Schnittstellen-Design für Produktserver

Jan Ortmann
(Dokumentversion 1610)

Einleitung

Ein Produktserver (Produkt-Engine, Produkt-Komponente) stellt operativen Systemen wie Bestandssystemen & Vertriebssystemen Produktinformationen und andere produktbezogene Services wie z. B. die Beitragsberechnung oder Prüfungen zur Verfügung. Die Definition der Versicherungsprodukte erfolgt über ein eigenes Produktdefinitionssystem.

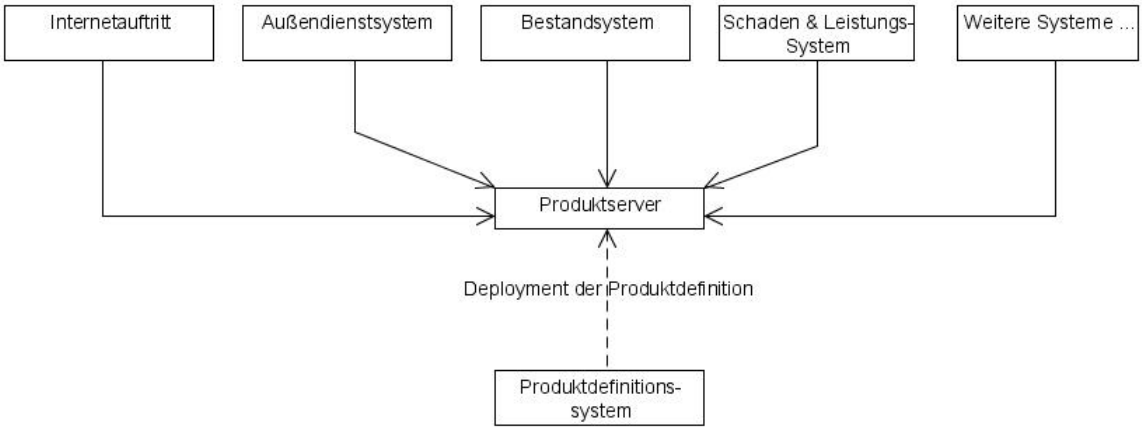

Abbildung 1: Produktserver & Produktdefinitionssystem

In diesem Artikel wird erläutert welche Varianten es bei der Konstruktion der Schnittstellen des Produktservers gibt und welche Vor- und Nachteile diese haben. Es wird hierbei davon ausgegangen, dass die Struktur der Verträge und Produkte im Produktserver anhand eines fachlichen Klassenmodells abgebildet ist. Angebote werden als angebotene Verträge betrachtet und besitzen somit die gleiche Struktur wie Verträge.

Bei den Beispielen gehen wir davon aus, dass der Produktserver seine Services über Stateless Session Beans zur Verfügung stellt. Die Services sollen auch über Remote Procedure Calls nutzbar sein. Aus diesem Grund werden an der Schnittstelle Data Transfer Objects (DTOs) verwendet [Fowler, PoEAA]. Die diskutierten Konstruktionsprinzipien sind aber unabhängig von der eingesetzten Technologie und können problemlos übertragen werden.

Strukturgleiche vs. flache Schnittstellen

Viele der Services eines Produktservers benötigen Informationen über einen Vertrag oder ein Angebot. Beispiele sind die Beitragsberechnung und die Annahmeprüfung. Die Methodensignaturen hierfür könnten zum Beispiel wie folgt aussehen:

```
public VertragDto berechneBeitrag(VertragDto vertrag);

public MessageList istAnnehmbar(VertragDto vertrag);
```

Die Methode zur Beitragsberechnung erhält als Parameter einen Vertrag, bei dem die Beitragsattribute (Nettobeitrag, Bruttobeitrag, etc.) nicht gefüllt sind und gibt wiederum einen Vertrag zurück, bei dem diese Attribute die berechneten Werte enthalten.

Es gibt zwei grundlegende Arten die Schnittstellen-Parameter einer Produktkomponente zu entwerfen:

- Schnittstellen-Parameter sind strukturgleich zum Modell
- Flache Schnittstellen-Parameter

Zwischen diesen beiden Extremen gibt es im konkreten Fall natürlich viele Abstufungen.

Die beiden Varianten lassen sich am besten an einem Beispiel erläutern. Die folgende Abbildung zeigt das verwendete fachliche Modell.

