interface und Implementierung können die Modellklassen auf unterschiedliche Packages aufgeteilt werden, ohne dass dies zur Folge hat, dass modellinterne Methoden auch für Clients sichtbar sind[3].

3 In Java müssen Methoden public sein, die von einer Klasse eines anderen Package aufgerufen werden. Es kann nicht unterschieden werden, ob es sich um die gleiche oder eine andere Schicht der Softwarearchitektur handelt.

Arbeiten mit Modell und Sourcecode

In zweitem Schritt des Tutorials erweitern wir unser Modell und arbeiten mit dem generierten Sourcecode.

Als erstes erweitern wir die Klasse *HausratVertrag* um ein Attribut *zahlweise*. Wenn der Editor mit der Vertragsklasse nicht mehr geöffnet ist, öffnen Sie diesen nun durch Doppelklick im Modell-Explorer. In dem Editor klicken Sie auf den Button New im Abschnitt Attributes. Es öffnet sich der folgende Dialog:

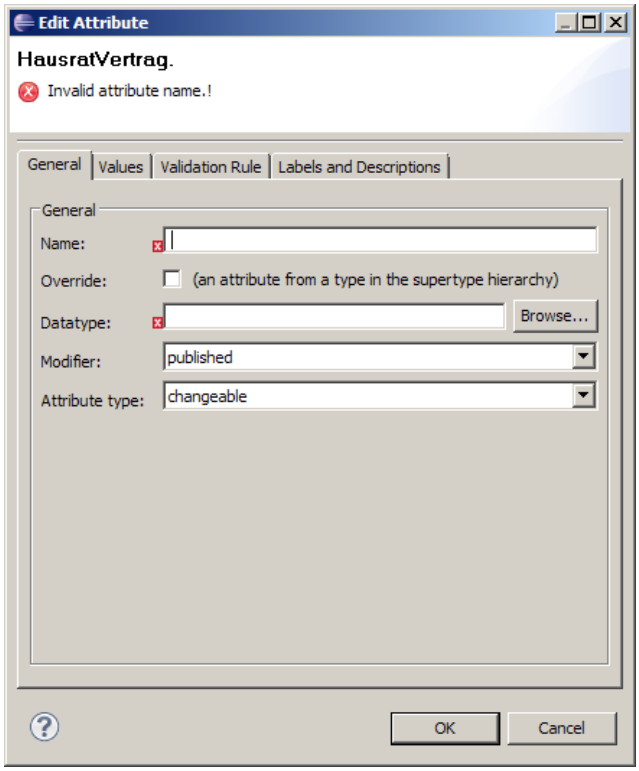


Abbildung 5: Dialog zum Anlegen eines neuen Attributs

Die Felder haben die folgende Bedeutung:

Feld	***Bedeutung***
Name	Der Name des Attributs.
Override	Indikator, ob dieses Attribut bereits in einer Superklasse definiert worden ist, und in dieser Klasse lediglich Eigenschaften wie z. B. der Default Value überschrieben werden[4].
Datatype	Datentyp des Attributs.
Modifier	Analog zum Modifier in Java. Der zusätzliche Modifier published bedeutet, dass die Eigenschaft ins published Interface aufgenommen wird.

4 Entspricht der `@override` Annotation in Java 5.

Feld	*Bedeutung*
Attribute type	Der Typ des Attributs. • changeable Änderbare Eigenschaften, also solche mit Getter- und Setter-Methoden. • constant Konstante, nicht änderbare Eigenschaft. • derived (cached, computation by explicit method call) Abgeleitete Eigenschaften im UML Sinne. Die Eigenschaft wird durch eine expliziten Methodenaufruf berechnet und das Ergebnis ist danach über die Getter-Methode abfragbar. Zum Beispiel kann die Eigenschaft bruttobeitrag durch eine Methode berechneBeitrag berechnet und danach über `getBruttobeitrag()` abgerufen werden. • derived (computation on each call of the getter method) Abgeleitete Eigenschaften im UML Sinne. Die Eigenschaft wird bei jedem Aufruf der Getter-Methode berechnet. Zum Beispiel kann das Alter einer versicherten Person bei jedem Aufruf von `getAlter()` aus dem Geburtstag ermittelt werden.

Geben Sie als Namen *zahlweise* und als Datentyp *Integer* ein. Wenn Sie auf den Browse Button neben dem Feld klicken, öffnet sich eine Liste mit den verfügbaren Datentypen. Alternativ dazu können Sie wie in Eclipse üblich auch mit Strg-Space eine Vervollständigung durchführen. Wenn Sie zum Beispiel „D“ eingeben und Strg-Space drücken, sehen Sie alle Datentypen, die mit „D“ beginnen. Die anderen Felder lassen Sie wie vorgegeben und drücken jetzt Ok, danach speichern Sie die geänderte Vertragsklasse.

Der Codegenerator hat nun bereits die Java-Sourcefiles aktualisiert. Das published Interface `IHausratVertrag` enthält nun Zugriffsmethoden für das Attribut. Die Implementierung `HausratVertrag` implementiert diese Methoden und speichert den Zustand in einer privaten Membervariable.

```
/**
 * Membervariable fuer zahlweise.
 *
 * @generated
 */
private Integer zahlweise = null;

/**
 * {@inheritDoc}
 *
 * @generated
 */
public Integer getZahlweise() {
    return zahlweise;
}

/**
 * {@inheritDoc}
 *
 * @generated
 */
public void setZahlweise(Integer newValue) {
    this.zahlweise = newValue;
}
```

Das JavaDoc für die Membervariable und für die Getter-Methode ist mit einem `@generated` markiert. Dies bedeutet, dass die Methode zu 100% generiert wird. Bei einer erneuten Generierung wird dieser Code genau so wieder erzeugt, unabhängig davon, ob er in der Datei gelöscht oder modifiziert worden ist. Änderungen seitens des Entwicklers werden also überschrieben. Möchten Sie die Methode modifzieren, so fügen Sie hinter die Annotation `@generated` ein `NOT` hinzu. Probieren wir das einmal aus. Fügen Sie jeweils eine Zeile in die Getter- und Setter-Methode ein, und ergänzen bei der Methode `setZahlweise() NOT` hinter der Annotation, also etwa

```
/**
 * {@inheritDoc}
 *
 * @generated
 */
public Integer getZahlweise() {
    System.out.println("getZahlweise");
    return zahlweise;
}
```

```
/**
 * {@inheritDoc}
 *
 * @generated NOT
 */
public void setZahlweise(Integer newValue) {
    System.out.println("setZahlweise");
    this.zahlweise  = newValue;
}
```

Generieren Sie jetzt den Sourcecode für die Klasse *HausratVertrag* neu. Dies können Sie wie in Eclipse üblich auf zwei Arten erreichen:

- Entweder bauen Sie mit Project▶Clean das gesamt Projekt neu, oder
- Sie speichern die Modellbeschreibung der Klasse *HausratVertrag* erneut.

Nach dem Generieren ist das `System.out.println(...)`-Statement aus der Getter-Methode

entfernt worden, in der Setter-Methode ist es erhalten geblieben.

Methoden und Attribute die neu hinzugefügt werden, bleiben nach der Generierung erhalten. Auf diese Weise kann der Sourcecode beliebig erweitert werden.

Nun erweitern wir noch die Modelldefinition der Zahlweise um die erlaubten Werte. Öffnen Sie dazu den Dialog zum Bearbeiten des Attributes und wechseln auf die zweite Tabseite. Bisher sind alle Werte des Datentyps als zulässige Werte für das Attribute erlaubt. Wir schränken dies nun auf 1, 2, 4, 12 für jährlich, halbjährlich, quartalsweise und monatlich ein. Ändern Sie hierzu den Typ auf „Enumeration“ und geben Sie in die Tabelle die Werte 1, 2, 4 und 12 ein[5].

Setzen Sie nun einmal den Default Value auf 0. Faktor-IPS markiert den Default Value mit einer Warnung, da der Wert nicht in der im Modell erlaubten Wertemenge enthalten ist.

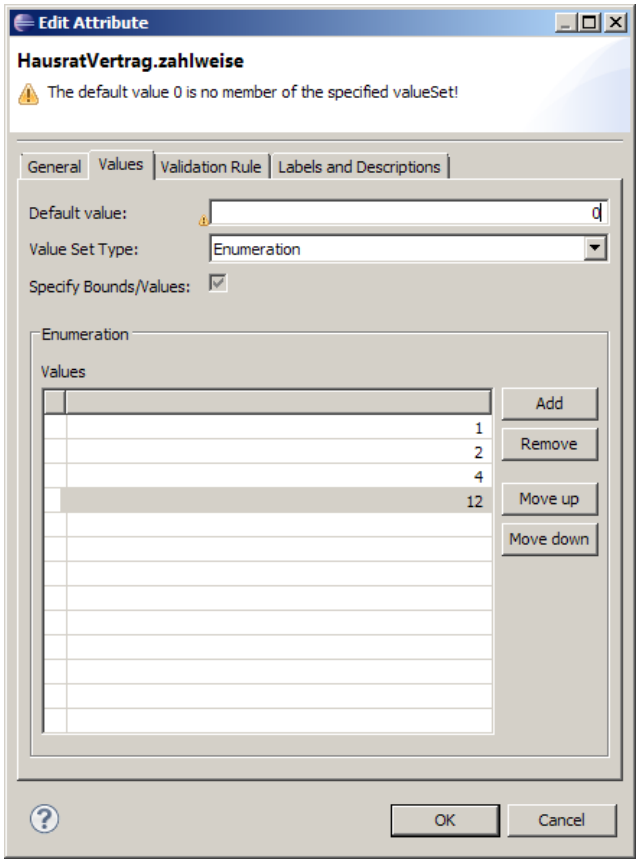


Abbildung 6: Wertebereich für zahlweise

Es handelt sich also um einen möglichen Fehler im Modell. Lassen wir das aber für einen Augenblick so stehen. Das gibt uns die Gelegenheit die Fehlerbehandlung von Faktor-IPS zu erläutern. Schließen Sie dazu den Dialog und speichern die Vertragsklasse. Im Problems-View von Eclipse wird nun die gleiche Warnung wie im Dialog angezeigt. Faktor-IPS lässt Fehler und Inkonsistenzen im Modell zu und informiert den Benutzer darüber im Eclipse-Stil, also in den Editoren und als so genannte Problemmarker, die im Problem-View und in den Explorern sichtbar

5 Faktor-IPS unterstützt auch die Definition von Enums. Auf dieses Feature wird an dieser Stelle aber verzichtet. Darüber hinaus können über einen Extension Point beliebige Java Klassen als Datentyp registriert werden. Diese Java-Klassen sollten dabei entsprechend des Musters ValueObject realisiert sein.

sind.

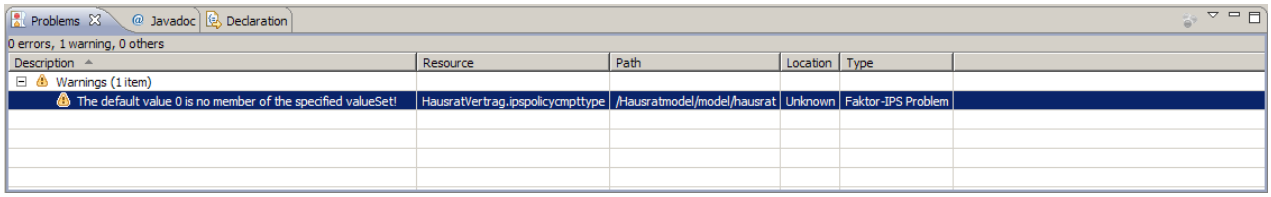


Abbildung 7: Anzeige von Fehlern in der Problems-View

Setzen Sie als Defaultwert wieder <null> ein und speichern die Vertragsklasse. Die Warnung wird damit wieder aus dem Problem-View entfernt.

Faktor-IPS generiert eine Warnung und keinen Fehler, da es durchaus sinnvoll sein kann, wenn der Defaultwert nicht im Wertebereich ist. Insbesondere gilt dies für den Defaultwert `null`*. Wird zum Beispiel ein neuer Vertrag angelegt, so kann es gewolltes Verhalten sein, dass die Zahlweise nicht vorbelegt ist sondern noch null ist, um die Eingabe der Zahlweise durch den Benutzer zu erzwingen. Erst wenn der Vertrag vollständig erfasst ist, muss auch die Bedingung erfüllt sein, dass die Eigenschaft Zahlweise einen Wert aus dem Wertebereich enthält.*

Das Kapitel schließen wir mit der Definition einer Klasse *HausratGrunddeckung* und der Kompositionsbeziehung zwischen *HausratVertrag* und *HausratGrunddeckung* gemäß dem folgenden Diagramm:

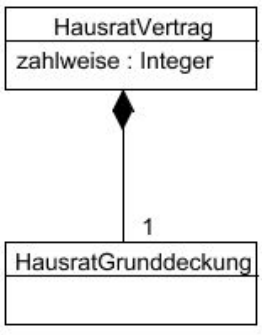


Abbildung 8: Modell der Vertragsseite

Legen Sie zunächst die Klasse *HausratGrunddeckung* analog zur Klasse *HausratVertrag* an. Danach wechseln Sie zurück in den Editor, der die Klasse HausratVertrag anzeigt. Starten Sie den Assistenten zur Anlage einer neuen Beziehung durch Klicken auf den New Button rechts neben dem Abschnitt, der mit Associations überschrieben ist[6].

