B-Wise Produktsystem

B-Factory

Lösungen für Versicherungsunternehmen

Februar 2016

BISS

Evolutionäre Ideen für Ihre Software

Inhalt

Fachlicher Hintergrund

Deregulierung und Digitalisierung erhöhen den Wettbewerbsdruck auf die Versicherungsunternehmen, dem zumeist mit einer Differenzierungsstrategie begegnet wird. Innovative Produkte, deren Entwicklungszyklen zunehmend kürzer werden, sind der naheliegendste Weg, diese Differenzierungsstrategie effektiv umzusetzen.

Die Fähigkeit, schnell neue Produkte einzuführen, zählt somit immer mehr zu den strategischen Wettbewerbsvorteilen von Versicherungsunternehmen. Die fachliche Komplexität des Problems sprengt häufig die Grenzen herkömmlicher IT-Systeme, wobei die unterschiedlichen Anforderungen von Fachabteilung, Anwendungsentwicklung und IT-Betrieb bisherige Lösungen an ihre Grenzen stoßen lassen. Die Entwicklung neuer Produkte, inkl. deren Konzeption, fachlicher Abstimmungen und der Anpassung aller relevanten Systeme vom Vertrieb bis zur Leistung nimmt heute immer noch sehr viel Zeit in Anspruch. Gerade die Digitalisierung mit ihren extrem kurzen Innovationszyklen stellt hier einen Treiber dar, dem sich heute kein Unternehmen entziehen kann, das am Markt erfolgreich operieren möchte.

Im industriellen Bereich kommt hier seit Jahren das sogenannte Rapid-Prototyping zum Einsatz. Durch den Einsatz neuer Werkstoffe und Technologien (z.B. 3D-Druck) lassen sich dabei in kürzester Zeit produktionstaugliche Kleinstserienerzeugnisse herstellen. Die Industrialisierung des Versicherungsbetriebs führt auch im Bereich der Entwicklung von Versicherungsprodukten zu einem immer stärker ausgeprägten Wunsch nach einem „Rapid-Prototyping von Versicherungsprodukten". Dabei steht nicht nur das Produkt selbst im Fokus des Interesses, sondern auch dessen Validierung in den unterschiedlichen Einsatzszenarien, allen voran im Rahmen digitaler Geschäftsmodelle. Produkte sind heutzutage häufig „Minimalprodukte", die nach einer kurzen und effizienten Entwicklungs-, Validierungs- und Testphase ausgerollt und dem Kunden zur „echten Prüfung" vorgelegt werden („live testing"). Setzt sich ein solches Produkt am Markt durch, müssen entsprechende BackOffice-Systeme flexibel auf die Bedürfnisse reagieren können. Der Produkt-Lifecycle endet hier jedoch nicht: Laufende Analysen, Kontrolle der Key Performance Indicators (KPI) sowie kontinuierliches Benutzer-Feed-Back müssen mit den spezifischen Produktdaten korreliert und laufend ausgewertet werden.

Ein entsprechend kurzer Produktentwicklungsprozess, der nicht nur die Fachabteilung beim Modellieren und Testen unterstützt, sondern eine schnelle und nahtlose Überführung der Produkte in den täglichen Betrieb ermöglicht, stellt einen direkten strategischen Wettbewerbsvorteil dar. Begleitet wird dieser von der Anforderung, Produkte schnellstmöglich innerhalb entsprechender Anwendungen live auf den Markt zu bringen sowie im Anschluss die BackOffice-Systeme bei der Wahrnehmung von deren Aufgaben zu unterstützen.

All diese Aufgaben übernimmt typischerweise ein Produktsystem. Bei der Betrachtung der Anforderungen und Leistungen eines Produktsystems müssen wir unterschiedliche Phasen im Leben eines Produkts unterscheiden:

Während der *Produktfindung* werden existierende Produkte, bisherige Bestands- sowie Marktdaten analysiert und statistisch ausgewertet, um das fachliche Design des neuen Produkts zu entwickeln. In dieser Phase spielen auch Marketingaspekte sowie die Unternehmensstrategie eine Rolle.

Bei der *Produktdefinition* werden die einzelnen Komponenten des Produkts im Detail definiert (implementiert) und getestet. Wichtig hierbei ist u.a. die korrekte Überführung die während der Produktfindung definierten Konzepte in das Zielsystem (IT-Produkt).

In der *Produktion* wird das Produkt schließlich verkauft und die abgeschlossenen Policen im Bestandssystem verwaltet, auftretende Schäden oder sonstige Leistungen mit Hilfe des Schadensystems geprüft und ggf. beglichen. Hier spielen klassische Vertriebssysteme sowie zunehmend digitale Vertriebswege eine wichtige Rolle. Auch das Bestands- und Schadensystem haben bestimmte Anforderungen an das Produkt bzw. das Produktsystem.

So muss ein Produktsystem sowohl die Konzeption und die Entwicklung eines Produkts unterstützen als auch den Vertrieb, die Bestandsführung sowie entsprechende Schadensysteme. Aber auch Analysesysteme sind (sowohl retrospektiv als auch prospektiv) ein wesentlicher Bestandteil der Wertschöpfungskette und somit vom Produktsystem zu unterstützen. Daher ist es wichtig, dass ein Produktsystem die unterschiedlichen Bereiche der Plausiblisierung, Tarifierung, Antrags- und Vertragsunterstützung bis hin zur Aufschlüsselung der vereinbarten Leistungen inkl. deren Plausibilisierung und Prüfung für den Einzelfall optimal unterstützt.

Ein Produktsystem ist daher umso besser geeignet, die genannten Anforderungen zu erfüllen, je strukturierter das Produktmodell ist, auf dem das Produktsystem aufsetzt. Das Produktmodell ist somit die strukturelle Basis eines Produktsystems und definiert die Komponenten, aus denen sich ein Produkt zusammensetzt. Hier hilft eine klare Definition nicht nur dem Produktsystem sondern insbesondere dem Anwender (Produktdesigner), da die Modelle einheitlich, klar verständlich und in sich stimmig definiert werden können. Entsprechende Werkzeuge, die teils grafisch, teils textuell eine Visualisierung der Zusammenhänge ermöglichen sowie eine Erfassung bzw. Änderung der einzelnen Parameter erlauben, sind daher inhärenter Bestandteil moderner Produktsysteme.

