Komponentisierung einer Produktsuite für die Gesamtbanksteuerung

Dieter Ebhart, msgGillardon AG, dieter.ebhart@msg-gillardon.de

Zusammenfassung

Die Produktsuite THINC von msgGillardon wird in den nächsten Jahren neben einer vorgegebenen Gesamtlösung, die sich durch Konfigurationen kundenspezifisch anpassen lässt, auch als komponentenbasierte Anwendungslandschaft zur Verfügung stehen. Bei dieser kann der Kunde einzelne Komponenten auswählen und zu individuellen Workflows auch unter Verwendung von Fremdkomponenten zusammenstellen. Im vorliegenden Artikel werden die grundlegenden Ansätze sowie die Erfahrungen nach Abschluss der ersten Migrationsschritte hin zu einer komponentenbasierten Anwendungslandschaft beschrieben.

Ausgangsbasis

Im Zuge der Umsetzung der aufsichtsrechtlichen Vorgaben sowie der 14 Grundsätze für die effektive Aggregation von Risikodaten und die Risikoberichterstattung des Basler Ausschusses für Bankenaufsicht [BCBS239] ergeben sich neue Anforderungen an die IT-Systeme im Umfeld der Gesamtbanksteuerung.

Insbesondere aufgrund der aus Grundsatz zwei „*Datenarchitektur und Infrastruktur*“, Grundsatz drei „*Genauigkeit und Integrität*“ und Grundsatz vier „*Vollständigkeit*“ ableitbaren Anforderungen beschäftigen sich viele Banken mit der Bereitstellung zentraler Datenschichten, Data-Warehouses und der Konsolidierung der Berechnungs- und Aggregations-Methoden in der Gesamtbanksteuerung.

Dadurch ergeben sich auch neue Anforderungen an die Standardsoftware in dieser Anwendungsdomäne:

- Die IT-Lösungen müssen sich über einen zentralen Datenbestand versorgen lassen.
- Die Ergebnisse müssen in den zentralen Datenhaushalt zurückgespielt werden.
- Alle Datenänderungen und Anpassungen müssen im zentralen Data-Warehouse vorgenommen werden.
- Es muss nachvollziehbar sein, auf welchem Datenstand eine Risikobetrachtung beruht.
- Nicht zuletzt sollen die eingesetzten Methoden und Algorithmen in der gesamten Lösung einheitlich sein.

Um all dies zu ermöglichen, müssen bei Komplettlösungen wie THINC einzelne Module in anderen Kontexten einsetzbar bzw. die Einbindung fremder Module möglich sein.

Im Folgenden beschreiben wir die Lösungsansätze und die Herausforderungen bei der Umsetzung dieser Anforderungen in der Produktsuite THINC.

THINC ist eine Lösung von msgGillardon für die Gesamtbanksteuerung, die sowohl eine periodische Ergebnisvorschau-Rechnung (im Folgenden EVR) sowie ein ertragsorientiertes Risikomanagement erlaubt.

Herausforderungen

Die gezielte Aufteilung von monolithischen Anwendungen in Komponenten oder Module mit einem klar abgegrenzten fachlichen Leistungsumfang und Datenhaushalt verspricht neben der Erfüllung oben genannter Anforderungen weitere Vorteile:

- Reduktion der Komplexität und der Abhängigkeiten innerhalb sowie zwischen den Anwendungen und damit einhergehend eine höhere Effektivität sowohl in der Entwicklung als auch im Test.
- Höhere Flexibilität durch Austausch einzelner Module.
- Bessere Passgenauigkeit der Produktsuite auf die Anwendungslandschaft der Kunden. Schon vorhandene Module bzw. Komponenten, z.B. zur Berechnung des Adressrisikos, können berücksichtigt werden.

Auf der anderen Seite entstehen ganz neue Herausforderungen durch die Aufteilung einer solchen Produktsuite:

Um eine durchgängige Lösung zu erhalten, müssen die einzelnen Komponenten orchestriert werden. Bei der Orchestrierung sind außer der

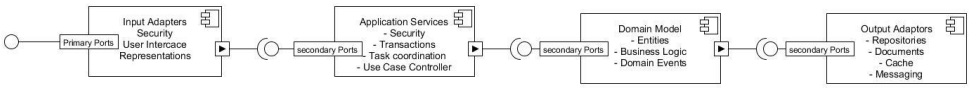

Abbildung 1: Ports and Adaptors

fachlichen Korrektheit auch alle regulatorischen Vorgaben sicherzustellen.