6 In Faktor-IPS wird in Übereinstimmung mit der UML der Begriff Association verwendet. In dem Tutorial verwenden wir den im Sprachgebrauch üblicheren Begriff Beziehung.

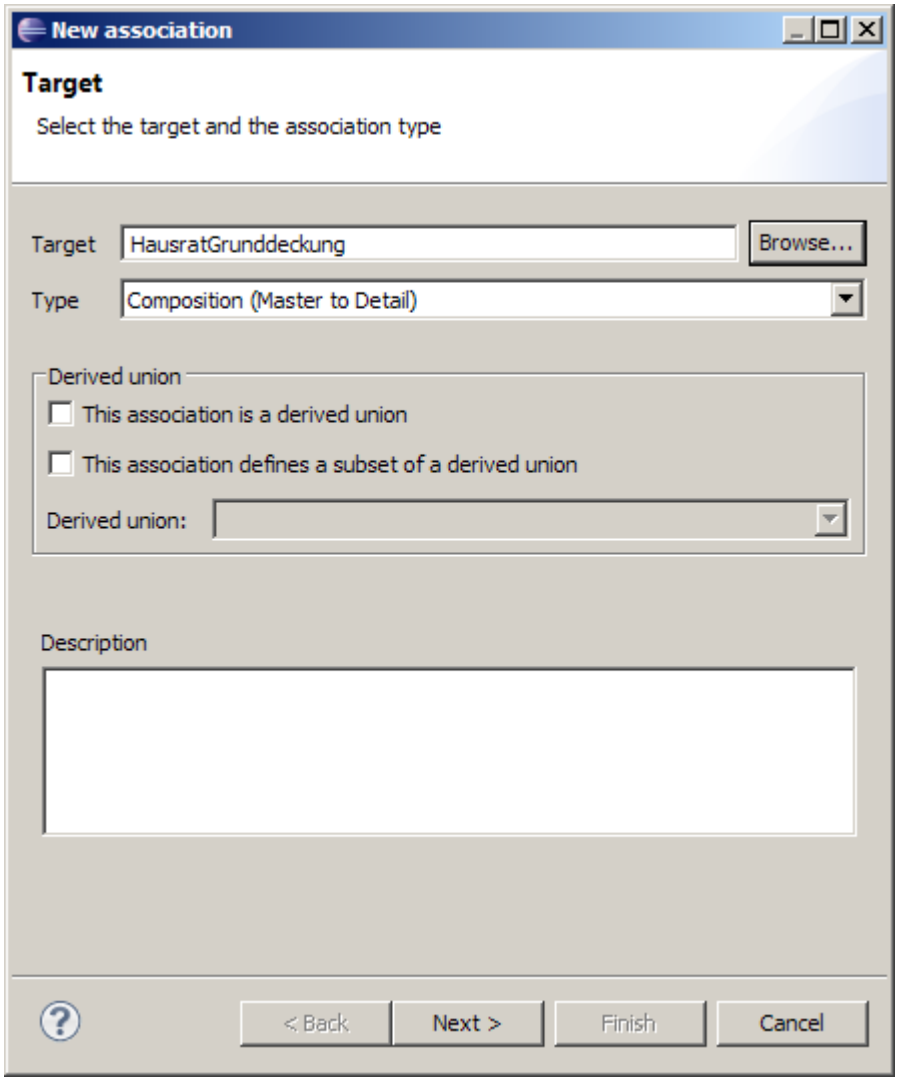


Abbildung 9: Anlegen einer neuen Beziehung

Als Target wählen Sie bitte die gerade angelegte Klasse *HausratGrunddeckung* aus. Hier steht ihnen wieder die Vervollständigung mit STRG-Space zur Verfügung. Die mit Derived Union überschriebene Box ignorieren Sie zunächst. Das Konzept wird im Tutorial zur Modellpartitionierung erläutert.

Auf der nächsten Seite geben Sie als minimale Kardinalität 1 und als maximale Kardinalität 1 ein, als Rollennamen HausratGrunddeckung bzw. HausratGrunddeckungen. Die Mehrzahl wird verwandt, damit der Codegenerator verständlichen Sourcecode generieren kann.

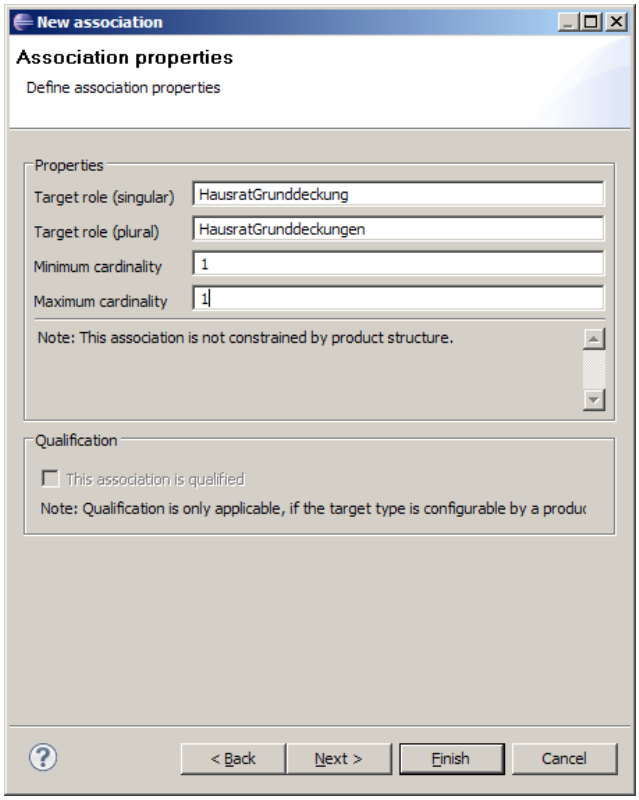


Abbildung 10: Rollennamen und Kardinalitäten einer Beziehung

Auf der nächsten Seite können Sie auswählen, ob es auch eine Rückwärtsbeziehung von *HausratGrunddeckung* zu *HausratVertrag* geben soll. Beziehungen in Faktor-IPS sind immer gerichtet, so ist es auch möglich die Navigation nur in eine Richtung zuzulassen. Hier wählen Sie bitte New inverse association und gehen zur nächsten Seite. Hier brauchen Sie nur noch die Rollenbezeichnung einzugeben. Mit Finish legen Sie die beiden Beziehungen (vorwärts und rückwärts) an und speichern danach noch die Klasse *HausratVertrag*. Wenn Sie sich jetzt die Klasse *HausratGrunddeckung* ansehen, ist dort die Rückwärtsbeziehung eingetragen.

Zum Abschluss werfen wir noch einen kurzen Blick in den generierten Sourcecode. In das published Interface `IHausratVertrag` wurden Methoden generiert, die Grunddeckung dem *HausratVertrag* hinzuzufügen etc. In dem Interface `IHausratGrunddeckung` gibt es eine Methode, um zum *HausratVertrag* zu navigieren, zu dem die *HausratGrunddeckung* gehört. Kompositbeziehungen werden also immer über die Containerklasse hergestellt (und aufgelöst). Wenn sowohl die Vorwärts- als auch die Rückwärtsbeziehung im Modell definiert ist, werden hierbei beide Richtung berücksichtigt. Das heißt nach dem Aufruf von `setHausratGrund-deckung(IHausratGrunddeckung d)` auf einer Instanz `v` von `HausratVertrag` liefert `d.getHausratVertrag()` wieder `v` zurück. Dies zeigt ein kurzer Blick in die Implementierung der Methode `setHausratGrunddeckung()`[7] in der Klasse `HausratVertrag`:

7 Die generierten Javadocs zeigen wir von nun an nicht mehr im Tutorial, da die Methoden im Text erläutert werden.

```
public void setHausratGrunddeckung(IHausratGrunddeckung newObject) {
        if (hausratGrunddeckung != null) {
                hausratGrunddeckung.setHausratVertragInternal(null);
        }
        if (newObject != null) {
                ((HausratGrunddeckung) newObject).setHausratVertragInternal(this);
        }
                hausratGrunddeckung = (HausratGrunddeckung) newObject;
}
```

Der Vertrag wird in der Deckung als der Vertrag gesetzt, zu dem die Deckung gehört (zweites `if`-Statement der Methode) .

Erweiterung des Hausratmodells

In diesem Abschnitt werden wir unser Hausratmodell vervollständigen. Die folgende Abbildung zeigt das Modell.

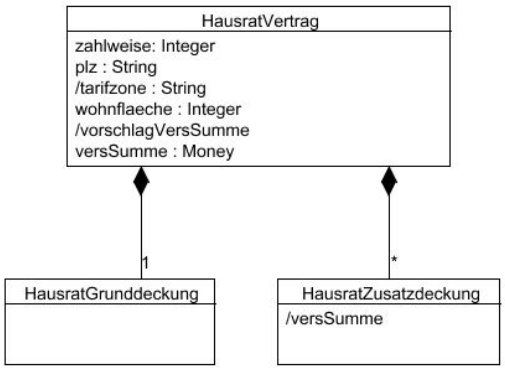


Abbildung 11: Hausratmodell mit Grunddeckung und Zusatzdeckung

Jeder Hausratvertrag muss genau eine Grunddeckung enthalten und kann beliebig viele Zusatzdeckungen enthalten. Die Grunddeckung deckt immer die im Vertrag definierte Versicherungssumme. Darüber hinaus kann ein Hausratvertrag optional Zusatzdeckungen enthalten, typischerweise sind dies Deckungen gegen Risiken wie zum Beispiel Fahrraddiebstahl oder Überspannungschäden. Die Zusatzdeckung werden wir bis zum zweiten Teil des Tutorials zurückstellen und konzentrieren uns bis dahin auf den Hausratvertrag und die Hausratgrunddeckung.

Wir öffnen die Klasse *HausratVertrag* und definieren die Attribute der Klasse:

Name : Datentyp	***Beschreibung, Bemerkung***
plz : String	Postleitzahl des versicherten Hausrats
/tarifzone : String	Die Tarifzone (I, II, III, IV oder V) ergibt sich aus der Postleitzahl und ist maßgeblich für den zu zahlenden Beitrag. => Achten Sie also bei der Eingabe darauf den AttributeType auf derived (computation on each call of the getter method) zu setzen!
wohnflaeche : Integer	Die Wohnfläche des versicherten Hausrats in Quadratmetern. Der erlaubte Wertebereich ist min=0 und unbeschränkt. Den Wertebereich definieren Sie auf der zweiten Seite des Dialogs. Für einen unbeschränkten Wertebereich wird das Feld max auf <null> gesetzt.
/vorschlagVersSumme : Money	Vorschlag für die Versicherungssumme. Wird auf Basis der Wohnfläche bestimmt. => Achten Sie bei der Eingabe darauf den AttributeType auf derived (computation on each call of the getter method) zu setzen!
versSumme : Money	Die Versicherungssumme. Der erlaubte Wertebereich ist min=0 EUR und max=<null>.

Der Editor, der die Klasse *HausratVertrag* anzeigt, sieht nun wie folgt aus:

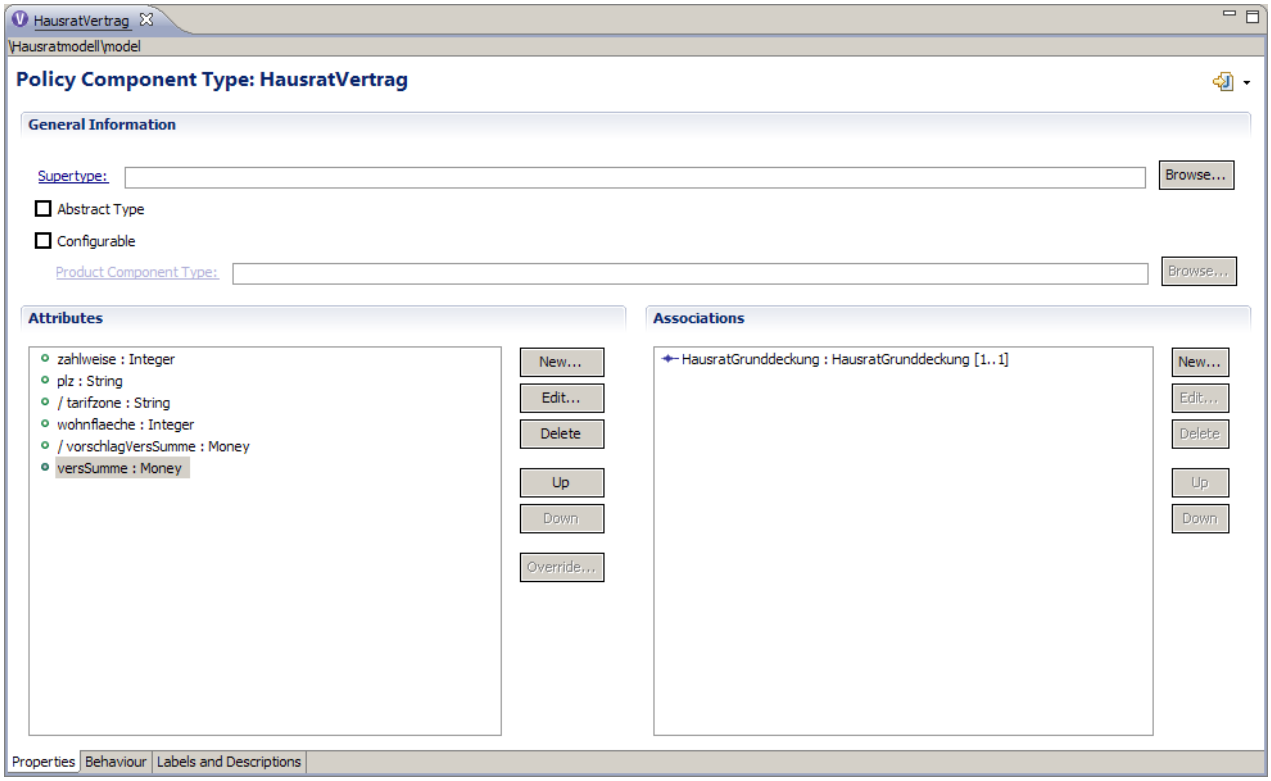


Abbildung 12: Klasse HausratVertrag

Die abgeleiteten Attribute werden UML-konform mit einem vorangestellten Schrägstrich angezeigt.