Insbesondere die Gestaltung und die Definition (Implementierung) von Produkten stellen hohe Anforderungen an das Design und die Usability des Produktsystems. Die Bedienung sollte gleichermaßen für Fachanwender wie für IT-Spezialisten leicht erlernbar sein, den Entwickler jedoch nicht in seinen Möglichkeiten einschränken. Der Anwender sollte Produkte aus bekannten Komponenten zusammenstellen und diese parametrisieren können. Aber auch komplexe Zusammenhänge und Algorithmen von Personenprodukten sollten effizient mit dem Produktsystem umsetzbar sein.

Ebenso wie die Frage, was ein Produktsystem enthalten und welche Eigenschaften es besitzen soll, ist auch die Frage nach dem, was nicht zum Produktsystem bzw. dessen Inhalten gehört, wesentlich. Im Rahmen heute gängiger Service-orientierter

Architekturen haben sich klare Verantwortlichkeiten der einzelnen Services herauskristallisiert. Eine Abgrenzung des Produktsystems zu anderen IT-Systemen und –Services ist daher von entscheidender Bedeutung für eine langfristig wartbare und erfolgreiche IT-Architektur. Einiges davon wird bereits aus der Schnittstelle (dem API, Application Programming Interface) des Produktsystems deutlich. Anderes sollte explizit als „ist anderswo zu definieren und zu implementieren" charakterisiert werden.

B-Wise Produktsystem B-Factory

B-Factory ist ein Werkzeug zur regelbasierten Entwicklung und zur produktiven Bereitstellung von Versicherungsprodukten. Es bildet das gesamte Produktwissen in Form von Daten, Merkmalen und Eigenschaften sowie in Form deklarativer Regeln ab und stellt dieses Wissen mit Hilfe einer Schnittstelle (API) den daran angeschlossenen Systemen (Vertrieb, Bestand, Schaden) zur Verfügung.

Das Produktwissen umfasst dabei das Produktmodell, alle zugehörigen fachlichen, technischen und organisatorischen Produktinformationen sowie die entsprechenden Plausibilisierungs-, Prüf und Berechnungsregeln. Dies gilt sowohl für die Verkaufs- als auch Leistungsseite.

B-Factory kommt zusammen mit einem Basis-Produktmodell („Produktbaukasten"), das sich fachlich an heute gängigen Fachstandards (GDV und BiPRO) orientiert. Dadurch ist es z.B. sehr einfach, entsprechende Schnittstellen wie BiPRO SOAP-WebServices zu bedienen[1].

Das Produktsystem richtet sich sowohl an fachliche Produktentwickler, die für die Produktfindung und Produktdefinition verantwortlich sind. Es dient aber ebenso dem IT-Entwickler, der mit der Aufgabe betraut ist, entsprechende Anwendungen zu implementieren.

Der fachliche Produktentwickler arbeitet auf vordefinierten und konfigurierbaren Produkten und wird durch entsprechende Werkzeuge unterstützt, welche die Aufgabe der Produkt-Konzeption und -Definition sowie der Produkt-Tests erleichtern. Hierzu zählen neben speziellen Editoren auch der Produkt-Debugger, der Regressionstester sowie die Möglichkeit, auf Knopfdruck Web-Oberflächen für interaktive Tests zu erzeugen. Diese interaktiven Tests werden von Komponenten des B-Wise-Frameworks unterstützt.

Die konfigurierbaren Produkte stellt im Normalfall ein Mitarbeiter aus der IT zur Verfügung. Dadurch wird einerseits maximale Flexibilität erreicht, andererseits

[1] Das Produktsystem verfügt neben dem spezifischen API mit C/C++, Java und WebService-Bindings über BiPRO-Schnittstellen, so dass eine Bereitstellung des Produktwissens auf diesem Wege „out-of-the-box" möglich ist.

leichte Anpassbarkeit und/oder Erweiterbarkeit durch die Fachabteilung ermöglicht.

Der Anwendungsentwickler erhält einen Service mit einem wohl definieren und für alle Produkte einheitlichen API, das die effektive und schnelle Implementierung entsprechender Anwendungen ermöglicht. Das API erlaubt dabei hoch-interaktive Anwendungslösungen, wie sie heute z.B. im Rahmen moderner Web-Anwendungen benötigt werden. Der Batch-Betrieb wird durch entsprechende Optimierungsalgorithmen unterstützt, die auf eine hohe Performance zielen.

B-Factory Eigenschaften

Bei B-Factory handelt es sich um ein Programmsystem zur Erstellung, Pflege und Anwendung von Versicherungsprodukten. Es dient der systematischen, formalen Beschreibung von Versicherungsprodukten und besteht im Wesentlichen aus:

- einer zentralen Speicherung und Verwaltung der Produkte
- einer Entwicklungsumgebung mit Werkzeugen zur Produktdefinition, Produktänderung und Produkttests
- Werkzeugen zur Organisation und Dokumentation der Produkte sowie einer
- Schnittstelle für externe Systeme (z.B. Programme der Vertragsverwaltung oder Angebotserstellung).

B-Factory dient im Sinne eines Service unterschiedlichen Dritt-Programmen bzw. -Anwendungen. Es bietet über eine definierte Schnittstelle (Produktsystem-API) Informationen zu den darin implementierten Produkten und ermöglicht unterschiedliche Prüfungen sowie Berechnungen. Darüber hinaus erlaubt es Abfragen zur Struktur und zu den Inhalten der enthaltenen Produkte sowie produktübergreifende Analysen (z.B. „alle Produkte, die sich auf Einfamilienhäuser als versicherte Objekte beziehen").

Bei der Entwicklung von B-Factory wurde sehr viel Wert gelegt auf Flexibilität, Transparenz, Kostensenkung bei der Entwicklung von Produkten, kurze Time-to-Market sowie maximale Kundenorientierung.

B-Factory erfüllt eine Reihe von Anforderungen (insbesondere die eingangs Definierten) und bietet eine Fülle von Leistungen sowohl im technischen wie im fachlichen Umfeld. Es ist dabei sehr kompakt und kann eigenständig (stand-alone) eingesetzt werden.

Die nachfolgende Liste gibt einen Überblick über die wesentlichen Eigenschaften von B-Factory.

- B-Factory ermöglicht die spartenübergreifende Abbildung aller Produkte eines Versicherungsunternehmens inkl. verwandter Produkte aus dem Allfinanzbereich. Beliebige, spartenübergreifende Bündelungen von Versicherungsprodukt-

ten sowie Allfinanzprodukten sind möglich und auch Assistance-Produkte werden unterstützt.

