Komponenten Architekturen im Versicherungsbereich

Ein Erfahrungsbericht

Jens Coldewey
Coldewey Consulting
Uhdestr. 12
D-81477 München
Tel: +49-89-74995702
Fax: +49-89-74995703
email: jens_coldewey@acm.org
http://www.coldewey.com

Die Deregulierung im Versicherungsbereich hat Umwälzungen ausgelöst, die häufig nicht mehr von den existierenden Systemen abgedeckt werden können. Viele Unternehmen haben sich daher entschlossen, ihre DV-Landschaft zu renovieren. Eine zentrale Aufgabe ist dabei die Festlegung der fachlichen Architektur. Am Beispiel zweier großen Versicherungen schildert dieser Vortrag Vorgehensweisen, sowie die Probleme und Widerstände, die während der Projekte auftraten.

Jens Coldewey ist unabhängiger Berater mit Schwerpunkt Architektur und Objekttechnologie in großen Organisationen. In den letzten Jahren hat er mehrere Versicherungen bei Architekturfragen unterstützt. Herr Coldewey ist Kolumnist in der Zeitschrift Objekt Spektrum.

Durch die rechtlichen Änderungen der letzten Jahre ist der Wettbewerb im Versicherungsbereich zunehmend härter geworden. Die Deregulierung der Märkte gibt den Unternehmen nach Jahrzehnten starren Reglements nun die Gelegenheit, sich über innovative Produkt- und Marketingkonzepte Wettbewerbsvorteile zu sichern. Die wichtigsten Innovationen sind flexiblere Produkte, spartenübergreifende Angebote sowie verbesserte Kundenbetreuung, die unter dem Akronym POS firmiert – je nach Interpretation wird darunter „Point of Service“ oder „Point of Sale“ verstanden, also die komplette Vertrags- und Schadensbearbeitung vor Ort beim Kunden.

Nachdem Finanzdienstleistungen sich heute zu einem wesentlichen Teil auf die Datenverarbeitung (DV) stützen, wurde die existierende DV in vielen Unternehmen zu einem der wesentlichen Hemmschuhe für die Flexibilisierung. Altsysteme waren nicht für die neuen Bedingungen ausgelegt, die Flexibilität der DV wurde innerhalb kurzer Zeit zu einem wichtigen Wettbewerbsfaktor. Zusätzlich zu diesen Anstößen führte die Einführung des Euro und die Jahr 2000 Umstellung der Systeme dazu, daß die gesamte Software erfaßt und angepaßt werden mußte.

Viele Unternehmen nehmen diese dreifache Herausforderung zum Anlaß, neben den unaufschiebbaren Anpassungen der Altsysteme auch über eine grundlegende Renovierung ihrer DV-Landschaft nachzudenken. Dabei geht es nicht so sehr um eine völlige Ablösung der bestehenden Systeme auf einen Schlag – das wäre betriebswirtschaftlich

unsinnig und wegen der damit verbundenen Risiken auch technisch nicht zu rechtfertigen. Vielmehr soll ein allmählicher Übergang ermöglicht werden, der es erlaubt, bestehende Systeme langfristig parallel zu neuer Software zu betreiben.

Eine wichtige Voraussetzung dafür ist eine Facharchitektur, die alle Bereiche des Versicherungsgeschäfts abdeckt und über ausreichend Flexibilität verfügt, um den derzeitigen Trend von der Versicherung hin zum Finanzdienstleister mit tragen zu können. Eine Facharchitektur definiert dabei, aus welchen Systemen und Komponenten sich die Software Landschaft zusammensetzt und wie diese miteinander arbeiten.

Im folgenden berichte ich über zwei verschiedene Projekte, in denen eine solche Architektur mit Hilfe von Komponenten modelliert wurden. In beiden Fällen laufen die Projekte noch, es handelt sich also eher um „Frontberichte“, als um abschließende Bewertungen. Bei beiden Kunden handelt es sich um große europäische Gesellschaften mit umgerechnet mehreren Milliarden DM Prämienaufkommen pro Jahr. Sie sind beide in einzelnen Sparten jeweils nationale Marktführer.