Öffnen Sie nun die Klasse `HausratVertrag` im Java-Editor und implementieren die Gettermethoden für die beiden abgeleiteten Attribute „tarifzone“ und „vorschlagVersSumme“ wie folgt:

```
/**
 * {@inheritDoc}
 *
 * @generated NOT
 */
public String getTarifzone() {
    return "I";        // TODO wird spaeter anhand einer Tarifzonentabelle ermittelt
}

/**
 * {@inheritDoc}
 *
 * @generated NOT
 */
public Money getVorschlagVersSumme() {
    // TODO: der multitplikator wird spaeter aus den produktdaten ermittelt.
    return Money.euro(650).multiply(wohnflaeche);
}
```

Denken Sie daran, hinter `@generated` ein `NOT` zu schreiben, damit der manuell eingefügte Code nicht überschrieben wird!

Aufnahme von Produktaspekten ins Modell

Nun beschäftigen wir uns – endlich – damit, wie Produktaspekte im Modell abgebildet werden. Bevor wir dies mit Faktor-IPS tun, diskutieren wir das Design auf Modellebene.

Schauen wir uns die bisher definierten Eigenschaften unserer Klasse *HausratVertrag* an und überlegen, was für diese Eigenschaften in einem Produkt konfigurierbar sein soll:

Eigenschaft von Hausratvertrag	***Konfigurationsmöglichkeiten***
zahlweise	Die im Vertrag erlaubten Zahlweisen. Der Vorgabewert für die Zahlweise bei Erzeugung eines neuen Vertrags.
wohnflaeche	Bereich (min, max), in dem die Wohnfläche liegen muss.
vorschlagVersSumme	Definition eines Vorschlagwertes für einen Quadratmeter Wohnfläche. Der Vorschlag für die Versicherungssumme ergibt sich dann durch Multiplikation mit der Wohnfläche[8].
versSumme	Bereich, in dem die Versicherungssumme liegen muss.

Wir wollen zwei Hausratprodukte erstellen. HR-Optimal soll einen umfangreichen Versicherungsschutz gewähren während HR-Kompakt einen Basisschutz zu einem günstigen Beitrag bietet. Die folgende Tabelle zeigt die Eigenschaften der beiden Produkte bzgl. der oben aufgeführten Konfigurationsmöglichkeiten:

Konfigurationsmöglichkeit	***HR-Kompakt***	***HR-Optimal***
Vorgabewert Zahlweise	jährlich	jährlich
Erlaubte Zahlweisen	halbjährlich, jährlich	monatlich, vierteljährlich, halbjährlich, jährlich
Erlaubte Wohnfläche	0-1000 qm	0-2000 qm
Vorschlag Versicherungssumme pro qm Wohnfläche	600 Euro	900 Euro
Versicherungssumme	10Tsd – 2Mio Euro	10Tsd – 5Mio Euro

Wir bilden dies im Modell ab, indem wir eine Klasse *HausratProdukt* einführen. Das Produkt enthält die Eigenschaften und Konfigurationsmöglichkeiten, die bei allen Hausratverträgen, die auf dem gleichen Produkt basieren, identisch sind. Die beiden Produkte HR-Optimal und HR-Kompakt sind Instanzen der Klasse *HausratProdukt*. Das folgende UML Diagramm zeigt das Modell:

8 Alternativ könnten wir auch eine Formel zur Berechnung des Vorschlags als Konfigurationsmöglichkeit verwenden. Zunächst beschränken wir uns aber auf den Faktor.

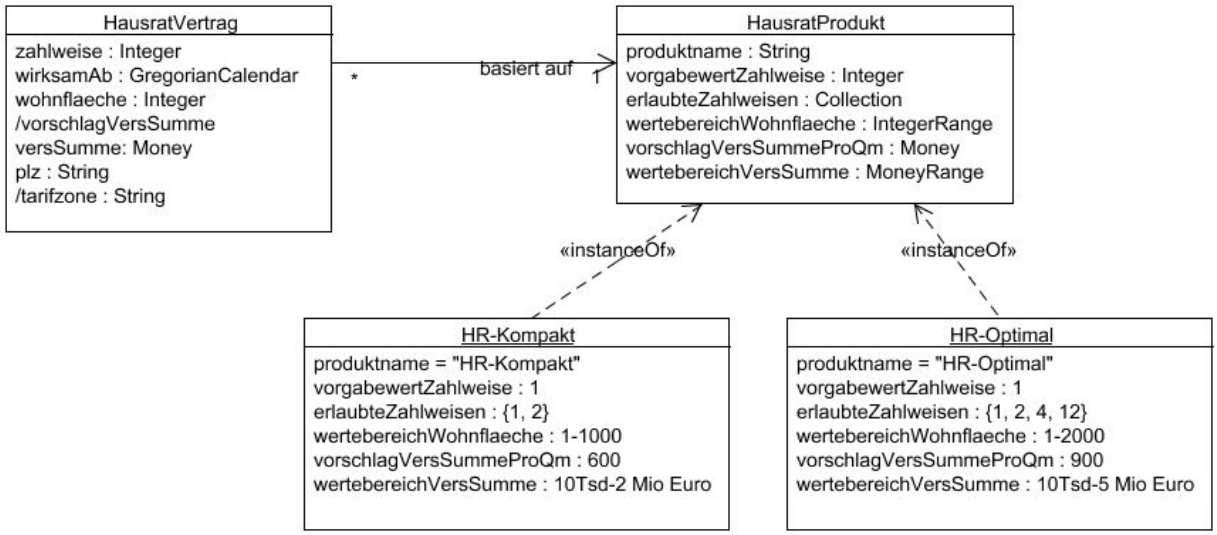


Abbildung 13: Hausratmodell mit Produktklassen ohne Berücksichtigung von Änderungen im Zeitablauf

Nun ändern sich Produkte aber im Laufe der Zeit. Bei Versicherungsprodukten unterscheidet man dabei zwischen zwei Arten der Änderung[9]:

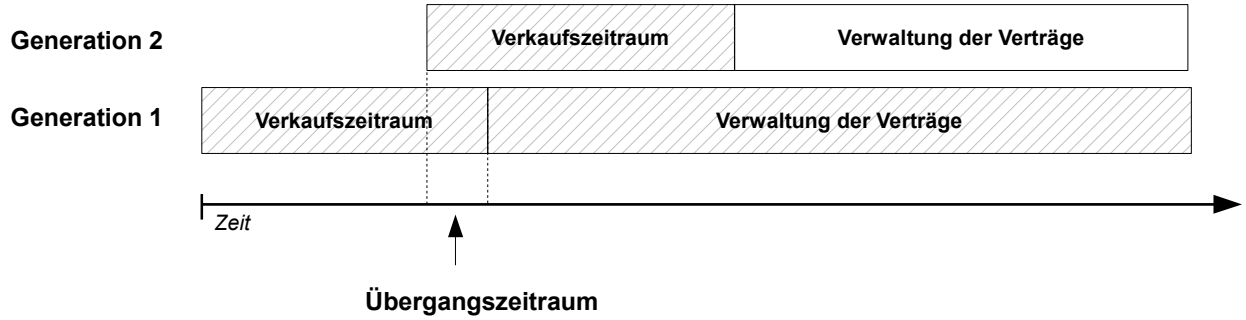
Generation

Generationen werden aufgelegt, um für das Neugeschäft geänderte Bedingungen anbieten zu können. Verträge können während des Verkaufszeitraums einer Generation zu den in der Generation definierten Bedingung abgeschlossen werden.

Bestehende Verträge bleiben von der Einführung einer neuen Generation unberührt, es sei denn, es findet ein expliziter Produkt(Generations)wechsel statt.

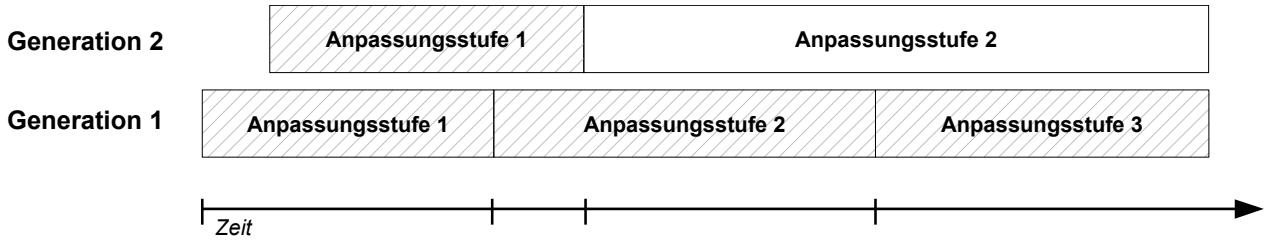
I.d.R. gibt es zu einem Zeitpunkt eine für das Neugeschäft gültige Generation. Das ist die, in deren Verkaufszeitraum der Zeitpunkt fällt. Es kann allerdings zu einem Zeitpunkt auch mehrere verkaufbare Generationen eines Produktes geben. In der Praxis kommt dies vor, wenn in der Übergangszeit von einer alten auf eine neue Generation beide verkauft werden. Dies zeigt die folgende Abbildung:

9 Leider existiert keine einheitliche Namensgebung für die beiden Arten der Produktänderung. Im Tutorial verwenden wir die Begriffe Generation und Anpassungsstufe (Faktor-IPS Schema). In Faktor-IPS kann die verwendete Namensgebung sowohl für die Benutzeroberfläche (Window►Preferences: Faktor-IPS: Naming scheme for changes over time) als auch für den generierten Sourcecode (in der „.ipsproject“ Datei im Abschnitt GeneratedSourcecode) konfiguriert werden.



Anpassungsstufe

Eine Anpassungsstufe ist eine Änderung, die für alle auf Basis einer Generation abgeschlossenen Verträge gelten soll. Innerhalb des Gültigkeitszeitraums einer Anpassungsstufe gibt es keine Änderungen. Zu einem gegebenen Zeitpunkt gibt es genau eine gültige Anpassungsstufe einer ProduktGeneration. Den Zusammenhang zwischen Generationen und Anpassungsstufen zeigt die folgende Abbildung:



Wie berücksichtigen wir die Änderungen im Zeitablauf im Modell? Wir gehen davon aus, dass sich alle Eigenschaften im Zeitablauf ändern können. Die oben modellierten Eigenschaften sind also keine Eigenschaften des Hausratproduktes sondern der ProduktAnpassungsstufe. Die folgende Abbildung zeigt das Modell im Überblick.

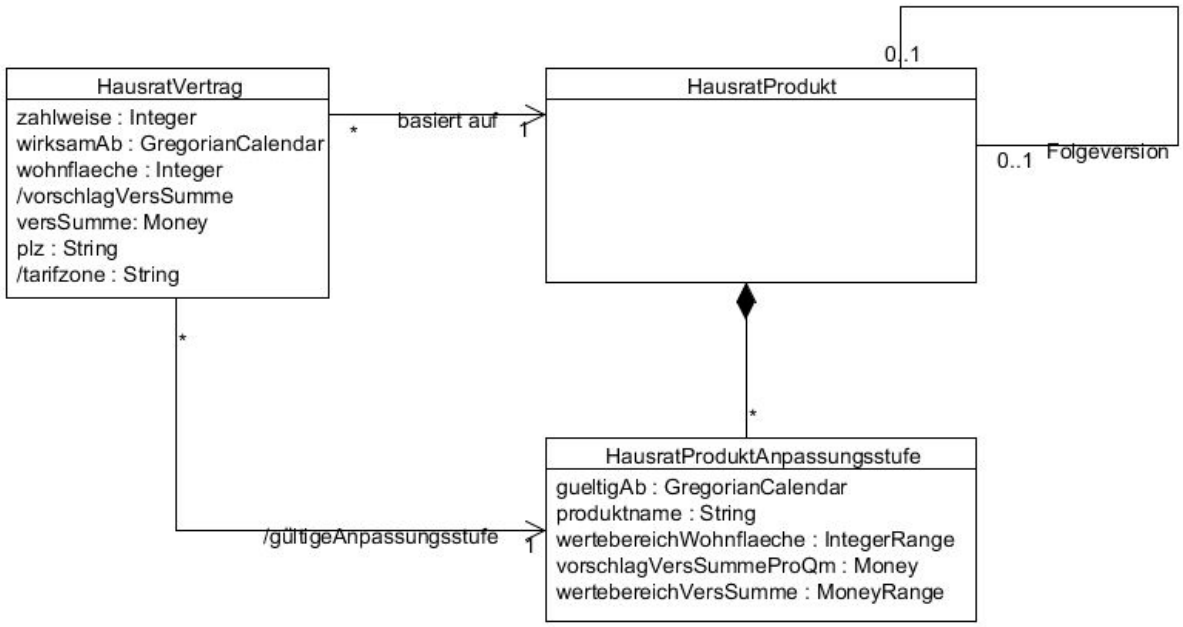


Abbildung 14: Hausratmodell mit Berücksichtigung von Änderungen im Zeitablauf

Das Konzept Produkt-Generation bilden wir über eine Beziehung zwischen Produkten ab, da die Unterscheidung zwischen einem neuen Produkt und einer neuen Generation sehr unscharf ist. Ob zum Beispiel die Umstellung des Kfz-Tarifs auf einen Scoringtarif eine neue Generation oder ein neues Produkt darstellt ist eine eher akademische Betrachtung[10].