- Alle produktrelevanten Daten, Informationen und fachliche Regeln können in B-Factory zentralisiert werden und stehen dadurch redundanzfrei innerhalb des Unternehmens zur Verfügung. Hierzu gehören Formeln zur Berechnung der Prämie, der Gewinnanteile und Renten, der Versicherungssumme, des Deckungskapitals, Rückkaufswerte, Einschlussbeiträge u.a. Die Logiken der Berechnungen sowie die beteiligten Attribute werden transparent gespeichert und über eine definierte Schnittstelle exportiert.
- Optionale syntaktische Erweiterungen erlauben die einfache Definition von Entscheidungstabellen, wie sie im Bereich von Sachprodukten häufig vorkommen; dies erleichtert die Erfassung und Pflege und steigert die Übersichtlichkeit innerhalb der Produkte. Diese Syntaxerweiterungen werden durch einen spezialisierten Java-Compiler zur Entwicklungszeit behandelt, so dass 100% Java-Bytecode entsteht, der mit Standard-Java-Laufzeitumgebungen (JRE) ohne zusätzliche Erweiterungen lauffähig ist. Der Compiler kann sowohl von der Kommandozeile aus, integriert in die Eclipse-Entwicklungsumgebung oder aus gängigen Build-Werkzeugen heraus verwendet werden.
- Das Produktsystem ermöglicht stücklistenförmige Konstrukte und eine Wiederverwendung der implementierten Bausteine. Neue Produkte können daher einfach durch Kombination und Verfeinerung oder Abwandlung aus bestehenden Elementen (Elementar- oder Marketingprodukten) gebildet werden. Diese Technologie ermöglicht eine kurze „Time-to-Market". Interaktive, teils graphische Werkzeuge unterstützen den Produktentwickler bei der Arbeit mit dem System. „Assistenten" („Wizzards") führen den Entwickler bei der Definition neuer Produkte. Automatisch unterstützte Checklisten erlauben die Einhaltung der Konsistenz sowie die Optimierung der technischen und fachlichen Qualität.
- Neben der Entwicklung neuer Produkte kann das Produktsystem selbstverständlich auch der technischen Unterstützung bestehender Produkte dienen. So können Prüf,- Tarifierungs- und ggfs. auch Annahmeregeln von Produkten im Produktsystem (nach)implementiert werden, um z.B. Berechnungen auf anderen technischen Plattformen (z.B. auf PC-Hardware oder auf Mobilgeräten) verfügbar zu machen oder um Berechnungen mit einem höheren Interaktivitätsgrad (z.B. mit feldweiser Validierung) zu ermöglichen. Ein stufenweises Vorgehen mit Integration bestehender Berechnungslösungen wie Rechenkerne oder -services in das Produktsystem ist in vielen Projekten sinnvoll und erlaubt so die Migration existierender Lösungen in eine moderne, Service-orientierte Welt mit einem modernen Produktsystem-API.
- Die Nutzung des Produktsystems wird durch den mitgelieferten Produktbaukasten unterstützt; der Produktbaukasten enthält Basisbausteine für alle Sparten (aktuell Personen-, Sach-, Krankenversicherung und Gewerbeversicherung) mit querschnittlich und unternehmensunabhängig definierten Merkmalen und Regeln.
- B-Factory ist Objekt-orientiert konzipiert, entwickelt und in Java implementiert worden, wobei zu Beginn das IAA-Modell und in Folge das von der BiPRO entwickelte Modell zu Grunde gelegt worden ist.

- Das Produktsystem ist als eigenständiges Modul mit einer definierten Schnittstelle (dem Produktsystem-API) implementiert. Es kann entweder direkt in eine Java-Anwendung eingebunden und dann mit einem Java-Interface angesprochen oder auch im Sinne eines autarken (Micro-) Service für sich allein in einem Application-Server deployt und über einen WebService genutzt werden.
- Das Produktsystem ist plattformübergreifend verfügbar (eine Java-Runtime-Umgebung vorausgesetzt).
- Eine Trennung zwischen dem Produktsystem und der zugrundeliegenden Datenbasis ermöglicht nicht nur die Verwendung unterschiedlicher Datenbanksysteme (derzeit SQL-basiert), sondern auch die Nutzung bestehender Daten unterschiedlicher Strukturen (beispielsweise die ad-hoc-Verwendung existierender DB-Tabellen unterschiedlicher Formate).
- B-Factory besitzt keine Abhängigkeiten zu anderen fachlichen B-Wise Modulen und lässt sich daher als schlankes Subsystem mit einem „small footprint" einsetzen; die JAR-Datei hat eine Größe von rund 600 kByte, der zugehörige Produktbaukasten lediglich 450 kByte
- Die konkreten Werte oder Wertebereiche von Merkmalen (z.B. Mehrwertsteuersatz bzw. Zahlweise) oder auch Entscheidungstabellen können sowohl innerhalb des Produkt-Codes als auch außerhalb des Produktsystems, z.B. innerhalb einer Datenbank abgebildet werden.
- Das Produktsystem erlaubt eine vollständige Historisierung der beinhalteten Produkte inkl. der parallelen Verarbeitung unterschiedlicher Versionen ein und desselben Produkts. Es ermöglicht darüber hinaus die Verwendung von Varianten (Mandantenfähigkeit z.B. in Form unterschiedlicher Vertriebskanäle).
- Produkte innerhalb des Produktsystem verfügen über unterschiedliche Stati (aktiv, gesperrt, im Test, usw.). Alle Status-Transitionen werden historisiert, so dass eine vollständige Nachvollziehbarkeit der Änderungen gewährleistet ist.
- Umfangreiche Prüfungen erlauben die Sicherung der fachlichen und technischen Konsistenz eines Produkttyps sowie einer Produktinstanz. Komplexe Prüfregeln können im Kontext von Geschäftsvorfällen verwendet werden.
- B-Factory beinhaltet optional zusätzlich zu den o.g. Faktoren und Regeln auch die textuelle Beschreibung der Produkte und Produkt-Bestandteile. Dieses dient nicht nur der Dokumentation der Produktentwicklung und Produktpflege, sondern auch der fachlichen Präsentation der Produkte und damit des Leistungsspektrums des Versicherers dem Kunden bzw. Interessenten gegenüber im Rahmen von Tarifierungs- bzw. Angebotssystemen.
- Neben der Beschreibung der Produkte selbst können optional auch die nötigen Textbausteine für Klauseln und allgemeine Texte innerhalb des Produktsystems gespeichert werden. Dabei handelt es sich vornehmlich um Textbausteine, die in Druckdokumenten (Angebote, Anträge, Policen) verwendet werden. Die Verwendung des HTML-Formats erlaubt dabei eine einfache und standardisierte Formatierung.
- Die Produkte innerhalb des Produktsystems sind fremdwährungsfähig. Es können parallel mehrere Währungseinheiten verwendet werden.
- Speziell für den Batch-Betrieb wurden Optimierungsverfahren implementiert, die eine effiziente Nutzung und kurze Ausführungszeiten erlauben.