Zum Komponentenbegriff

Bevor ich in die Details gehe, möchte ich zunächst den hier verwendeten Komponentenbegriff klären. Häufig werden unter Komponenten frei zusammensteckbare Bausteine verstanden, die meist abstrakte Dienste anbieten [Sim94]. Wichtig ist dabei vor allem die Middleware, die freie Austauschbarkeit gewährleistet und für die Kommunikation zwischen den Komponenten sorgt. Java Beans ist ein bekannter Protagonist dieser Interpretation.

So wichtig dieser Ansatz für die technische Umsetzung von Komponenten ist, so wenig hilfreich ist er für den Aufbau einer Facharchitektur. Dabei geht es ja primär darum, die Schnittstellen zwischen den Komponenten auch fachlich zu beschreiben und eben nicht völlig frei zusammensteckbar zu gestalten. Vielmehr soll definiert werden, welche Zuständigkeiten bei welcher Komponente liegen, wie die einzelnen Komponenten zusammenarbeiten und wie sie angepaßt und erweitert werden können. Eine Komponente ist hier also eher als ein Paket von Geschäftsobjekten zu verstehen, denn als Anforderung an die Middleware.

Das erste Projekt – Entwurf auf der grünen Wiese

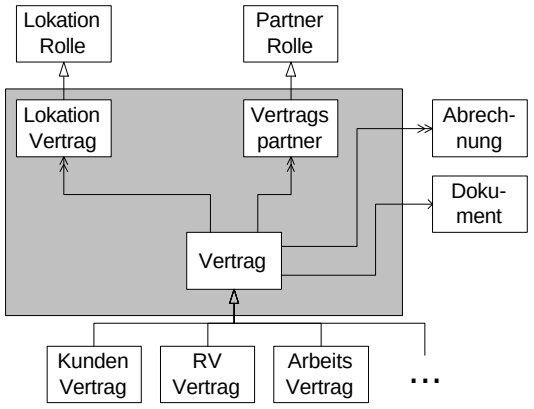
Das erste Unternehmen, von dem hier berichtet werden soll, hatte etwa ein Jahr vor dem Start des Projektes eine größere Fusion hinter sich gebracht. Im Rahmen dieser Fusion fand eine strategische Neuausrichtung der gesamten DV statt. Teil dieser Neuausrichtung war der Entwurf einer Architekturvision, die als Leitschnur zur Planung weiterer Projekte dienen sollte. In einem kleinen Team von vier erfahrenen Architekten bearbeiteten wir die Aufgabe innerhalb von vier Monaten.

Beim Vorgehen orientierten wir uns an Techniken zur objektorientierten Analyse. Die oben beschriebenen Komponenten weisen große Ähnlichkeiten auf mit Packages, wie man sie aus dem objektorientierten Design kennt [Mar95]. Was lag also näher, als die Komponenten auch aus einem groben Objektmodell heraus zu definieren? Wir erstellten

also zunächst ein grobes Modell der wichtigen Geschäftsobjekte unter starkem Einsatz von Analysemustern [Fow97]. Diese Geschäftsobjekte wurden dann nach vier Kriterien zu Komponenten zusammengefaßt:

- Fachliche Zusammengehörigkeit
- Schmale Schnittstellen
- Stabilität und Flexibilität der Objekte
- Erweiterungspotential

Die Größe einer Komponente betrug ca. 3-10 Klassen, die auch nur sehr grob umrissen wurden. Die nebenstehende Abbildung zeigt ein Beispiel für die Komponente „Vertrag“.