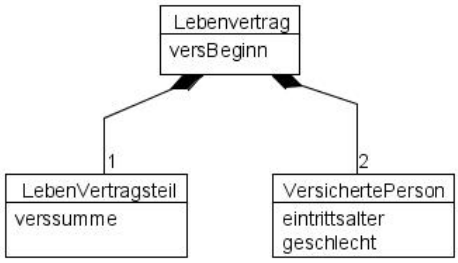

Abbildung 2: Beispielmodell 1

Eine strukturgleiche Schnittstelle würde – wie der Name schon sagt – die „gleichen“ Klassen mit den gleichen Attributen und Beziehungen haben (natürlich ohne fachliche Methoden). Die strukturgleichen Schnittstellenklassen werden dabei i.d.R. die Struktur des Fachmodells vollständig abbilden, also alle Klassen, Attribute und Beziehungen enthalten.

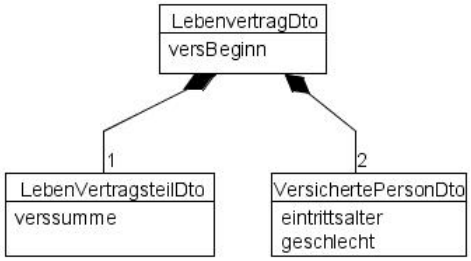

Abbildung 3: Strukturgleiches Schnittstellenmodell

Eine flache Schnittstelle für eine Beitragsberechnung könnte dagegen aus einer einzigen Klasse `LebenvertragDto` mit den Attributen aller Klassen des Fachmodells bestehen. Die zwei versicherten Personen würden durch ein Präfix an den Attributen abgebildet werden, also etwa eintrittsalterVP1 und eintrittsalterVP2.

Abbildung 4: Flaches Schnittstellenmodell

Flexible vs. Produkt-spezifische Schnittstellen

Zur Diskussion verwenden wir folgendes Modell als Beispiel.

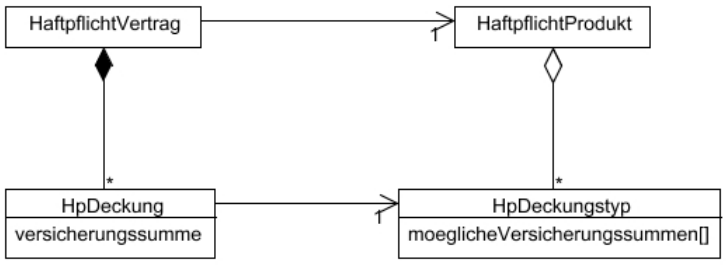

Abbildung 5: Beispielmodell 2

Ein Haftpflichtvertrag kann beliebig viele Deckungen enthalten. Analog enthält auf Produktseite das Haftpflichtprodukt beliebig viele Deckungstypen. Pro Deckungstyp kann der Vertrag maximal eine Deckung beinhalten. Jede Deckung hat eine Versicherungssumme. Die in einer Deckung erlaubten Versicherungssummen sind in dem zugehörigen Deckungstyp definiert.

Auf Basis des Modells ist folgendes Produkt definiert.

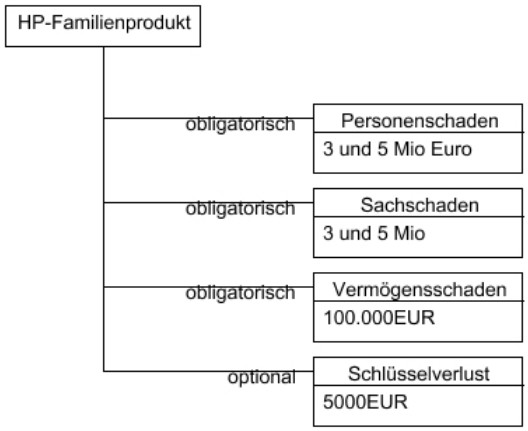

Abbildung 6: Struktur des Beispielproduktes

Möchte man für externe Partner einen Webservice zur Beitragsberechnung bereitstellen, so kann man die Produktinformationen bei der Definition der Schnittstelle berücksichtigen und mit folgenden Parametern in einer flachen Schnittstelle auskommen.

- versicherungssummePersonenschaden
- versicherungssummeSachschaden
- schluesselverlustEnthalten

Dabei impliziert man die folgenden Produktinformationen:

- Es gibt die vier oben aufgeführten Deckungstypen und keine weiteren
- Nur Schlüsselverlust ist optional, alle anderen obligatorisch
- Für Schlüsselverlust und für Vermögensschaden ist jeweils nur eine Versicherungssumme mögliche. Man muss die Höhe der Versicherungssumme also nicht an der Schnittstelle übertragen.

Die Konsequenz ist natürlich, dass entsprechende Produktänderungen zu einer Änderung der Schnittstelle führen.