Neben der fachlichen Orchestrierung ist außerdem die technische Integration der einzelnen Komponenten notwendig. Für die Integration der Komponenten müssen ähnliche querschnittliche Konzepte (Benutzerkonzept, Historisierung, Protokollierung, Berechtigungen etc.) verwendet werden.

Obige Herausforderungen setzen ein Software-Ökosystems mit einer definierten technologischen Plattform (technische Infrastruktur und querschnittliche Konzepte mit definierten Schnittstellen) sowie definierten fachlichen Komponenten voraus. Jede Komponente muss die dort beschriebenen Anforderungen erfüllen, um in das Ökosystem aufgenommen zu werden. Wir nennen diese Anforderungen „EcoSystem-Ready“.

Innerhalb dieses Ökosystems lassen sich die „EcoSystem-Ready“ Komponenten zu fachlichen Workflows orchestrieren.

Die Granularität der Komponenten beeinflusst dabei maßgeblich die Lösung:

- **Flexibilität**: Je kleiner die Granularität, umso flexibler, aber auch komplexer wird das Zusammenfügen der Komponenten zu komplexeren Abläufen.
- **Verantwortung**: Die Verantwortung für die Korrektheit eines fachlichen Ablaufs geht auf denjenigen über, der die Komponenten zu fachlichen Abläufen zusammenfügt. Je granularer die Komponenten sind, umso mehr Verantwortung und fachliche Kompetenz benötigt derjenige, der diese orchestriert.
- **Aufwand**: Je kleiner die Granularität der Komponenten gewählt wird umso höher wird die Anzahl der Schnittstellen zwischen dem Ökosystem und den Komponenten und somit auch der Aufwand für die Integration und Orchestrierung.

Stellt man diese Parameter gegenüber, entsteht eine Entscheidungsmatrix, bestehend aus den Dimensionen Flexibilität, Verantwortung und Aufwand, die als Grundlage für eine Entscheidung zur Granularität jeder einzelnen Komponente dienen kann.

Die Granularität der Komponenten kann verfeinert werden, indem die Migrationsschritte iterativ angewendet werden.

Kann die Migration nicht innerhalb eines Releases abgeschlossen werden, ist ein stufenweises Vorgehen erforderlich. Innerhalb eines Releases ist sicherzustellen, dass parallel zur Migration weiterhin Fehlerkorrekturen sowie die Umsetzung neuer regulatorischer Anforderungen bzw. anderer Produkterweiterungen möglich sind.

Lösungsansatz

In [KNHA18] wird die Migration einer großen Anwendung in Microservices in fünf Schritten beschrieben:

- **Schritt 1**: Definition einer externen Service-Fassade.
- **Schritt 2**: Implementieren und Verwenden der Service-Fassade innerhalb der Anwendung.
- **Schritt 3**: Migration der Clients auf die Verwendung der Service-Fassade.
- **Schritt 4**: Einführung interner Service-Fassaden.
- **Schritt 5**: Ersetzen der Implementierung hinter den Service-Fassaden durch Microservices.

In Anlehnung dazu erfolgt die Migration von THINC in fünf Schritten:

- **Schritt 1**: Aufteilung der Domäne EVR in vier große fachliche Komponenten (Planung, Maßnahmen, Szenarien und Simulation).
- **Schritt 2**: Definition der fachlichen Schnittstellen für die EVR-Komponenten.
- **Schritt 3**: Implementierung der Schnittstellen als Service-Fassaden je Komponente.
- **Schritt 4**: Auflösen der Abhängigkeiten der Komponenten untereinander durch interne Service-Fassaden und Auftrennen der Komponenten in eigene Deployment-Einheiten.
- **Schritt 5**: Iterative Anwendung der Schritte 1 bis 4 auf weitere Domänen.

Bei der Implementierung der Service-Adapter werden alle schnittstellenrelevanten Implementierungsteile, wie z.B. Validierungen, Formatwandlungen und Datenveredelung getrennt

von der fachlichen Komponente in eigenen Klassen implementiert. Dieses Grundkonzept ähnelt dem Architekturmuster „Ports and Adaptors“.