Insbesondere die möglichen Erweiterungen stellten uns vor eine erhebliche Herausforderung: Es ist heute kaum absehbar, welche Anforderungen in den nächsten Jahren auf die DV einer Versicherung zukommen. So mußte das Objektmodell sowohl versicherungsfremde Dienste abdecken, als auch einen möglichen Weg hin zu anderen Finanzdiensten aufzeigen. Zu lösen war dieses Problem nur dadurch, daß die Konzepte einen ausreichenden Abstraktionsgrad besaßen und entsprechend erweiterbar waren. So ist in der obigen Abbildung zu sehen, daß die Vertragskomponente nur ganz allgemeine Vertragsdienste anbietet, während die Verträge in anderen Komponenten spezialisiert werden.

Natürlich ignoriert ein solches Objektmodell wesentliche Teile der Fachlichkeit. Das Vorgehen ist allerdings vertretbar, wenn man sich das Ziel des Modells vor Augen führt: Auffinden sinnvoller Komponenten im System. Gerade wegen ihrer Unschärfe konnten die Objektmodelle also nur zum Auffinden, nicht jedoch zur Definition einer Komponente dienen. Statt dessen wurde jede Komponente auf einer einzelnen Seite mit einer einheitlichen Gliederung beschrieben. Am Beispiel des

- Verantwortlichkeit

  Vertrag definiert die allgemeine Schnittstelle, die jeder Vertrag anbieten muß. Zusätzlich enthält er Funktionen zum Verwalten von Verträgen sowie für beliebige Auswertungen über sie

- Exportschnittstelle

  Keine

- Erweiterungsschnittstelle

  Komponenten definieren spezielle Verträge, indem sie Subklassen von Vertrag bilden

- Importschnittstelle

  Spezialisiert PartnerRolle

  Spezialisiert LokationsRolle

  Vertrag wird mit einem Dokument dokumentiert

  Abrechnungen führen zu Zahlungsaufträgen an In- und Exkasso

- Mögliches Objektmodell

- Offene Punkte

Vertrags wird diese Gliederung im Kasten demonstriert. Insgesamt wurden auf diesem Weg 24 Komponenten definiert.

Auf der Basis dieser 24 Komponenten wurde eine sogenannte „Schnittstellenarchitektur“ spezifiziert. Systeme, die Leistungen einer Komponente übernehmen, sollen mittelfristig die Schnittstellen aufweisen, die durch die Komponenten definiert sind. Dies müssen mindestens die definierten Exporte sein, optional auch die Erweiterungsschnittstellen. Diese Festlegung ist der Schlüssel zu einer sanften Evolution der DV-Landschaft in Richtung einer Komponentenarchitektur im Rahmen weiterer Projekte. Werden neue Systeme entworfen, so halten diese sich an die definierten Schnittstellen, die freilich entsprechend verfeinert werden müssen. Benötigen diese neuen Systeme Schnittstellen zu bestehenden System, so werden auch diese konform ausgelegt. Dadurch entstehen im Laufe der Zeit Kapseln um die bestehenden Systeme, die so auch als Komponenten genutzt werden können. Die Ablösung bestehender Software erfolgt nur dann, wenn dies aus anderen wirtschaftlichen Gründen sinnvoll erscheint.

Das Projekt war zunächst als interne Studie gestartet worden, dementsprechend waren die Ergebnisse auch vorsichtig aufgenommen worden. Der Paradigmawechsel von einer unternehmensweiten Datenbank zu einer Komponentenarchitektur mußte erst noch nachvollzogen werden. Mittlerweile orientiert sich jedoch das erste große Projekt an den Überlegungen und ein eigenes Projekt wurde aufgesetzt, um die Einkapselung der Altsysteme auf technischer Ebene zu untersuchen. Dafür wird ein Ausschnitt des Leistungssystems für Personenversicherungen mit Hilfe von Orbix und Java eingekapselt. Die Kopplung zu den CICS-Transaktionen erfolgt über ein selbst geschriebenes Framework. Das System soll im Laufe von 1999 in Produktion gehen.