Alternativ kann man die Struktur der Schnittstelle so aufbauen, dass sie mit beliebig vielen Deckungen umgehen kann (also im wesentlichen strukturgleich zum Fachmodell ist). Die Referenz von den Vertragsklassen auf die entsprechenden Produktklassen würde man dabei durch die ID der Produktklasse ersetzen, also etwa so:

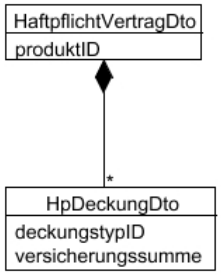

Abbildung 7: Strukturgleiche Schnittstelle für Beispiel 2

Was hat das für Konsequenzen für das aufrufende System? Hier gibt es zwei Alternativen:

- Produktflexibilität im aufrufenden System
  Das aufrufende System kann so konstruiert sein, dass es die Produktinformationen zur Laufzeit abfragt. Es kennt das fachliche Modell aber nicht die konkreten Produktdaten. Es werden also die erlaubten Deckungstypen inkl. der Information, ob sie optional oder obligatorisch sind und die erlaubten Versicherungssummen abgefragt. Die Oberfläche des Systems kann die Deckungen inkl. der Eingabemöglichkeiten für die Versicherungssummen dynamisch auf Basis der Produktinformationen darstellen. Die ID des Deckungstyps auf dem eine Deckung basiert wird im aufrufenden System gespeichert.
- Annahmen über Produktstruktur im aufrufenden System
  Ein Tarifrechner, den die Versicherung im Internet zur Verfügung stellt, kann zum Beispiel zwei Comboboxen „Versicherungssumme Personenschaden“, „Versicherungssumme Sachschaden“ und eine Checkbox „Schlüsselverlust J/N“ enthalten. Damit sind die oben aufgeführten Produktinformationen in den Internet-Tarifrechner codiert. Beim Aufruf ist ein Mapping des im Tarifrechner verwendeten (flachen) Modells auf die flexible Schnittstelle erforderlich. Dazu müssen u.a. die IDs der Deckungstypen im Mappingcode verwendet werden. Ist zum Beispiel die Checkbox angehakt, muss eine Deckung mit der DeckungstypID des Schlüsselverlusts zum Vertrag hinzugefügt werden.

Zusammenfassend gibt es also die folgenden beiden Extreme bei der Konstruktion von Schnittstellen:

- flache Schnittstellen mit Annahmen über die konkrete Produktstruktur
- flexible Schnittstellen, die strukturgleich (oder sehr ähnlich) zum Fachmodell sind

Letztere beinhalten keine Annahmen über die Produktstruktur, die über die Abbildung im Fachmodell hinausgehen.

Bewertung

Letztendlich möchten Versicherungsunternehmen mit der Einführung einer zentralen Produktkomponente mehr Produktflexibilität und kürzere Produktentwicklungszeiten erreichen. Um dies zu erreichen, muss Produktflexibilität vor allem in die operativen Systeme hinein konstruiert werden. Voraussetzung hierfür ist, dass die Produktkomponente eine entsprechende flexible Schnittstelle anbietet. Eine Produktkomponente sollte also in jedem Fall eine flexible, dem fachlichen Modell entsprechende Schnittstelle anbieten. Das heißt aber nicht unbedingt, dass

Fachmodelle und Schnittstelle 100% strukturgleich sind. So sind zum Beispiel Subklassen, die sich ausschließlich in der Implementierung von Methoden von ihren Superklassen unterscheiden im Schnittstellenmodell nicht erforderlich.

Ein Nachteil dieser flexiblen Schnittstellen ist allerdings, dass sie für Entwickler von solchen Systemen schwerer zu verstehen sind, die auf einem anderen fachlichen Modell arbeiten. Der Entwickler des oben bereits als Beispiel verwendeten Tarifrechners muss das einfache (unflexible) Modell des Tarifrechners auf das flexiblere Schnittstellenmodell abbilden. Dabei muss er nicht nur das Modell, sondern auch mindestens die IDs der einzelnen Produktbausteine kennen.

Häufig arbeiten Vertriebssysteme und Systeme externer Partner wie Maklersysteme und Vergleichsportale im Internet auf anderen, einfacheren Modellen. Für externe Partner auf deren Systeme man keinen Einfluss hat, bietet es sich an Services mit einer relativ flachen Schnittstelle zur Verfügung zu stellen. Insbesondere ist dies sinnvoll, wenn jeweils nur die aktuelle Produktgeneration unterstützt werden muss. Müssen mehrere Produktgenerationen unterstützt werden, kann dies dazu führen, dass die Schnittstelle flexibler sein muss, um alle Produktgenerationen zu unterstützen.