Das Prinzip dieses Architekturmusters ist eine saubere Trennung der Fachlogik von externen Systemen wie User Interfaces, Datenbanken oder Messaging. Da die Fachlogik (Domain Model) intern über Anwendungs-Services (Application Services) und nach extern durch Input- und Output-Adaptoren getrennt wird, entstehen so abgeschlossene Komponenten. Durch Ansprechen der Input-Adaptoren ist es möglich, Verarbeitungsketten, die einen fachlichen Workflow abbilden, aus der Verkettung dieser Komponenten zu bilden. Dieses Verketten ähnelt dem Architekturmuster „Pipes and Filters“, das es erlaubt, einen Datenfluss durch die Hintereinanderschaltung von Pipe- und Filter-Elementen flexibel zu konfigurieren. In unserem Fall übernehmen die Pipe-Elemente die Aufbereitung der Roh- oder Ergebnisdaten vom Aufrufer für die Versorgung des Input-Adapters der Folgekomponente. Die Filter-Elemente implementieren die Input-, Output-Adapter sowie die Application-Services und das Domänen-Modell.

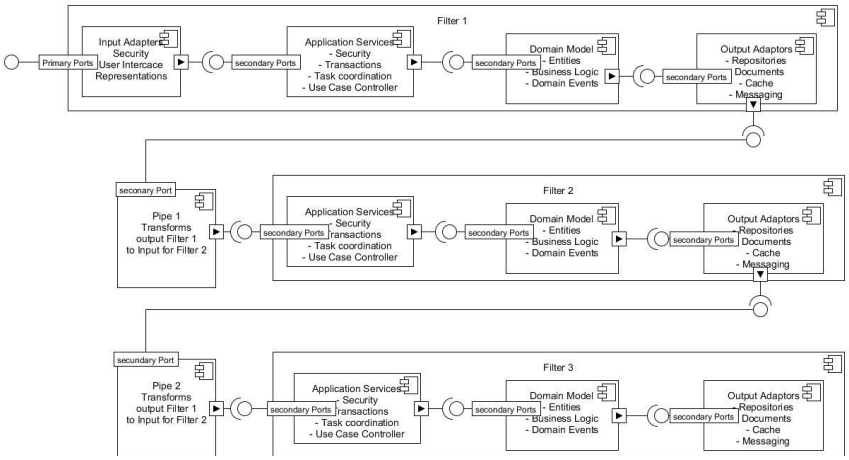

Abbildung 3: Orchestrierte Komponenten

Abbildung 2: Hintereinandergeschaltete Komponenten

Somit können die Komponenten unabhängig von ihrem Kontext durch Anpassung bzw. Austausch der Datenversorgungs-Elemente zu Verarbeitungsketten, sog. orchestrierten Komponenten, zusammengefügt werden. Dies gilt prinzipiell auch für Komponenten von Drittanbietern.

Da eine einfache Hintereinanderschaltung in Kettenform bei komplexeren fachlichen Workflows oft nicht genügt, erfolgt das Zusammenfügen von Komponenten zu Workflows durch eine oder mehrere Orchestrierungs-komponenten. Diese legt die Reihenfolge der Verarbeitungsschritte (also die Reihenfolge, in der die Komponenten aufgerufen werden) fest. Die Reihenfolge der Aufrufe kann dann z.B. regelbasiert erfolgen.

Die Datenversorgungs-Elemente in einer Orchestrierungskomponente greifen nur auf Daten dieser Orchestrierungskomponente zu. Alle notwendigen Datenversorgungen, Datenspeicherungen oder Datenweitergaben zu anderen Komponenten (z.B. andere Orchestrierungskomponenten) werden durch spezifische Versorgungskomponenten (Pipes) implementiert. Die so entstehenden Orchestrierungskomponenten bilden fachliche

Bausteine, die selbst wiederum analog zu dem „Ports & Adapters“-Muster aufgebaut sind.

Somit entstehen unterschiedlich komplexe Bausteine, die sich aufgrund des gleichen Aufbaus auch zu noch größeren Einheiten (z.B. Anwendungen oder „Apps“) verbinden lassen.

Diese Bausteine lassen sich anhand ihrer Granularität unterteilen in:

- **Methoden**: bilden die kleinsten Bausteine. Methoden werden mit Daten aus dem zentralen Datenhaushalt versorgt, führen Berechnungen und Veredelungen von Daten durch und stellen Zwischenergebnisse für die Weiterverarbeitung bereit. Als Beispiel kann die bankfachliche Methode zur Berechnung von Barwerten genannt werden.
- **Workflows**: fassen mehrere Methoden zu einem höherwertigen Dienst zusammen. Sie schalten mehrere bankfachliche Methoden zu einem konsistenten Verarbeitungsablauf zusammen. Als Beispiel kann die Berechnung höherwertiger Kennzahlen – z.B. Value at Risk (VaR) – genannt werden.
- **Anwendungsfälle**: fassen mehrere orchestrierte Workflows zu abgeschlossenen fachlichen Anwendungsfällen zusammen. Als Beispiel kann die Planung genannt werden.