Produktabbildungen innerhalb von B-Factory

Versicherungs- oder Allfinanzprodukte werden innerhalb von B-Factory mit Hilfe von Merkmalen (z.B. Versicherungssumme oder Wohnfläche) sowie mit Hilfe deklarativer Regeln beschrieben.

Merkmale und deren Ausprägungen sind selbsterklärend. So ist z.B. die Wohnfläche als Merkmal (eines definierten Typs, hier: Integer) Teil eines versicherbaren Objekts und beschreibt damit eine ganz bestimmte Eigenschaft einer spezifischen Objektinstanz (z.B. „ein Haus mit 200qm Wohnfläche"). Neben Merkmalen, die für Plausibilitäten und/oder Berechnungsregeln benötigt werden, existieren noch eine Reihe von Merkmalen, die rein deskriptiven oder administrativen Charakter haben, wie der Produktname oder die Produktversion. Neben den eigentlichen Werten (oder Wertebereichen) enthalten die Merkmale auch Stati wie z.B. „erforderlich" (required), „fehlerhaft" (erroneous) oder „im aktuellen Zustand nicht benötigt" (outtree).

Im Wesentlichen werden Produkte mit Hilfe von Regeln beschrieben. Regeln lassen sich prinzipiell in zwei Gruppen klassifizieren: Prüfregeln (Plausibilitäten) sowie Berechnungsregeln (Computations). Beide Gruppen (letztlich alle Regeln) werden als Java-Code implementiert. Für die Belange der Fachabteilung existiert „darüber gestülpt" eine Formelsprache, die einfach zu erlernen ist, allerdings (bewusst) nicht alle Freiheiten bei der Implementierung von Produkten erlaubt.

Plausibilitäten (Prüfregeln) schränken u.a. die möglichen Merkmalswerte ein oder setzen diese in Relation zu anderen Merkmalen. So könnte eine Prüfregel definieren, dass die Wohnfläche innerhalb eines Produkts den maximalen Wert von 250qm nicht überschreiten darf (hier wird nur auf ein Merkmal Bezug genommen). Eine andere Prüfregel könnte bestimmen, dass Männer nur bis zu einem Eintrittsalter von 50 Jahren versicherbar sind (hier wird auf zwei Merkmale – Geschlecht und Alter – Bezug genommen; lassen wir Unisex-Tarife einmal außen vor). Leistungsseitige Regeln könnten z.B. prüfen, ob die Umstände des eingetretenen Schadenfalls den versicherten Bedingungen entsprechen.

Berechnungsregeln definieren entweder in Java oder mit Hilfe der genannten Formelsprache Berechnungsschritte, die zur Ermittlung eines definierten Ziels (z.B. einer Prämie oder eines Auszahlungsbetrags) führen.

Alle Regeln werden deklarativ d.h. unabhängig voneinander erfasst. Der Entwickler muss sich damit nicht um die Reihenfolge der Ausführung kümmern. Diese Abhängigkeiten errechnet die B-Factory Regel-Engine aus den innerhalb der Regeln referenzierten Merkmalen. Die Regel-Engine erstellt aus diesen Informationen einen Berechnungsbaum, der bei jeder Zustandsänderung zur Laufzeit des Produktsystems aktualisiert wird.

Neben den beschriebenen Merkmalen und Regeln werden Produkte zusätzlich durch weitere, deskriptive Elemente beschrieben. Hierzu gehören u.a. die Beschreibungen des Leistungsumfangs definierter Bausteine, die z.B. unter Vergleichsaspekten

oder unter Marketings-/Vertriebsaspekten relevant sind (wie man sie heutzutage z.B. bei gängigen Vergleichssystemen im Internet vorfinden kann).

Optionale Erweiterungen

Optional existiert eine Java-Sprach- bzw. Syntaxerweiterung, welche speziell auf die Belange der Produktentwicklung konzipiert und implementiert worden ist. Die Syntaxerweiterung ermöglicht dabei u.a.:

- Einfachere Definition von Entscheidungstabellen
- Typvereinfachungen
- Test-Conditions

Gerade Entscheidungstabellen spielen bei Sachprodukten eine wichtige Rolle, so dass deren einfache und übersichtliche Erfassung die Implementierung und Pflege von Produkten entscheidend vereinfachen können. Neben der klassischen Implementierung mit Java-Bordmitteln kann auf eine übersichtlichere Darstellung zurückgegriffen werden. Das folgende Beispiel möge hierfür als Illustration dienen (Berechnung eines Beitragssatzes und eines Mindestbeitrags aus der Versicherungssumme und der Elektronikart im Rahmen einer Gewerbeversicherung):

```
computation PromilleMindestSatzBasisschutz   {

    switch if (Basisschutz.Versicherungssumme, Basisschutz.Elektronikart ) // IN
              (Basisschutz.Beitragssatz, JahresMindestbeitrag)             // OUT
    {
            //   VersSumme  ,    Elektronikart    : BeitragsSatz, JahresMindestbeitrag
            //   ------------------------------------------------------------------------
            case <= 37500   ,    Elektronikart.A  :  6.05,        125.00;
            case            ,    Elektronikart.B  :  8.80,        175.00;
            case            ,    Elektronikart.C  : 13.20,        200.00;

            case <= 75000   ,    Elektronikart.A  :  3.85,        205.00;
            case            ,    Elektronikart.B  :  6.60,        300.00;
            case            ,    Elektronikart.C  :  9.90,        450.00;

            case <=150000   ,    Elektronikart.A  :  2.75,        263.00;
            case            ,    Elektronikart.B  :  4.95,        450.00;
            case            ,    Elektronikart.C  :  6.60,        675.00;

            default: 0.00, 0.00;
        }
    }
}
```

Produktbaukasten

Der Produktbaukasten definiert ein fachliches Produkt-Modell und implementiert auf dessen Basis die versicherungsfachlichen Grundbausteine der Produkte. Er umfasst dabei alle Sparten (Sach-, Leben-, Kranken- – Voll- und Zusatzversicherung –, Kfz- und Gewerbe-Versicherung) und implementiert alle VU-unabhängigen (also generischen) Merkmale und Regeln.