Anwendung des Builder-Patterns zur Erzeugung der Schnittstellenobjekte

Um den Entwicklern der aufrufenden Systeme, die auf einfacheren Modellen arbeiten die Integration zu erleichtern, kann man einen Builder[1] bereitstellen, mit dem man die Schnittstellenobjekte einfach erzeugen kann. So könnte man für das obige Schnittstellenmodell einen `HpVertragDtoBuilder` wie folgt schreiben:

1 Die hier beschriebene Verwendung des Builder-Patterns entspricht [RtoP], weniger der Erklärung in [GoF].

```
public class HpVertragDtoBuilder {

    public final static String PRODUKT_FAMILIE_ID = "HpFam";
    public final static String PERSONENSCHADEN_ID = "PS";
    public final static String SACHSCHADEN_ID = "SS";
    public final static String SCHLUESSELVERLuST_ID = "SV";

    private HpVetragDto vertrag;

    public void createVertrag(
            String produktId,
            Money vsPersonenschaden,
            Money vsSachschaden) {

        vertrag = new HpVetragDto(produktId);

        HpDeckungDto psDeckung = new HpDeckungDto(PERSONENSCHADEN_ID);
        psDeckung.setVersicherungssumme(vsPersonenschaden);
        vertrag.addDeckung(psDeckung);

        HpDeckungDto ssDeckung = new HpDeckungDto(SACHSCHADEN_ID);
        ssDeckung.setVersicherungssumme(vsSachschaden);
        vertrag.addDeckung(ssDeckung);
    }

    public void addSchluesselverlust() {
        HpDeckungDto svDeckung = new HpDeckungDto(SCHLUESSELVERLuST_ID);
        vertrag.addDeckung(svDeckung);
    }

    public HpVetragDto getVertrag() {
        return vertrag;
    }
}
```

Der Tarifrechner könnte diesen Builder nun wie folgt nutzen, um das `HpVertragDto` zu erzeugen.

```
public HpVertragDto erzeugeHpVertragDto(
    Money vsPersonenschaden,
    Money vsSachschaden,
    boolean schluesselverlust) {

    HpVertragDtoBuilder builder = new HpVertragDtoBuilder();
    builder.createVertrag(HpVertragDtoBuilder.PRODUKT_FAMILIE_ID,
        vsPersonenschaden, vsSachschaden);
    if (schluesselverlust) {
        builder.addSchluesselverlust();
    }
    return builder.getVertrag();
}
```

Mit dem Builder lässt sich also die Integration für die Entwickler der aufrufenden Systeme vereinfachen, ohne dass man eine explizite flache Schnittstelle anbieten muss.

Flexibilität mit „Dynamic Properties“

Bisher sind wir davon ausgegangen, dass die Attribute der Schnittstellenklassen zur Compilezeit festgelegt werden, indem man für jedes Attribut entsprechende Getter- und Settermethoden definiert. Das bedeutet natürlich, dass die Einführung neuer Attribute zwangsläufig mit Änderungen

des Sourcecodes der Schnittstelle verbunden ist. Eine Möglichkeit dies zu vermeiden, ist die Verwendung von „Dynamic Properties“. Eine ausführliche Diskussion findet sich in [Fowler, Properties].

Anstatt für jedes Attribut spezielle Getter/Setter zu definieren, definiert man generische Getter/Setter, die den Attributnamen als Parameter verwenden (an jeder Schnittstellenklasse bzw. in einer Basisklasse) wie folgt.

```
private Map<String, Object> attributeValues;

public void setAtributeValue(String attributeName, Object value) {
     attributeValues.put(attributeName, value);
}

public Object getAttributeValue(String attributeName) {
    return attributeValues.get(attributeName);
}
```

Dynamische Eigenschaften sind sinnvoll für Aspekte, die häufigen Änderungen unterworfen sind. So werden zum Beispiel in neuen Produktgenerationen öfters neue Tarifierungsmerkmale eingeführt. Hierbei handelt es sich i.d.R. um weitere Eigenschaften des versicherten Objektes bzw. der versicherten Person, die es der Versicherung ermöglicht das zu versichernde Risiko genauer zu bewerten. Andere Modellaspekte bleiben dagegen über einen langen Zeitraum konstant. Zum Beispiel haben Versicherungsverträge immer einen Versicherungsbeginn, einen Ablauf bzw. Laufzeit, eine Zahlungsweise etc. Für diese „konstanten“ Eigenschaften, benötigt man keine Flexibilität an der Schnittstelle und man sollte auf die Anwendung des Dynamic Property Pattern verzichten, da die Anbindung der aufrufenden Systeme unnötig verkompliziert wird, ohne dass man daraus einen Nutzen ziehen kann. In einer Schnittstellenklasse sollten also die unveränderlichen Aspekte über explizite Methoden abgebildet werden und zusätzlich das Dynamic Property Pattern unterstützt werden. Der folgende Sourcecode zeigt dies für die bereits weiter oben als Beispiel verwendete Klasse `VersichertePersonDto`.

```
public class VersichertePersonDto {

    public final static String EINTRITTSALTER = "eintrittsalter";
    public final static String GESCHLECHT = "geschlecht";

    private Map<String, Object> attributeValues;

    public void setAtributeValue(String attributeName, Object value) {
         attributeValues.put(attributeName, value);
    }

    public Object getAttributeValue(String attributeName) {
        return attributeValues.get(attributeName);
    }

    public Integer getEintrittsalter() {
        return (Integer)attributeValues.get(EINTRITTSALTER);
    }

    public void setEintrittsalter(Integer eintrittsalter) {
        attributeValues.put(EINTRITTSALTER, eintrittsalter);
    }

    public String getGeschlecht() {
        return (String)attributeValues.get(GESCHLECHT);
    }

    public void setEintrittsalter(String geschlecht) {
        attributeValues.put(GESCHLECHT, geschlecht);
    }

}
```