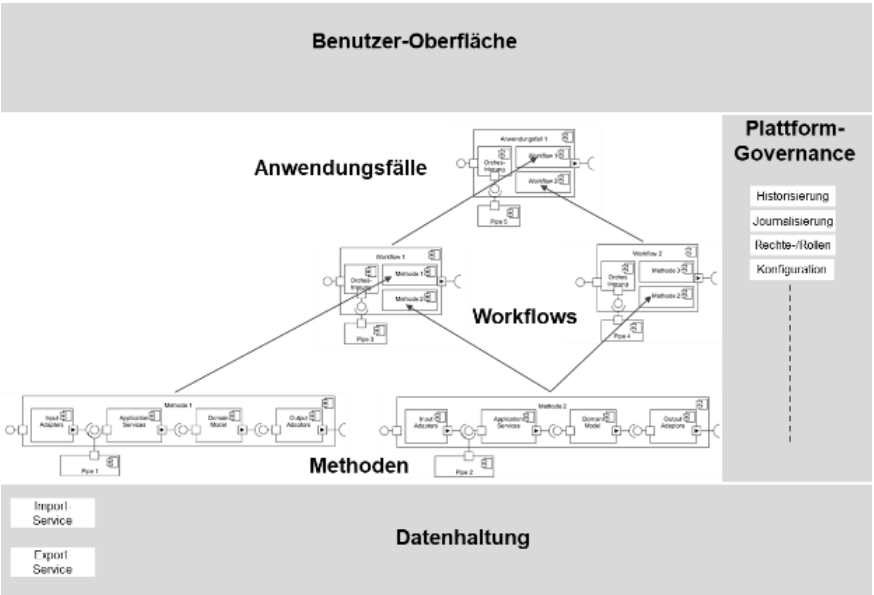

Abbildung 4: Prinzipielle Systemarchitektur

Sind diese Bausteine „Eco-System-Ready“, lassen sie sich in das Ökosystem einbinden, wie Abbildung 4 schematisch zeigt. Das Ökosystem übernimmt dabei die Einbindung der Bausteine in eine Infrastruktur-Umgebung. Dazu zählen wir:

- Die **Benutzeroberfläche**, da Bausteine in verschiedenste Benutzeroberflächen eingebunden werden können.
- Die **Plattform-Governance,** die die Anbindung an eine spezifische Anwendungslandschaft sicherstellt.
- Die **Datenhaltung**, die die benötigten Daten bereitstellt und die Ergebnisse aufnimmt.

Umsetzung

Aktuell haben wir die Schritte 1-3 in oben beschriebenem Migrationsplan für die erste Anwendungsdomäne EVR umgesetzt. Im Zuge der Umsetzung sind wir an einigen Stellen auf unerwartete Herausforderungen gestoßen. Eine kleine Auswahl wird im Folgenden beschrieben.

Vor allem in der Komponente „Planung“ müssen die Eigen- und Kundengeschäfte der Bank übergeben werden. Dabei kommen beträchtliche Datenmengen zusammen, die vor der Ablage in der lokalen Komponenten-Datenhaltung auf Ihre syntaktische und semantische Korrektheit geprüft werden müssen. Um hier eine performante Lösung zu erhalten, wurden die Validierungen in insgesamt vier Kategorien unterteilt:

- **Kategorie 1**: Formelle Prüfungen auf Typkorrektheit und Einhaltung der Wertebereiche (z.B. Prozentwert zwischen 0 und 100).
- **Kategorie 2**: Wertebereich im verwendeten Kontext (Geldbetrag < 1 Mio. €).
- **Kategorie 3:** Kontextprüfungen im selben Geschäftsobjekt (z.B. Wenn Passiv, dann Buchwert <0).
- **Kategorie 4:** Kontextprüfungen über Geschäftsobjekte hinweg (z.B. wenn Abgrenzungsdaten geliefert werden, dann muss bereits ein Cashflow existieren).

Bei den oben beschriebenen Eigen- und Fremdgeschäftsschnittstellen werden während der Datenanlieferung nur die Prüfungen der Kategorien 1 und 2 durchgeführt. Nach diesen Prüfungen können die Daten ohne Syntax-Fehler in der Datenhaltung abgelegt werden. Die zeitaufwändigeren semantischen

Kontextprüfungen der Kategorien 3 und 4 erfolgen erst nach der Datenspeicherung aber noch vor der weiteren Verarbeitung.