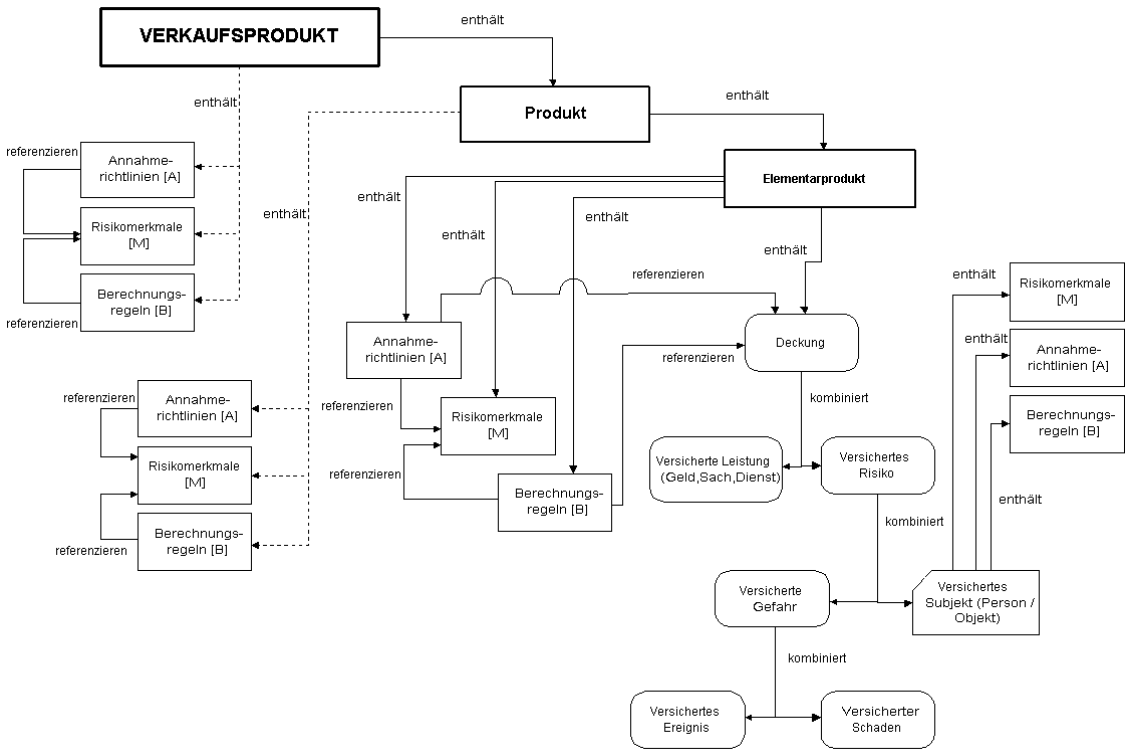
Das Modell dient in einem weiteren Schritt dazu, spartenspezifische Komponenten aus den einzelnen Modellbestandteilen abzuleiten, die für die fachliche und technische Entwicklung individueller Produkte eines Versicherungsunternehmens als Basisbausteine herangezogen werden können.

Das „Bauen" eines Versicherungsprodukts unter Verwendung dieser Basisbausteine erfolgt im Anschluss wie mit einem Baukasten (daher wird hier der Begriff „Produktbaukasten" verwendet):

Aus einer vorgegebenen Grundmenge an Bausteinen (= Komponenten) wird zunächst eine Art Fundament (= Basisstruktur) für das individuelle Produkt erzeugt. Das konkrete Produkt wird dabei als Kombination der unterschiedlichen Teilkomponenten zusammengesetzt.

Zusätzlich enthält der Baukasten diverse Baumuster für weitere, nach individuellen Wünschen kreierbare Bausteine, die auf das Fundament aufgesetzt werden können.

Die Grundstruktur des Modells zeigt die nachfolgende Abbildung.



Werkzeuge

B-Factory implementiert alle Eigenschaften eines Produkts in Java. Als Basis-Werkzeug für die Entwicklung von Produkten dient somit eine (prinzipiell beliebige) Java-IDE[2]. Sollen die in diesem Abschnitt aufgeführten Werkzeuge bzw. die oben genannten optionalen Spracherweiterungen verwendet werden, ist der Einsatz der Entwicklungsumgebung Eclipse (https://eclipse.org/) Voraussetzung, da die Werkzeuge von B-Factory als Eclipse-PlugIns ausgeführt sind.

Produkte können letztlich mit den bewährten Werkzeugen der im Einsatz befindlichen Java-Umgebung entwickelt, getestet und deployt werden. Für IT-Mitarbeiter entsteht damit praktisch kein Einarbeitungsaufwand, da bewährte und aus dem täglichen Einsatz bekannte Tools zum Einsatz kommen. Die Lernkurve beim Einsatz des Produktsystems und/oder des Produktbaukastens ist minimal, da bekannte Entwurfsmuster zum Einsatz kommen und die Struktur des Produktbaukastens klaren Regeln gehorcht.

Die in der Folge genannten Produktsystem-spezifischen Werkzeuge sind als Eclipse-PlugIns implementiert und damit in das Eclipse-Ökosystem eingebettet. Dies betrifft sowohl Werkzeuge für den IT-Entwickler als auch Werkzeuge für den Mitarbeiter der Fachabteilung. Der IT-Entwickler bekommt damit eine nahtlose Erweiterung seines bewährten Tools und kann sein vorhandenes Java-Know-How einsetzen. Der Mitarbeiter der Fachabteilung erhält auf seine Bedürfnisse zugeschnittene, jedoch auf der Basis-Umgebung aufbauende Werkzeuge.

