Der SOA-Entwicklungsprozess

Dierk Pingel

Zusammenfassung

Der SOA-Entwicklungsprozess ist ein System-Entwicklungsprozess, der die Besonderheiten der Service-Orientierung berücksichtigt: das Wiederauffinden von bereits existierenden Services und das Entwickeln neuer Services mit der „richtigen" Granularität.

Die Wiederverwendung von Services garantiert auf lange Sicht eine signifikante Produktivitätssteigerung. Die Geschäftsprozesse treiben den Servicezuschnitt. Der Nutzen von SOA liegt bei den fachlichen Prozessen und nicht in der technischen Architektur.

Ein guter SOA-Entwicklungsprozess beschreibt ein durchgängiges Verfahren von der Spezifikation der Anforderungen, die später den Service-Zuschnitt mit beeinflussen, bis zur Implementierung der Services in SOA-Domänen. Der Übergang von der technologieunabhängigen Systemanalyse zum technologieabhängigen Systemdesign spielt dabei eine besondere Rolle.

1 Schichten-Architektur

Gut strukturierte Systeme setzen sich aus mehreren mehr oder minder deutlich voneinander getrennten Schichten zusammen:

- eine Präsentationsschicht (engl. View-Layer)
- eine Verarbeitungsschicht (Controller-Layer)
- eine Datenschicht (Model-Layer)

Diese als MVC-(Model-View-Controller)-Architektur bekannte 3-Schichten-Architektur wird bei einer SOA (Service-Orientierten Architektur) zu einer 4-Schichten-Architektur. Die Verarbeitungsschicht (Controller-Layer) wird aufgeteilt in eine Transformations- und eine Steuerungsschicht. Die Steuerungsschicht steuert die Verarbeitung und die Transformationsschicht verarbeitet (transformiert) die Daten.

Die Steuerungsschicht repräsentiert den Prozess und die Transformationsschicht den funktionalen Teil einer Anwendung. Die Funktionen werden klassisch realisiert.

Im SOA-Kontext wird die Steuerungsschicht getrennt vom funktionalen Teil entwickelt. Häufig kommt für die Modellierung der Steuerungsschicht die BPMN (Business Process Modeling Notation) zum Einsatz. Aber auch jedes andere Ablaufdiagramm wie z.B. ein Programmablaufplan oder ein UML- (Unified Modeling Language-) Aktivitätendiagramm kann verwendet werden. Der Prozess legt fest, in welcher

Reihenfolge die funktionalen Teile aufgerufen werden. Das Modellieren der Steuerungsschicht nennt man auch Orchestrieren.

2 Vorgehensmodell

Als Vorgehensmodell eignet sich jedes Modell, das eine Trennung zwischen einer technologieunabhängigen und einer technologieabhängigen Modellierung erlaubt.

Innerhalb eines Projektes entscheiden wir uns für ein Wasserfallmodell mit den Phasen Spezifikation, Analyse, Design, Realisierung und Test, mit dem auch sehr große Projekte „gemanagt“ und „controllt“ werden können.

Innerhalb der Phasen arbeiten wir iterativ zwischen den Aktivitäten. Beim sequenziellen Abarbeiten der Phasen sind natürlich auch Rückschritte in vorhergehende Phasen möglich, so dass wir insgesamt von einem LifeCycle-Vorgehensmodell sprechen.

Die Aktivitäten Projekt-, Konfigurations-, Qualitäts- und Testmanagement ergänzen als phasenübergreifende Aktivitäten unseren SOA-Entwicklungsprozess.

Im Gegensatz zu testgetriebenen und agilen Vorgehensmodellen bevorzugen wir einen modellgetriebenen Entwicklungsprozess, bei dem Modelle zunächst einmal spezifiziert und unmittelbar nach Fertigstellung getestet werden. Schon während der Erarbeitung der Modelle achten wir auf eine wirtschaftlich angemessene Qualität. Zusätzliche Quality Gates sichern die Qualität beim Übergang in eine nachfolgende Phase.

Im Unterschied zu Extreme Programming (XP) liegt bei unserem SOA-Entwicklungsprozess der Schwerpunkt auf Analyse und Design und nicht auf Test und Coding. Tests, Coding und Dokumentation werden sofern möglich bei uns aus Modellen generiert.

Große Projekte portionieren wir in Projektinkremente (= Projektstufen), die in einer Spezifikationsphase gebildet werden. Damit erreichen wir ähnlich wie in agilen Ansätzen eine Verteilung großer Aufgaben in kleinere, handhabbare, eigenständig realisierbare Teile. Unsere Projektinkremente entsprechen in etwa den Releases beim Extreme Programming, sind aber mit drei bis zu maximal zwölf Monaten tendenziell länger.

Anders als bei agilen Ansätzen verzichten wir auf eine wöchentliche oder gar tägliche Integration von Systemkomponenten. Wir investieren dafür stärker in Anwendungs- und Systemarchitekturen während der Analyse und des Designs (Stichworte: Wiederverwendung von Services, Organisation der Services in SOA-Domänen).

Mehrere zusammenhängende Projekte fügen wir zu einem Programm zusammen, in dem die Projekte parallel bearbeitet werden (können).

Viele Prinzipien und Praktiken aus agilen Ansätzen (z.B. Akzeptierte Verantwortung, Einbeziehen des Kunden in die Projektarbeit, Kommunikation) werden bei uns dem Thema Projektkultur zugeordnet und sind bei unseren Projekten gelebte Praxis.