Da naturgemäß Komponenten auch in anderen Anwendungsdomänen verwendet werden, steigen die in der Planung zu berücksichtigenden Abhängigkeiten stark an. Um diese zu minimieren haben wir zunächst die Schnittstellensignaturen mit festen Antworten implementiert. Auf Basis dieser Schnittstellenattrappen können die Aufrufer sowohl die Anbindung implementieren als auch erste Funktionstests durchführen. Somit lassen sich die zeitlichen Abhängigkeiten zwischen den einzelnen Projekten stark reduzieren. Dies birgt das Risiko, dass Änderungen an den Signaturen oder beim - insbesondere in den Attrappen nicht implementiertem - Verhalten auf die Verwender durchschlagen können.

Wegen des entstehenden Abhängigkeitsnetzes ist die Einhaltung der Liefertermine jeder Komponente für den Gesamtfortschritt essentiell. Wir haben daher eine Earned Value Analyse in zweiwöchigem Rhythmus in Kombination mit Timeboxing und einem „Design to Budget“ Ansatz eingeführt und damit gute Erfahrungen gemacht. Auftretende Schwierigkeiten konnten frühzeitig erkannt und ihnen damit effizient entgegengesteuert werden.

Aufgrund des vorgegebenen Zeitrahmens und der Größe der Anwendungsdomäne war ein schneller Teamaufbau zu Beginn der Designphase notwendig. Innerhalb kürzester Zeit mussten sich die Entwicklungsteams mit der Aufgabenstellung und den in der Anforderungsanalyse angedachten Lösungen vertraut machen. Um dies sicherzustellen, haben wir den Entwicklerteams für typische Problemstellungen Blaupausen mit einer beispielhaften Implementierung zur Verfügung gestellt. Diese wurden dann von den Entwicklern nur noch an die spezifischen Gegebenheiten angepasst. Zudem haben wir einen Musterkatalog für typische Design-Lösungen im Projekt erstellt. Mit diesen Maßnahmen wurde teamübergreifend eine sehr homogene Codestruktur erreicht und sichergestellt, dass die Vorgaben aus der Systemarchitektur umgesetzt werden.

Fazit

Die Migration einer Produktsuite mit mehr als 500.000 Lines of Code, die weiterhin gewartet und weiterentwickelt werden muss, lässt sich am besten schrittweise durchführen. Der Ansatz, die Migration mit der Definition und Implementierung der Schnittstellen zu beginnen, hat sich bewährt. Durch die Definition der Schnittstellen und deren Verwendung stellt sich frühzeitig heraus, ob die gewählten Komponenten, deren Granularität und die Schnittstellensignaturen auch in der Praxis bestehen.

Eine schrittweise Migration ist auch sinnvoll, um Entscheidungspunkte definieren zu können, an denen der erreichte Stand überprüft und das weitere Vorgehen hinterfragt und ggfs. angepasst oder sogar abgebrochen werden kann.

Neben dem Vorgehen sind die Softwarearchitektur und die Projektsteuerung für den Erfolg einer solchen Migration erfolgskritisch. Im Laufe der Umsetzungsphase kommen immer wieder neue Sachverhalte auf, die eine Anpassung in der Umsetzung erfordern. Anpassungen, die sich auf mehrere, unter Umständen schon fertig implementierte Lösungen auswirken, lassen sich am Einfachsten in möglichst gleichförmigem Code umsetzen. Hier ist im Softwaredesign eine Abwägung zwischen einer möglichst hohen Generalisierung und bewusster Redundanz zu treffen.

Bei der Steuerung eines solchen Vorhabens ist entscheidend, dass frühzeitig Abweichungen und Entscheidungsbedarfe erkannt werden, um rechtzeitig benötigte Entscheidungen treffen zu können. Sehr gute Erfahrungen haben wir damit gemacht, den Planaufwand für die vollständige Umsetzung (also fachliches Design, technisches Design, Realisierung und Test) jedes Features schon bei der Erstellung oder Auswahl einer Lösung mit zu berücksichtigen. Diese Vorgaben dienen im weiteren Projektverlauf als Grundlage für das Projektcontrolling.

Literatur

[BCBS239]: Basler Ausschuss für Bankenaufsicht: Grundsätze für die effektive Aggregation von Risikodaten und die Risikoberichterstattung; Januar 2013

[KNHA18]: Knoche, Holger & Hasselbring, Wilhelm. (2018). Using Microservices for Legacy Software Modernization. IEEE Software. 35. 44-49. 10.1109/MS.2018.2141035.