Compiler für Syntax-Erweiterungen

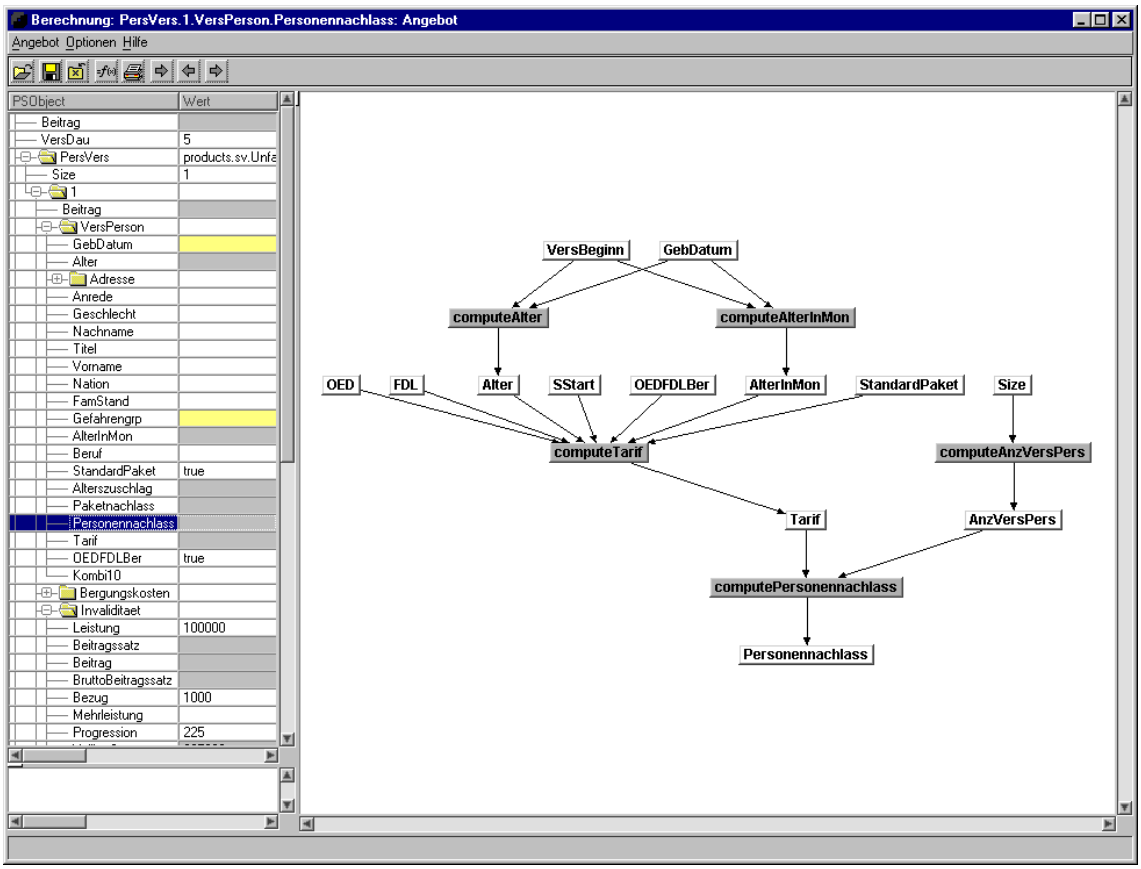
Der Compiler für die Sprach- und Syntaxerweiterungen (siehe z.B. die Erweiterung für Entscheidungstabellen weiter oben) ist als Eclipse-PlugIn implementiert und steht dem Entwickler damit transparent zur Verfügung. Der Compiler erzeugt reinen Java-Bytecode, so dass der Charakter der Java-Sprache und der Umgebung gewahrt bleiben. Durch Erweiterungen auch des Eclipse-Debuggers ist ein Debugging auf Quellcode-Ebene möglich, so dass auch hier die bewährte Arbeitsweise erhalten bleibt.

Produkt-Debugger

Der Produkt-Debugger[3] dient der Visualisierung des Berechnungsbaums sowie des Datenflusses. Er ermöglicht fachlich gerichtete Tests. Die nachfolgende Abbildung zeigt das Benutzerinterface des Produkt-Debuggers.

[2] Da keines der in der Folge genannten Werkzeuge Voraussetzung für den Einsatz des Produktsystems ist, kann auf eine beliebige IDE zurückgegriffen werden.

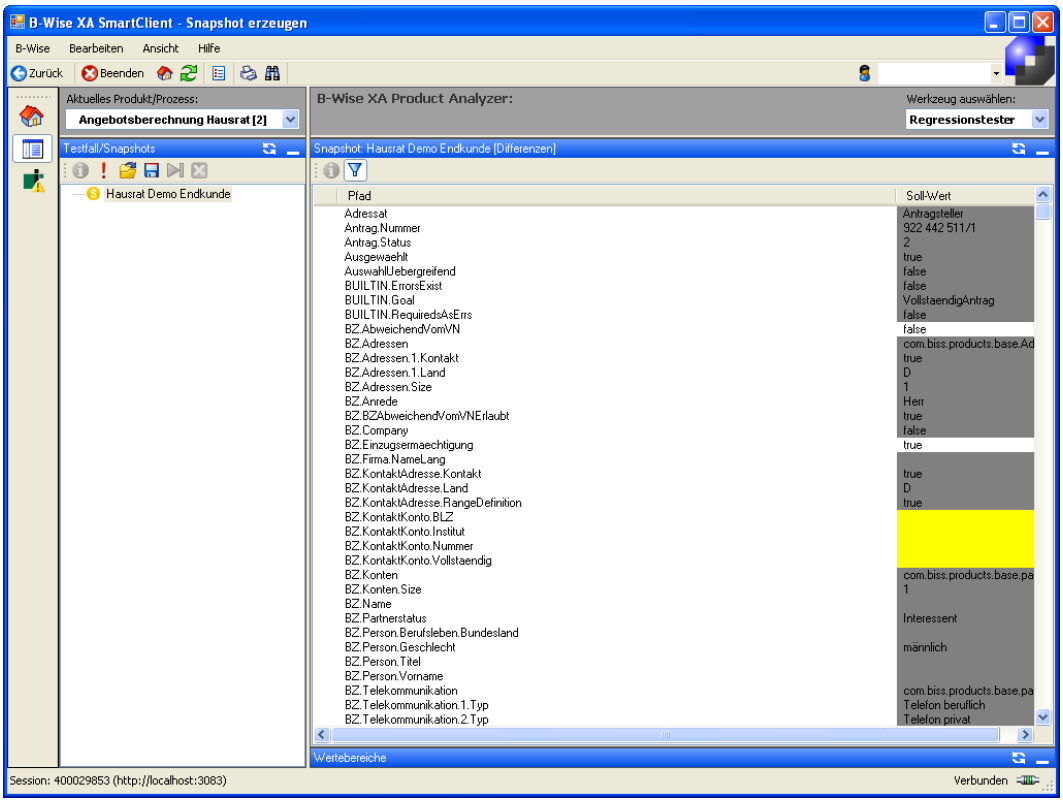
[3] Der Produkt-Debugger ist aktuell ein Stand-alone Tool; die Eclipse-Integration befindet sich aktuell in Entwicklung.