Wenn der Vertrag auf Produkteigenschaften zugreifen muss, wird offensichtlich ein Datum benötigt, um die gültige ProduktAnpassungsstufe zu ermitteln. In einem Bestandssystem wird hierzu das Wirksamkeitsdatum, ab dem der Vertragsstand gültig ist, verwendet. In einem Angebotssystem kann auch der Versicherungsbeginn verwendet werden. Wir verwenden in dem Tutorial ein Attribut wirksamAb an der Klasse HausratVertrag. Genau genommen repräsentiert die Klasse HausratVertrag damit eher den Stand eines Vertrags, der von diesem Datum an wirksam ist (bis zum nächsten Vertragsstand)[11].

Genug Theorie, erweitern wir unser Modell in Faktor-IPS um die Produktklassen. Als erstes definieren wir die Klasse *HausratProdukt*. Zum Erzeugen klicken Sie in der Toolbar auf den Button  . In dem Wizard tragen Sie den Namen der neuen Klasse an („HausratProdukt“) und geben im Feld Policy Component Type an, welche Klasse konfiguiert wird, in diesem Fall also *HausratVertrag* und drücken Finish. Es öffnet sich der Editor zur Bearbeitung von Produktklassen.

10 Die „Nachfolge-Generation“ ist nicht auf Instanzen der selben Klasse beschränkt. So kann auch ein strukturell sehr unterschiedliches Produkt (wie z. B. ein neuer Scoringtarif) Nachfolger eines bestehenden Produktes sein.

11 Da der Vertrag selbst lediglich die identifizierende Klammer über die Vertragsstände darstellt und kaum benötigt wird, nennen wir den Vertragsstand verkürzend Vertrag. Sie können dies natürlich auch anders handhaben.

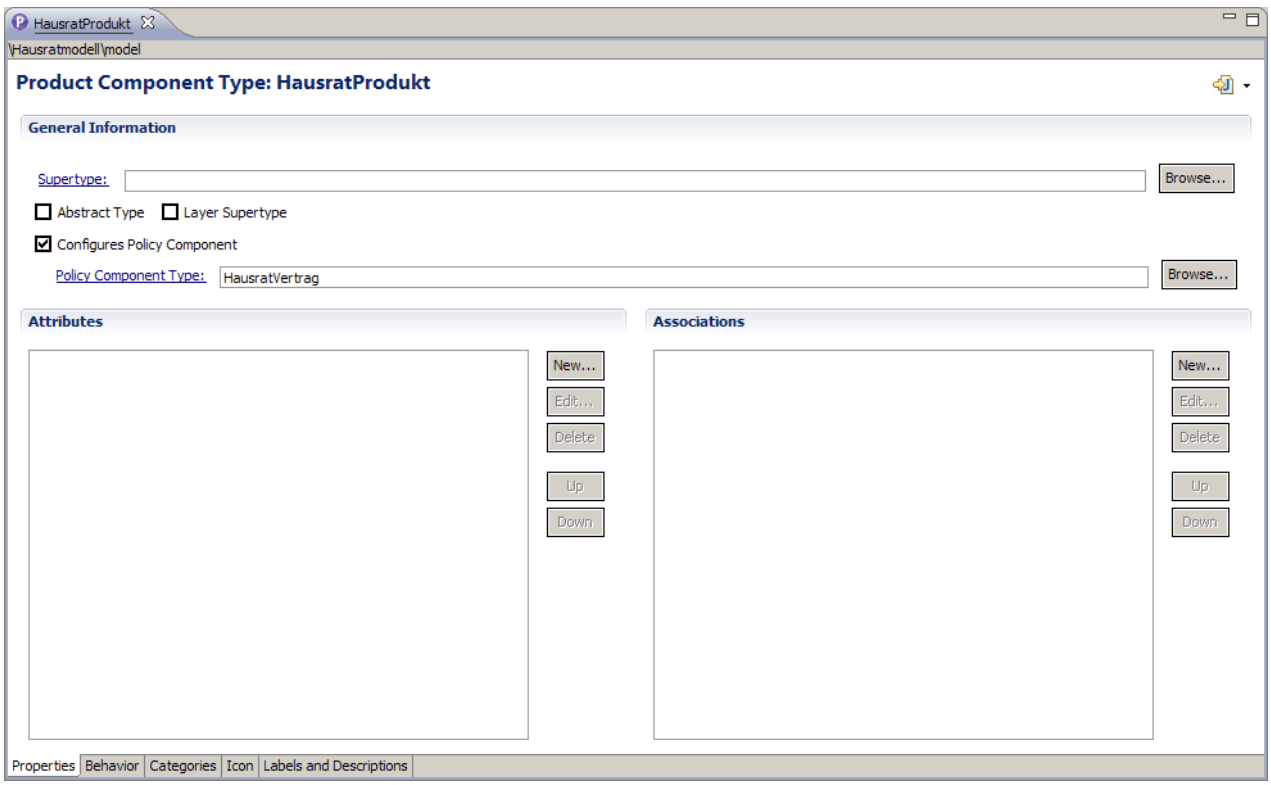


Abbildung 15: Editor für Produktklassen

Im Abschnitt General Information sehen wir die gerade im Wizard eingegebene Information, dass die Klasse *HausratProdukt* die Klasse *HausratVertrag* konfiguriert. Ansonsten ist der Aufbau der ersten Seite des Editors analog zum Editor für Vertragsklassen[12].

Gleichzeitig hat Faktor-IPS nun die published Interfaces `IHausratProdukt` und `IHausratProduktGen` sowie die zugehörigen Implementierungsklassen `HausratProdukt` und `HausratProduktGen` generiert. Im published Interface des Produktes gibt es eine Methode, um zu einem Stichtag die gültige Anpassungsstufe zu ermitteln.

Folgende Aspekte sollen in der Klasse *HausratProdukt* konfigurierbar sein:

- Der Name des Produktes sowie
- die erlaubten Zahlweisen und der Vorbelegungswert für die Zahlweise.

Beginnen wir mit dem Produktnamen. Hierzu legen Sie ein neues Attribut „produktname“ vom Datentyp String an. Dies geschieht analog zum Anlegen eines Attributes für Vertragsklassen. Wie für Attribute von Vertragsklassen können die erlaubten Werte über Bereiche oder Aufzählungen eingeschränkt werden. Für Produktnamen machen wir von dieser Möglichkeit aber keinen Gebrauch. Die folgende Abbildung zeigt den entsprechenden Dialog:

12 Sie können in den Preferences einstellen, ob Sie alle Informationen zu einer Klasse auf einer Seite oder auf zwei Seiten dargestellt bekommen möchten.

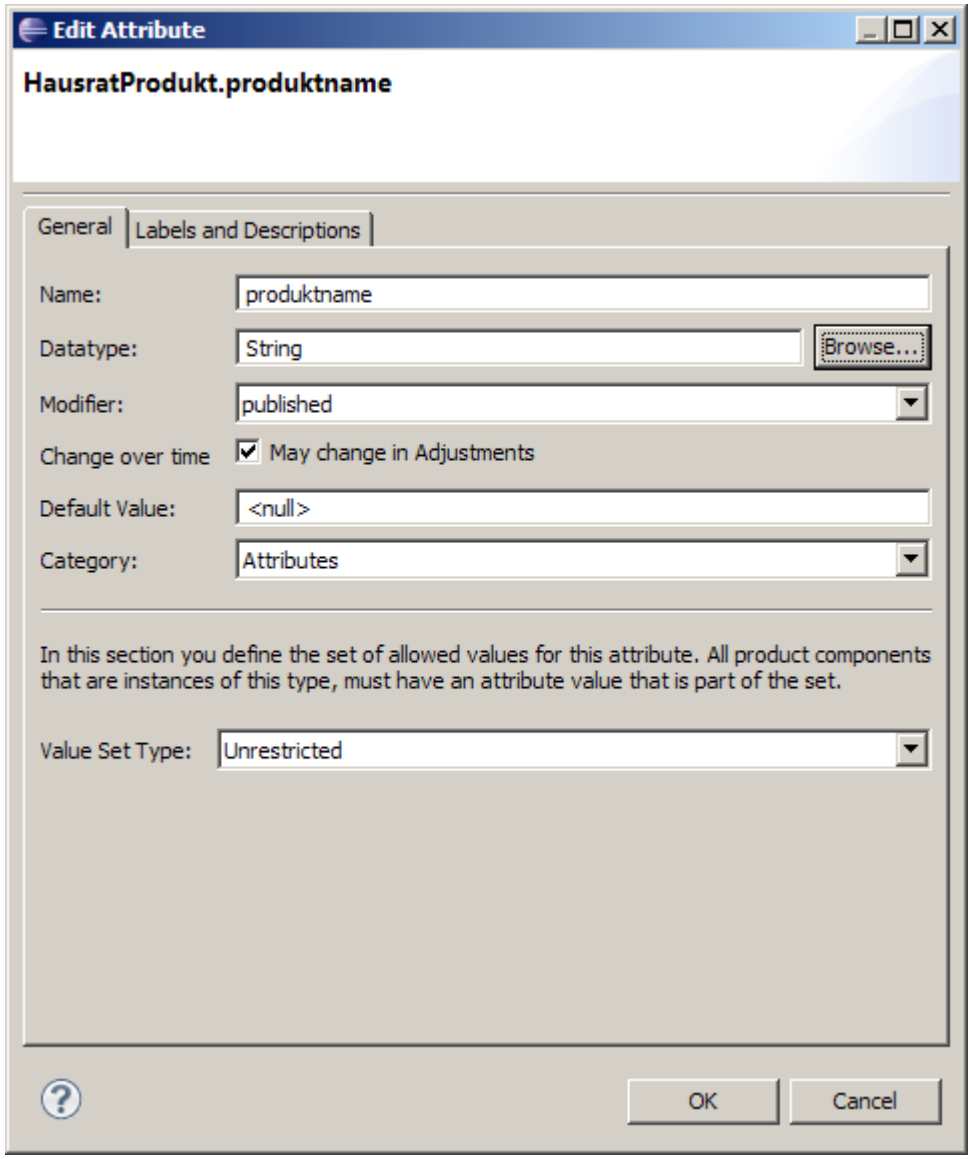


Abbildung 16: Dialog zum Editieren von Produktattributen

Nun definieren wir, dass die in einem Hausratvertrag erlaubten Zahlungsweisen und der Vorgabewert für die Zahlweise im Produkt konfiguriert werden können. Hierzu öffnen wir zunächst den Editor für die Klasse *HausratVertrag*. In den Abschnitt General Information hat der Wizard eingetragen, dass die Klasse *HausratVertrag* durch die Klasse *HausratProdukt* konfiguriert wird.

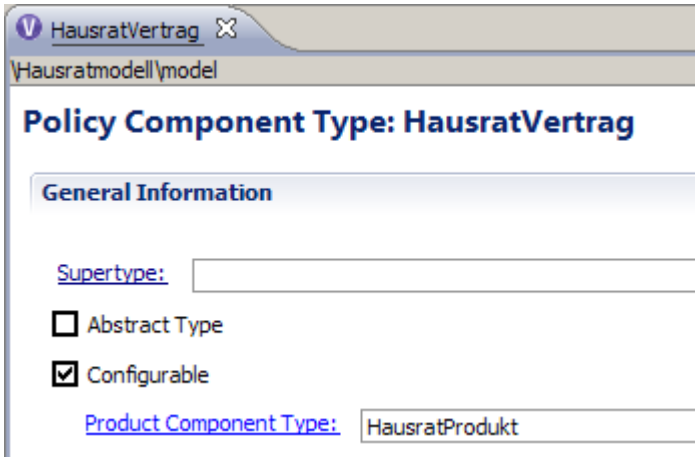


Abbildung 17: General Information Abschnitt im Editor für die Klasse Vertrag

Nun öffnen Sie den Dialog zum Editieren des Attributes „zahlweise“. Da Verträge nun als konfigurierbar definiert sind, gibt es im Dialog nun einen Bereich Configuration. In diesem kann festgelegt werden, ob und wenn ja, wie ein einzelnes Attribut konfigurierbar ist. Die Möglichkeiten unterscheiden sich in Abhängigkeit vom Typ des Attributes.