Flexibilität bzgl. der Einführung neuer Produkte bringt die Verwendung von Dynamic Properties natürlich nur dann, wenn die operativen Systeme auch entsprechend flexibel gebaut sind. Das heißt die Namen der Attribute dürfen nicht im operativen System hart codiert sein, sondern müssen zur Laufzeit abgefragt werden. Hierzu muss die Produktkomponente entsprechende Auskunftsmethoden zur Verfügung stellen. Dabei ist zu beachten, dass Attribute unter Umständen nur in bestimmten Produktgeneration verwendet werden. Eine Methodensignatur für einen entsprechenden Auskunftsservice könnte also in etwa wie folgt aussehen:

```
public String[] getDynamicAttributesFuerHpVertrag(String produktGenerationId);
```

Generell berücksichtigen sollte man beim Entwurf der Schnittstelle, dass man Produktflexibilität nicht durch eine flexible Schnittstelle des Produktsystems erhält, sondern die gesamte IT-Landschaft des Versicherungsunternehmens betrachten muss. Letztendlich müssen die operativen Systeme die angestrebte Flexibilität idealer Weise von der Datenbank bis zum User Interface und den Druckfunktionen unterstützen. Versuche extrem generische operative Systeme zu entwickeln, scheitern immer daran, dass die generischen Benutzeroberflächen nicht benutzerfreundlich sind und die Systeme vor allem im Batchbetrieb nicht performant genug sind. Ein guter Kompromiss zwischen Produktflexibilität, Benutzerfreundlichkeit & Performance ist es das operative System flexibel bzgl. der Einführung neuer Attribute für bestimmte fachliche Klassen wie das versicherte Objekt, die versicherte Person etc. zu gestalten. Die Struktur, also die Klassen und deren Beziehungen, sollte dagegen zur Compilezeit festgelegt werden.

Dieses Prinzip ist zum Beispiel im RAP-Client für die im Faktor-IPS Einführungstutorial entwickelte Hausratversicherung umgesetzt. Die folgende Abbildung zeigt die Maske zur Eingabe eines Angebotes über eine Hausratversicherung.