Der Druck aus dem Management wächst, die Schnittstellen auch auf breiterer Basis umzusetzen und so die Architektur zum Leben zu erwecken. Allerdings muß dabei auch vereinzelt gegen unrealistische Vorstellungen über den benötigten Aufwand angegangen werden. Auch leiden solche visionären Projekte immer wieder unter dem Druck des Tagesgeschäfts, so daß eine Priorisierung nicht immer im Sinne des Pilotprojektes ausfällt.

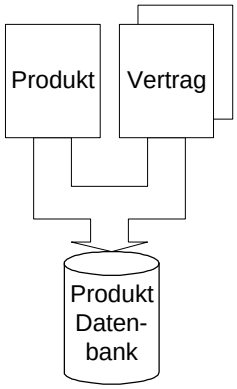
Im großen und ganzen sieht es jedoch so aus, als würde die Architektur zum Erfolg werden. Endgültig läßt sich dies aber erst nach vielen Jahren der Entwicklung feststellen.

Das zweite Projekt – Von der Datenarchitektur zur Komponente

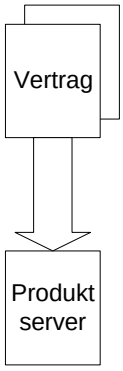
Einige Unternehmen arbeiten schon seit geraumer Zeit an der Erneuerung ihren DV. Die Versicherung, von der ich im zweiten Teil berichten möchte, hat bereits seit fast zehn Jahren ein entsprechendes Projekt zu laufen. Die ursprünglichen Entwürfe sind demnach auch mit den Techniken der achtziger Jahre entstanden, bevor Objektorientierung oder gar Komponenten die notwendige Reife erreicht hatten. So ging die Architektur im wesentlichen von dem Datenmodell aus, das die „Insurance Application Architecture“ der IBM [IAA93] definiert. Die Dynamik wurde zu großen Teilen mit Hilfe eines Regelsystems definiert.

Schon bei den ersten Teilprojekten zeigte sich jedoch, daß dieser Ansatz wegen der fehlenden Kapselung nicht tragfähig genug war, ein so großes System zu realisieren. So wurde die Realisierung selbst mit objektorientierten Techniken vorgenommen, ohne jedoch eine grundsätzliche Anpassung der Architektur vorzunehmen.

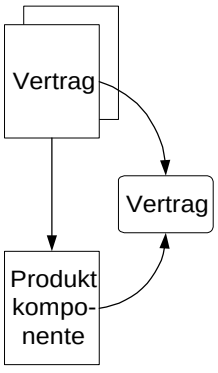
Nachdem Teile der Software für Lebensversicherungen auf das neue System umgestellt waren, bestand die Aufgabe darin, ein System zur Definition neuer Produkte zu entwerfen. Die ursprünglichen Vorgaben waren noch stark auf die (relationale) Datenbank fixiert, wie die nebenstehende Skizze verdeutlicht: Als Schnittstelle des Produktsystems war eine Produktdatenbank vorgesehen, auf die dann die Nutzersysteme zugreifen. Diese Nutzersysteme können Verträge verwalten, wie in der Abbildung, oder auch andere Aufgaben übernehmen, wie z.B. Schadensbearbeitung, oder Außendienstunterstützung.



Nun lassen sich Versicherungsprodukte gut in Form hierarchischer Stücklisten modellieren [SLe96]. Zusätzlich enthält eine Produktinformation nicht nur Daten, sondern auch umfangreiche Regeln zur Berechnung von Tarifen und Regeln. Beides ist jedoch mit einer relationalen Datenbank nur schwer performant zu realisieren. Auf der anderen Seite sind Produktinformationen Stamminformationen, die nicht unbedingt einer Kontrolle durch Transaktionen unterliegen müssen. Der erste Schritt bestand also darin, von der Datenbanksicht abzugehen und eine stärker objektorientierte Sicht auf das Problem zu erreichen, wie sie in der zweiten Abbildung zu sehen ist. Hier stellt ein Produktserver Informationen über die verschiedenen Produkte zur Verfügung und bietet auch spezifische Services an, wie z.B. die Tarifberechnung. Die Speicherung der Produktinformation wird dadurch vollständig zur internen Angelegenheit des Produktservers. Es handelt sich also um ein reflexives System, in dem der Produktserver die Meta-Ebene darstellt, die von den Nutzersystemen interpretiert wird [BMR+96].