Produkt-Tester (Regressions-Tester)

Der Produkt-Tester[4] dient der Aufnahme und dem Abspielen von Produkt-Testfällen und ist somit als Regressionstester konzipiert. Er ermöglicht die Aufnahme der relevanten Datenkommunikation zwischen Client und Produkt sowie das nachfolgende Abspielen dieser Datenkommunikation und somit den Vergleich der Ergebnisse einer aktuellen mit einer gespeicherten (Referenz-) Version. Die nachfolgende Abbildung zeigt das aktuelle Benutzerinterface des Produkt-Testers.

[4] Der Produkt-Tester ist aktuell ein Stand-alone Werkzeug; die in Eclipse integrierte Variante befindet sich in Entwicklung.



Produkt-Editor

Der (teilweise grafische) Produkt-Editor[5] dient der Erstellung (teils auch Zusammenstellung) von Produkten aus Teilkomponenten sowie der einfachen Definition der enthaltenen Merkmale und Regeln mit Hilfe einer vereinfachten Formelsprache. Alle mit dem Produkt-Editor erstellten Elemente werden letztlich in Java übersetzt und fügen sich damit nahtlos in die Java-Entwicklungs- und Runtime-Umgebung ein.

[5] Der Produkt-Editor befindet sich aktuell in einem recht frühen Entwicklungsstadium.

B-Factory zur Laufzeit

Versicherungsprodukte werden innerhalb von B-Factory zu 100% in Java implementiert. Sowohl B-Factory als auch die damit implementierten Produkte sind daher innerhalb einer Java-Runtime (ab Version 1.6) lauffähig und können insbesondere auch in gängigen Application-Servern wie IBM WebSphere oder JBoss/Wildfly verwendet werden.

Bei der Nutzung von B-Factory gibt es verschiedene Szenarien: Ist die nutzenden Anwendung selbst in Java (oder einer anderen JVM-basierten Sprache) implementiert, so kann B-Factory als Library eingebunden werden, so dass Produktsystem-Instanzen in der eigenen Anwendung erzeugt und per Interface direkt angesprochen werden können.

Alternativ kann B-Factory auch eigenständig betrieben werden (entweder „stand-alone" in einer eigenen VM oder aber „managed" in einem Application-Server). In diesem Fall erfolgt die Kommunikation dann über einen WebService. Diese Betriebsart erlaubt dann auch die Nutzung aus Anwendungen heraus, die nicht auf Basis der Java-VM entwickelt werden (z.B. PHP).

Das Interface sieht in beiden Szenarien ähnlich aus. In der Folge sind beispielhaft die wichtigsten API-Methoden des Java-Interface beschrieben:

connect

```
public UID connect()
throws PSException;
```

Verbindet den Consumer mit dem Produktsystem.

Parameters:
-
Returns:
Global Unique ID der Verbindung
Throws:
PSException

loadProduct

```
public void loadProduct( UID connId, String prodName)
throws PSException;
```

Lädt das Produkt mit dem angegebenen Klassennamen in das Produktsystem.

Parameters:
ID connId - ID der Verbindung
String prodName – vollqualifizierter Name der Java-Klasse mit der Produktdefinition
Returns:
-
Throws:
PSException

setGoal

```
public void setGoal( UID connId, String goalName)
throws PSException;
```

Setzt das Berechnungsziel goalName. Damit wird der interne Berechnungsbaum aufgebaut, d.h. es wird ermittelt, welche Berechnungen notwendig sind, um die Ergebnisvariable goalName zu berechnen und welche Produktmerkmale in diesen Berechnungen referenziert werden. Dadurch kann u.a. bestimmt werden, welche Eingaben notwendig sind, um zum Ergebnis zu gelangen.

Parameters:
ID connId - ID der Verbindung
String goalName – Name der zu berechnenden Ergebnisvariablen
Returns:
-
Throws:
PSException

setValue

```
public boolean setValue( UID connId, String propertyName, String
unformattedValue)
throws PSException;
```

Setzt den Wert eines Merkmals auf unformattedValue. Der Wert wird dabei unabhängig vom Typ des Merkmals immer als unformatierter String übergeben. Ist dieser String keine gültige Repräsentation für den Datentyp der Eigenschaft, wird die Zuweisung trotzdem wirksam; die Eigenschaft geht allerdings in Zustand „erroneous" (fehlerhaft) über.

Parameters:

ID connId - ID der Verbindung
String propertyName – Name des Merkmals (z.B. „VersSumme")
String unformattedValue – unformatierte String-Repräsentation des neuen Wertes des Merkmals (z.B. „200000")

Returns:

-

Throws:

PSException

getValue

```
public String getValue( UID connId, String propertyName)
throws PSException;
```

Fragt den Wert eines Merkmals ab. Der Wert wird dabei als formatierter String übergeben.

Parameters:

ID connId - ID der Verbindung
String propertyName – Name der Eigenschaft

Returns:

formatierte String-Repräsentation des aktuellen Wertes der Eigenschaft

Throws:

PSException

getRange

```
public List<?> getRange( UID connId, String propertyName)
throws PSException;
```

Fragt den für ein Merkmal gültigen Wertebereich ab. Die Operation wird nur für Merkmale vom Aufzählungstyp (EnumProperties) unterstützt.

Parameters:

ID connId - ID der Verbindung
String propertyName – Name des Merkmals

Returns:

Liste der möglichen Werte, die das Merkmal annehmen kann/darf

Throws:
PSException

getChanges

```
public List<ChangeInfo> getChanges( UID connId)
throws   PSException;
```

Fragt den seit der Initialisierung oder dem letzten Aufruf von resetChanges (sofern ein solcher bereits erfolgt ist) als Folge von Änderungen an Eingabewerten geänderten Zustand (Liste von Änderungen) ab.

Parameters:
ID connId - ID der Verbindung
Returns:
Liste der seit der Initialisierung oder dem letzten Aufruf von resetChanges als Folge von Änderungen an Eingabewerten geänderten Merkmalen. Dabei enthalten die gelieferten Änderungsinformationen jeweils den Namen des Merkmals, die Art der Änderung, den aktuellen Wert sowie den aktuellen Zustand (Status wie z.B. „erroneous").
Throws:
PSException

resetChanges

```
public void resetChanges( UID connId)
throws PSException;
```

Setzt die Liste der als Folge von Änderungen an Eingabewerten geänderten Zustände zurück.