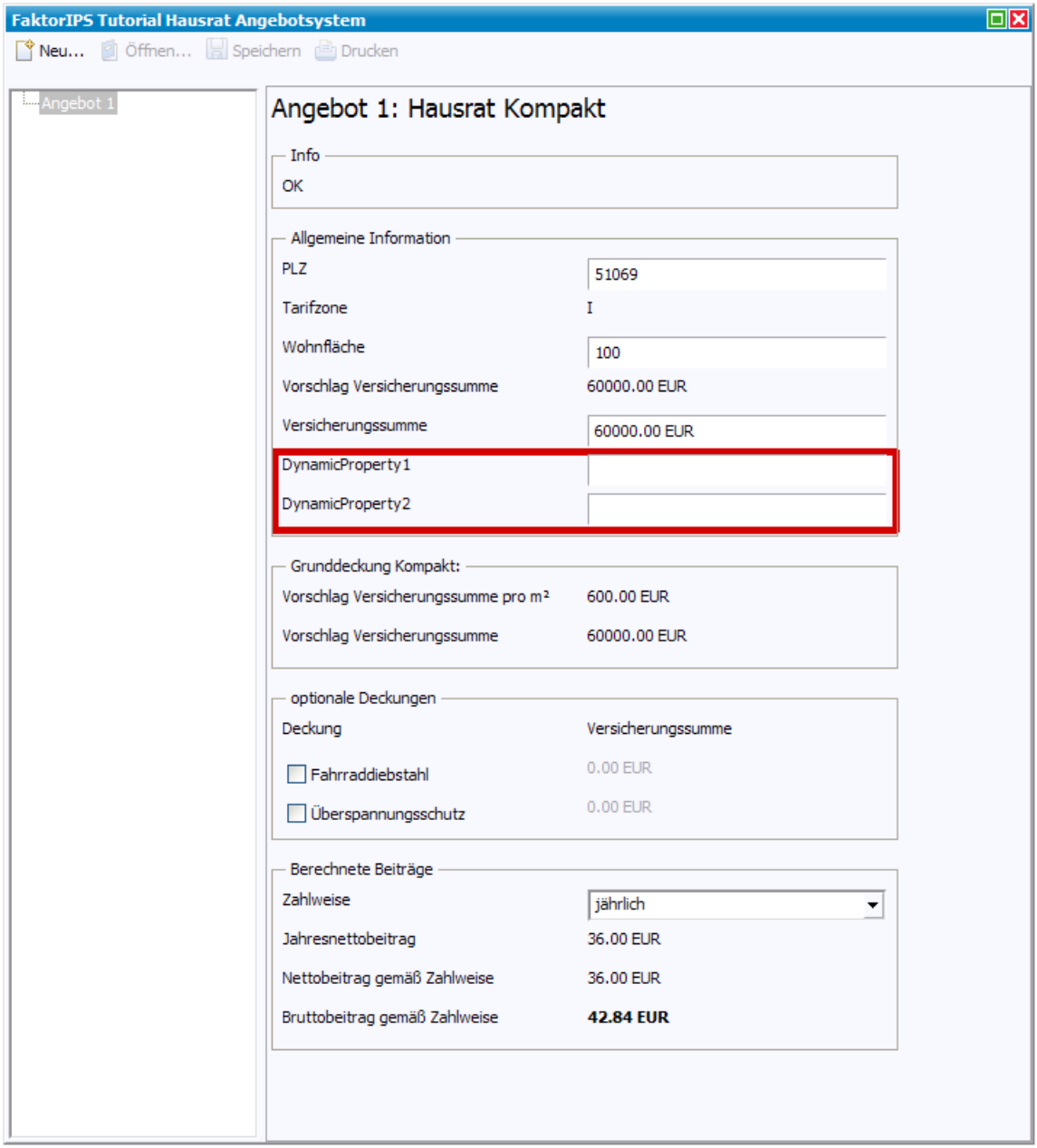

Abbildung 8: Eingabemaske für ein Angebot über eine Hausratversicherung

Der Aufbau der Maske ist zu großen Teilen fest im Programm implementiert, ebenso die Anzahl und Positionierung der Labels und Eingabecontrols. Am Ende des Abschnitts zum versicherten Hausrat ist allerdings ein Bereich vorgesehen, in dem weitere Attribute der Klasse HausratVertrag dynamisch angezeigt werden. Der Bereich ist in der Abbildung rot markiert. Beispielhaft wurden mit Faktor-IPS im Modell zwei neue Attribute DynamicProperty1 und DynamicProperty2

angelegt. Der RAP-Client erzeugt dynamisch entsprechende Controls zur Bearbeitung dieser Eigenschaften. Details können der Faktor-IPS Dokumentation „Tutorial zur Entwicklung einer Anwendung“ entnommen werden.

Berücksichtigung von Produktänderungen im Zeitablauf

Versicherungsunternehmen führen in regelmäßigen Abständen Änderungen an ihren Produkten durch. Unterschieden wird dabei zwischen Änderungen, die nur für das Neugeschäft maßgeblich sind und Änderungen wie zum Beispiel Beitragsanpassungen, die sich auch auf bestehende Verträge auswirken. Die Begriffe für diese beiden Arten von Änderungen sind leider nicht standardisiert. Wir verwenden in diesem Artikel den Begriff Generation für Änderungen, die nur für das Neugeschäft maßgeblich sind und Anpassungsstufe für Änderungen, die auch für bestehende Verträge relevant sind[2].

Produktgenerationen

Im folgenden konzentrieren wir uns zunächst auf Generationen und behandeln danach Anpassungsstufen. Eine neue Generation wird i.d.R. zu einem bestimmten Stichtag eingeführt. Damit ist gemeint, dass alle Verträge für das Produkt ab diesem Datum auf Basis der neuen Generation abgeschlossen werden.

Beispiel
Zum 1.1.2008 wird eine neue Generation Haftpflicht eingeführt. Neue Haftpflichtverträge mit Versicherungsbeginn ab dem 1.1.2008 sollen auf Basis dieser Generation angeboten (und abgeschlossen) werden. Verträge mit einem Versicherungsbeginn vor dem 1.1.2008 werden auf Basis der alten Generation abgeschlossen.