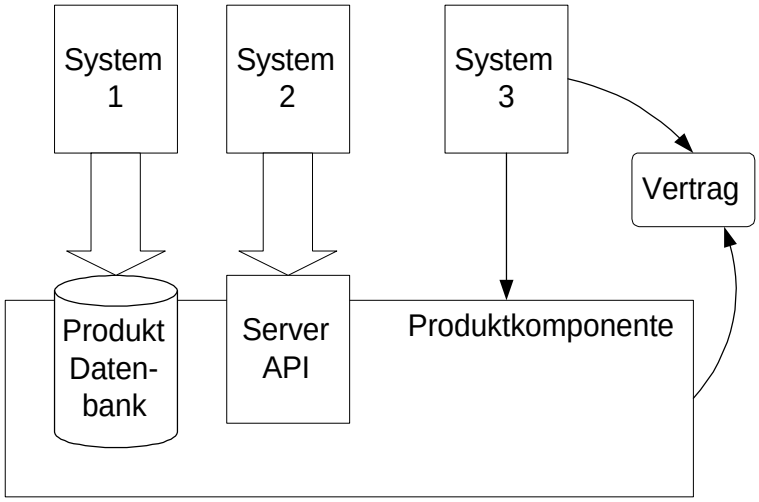
Aber auch diese Sicht erwies sich als nicht tragfähig. Zwar lassen sich so Außendienstsysteme betreiben, die eine relativ geringe Transaktionslast aufweisen. Im Backoffice sind jedoch die Transaktionslasten so groß, daß die Arbeit mit einem reinen Meta-Modell keine ausreichende Performance mehr ermöglicht. Zumindest für Massenprodukte sind komplexere Optimierungen notwendig, welche die Schnittstelle zu den Nutzersystemen weiter verkomplizieren würden. Zudem muß jedes Nutzersystem den Code zur Interpretation der Meta-Informationen enthalten, die der Produktserver bereitstellt. Es entstünde also unnötige Redundanz.



Einen Ausweg aus dieser Misere bietet der nebenstehende Ansatz. Hier ist das Produktsystem kein Server mehr, sondern eine eigene Komponente, deren Aufgabe darin besteht, für die Nutzersysteme performante versicherungstechnische Objekte zu erzeugen, also z.B. Verträge, Schadensakten, Angebote und so weiter. Diese Ob-

jekte kapseln nicht nur das Wissen über das Produkt, zu dem sie gehören, sondern auch technische Optimierungen, die hohe Transaktionslasten sicherstellen. Dadurch entwickelt sich die Produktkomponente zu einer Fabrik für Objekte, die dann von den Nutzersystemen verwaltet werden [GHJ+95].

Im realen Betrieb wäre es jedoch naiv, anzunehmen, daß alle Nutzersysteme diesem Ansatz folgen würden. Immerhin ist ein Teil der Landschaft ja noch ausgehend von der Datenbank entworfen worden. Zusätzlich müssen bestehende Altsysteme bedient werden. Für den Erfolg des Systems ist es daher unabdingbar, daß auch diese Systeme in der Lage sind, ihre produktspezifischen Informationen zu erhalten. Dafür muß die Produktkomponente nach wie vor auch andere Schnittstellen anbieten, wie in der Abbildung unten zu sehen ist. Der wichtige Unterschied zu den zuvor diskutierten Lösungen besteht darin, daß hier die Datenbank ebenso wie die Serverschnittstelle reine Schnittstellen sind, also auf die Architektur der Produktkomponente selbst kaum Einfluß haben. Dadurch lassen sich beliebig viele solcher Schnittstellen an die Produktkomponente anstecken.