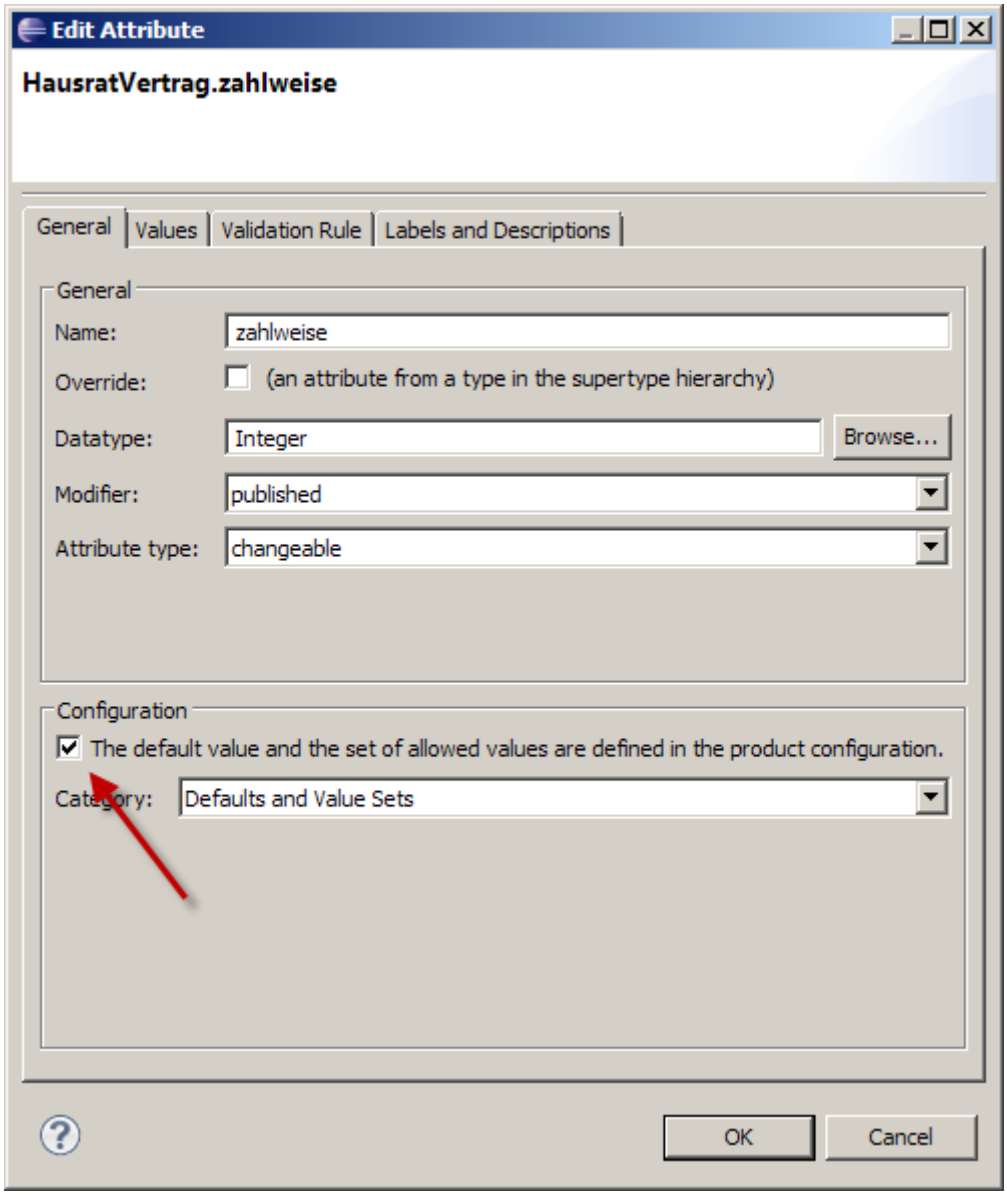


Abbildung 18: Dialog für ein Vertragsklassenattribut mit Konfigurationsmöglichkeit

Um die erlaubten Zahlweisen und den Vorgabewert für die Zahlweisen im Produkt definierten zu können, müssen Sie die entsprechende Checkbox anhaken. Nun schließen Sie den Dialog wieder und speichern.

Werfen wir nun einen Blick in den Sourcecode. Im published Interface der ProduktAnpassungsstufe gibt es jetzt jeweils eine Methode, um den Produktnamen, den Vorgabewert für die Zahlweise und die erlaubten Werte für die Zahlweise abzufragen.

```
public String getProduktname();

public Integer getDefaultZahlweise();

public OrderedValueSet<Integer> getAllowedValuesForZahlweise(String businessFunction);
```

Im published Interface des Vertrags gibt es jetzt Methoden, um auf das Produkt und die ProduktAnpassungsstufe auf der der Vertrag basiert, zuzugreifen.

```
public IHausratProdukt getHausratProdukt();

public void setHausratProdukt(
        IHausratProdukt hausratProdukt,
        boolean initPropertiesWithConfiguratedDefaults);

public IHausratProduktGen getHausratProduktAnpassungsstufe();
```

Nun legen Sie noch das Vertragsattribut „wirksamAb“ mit Datentyp GregorianCalendar an. Dieses Attribut ist nicht produktseitig konfigurierbar. Woher weiß nun die Vertragsklasse, dass sie mit dem Attribut „wirksamAb“ die gültige Anpassungsstufe ermitteln muss? Hierzu hat Faktor-IPS in die Klasse HausratVertrag bereits folgende Methode generiert:

```
/**
 * {@inheritDoc}
 *
 * @generated
 */
public Calendar getEffectiveFromAsCalendar() {
    return null; // TODO Implementieren des Zugriffs auf das Wirksamkeitsdatum.
    // Damit diese Methode bei erneutem Generieren nicht neu ueberschrieben
    // wird, muss ein NOT hinter die Annotation @generated geschrieben werden!
}
```

Diese Methode wird von der Methode `getHausratProduktAnpStufe()` aufgerufen, um den Stichtag zur Ermittlung der Produkt-Anpassungsstufe zu erhalten[13]. Wir geben hier einfach das wirksamAb zurück (und schreiben ein `NOT` hinter die `@generated` Annotation, also

```
/**
 * {@inheritDoc}
 *
 * @generated NOT
 */
public Calendar getEffectiveFromAsCalendar() {
    return wirksamAb;
}
```

Die Implementierung der published Interfaces `IHausratProdukt` und `IHausratProduktAnpStufe` sorgen dafür, dass die Produktdaten zur Laufzeit zur Verfügung stehen. Die Details sprengen allerdings den Rahmen dieses Tutorials.

Die Implementierung der Methode `getEffectiveFromAsCalendar()` ist nur in der Klasse HausratVertrag zu implementieren. In abhängigen Klassen wie z.B. den Deckungen wird standardmäßig die `getEffectiveFromAsCalendar()` Methode der jeweiligen Vaterklasse aufgerufen.

Markieren Sie abschließend noch die Attribute „wohnflaeche“ und „versSumme“ analog zur „zahlweise“ als konfigurierbar.

Nun überarbeiten wir die Berechnung des Vorschlags der Versicherungssumme. In Kapitel 4 hatten wir die Methode `getVorschlagVersSumme()` der Klasse `HausratVertrag` bisher wie folgt

13 Es handelt sich also um eine TemplateMethod (im Sinne der GoF-Patterns), die in der abstrakten Basisklasse PolicyComponent definiert ist.

implementiert:

```
/**
 * {@inheritDoc}
 *
 * @generated NOT
 */
public Money getVorschlagVersSumme() {
    // TODO: der multitplikator wird spaeter aus den produktdaten ermittelt.
    return Money.euro(650).multiply(wohnflaeche);
}
```

Nun wollen wir die Höhe des Multiplikators im Hausratprodukt konfigurieren können. Hierzu legen Sie zunächst an der Klasse *HausratProdukt* ein neues Attribute „vorschlagVersSummeProQm“ vom Datentyp Money an. Dies ist der Vorschlagswert für einen Quadratmeter Wohnfläche. Nach dem Speichern der Klasse HausratProdukt hat Faktor-IPS an der Klasse `HausratProduktAnpStufe` die entsprechende Gettermethode `getVorschlagVersSummeProQm()` generiert. Diese nutzen wir nun in der Berechnung des Vorschlags für die Versicherungssumme. Passen Sie den Sourcecode in der Klasse `HausratVertrag` dazu wie folgt an:

```
/**
 * {@inheritDoc}
 *
 * @generated NOT
 */
public Money getVorschlagVersSumme() {
    IHausratProduktAnpStufe gen = getHausratProduktAnpStufe();
    if (gen==null) {
        return Money.NULL;
    }
    return gen.getVorschlagVersSummeProQm().multiply(wohnflaeche);
}
```

Definieren wir nun noch die Produktseite des Modells für die Grunddeckung. Dazu markieren wir die Klasse *HausratGrunddeckung* als configurable. Die neu anzulegende Produktbausteinklasse nennen wir *HausratGrunddeckungstyp* (s. Abbildung 19). Wir definieren zunächst nur ein Attribut "Bezeichnung" an dieser Klasse. Im weiteren Verlauf des Tutorials werden wir die Klasse erweitern.

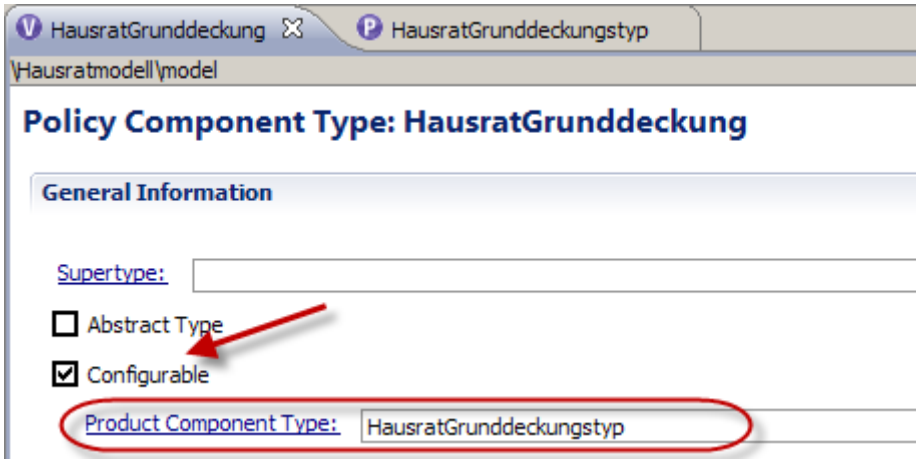


Abbildung 19: HausratGrunddeckung wird durch HausratGrunddeckungstyp konfiguriert

Zum Abschluss dieses Kapitels beschäftigen wir uns noch mit den Beziehungen zwischen den Klassen der Produktseite. Wir wollen hierüber abbilden, welche (Hausrat)Deckungstypen in welchen (Hausrat)Produkten enthalten sind. Das folgende UML Diagramm zeigt das Modell:

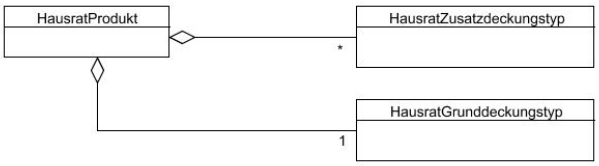


Abbildung 20: Modell der Produktkonfigurationsklassen

Das Hausratprodukt verwendet genau einen Grunddeckungstyp und beliebig viele Zusatzdeckungstypen. Andersherum kann ein Grunddeckungstyp bzw. ein Zusatzdeckungstyp in beliebig vielen Hausratprodukten verwendet werden. Die primäre Navigation ist immer vom Hausratprodukt zum Grund- bzw. Zusatzdeckungstyp, nicht umgekehrt, da ein Deckungstyp immer unabhängig von den Produkten sein sollte, die ihn verwenden.

Definieren wir die Beziehung zwischen *HausratProdukt* und *HausratGrunddeckungstyp* also nun in Faktor-IPS. Öffnen Sie hierzu zunächst den Editor für die Klasse HausratProdukt und legen im Bereich Associations durch klicken auf New eine neue Beziehung an. Es öffnet sich der folgende Dialog, in den Sie die Daten wie hier abgebildet eintragen. Achten Sie darauf die maximale Kardinalität auf eins zu setzen. Den Zusatzdeckungstypen legen wir im Teil 2 des Tutorials an.