Da Bestandssysteme Verträge unterschiedlicher Produktgenerationen verwalten können, ist es für sie relativ einfach die ID der verwendeten Produktgeneration an der Schnittstelle zu übergeben. Die Ermittlung der Produktgeneration anhand des Versicherungsbeginns ist dagegen an dieser Stelle wenig empfehlenswert, da zu einem Zeitpunkt auch mehrere Produktgeneration verkauft werden können. Bei der Neuanlage von Verträgen wird die Generation entweder von den Vertriebssystemen mit übergeben oder durch den Sachbearbeiter eingegeben. Im letzteren Fall kann natürlich ein Service der Produktkomponente genutzt werden, der zum Versicherungsbeginn alle verkaufbaren Generationen zurück liefert.

Bestandssysteme müssen hierzu natürlich die ID der Generation in ihrer Datenbank speichern. Da die ID aber wie der Name schon sagt die Generation identifiziert, stellt dies kein Problem dar. (Natürlich darf dabei im Produktsystem die ID einer Generation nachdem sie operativ genutzt wird nicht mehr geändert werden. Wie schon gesagt, es ist eine ID.)

Vertriebssysteme sind dagegen häufig so gebaut, dass sie lediglich mit einer einzigen Produktgeneration arbeiten können. D.h. der Verkaufsbeginn der neuen Produktgeneration in einem Vertriebssystem wird bestimmt durch den Releasewechsel auf eine Version der Software. Häufig werden in Vertriebssystemen nur der 1. (und manchmal noch der 15.) eines Monats als Versicherungsbeginn für neue Angebote zugelassen. Die neue Softwareversion kann damit

2 In der VAA des GDV werden Änderungen für das Neugeschäft als Versionen und Änderungen, die auch bestehende Verträge betreffen als Generationen bezeichnet. Eine ausführlichere Beschreibung befindet sich im Faktor-IPS Tutorial.

problemlos im Monat vor der Einführung der neuen Produktgeneration in Betrieb genommen werden. In unserem obigen Beispiel also im Dezember 2007. Alle Angebote die mit der neuen Softwareversion erstellt werden, haben damit automatisch einen Versicherungsbeginn nach dem 1.1.2008 und es muss die neue Generation verwendet werden. Angebote mit einem Versicherungsbeginn, die die Verwendung der alten Generation erforderlich machen würden, können nicht erstellt werden, wenn man voraussetzt, dass die Software in diesem Fall keine Angebote mit Versicherungsbeginn in der Vergangenheit zulässt.

Da die Produktkomponente alle Produktgeneration unterstützt, muss beim Aufruf aus einem Vertriebssystem also die zu verwendende Generation bestimmt werden. Hierzu gibt es einige Alternativen.

Verwendung Versicherungsbeginn zur Ermittlung der Generation
Innerhalb der Produktkomponente hat eine Generation einen frühestmöglichen Versicherungsbeginn. Zu einem gegeben Versicherungsbeginn kann damit die aktuelle Generation ermittelt werden. Dies ist aber keine Lösung, wenn die Produktkomponente unabhängig vom Vertriebssystem installiert wird. Wieso das so ist, detaillieren die folgenden beiden Szenarien:

- Szenario A
  Der neue Haftpflichttarif aus unserem Beispiel erfordert die Eingabe eines neuen Tarifmerkmals. Die Produktkomponente wird am 15.12.2007 auf einem zentralen Server installiert, das Vertriebssystem aber erst am 20.12.2007. Im Zeitraum dazwischen können keine Angebote mit Versicherungsbeginn in 2008 erstellt werden, da die Vertriebssoftware noch nicht die Eingabe des neuen Tarifmerkmals erfordert, die Produktkomponente aber bereits die neue Generation als die dann gültige ermittelt. :-(
- Szenario B
  Die Struktur des Haftpflichttarif bleibt gleich, es ändern sich nur die Beiträge. In diesem Fall würde es softwaretechnisch keine Probleme geben. Fachlich ist aber nun der Verkaufsbeginn der neuen Produktgeneration im Vertriebssystem von der Installation der Produktkomponente abhängig.

Verwendung Versicherungsbeginn zur Ermittlung der Generation & Synchronisation der Installation

Zusätzlich zum oben beschriebenen Verfahren werden Produktkomponente und Vertriebssystem immer zeitgleich installiert. Dieses Verfahren funktioniert theoretisch, d.h. es wird immer die korrekte Generation verwendet. Für die Praxis ist es untauglich, da damit eine unnötige Kopplung zwischen den Komponenten/Systemen erzeugt wird, die den Betrieb erschweren. Die Komplexität steigt mit der Anzahl der Systeme, die die Produktkomponente nutzen.
Das Verfahren kann angewendet werden, wenn die Produktkomponente nicht auf zentralen Servern installiert wird, sondern z. B. als Teil eines Offline-Tarifrechner ausgeliefert wird.