Allerdings gab es zunächst erhebliche Widerstände gegen diese Lösung. Je nach Sichtweise handelt es sich hier um den Keim einer Komponentenarchitektur oder um einen Fremdkörper in der existierenden Datenarchitektur. In den Diskussionen stellte sich heraus, daß nach wie vor häufig noch in der unternehmensweiten Datenbank gedacht wird, über die Applikationen miteinander kommunizieren. Die Vorstellung, daß Komponenten auch lebensfähige Objekte übergeben können und nicht nur eine RPC Schnittstelle aufweisen, verwirrt. Viele Vorbehalte beruhten – und beruhen noch immer – auf Mißverständnissen und fehlende Erfahrung im Umgang mit solchen Komponenten.

Ausgeräumt werden konnten viele der Vorbehalte bisher durch ein konsequentes iteratives Vorgehen, das sowohl das Risiko minimiert, als auch Vertrauen schafft in die Leistungsfähigkeit des Teams.

Quintessenz

Die beiden Projekte haben völlig unterschiedliche Ansätze verfolgt, um Komponentengedanken in die Architektur einzubringen. In beiden Fällen zeigte sich, daß mittlerweile

die Techniken zur Verfügung stehen, um saubere Komponenten zu finden. Der Mut zu unkonventionellen Ansätzen ist dabei allerdings eine notwendige Voraussetzung.

Im Umfeld großer DV Landschaften darf keinesfalls der Aufwand vernachlässigt werden, den die Einbindung von Altsystemen verursacht. Hier schlummert die eigentliche technische Komplexität.

Noch wichtiger als die Lösung technischer Probleme ist es jedoch, das politische Umfeld konsequent zu beackern. Gerade visionäre Projekte stehen hier in einem Minenfeld, was von allen Beteiligten hohe Aufmerksamkeit erfordert. Hier können konsequentes Risikomanagement und iterative Entwicklung Entlastung bringen. Sie ersetzen jedoch nicht das Fingerspitzengefühl des Projektleiters im Umgang mit den Sorgen und Ängsten seiner Kolleginnen und Kollegen. Wer umfangreiche Änderungen an der DV-Landschaft eines Unternehmens entwirft, sollte stets Tom DeMarcos Zitat im Kopf behalten: „People hate changes“ – Menschen hassen Veränderungen.

Danksagung

Für die Unterstützung bei der Durchführung der Projekte und der Erstellung dieses Berichts möchte ich den beteiligten Teams danken, sowie Ruth Leuzinger, Christa Schwanninger, Frank Buschmann, Rudolf Donko, Ralph Johnson und Bruce Anderson.

Referenzen

[BMR+96] Frank Buschmann, Regine Meunier, Hans Rohnert, Michael Stal, Peter Sommerlad: *Pattern-Oriented Software Architecture - A System of Patterns;* John Wiley & Sons, Chichester, 1996; ISBN 0-471-95869-7

[Fow97] Martin Fowler: *Analysis Patterns - Reusable Object Models;* Addison-Wesley, Reading, Massachusetts, 1997; ISBN 0-201-89542-0

[GHJ+95] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: *Design Patterns - Elements of Reusable Object-Oriented Software;* Addison-Wesley, Reading, Massachusetts, 1995; ISBN 0-201-63361-2

[IAA93] IBM Coorp.: *Insurance Application Architecture;* Lizensierte Dokumente der IBM, 1993

[Mar95] Robert C. Martin: *Designing Object-Oriented C++ Applications Using the Booch Method;* Prentice Hall, Eaglewood Cliffs, New Jersey, 1995; ISBN 0-13-203837-4

[Sim94] Oliver Sims: *Business Objects - Delivering Cooperative Objects for Client-Server;* McGraw-Hill, Berkshire, 1994; ISBN 0-07-707957-4

[SLe96] Paul Schönsleben, Ruth Leuzinger: *Innovative Gestaltung von Versicherungsprodukten - Flexible Industriekonzepte in der Assekuranz;* Gabler Verlag, Wiesbaden, 1996; ISBN 3-409-19470-3