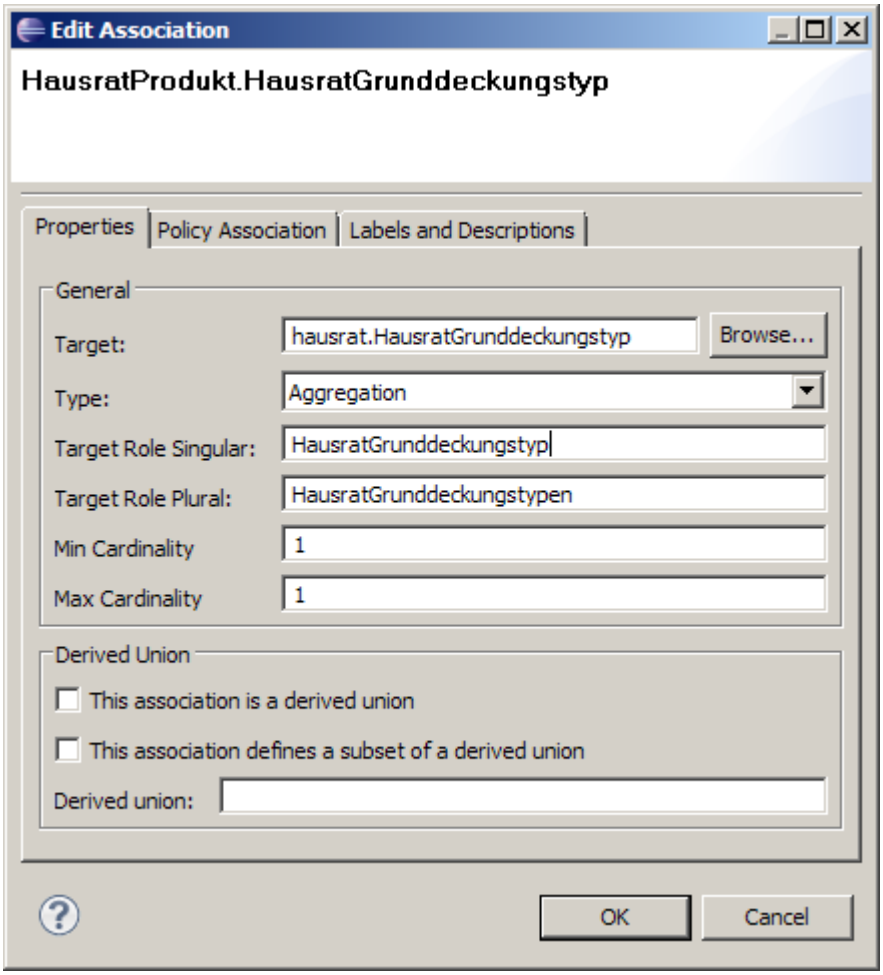


Abbildung 21: Dialog für Beziehungen zwischen Produktklassen

Definition der Produkte

In diesem Kapitel definieren wir nun die Produkte HR-Optimal und HR-Kompakt in Faktor-IPS. Hierzu wird die speziell für den Fachbereich entwickelte Produktdefinitionsperspektive verwendet.

Als erstes richten Sie ein neues Projekt mit dem Namen „*Hausratprodukte*“ und dem Sourceverzeichnis „*produktdaten*“ ein. Dazu wieder ein neues Java-Projekt erzeugen und dann Add IpsNature im Package-Explorer aufrufen. Als Typ wählen Sie diesmal Product definition project, als Packagename „*org.faktorips.tutorial.produktdaten*“ und als Runtime-ID Prefix „*hausrat*.“. Achten Sie darauf, dass der Prefix mit einem Punkt (.) endet. Faktor-IPS erzeugt für jeden neuen Produktbaustein eine Id, mit der der Baustein zur Laufzeit identifiziert wird. Standardmäßig setzt sich diese RuntimeId aus dem Prefix gefolgt von dem (unqualifizierten) Namen zusammen[14]. Zur Laufzeit wird nicht der qualifizierte Name eines Bausteines zur Identifikation verwendet, da die Packagestruktur zur Organisation der Produktdaten zur Entwicklungszeit dient. Auf diese Weise können die Produktdaten umstrukturiert (refactored) werden, ohne dass dies Auswirkungen auf die nutzenden operativen Systeme hat.

Die Verwaltung der Produktdaten in einem eigenen Projekt erfolgt vor dem Hintergrund, dass die Verantwortung für die Produktdaten bei anderen Personen liegt und sie auch einen anderen Releasezyklus haben können. So könnte die Fachabteilung zum Beispiel ein neues Produkt „HR-Flexibel“ erstellen und freigeben, ohne dass das Modell geändert wird. Damit in dem neuen Projekt auf die Klassen des Hausratmodells zugegriffen werden kann, muss in Faktor-IPS im Produktdatenprojekt eine Referenz auf das Hausratmodell-Projekt definiert werden. Das funktioniert in Faktor-IPS analog zur Definition des Buildpath in Java. Der Dialog zur Definition des Faktor-IPS Buildpath ist auch genauso aufgebaut wie der entsprechende Dialog für den Java Buildpath und kann über die Projekteigenschaften aufgerufen werden. Die folgende Abbildung zeigt den Dialog für den Faktor-IPS Buildpath des Produktdatenprojektes mit Referenz auf das Modellprojekt.

Gespeichert wird der Buildpath in der „.ipsproject“ Datei im XML-Element IpsObjectPath. Nach dem Anlegen der Referenz enthält das Element den folgenden neuen Entry.

<Entry type="project" referencedIpsProject="Hausratmodell"/>

Neben Sourceverzeichnissen und Projektreferenzen unterstützt Faktor-IPS (wieder analog zu Java) auch Archive/Bibliothek im Buildpath. Erstellt werden können Faktor-IPS Archive analog zu JARs über einen Export-Wizard.

14 Für die Implementierung eigener Verfahren zur Vergabe der RuntimeId wird ein entsprechender Extension Point bereitgestellt werden.

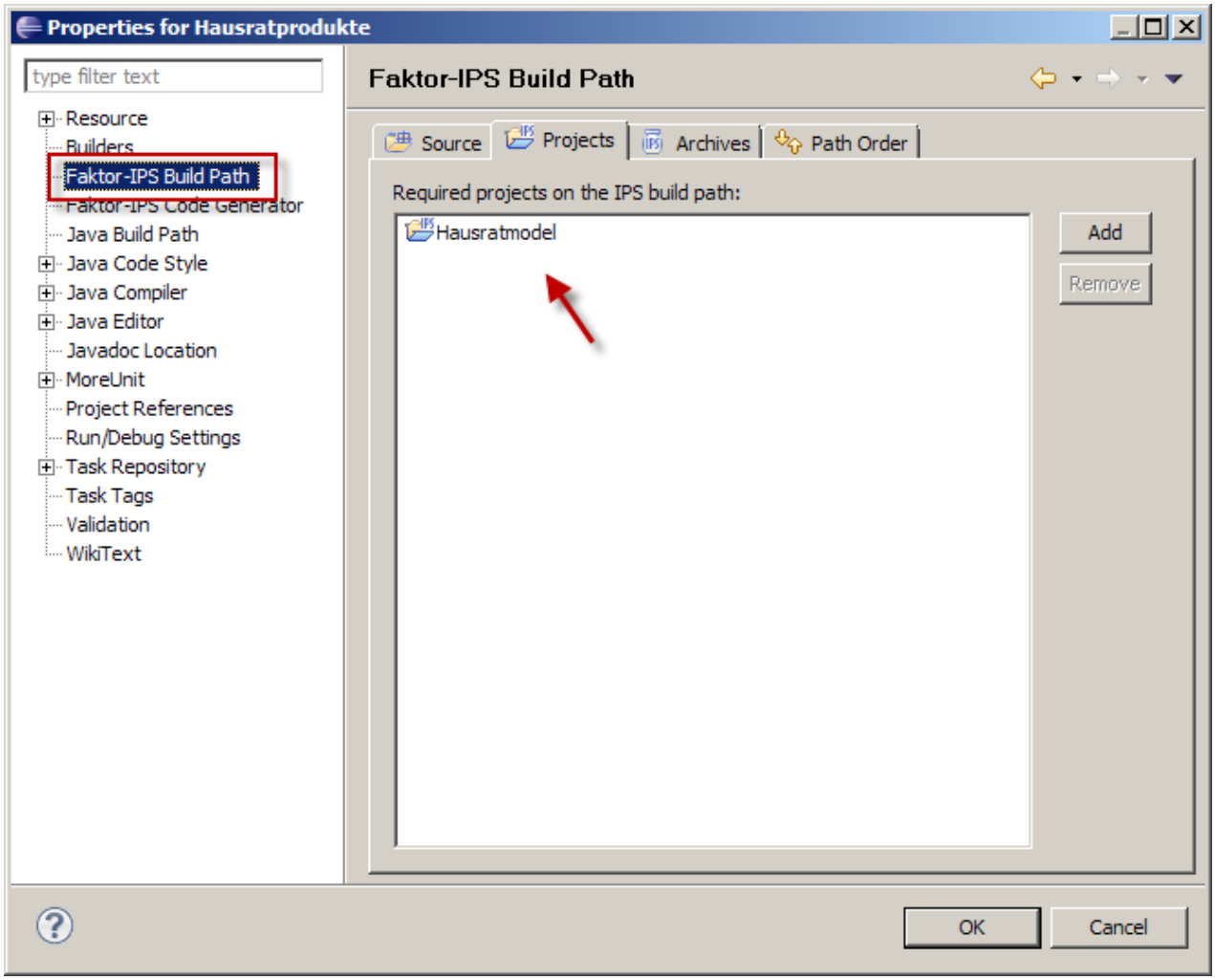


Abbildung 22: Dialog für den Faktor-IPS Buildpath

Damit auch die Javaklassen in dem Projekt verfügbar sind, müssen Sie nun noch den Java-Buildpath um das Java-Projekt Hausratmodell erweitern.

Öffnen Sie nun zunächst die Produktdefinitionsperspektive über Window►Open Perspective►Other, in der Liste dann Produktdefintion auswählen[15]. Falls Sie noch Editoren geöffnet haben, schließen Sie diese jetzt, um die Sichtweise der Fachabteilung auf das System zu haben. Damit Sie im Problemsview ausschließlich die Marker von Faktor-IPS sehen (und nicht auch Java-Marker u.a.) müssen Sie im Problemsview den Faktor-IPS Filter ein- und alle anderen Filter (standardmäßig mindestens der Defaultfilter) ausschalten.

Zunächst legen wir zwei IPS Packages an, einen für die Produkte und einen für die Deckungen. Dies geschieht wie in der Java Perspektive entweder über das Kontextmenü oder die Toolbar.

Sie können in dem Projekt im übrigen auch beliebige andere Verzeichnisse anlegen, zum Beispiel ein doc-Verzeichnis, um Dokumente zu den Produkten zu verwalten.

15 Für die Verwendung von Faktor-IPS durch die Fachabteilung gibt es auch eine eigene Installation (In Eclipse Terminologie: ein eigenes Produkt), bei der ausschließlich die Produktdefinitionsperspektive verfügbar ist.

Unsere Produkte sollen ab dem nächsten Quartalsbeginn verfügbar sein. Bevor wir nun also mit dem Anlegen der Produkte beginnen, setzen wir noch das Wirksamkeitsdatum, mit dem wir arbeiten wollen. Klicken Sie hierzu in der Toolbar auf , geben den nächsten Quartalsbeginn ein und drücken auf OK. Alle Änderungen werden von nun an mit diesem Wirksamkeitsdatum durchgeführt.

Als erstes legen wir jetzt das Produkt HR-Optimal an. Markieren Sie dazu das gerade angelegte Package „`produkte`" und klicken dann in der Toolbar auf . Es öffnet sich der Wizard zum Erzeugen eines neuen Produktbausteins. Der Wizard bietet Ihnen nun die im Modell verfügbaren Produktklassen zur Auswahl, zu denen Sie Produktbausteine erstellen können. Wählen Sie HausratProdukt.