Verwendung von Versicherungsbeginn & Antragsdatum

Anstatt die Installation zu synchronisieren, könnte man die aktuelle Produktgeneration anhand des Versicherungsbeginns und des Antragsdatums ermitteln. Hierzu müsste die Produktkomponente für jede Generation zusätzlich zum frühesten Versicherungsbeginn einen Verkaufsbeginn führen. In unserem Beispiel würden wir definieren, dass die neue Haftpflichtgeneration einen frühestmöglichen Versicherungsbeginn 1.1.2008 hat, aber bereits ab dem 20.12.2007 verkauft wird. Würde nun vom Vertriebssystem ein Angebot mit Versicherungsbeginn 1.1.2008 und Antragsdatum 19.12.2007 berechnet werden, so würde die Produktkomponente die alte Generation

wählen, mit Antragsdatum 20.12.2007 würde es die neue Generation wählen.

Das Problem an der Lösung ist, dass der Verkaufsbeginn vom konkreten Vertriebssystem abhängt. So erfolgt der Releasewechsel des Internetauftritts i.d.R. unabhängig vom Releasewechsel des Außendienstsystems. Die Produktkomponente müsste also für jedes nutzende System den Verkaufsbeginn führen. Dieser müsste für jedes Vertriebssystem synchron zur Installation sein, da die Systeme nur eine Generation unterstützen. Wird die Installation verschoben müsste der Verkaufsbeginn in der Produktkomponente angepasst werden, eine unschöne Abhängigkeit.

Festlegung der Generation durch das nutzende System

Das nutzende System gibt der Produktkomponente die ID der zu verwendenden Generation mit. Dadurch wird der Verkaufsbeginn mit dem Releasewechsel der Vertriebssoftware synchronisiert. Allerdings muss jedes Vertriebssystem die IDs der aktuellen Produktgenerationen kennen. Beim Wechsel auf eine neue Produktgeneration muss die ID der zu verwendenden Generation in der Software angepasst werden. Da die Einführung einer neuen Produktgeneration i.d.R. mit Softwareänderungen verbunden ist und zudem immer ein Systemtest stattfindet, ist der Nachteil aber vernachlässigbar.

Empfehlung zur Konstruktion der Schnittstelle der Produktkomponente

Die Produktkomponente sollte nicht pro System den Verkaufsbeginn verwalten. Statt dessen kann die ID der zu verwendenden Generation explizit an der Schnittstelle übergeben werden. Die Vertriebssoftware kann aber auch die für den Versicherungsbeginn aktuelle Generation verwenden, indem er die ID auf NULL setzt. Welches Verfahren gewählt wird, entscheiden die Entwickler der Vertriebssoftware.

Anpassungsstufen

Da sich eine Produktanpassung auf alle bestehenden Verträge auswirkt, bleibt die Struktur des Modells i.d.R. unverändert, es werden also z. B. keine neuen Tarifmerkmale eingeführt, da diese Werte für die alten Verträge nicht bekannt sind. Operative Systeme können damit i.d.R. neue Anpassungsstufen unterstützen, ohne dass Änderungen an der Software notwendig wären. Aus diesem Grund wird die zu verwendende Anpassungsstufe von der Produktkomponente auf Basis eines Stichtags (Versicherungsbeginn, letzte Hauptfälligkeit) ermittelt. Die Produktänderungen werden damit mit der Installation der Produktkomponente für die Systeme wirksam. In Bestandssystemen werden die bestehenden Verträge i.d.R. über einen Batchjob angepasst.

Zusammenfassung

In diesem Artikel wurden die grundlegenden Konstruktionsprinzipien der Schnittstellen von Produktservern diskutiert. Insbesondere wurde erläutert wie Produktinformationen implizit in (flachen) Schnittstellen enthalten sind. Produktänderungen bedingen hierdurch häufig auch Änderungen der Schnittstelle. Produktflexibilität kann man durch die Verwendung von Schnittstellen erreichen, die strukturgleich zum fachlichen Modell sind und die Einführung neuer Tarifierungsmerkmale durch „Dynamik Properties“ unterstützen. Die Herausforderung besteht darin diese Produktflexibilität nicht nur an der Schnittstelle des Produktservers bereitzustellen, sondern diese Flexibilität auch in den operativen Systeme zu unterstützen.

Referenzen

[Fowler, Properties] Fowler, Martin. Dealing With Properties, 1997

[Fowler, PoEAA] Fowler, Martin. Pattern of Enterprise Application Architecture, Addison Wesley 2003

[RtoP] Kerievsky, Joshua. Refactoring to Patterns, Addision-Wesley 2004

[GoF] Gamma et al. Design Patterns - Elements of Reusable Object Oriented Software. Addision-Wesley 1995.