Parameters:
ID connId - ID der Verbindung
Returns:
-
Throws:
PSException

getViolations

```
public List<PSViolation> getViolations( UID connId)
throws PSException;
```

Fragt die Liste der seit der Initialisierung oder dem letzten Aufruf von resetViolations (sofern ein solcher bereits erfolgt ist) aufgetretenen Fehlermeldungen ab. Geliefert werden

nur solche Fehlermeldungen, deren Ursache immer noch besteht; Fehler, die aufgrund eines mittlerweile korrigierten Zwischenzustandes aufgetreten waren, sind in der List nicht mehr enthalten.

Parameters:
ID connId - ID der Verbindung
Returns:
Liste der Fehlermeldungen zum aktuellen Produktzustand
Throws:
PSException

resetViolations

```
public void resetViolations( UID connId)
throws PSException;
```

Setzt die Liste der Fehlermeldungen in Bezug auf die Lieferung mit der Methode getViolations zurück. Der Fehlerzustand selbst sowie die Fehlerhaftigkeit der zugehörigen Eingabewerte bleiben bestehen; nur die Fehlermeldung wird aus der Liste entfernt.

Parameters:
ID connId - ID der Verbindung
Returns:
-
Throws:
PSException

writeStateOnStream

```
public void writeStateOnStream( UID connId, OutputStream os)
throws PSException;
```

Schreibt den aktuellen Zustand in den übergebenen Zieldatenstrom.

Parameters:
ID connId - ID der Verbindung
OutputStream os - Zieldatenstrom
Returns:
-
Throws:
PSException

readStateFromStream

```
public void readStateFromStream( UID connId, InputStream is)
throws PSException;
```

Liest einen zuvor gespeicherten Produktzustand ein und überschreibt damit den aktuellen Zustand.

Parameters:
ID connId - ID der Verbindung
InputStream is - Quelldatenstrom
Returns:
-
Throws:
PSException

disconnect

```
public void disconnect( UID connId) throws PSException;
```

Beendet eine Verbindung mit dem Produktsystem.

Parameters:
ID connId - ID der zu beendenden Verbindung
Returns:
-
Throws:
PSException

Eine typische Anwendung, welche das o.g. API verwendet, kann zur Laufzeit dabei wie folgt aussehen:

Zunächst wird zwischen dem nutzenden System (z.B. Angebotssystem) und B-Factory eine Verbindung aufgebaut. Anschließend wird mit Hilfe der Methode `loadProduct` ein definiertes Produkt (z.B. Hausrat) in das Produktsystem geladen. Das Produkt wird dabei automatisch anhand der vorhandenen Regeln initialisiert, ggf. werden Vorbelegungen vorgenommen. Das nutzende System definiert die gewünschte Zielvariable (z.B. den Bruttobeitrag bei einer Tarifierung oder die Vollständigkeit der annahmerelevanten Daten bei der Antragserstellung). Das Produktsystem errechnet aus der Zielvariablen und dem dazu erforderlichen Berechnungsbaum die erforderlichen Merkmale und setzt diese auf den Zustand „required". Das Nutzersystem kann nun mit Hilfe von `setValue` einzelne Werte im Produkt setzen. Nach jedem dieser Aufrufe errechnet das Produktsystem einen neuen Produktzustand, in dem einzelne Werte anhand der implementierten Regeln eingeschränkt, vorbelegt auf „erforderlich" oder „disabled" gesetzt werden etc. So werden z.B. auch Wertebereiche auf alle erlaubten Werte eingeschränkt. Werden bestimmte Merkmale fehlerhaft (wird z.B. im Verlauf einer Berechnung eine Summe überschritten), setzt das Produktsystem nicht nur den Wert auf „fehlerhaft" sondern liefert auf Anfrage die dazu passende Fehlermeldung.

Das Nutzer-System kann die von B-Factory ermittelten Informationen (Werte der Merkmale und deren Zustände) abfragen und z.B. zur entsprechenden Darstellung der Werte innerhalb eines Dialogs verwenden. B-Wise Module wie das B-Wise Angebotssystem zusammen mit dem B-Wise Dialogsystem können diese Informationen unmittelbar für die Dialoganzeige innerhalb eines HTML-Browsers umsetzen (z.B. die Darstellung fehlerhafter Werte in rot zusammen mit der entsprechenden Fehlermeldung).

Nach erfolgreichem Abschluss einer Berechnung (z.B. nach vollständig erfolgter Tarifierung) kann das Nutzersystem definierte (oder alle) Werte in serialisierter Form erhalten, um diese z.B. an ein Dokumenten-Management-System weiterzureichen. Anschließend wird die Verbindung mit `disconnect` beendet.

Über BISS

Die BISS GmbH entwickelt seit über 20 Jahren – auf der Basis multifunktionalen Software-Frameworks B-Wise – umfangreiche maßgeschneiderte Lösungen für die Points of Sales and Service von Versicherungsunternehmen und Finanzdienstleistern. Zum Leistungsspektrum gehören zudem Systemintegration und Beratung. Als Branchenspezialist bietet BISS einerseits die Sicherheit einer in der Praxis bewährten Software-Plattform und andererseits die Möglichkeit, Funktionalitäten, Prozesse und wichtige technische Eigenschaften genau auf die spezifischen Anforderungen abzustimmen.

Zum Leistungsspektrum gehören neben der Analyse und Beratung die Implementierung und Systemintegration sowie die Pflege und der Betrieb der fertigen Systeme. Dabei werden alle fachlichen (Ausschließlichkeit, Makler, Kooperationspartner, Internet) wie auch technischen (PCs, Tablets und Smartphones) Kanäle optimal unterstützt.

BISS-Lösungen sind wie wenige andere dazu geeignet, die Produktivität und Effektivität von Geschäftsprozessen im Vertrieb und Kundenmanagement von Versicherungen und Finanzdienstleistern zu steigern. Neutrale Instanzen wie VersicherungsMagazin, Charta-Qualitätsbarometer, Service-Rating, YouGov-Psychonomics haben die Qualität der BISS-Systeme durch Befragungen, Untersuchungen und Jury-Entscheidungen immer wieder bestätigt.

Der direkte Kontakt zu Ihrem Produktsystem:

BISS
Gesellschaft für Büroinformationssysteme mbH
Marie-Curie-Straße 4
D - 26129 Oldenburg
Telefon: +49 (0) 441 36 10 76-0
Telefax: +49 (0) 441 36 10 76-99
Web: www.biss-net.com
E-Mail: vertrieb@biss-net.com

BISS

Effiziente Systeme für das Kundenmanagement in der Versicherungswirtschaft