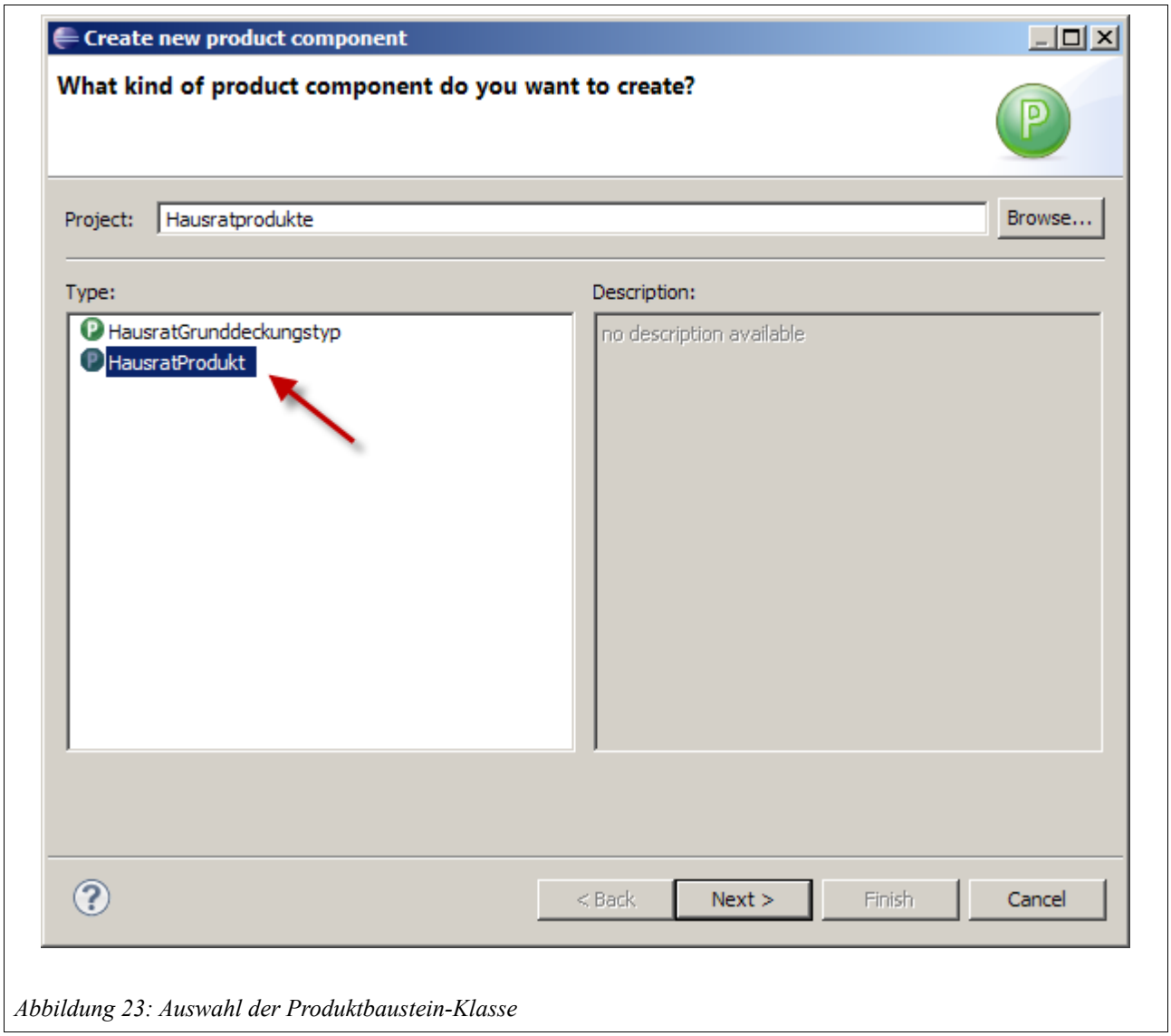


Abbildung 23: Auswahl der Produktbaustein-Klasse

Falls Sie keine Produktklasse finden, fehlt noch die Referenz auf das Hausratmodell-Projekt im Faktor-IPS Buildpath (s.o.). Gehen sie mit Next > zur nächsten Seite des Wizards.

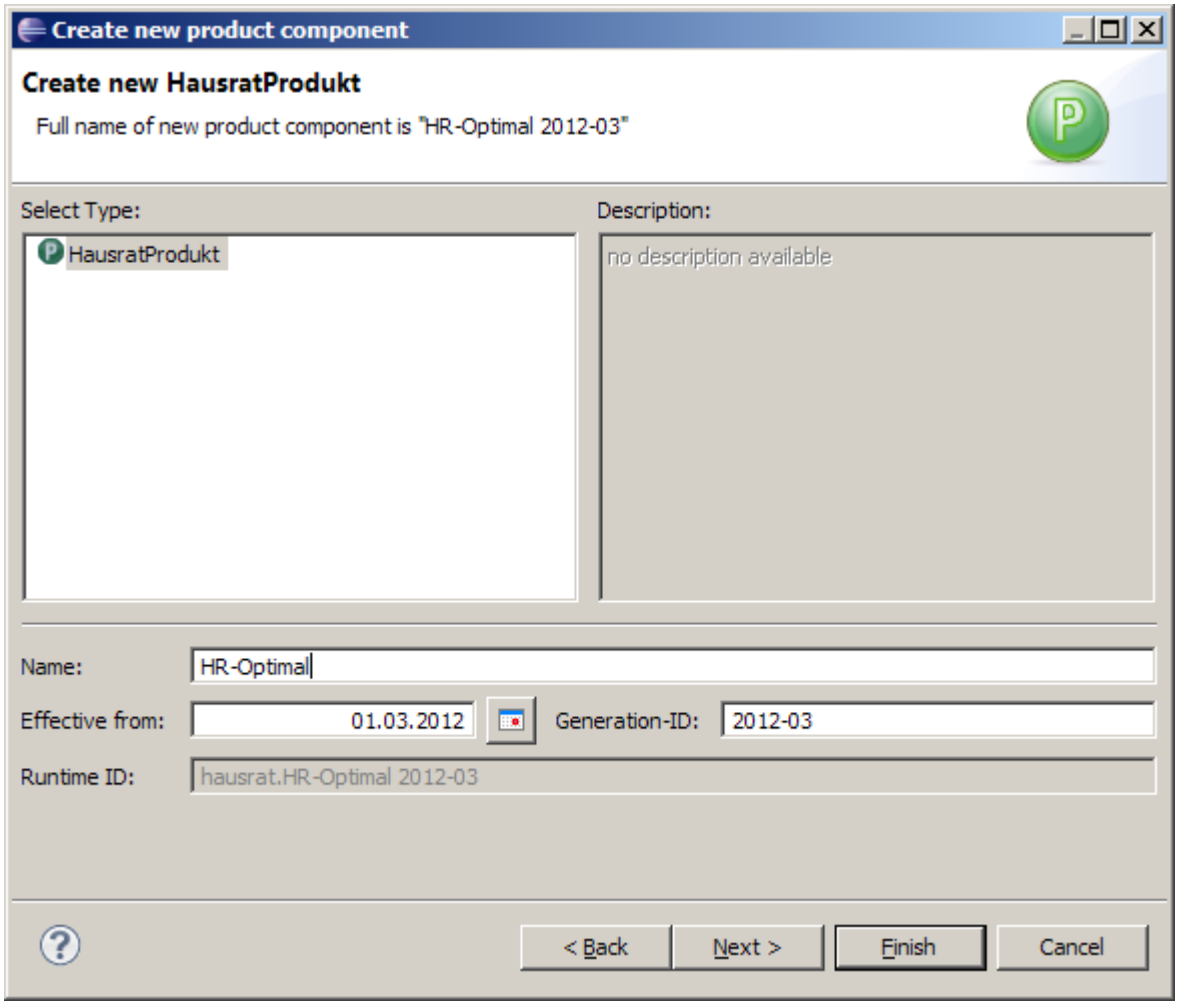


Abbildung 24: Anlegen eines neuen Produkts

Hier geben Sie den Namen des Bausteins ein. Die Generations-ID des Bausteins wird anhand des Wirksamkeitsdatum vorbelegt. Das Format der Generationsnummer kann in der „.ipsproject“-Datei definiert werden. Standardmäßig ist es „JJJJ-MM“ mit einem zusätzlichen optionalen Postfix, also z. B. „2012-03b“. Ebenfalls wird die Runtime ID anhand des im Projekt definierten Prefixes und dem unqualifizierten Namen vorbelegt.

Standardmäßig ist diese Vorbelegung nicht änderbar. Dies kann aber in den Einstellungen (Window►Preferences...►Can modify runtime id) geändert werden.

Wenn Sie nun Finish drücken wird der Produktbaustein im Dateisystem angelegt. Mit Doppelklick auf den Produktbaustein im Produkt Struktur Explorer öffnen Sie den Editor für den Baustein.

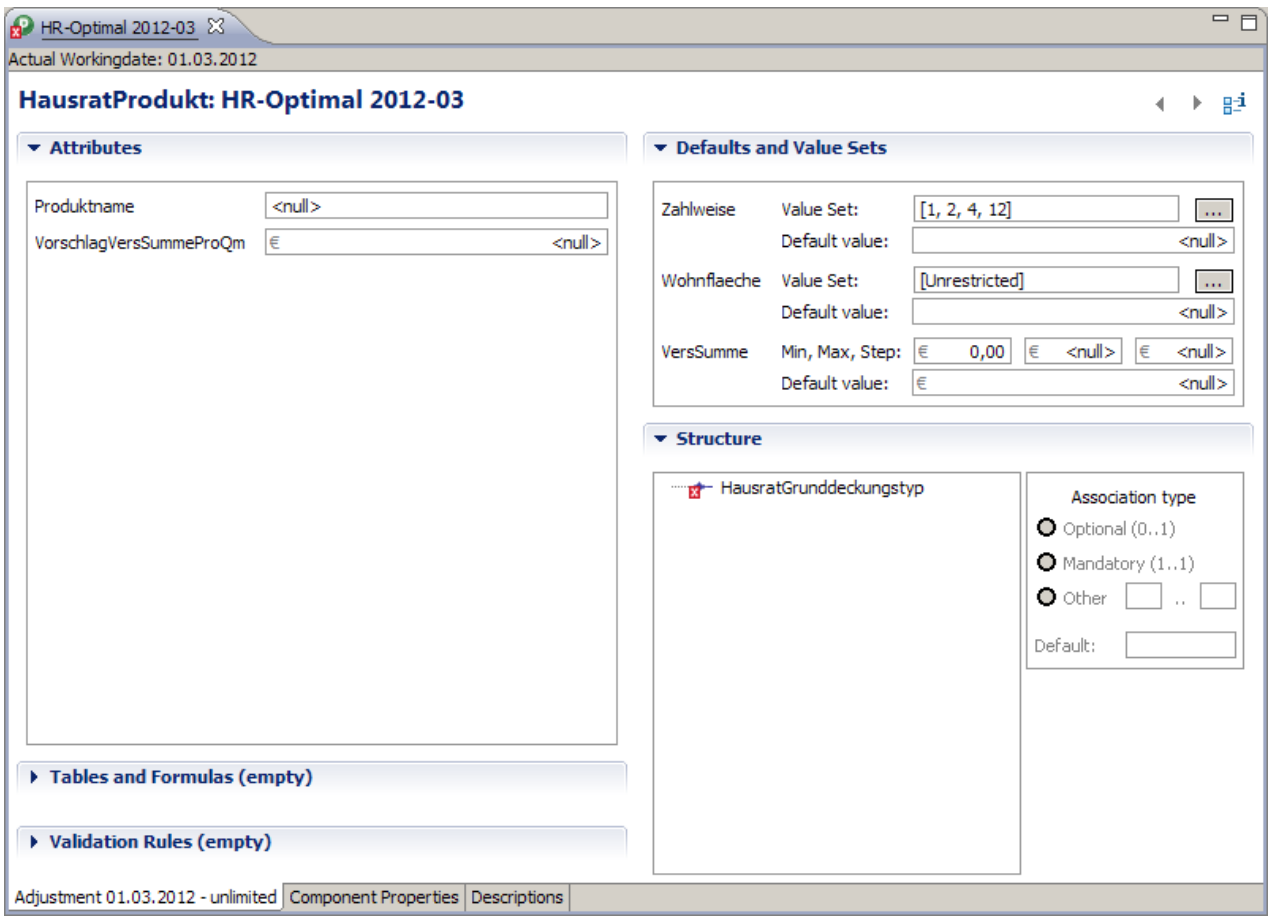


Abbildung 25: Produktbaustein-Editor für HR-Optimal

Die erste Seite des Editors zeigt die in Bearbeitung befindliche Anpassungsstufe des Produktbausteins. Nach der Anlage hat ein Baustein immer eine Anpassungsstufe, die ab dem eingestellten Wirksamkeitsdatum gültig ist. Die Seite ist standardmäßig in fünf Bereich eingeteilt [16]:

Attributes

Enthält die Eigenschaften der Anpassungsstufe. Hier ist jedes in der Produktklasse definierte Attribute aufgelistet.

Tables and Formulas

Enthält die Berechnungsvorschriften und Referenzen auf Tabellen. Hierzu mehr bei der Implementierung der Beitagsberechnung in Teil 2.

Validation Rules

Enthält die im Modell als konfigurierbar gekennzeichneten Valdierungsregeln.

Defaults and Value Sets

Enthält die Vorbelegungswerte und Wertebereiche für die Vertragseigenschaften.

16 Mit Faktor-IPS 3.6 wurde die Möglichkeit eingeführt, den Seitenaufbau zu konfigurieren und so die Produktbaustein-Eigenschaften nach fachlichen Kriterien auf der Oberfläche zu gruppieren.

Structure
Enthält die verwendeten anderen Produktbausteine.

Geben Sie also nun die Daten für das Produkt HR-Optimal entsprechend der folgenden Tabelle ein:

Konfigurationsmöglichkeit	***HR-Optimal***
Produktname	Hausrat Optimal
Vorschlag Versicherungsumme pro qm Wohnfläche	900EUR
Vorgabewert Zahlweise	1 (jährlich)
Erlaubte Zahlweisen	1, 2, 4, 12
Vorgabewert Wohnflaeche	<null>
Erlaubte Wohnflaeche	0-2000
Vorgabewert Versicherungssumme	<null>
Versicherungssumme	10000EUR – 5000000EUR

Nun legen wir den Grunddeckungstyp für das Produkt an. Markieren Sie hierzu den Ordner Deckungen und legen einen neuen Produktbaustein mit Namen „HRD-Grunddeckung-Optimal“ an basierend auf der Klasse *HausratGrunddeckungstyp*.

Nun müssen wir noch den Deckungstyp dem Produkt HR-Optimal zuordnen. Dies kann man bequem per Drag&Drop aus dem Model-Explorer erledigen. Öffnen Sie das Produkt *HR-Optimal.* Ziehen Sie die *HRD-Grunddeckung-Optimal* aus dem Produktdefinitions-Explorer auf den Knoten *HausratGrundeckungstyp* im Bereich Structure.

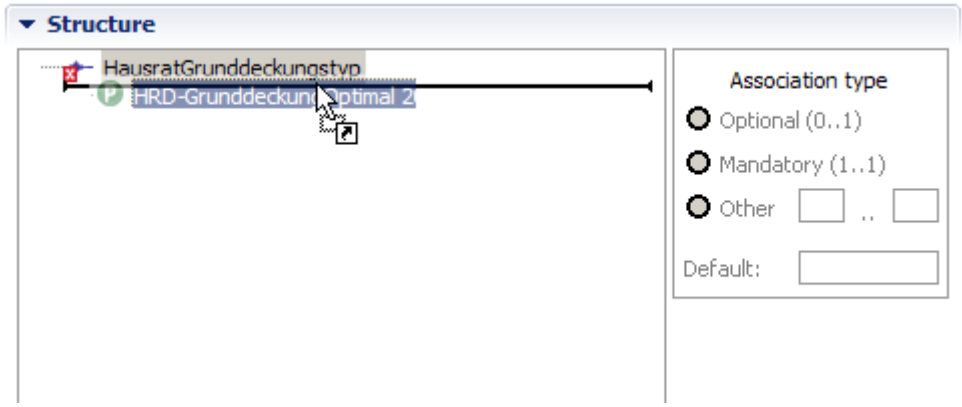


Abbildung 26: Zuordnung von Deckungstypen zu einem Produkt

Nun legen wir noch das Produkt *HR-Kompakt* inklusive der Grunddeckung *HRD-Kompakt* an. Dies können Sie analog zum Produkt *HR-Optimal* machen. Alternativ können Sie einen Kopierassistenten verwenden, mit dem Sie einen Produktbaustein inklusive aller verwendeter Bausteine kopieren können. Wenn Sie dies ausprobieren möchten markieren Sie das Produkt *HR-Optimal* im Produktdefinitions-Explorer und wählen im Kontextmenü New►Copy Product ... Auf der ersten Seite geben Sie als Search Pattern „Optimal“ und als Replace Pattern „Kompakt“ ein, klicken Next und dann Finish. Faktor-IPS legt die Bausteine *HR-Kompakt* und *HRD-Grunddeckung*

neu an. In der Praxis wird der Kopierassistent i.d.R. zum Anlegen neuer Generationen verwendet, der Aufruf erfolgt im Kontextmenü über New►Create New Generation ...

Öffnen Sie nun das neue Produkt und geben seine Daten ein.

Konfigurationsmöglichkeit	***HR-Kompakt***
Produktname	Hausrat Kompakt
Vorschlag Versicherungsumme pro qm Wohnfläche	600EUR
Vorgabewert Zahlweise	jährlich
Erlaubte Zahlweisen	halbjährlich, jährlich
Vorgabewert Wohnflaeche	<null>
Erlaubte Wohnflaeche	0-1000 qm
Vorgabewert Versicherungssumme	<null>
Versicherungssumme	10000EUR – 2000000EUR

Damit ist die Definition der beiden Produkte zunächst abgeschlossen. Im Produktdefinitions-Explorer sollten Sie wie nun folgt angezeigt werden.

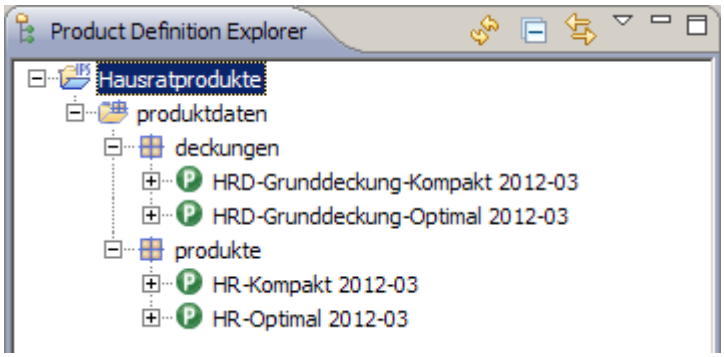


Abbildung 27: Darstellung der Produkte im Produktdefinitions-Explorer

Neben dem Produktdefinitionsexplorer stehen ihnen zwei weitere Werkzeuge zur Analyse der Produktdefinition zur Verfügung. Die Struktur eines Produktes können Sie sich mit Show Structure im Kontextmenü anzeigen lassen. Die unterschiedlichen Verwendung eines Bausteins mit Search References. Darüber hinaus können Sie die Reihenfolge der Pakete über den Menüpunkt Edit Sort Order frei festlegen.

Rechts neben dem Editor zur Eingabe der Produktdaten befindet sich der Model Description View. Dieser zeigt passend zum in Bearbeitung befindlichen Produktbaustein die Dokumentation der zugehörigen Produktklasse. Wenn Sie die einmal ausprobieren wollen, dokumentieren Sie z. B. das Attribut „produktname“ im Modell, schließen Sie den Bausteineditor und öffnen ihn erneut.

Zugriff auf Produktinformationen zur Laufzeit

Nachdem wir die Produktdaten erfasst haben, beschäftigen wir uns nun damit, wie man zur Laufzeit (in einer Anwendung/einem Testfall) auf diese zugreift. Hierzu werden wir einen JUnit-Test schreiben, den wir im Laufe des Tutorials weiter ausbauen.

Zum Zugriff auf Produktdaten stellt Faktor-IPS das Interface `IRuntimeRepository` bereit. Die Implementierung `ClassloaderRuntimeRepository` erlaubt den Zugriff auf die mit Faktor-IPS erfassten Produktdaten und lädt die Daten über einen Classloader. Damit dies möglich ist, macht Faktor-IPS zwei Dinge:

1. Die Dateien, die die Produktinformationen enthalten, werden in den Java Sourcefolder mit dem Namen „derived“ kopiert. Damit sind diese Dateien im Buildpath des Projektes enthalten und können über den Classloader geladen werden.
2. Welche Daten sich im ClassloaderRuntimeRepository befinden, ist in einem Inhaltsverzeichnis vermerkt. Dieses Inhaltsverzeichnis (Englisch: table of contents, toc) wird von Faktor-IPS ebenfalls in eine Datei generiert, die als Toc-File bezeichnet wird. Die Datei heißt standardmäßig „`faktorips-repository-toc.xml`“[17].

Der folgende Screenshot zeigt den Inhalt des Sourcefolders „derived“ im Package-Explorer.

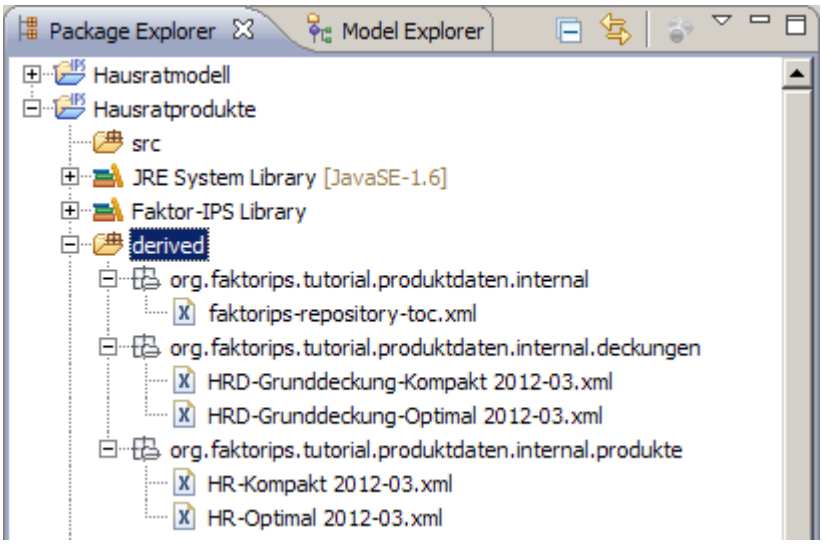


Abbildung 28: Inhalt des Sourcefolders „derived“

Ein ClassloaderRuntimeRepository wird über die statische `create(...)` Methode der Klasse erzeugt. Als Parameter wird der Pfad zum Toc-File übergeben. Das Toc-File wird direkt beim Erzeugen des Repositorys über `Classloader.getResourceAsStream()` gelesen. Alle weiteren Daten werden erst (wiederum über den Classloader) geladen, wenn auf sie zugegriffen wird.

Das Laden der Daten über den Classloader hat im Gegensatz zum Laden aus dem Filesystem den großen Vorteil, dass es völlig plattformunabhängig ist. So kann der Programmcode z. B. ohne Änderungen auf z/OS laufen.

17 Die Namen lassen sich in der „.ipsproject“ Datei im Abschnitt IpsObjectPath konfigurieren.

Einen Produktbaustein kann man über die Methode `getProductComponent(...)` erhalten. Als Parameter übergibt man die RuntimeId des Bausteins. Da das Interface `IRuntimeRepository` unabhängig vom konkreten Modell (in unserem Fall also dem Hausratmodell) ist, muss man das Ergebnis noch auf die konkrete Produktklasse casten.

Probieren wir dies doch einmal in einem JUnit Testfall aus. Legen Sie hierzu zunächst im Projekt Hausratprodukte einen neuen Java Sourcefolder „test“ an. Am einfachsten geht dies, indem Sie im Package-Explorer das Projekt markieren und im Kontextmenü Buildpath►New Source Folder... aufrufen.

Danach markieren Sie den neuen Sourcefolder und legen einen JUnit Testfall an, indem Sie in der Toolbar auf  klicken und dann JUnit Test Case auswählen. In dem Dialog geben Sie als Namen für die Testfallklasse „TutorialTest“ ein und haken an, dass auch die `setUp()` Methode generiert werden soll. Die Warnung, dass die Verwendung des Defaultpackages nicht geraten wird, ignorieren wir in dem Tutorial. Im unteren Teil des Dialogs gibt es einen Link, mit dem Sie die benötigte JUnit Library zum Java Buildpath des Projektes hinzufügen können. Der nächste Kasten enthält den Sourcecode der Testfallklasse.

Anstatt die Korrektheit der Produktdaten mit `assert*`-Statements zu testen, geben wir sie hier mit

```
public class TutorialTest extends TestCase {

    private IRuntimeRepository repository;
    private IHausratProdukt kompaktProdukt;
    private IHausratProduktAnpStufe kompaktAnpStufe;

    public void setUp() {
      // Repository erzeugen
      repository = ClassloaderRuntimeRepository.create(
          "org/faktorips/tutorial/produktdaten/internal/faktorips-repository-toc.xml");

      // Referenz auf das Kompaktprodukt aus dem Repository holen
      IProductComponent pc = repository.getProductComponent("hausrat.HR-Kompakt 2012-03");

      // Juengste ProductAnpassungsstufe holen (wir wissen es gibt nur eine).
      IProductComponentGeneration pcGen = pc.getLatestProductComponentGeneration();

      // Auf die eigenen Modellklassen casten
      kompaktProdukt = (IHausratProdukt) pc; // wird im weiteren Verlauf benoetigt.
      kompaktAnpStufe  = (IHausratProduktAnpStufe) pcGen;
    }

    public void testProduktdatenLesen() {
        System.out.println("Produktname: " + kompaktAnpStufe.getProduktname());
        System.out.println("Vorschlag Vs pro 1qm: "
               +kompaktAnpStufe.getVorschlagVersSummeProQm());
        System.out.println("Default Zahlweise  : " + kompaktAnpStufe.getDefaultValueZahlweise());
        System.out.println("Erlaubte Zahlweisen: "
               +kompaktAnpStufe.getAllowedValuesForZahlweise(null));
        System.out.println("Default Vs: " + kompaktAnpStufe.getDefaultValueVersSumme());
        System.out.println("Bereich Vs: " + kompaktAnpStufe.getRangeForVersSumme(null));
        System.out.println("Default Wohnflaeche: "
               +kompaktAnpStufe.getDefaultValueWohnflaeche());
        System.out.println("Bereich Wohnflaeche: "
               +kompaktAnpStufe.getRangeForWohnflaeche(null));
      }
}
```

`println` auf der Konsole aus. Wenn Sie den Test nun ausführen, sollte er folgendes ausgeben:

```
Produktname: Hausrat Kompakt
Vorschlag Vs pro 1qm: 600.00 EUR
Default Zahlweise  : 1
Erlaubte Zahlweisen: [1, 2]
Default Vs: MoneyNull
```

```
Bereich Vs: 10000.00 EUR-2000000.00 EUR
Default Wohnflaeche: null
Bereich Wohnflaeche: 0-1000
```

Damit haben wir einen Einblick in die Modellierung mit Faktor-IPS bekommen, haben erste, einfache Produkte angelegt und zur Laufzeit auf Produktdaten zugegriffen.

In Teil II dieses Tutorials werden wir in die Verwendung von Tabellen & Formeln einführen. Diese werden wir nutzen, um das Hausratmodell so zu erweitern, dass den Produkten flexibel Zusatzdeckungen durch Anwender aus der Fachabteilung hinzu konfiguriert werden können, ohne dass das Modell erweitert werden muss.