3 Spezifikation

Die Systemspezifikation

- grenzt das zu entwickelnde System ab,
- spezifiziert die [Geschäfts-] Prozesse und externen Akteure und
- definiert die Anforderungen an das System.

3.1 Anforderungsmanagement

Das Anforderungsmanagement spezifiziert

- attributive Anforderungen,
- qualitative Anforderungen und
- prozessuale Anforderungen.

Ein System, das eine attributive Anforderung erfüllt, erfüllt diese ganz oder gar nicht. Beispiel: Der Prozess xy druckt Kontoauszüge.

Bei qualitativen Anforderungen wird ein Bereich möglicher Erfüllungsgrade vorgegeben. Performance ist ein gutes Beispiel für eine qualitative Systemanforderung. Ein System kann die vorab definierte *optimale* Performance (z.B. Antwortzeit von 1/10 Sekunde), die *geplante* Performance (0,5 Sekunden) oder "nur" die noch *tolerierbare* Performance (1 Sekunde) erreichen.

Das Spezifizieren von attributiven und qualitativen Systemanforderungen ist nicht SOA-spezifisch und wird daher hier nicht weiter erörtert.

Mit den prozessualen Anforderungen werden die [Geschäfts-] Prozesse spezifizieren. Wir empfehlen, die [Geschäfts-] Prozesse im Rahmen der Prozessmodellierung mithilfe eines Kontextdiagramms zu benennen.

3.2 [Geschäfts-] Prozessmodellierung

Ob mit einer Prozess- oder Geschäftsprozessmodellierung begonnen wird, hängt vom Kontext ab. Wenn die Modellierung im Unternehmenskontext beginnt, sprechen wir von einer Geschäftsprozessmodellierung, andernfalls von einer Prozessmodellierung.

Methodisch nutzen wir für die Geschäftsprozess- und für die Prozessmodellierung dieselben Methodenbausteine, so dass eine Unterscheidung zwischen Geschäftsprozess- und Prozessmodellierung für uns nicht relevant ist. Ohnehin gibt es keine allgemein gültige Definition von Geschäftsprozessen.

Als Methodenbaustein nutzen wir ein Kontextdiagramm in Form eines UML-Anwendungsfalldiagramms mit sehr eingeschränktem Modellelemente-Umfang. Der Anwendungsfall ist gleichzusetzen mit einem [Geschäfts-] Prozess. Durch eine hierarchische Zerlegung der Anwendungsfälle (=Prozesse) über mehrere Kontextdiagramme kann jedes System jeder Größenordnung, also auch ein komplettes Unternehmen, modelliert werden. In der Analyse werden wir dann später zusätzlich noch UML-Aktivitätendiagramme für die Modellierung des Ablaufs von Prozessen verwenden.

Hier ein Beispiel für ein Kontextdiagramm einer fiktiven Bank, der Helgoland-Bank:

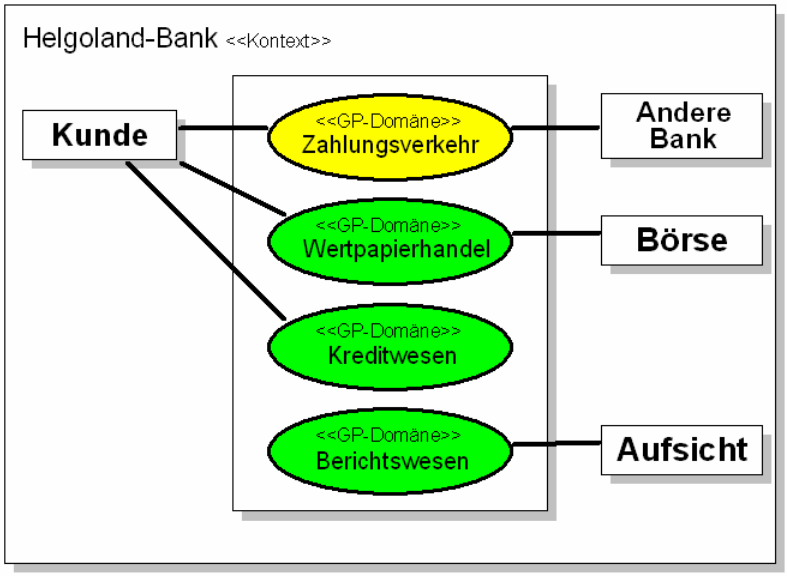

Fig. 1: Prozessmodell (Kontextdiagramm) der Helgoland-Bank (Ebene 1)

Die Zusammenfassung mehrerer Geschäftsprozesse bezeichnen wir mit Geschäftsprozess-Domäne. Als Stereotyp im UML-Anwendungsfalldiagramm verwenden wir *<<GP-Domäne>>*.

Eine Hierarchie-Ebene tiefer würde die "*<<GP-Domäne>>* Zahlungsverkehr" wie folgt aussehen (siehe Fig. 2:

Dort, wo der [Geschäfts-] Prozessname nicht aussagefähig genug ist oder dort, wo Vorbedingungen für den Prozessstart benannt werden müssen, wird eine zusätzliche Prozessbeschreibung benötigt.

Die Prozesse (Anwendungsfälle) im Kontextdiagramm (UML-Anwendungsfalldiagramm) sind grundsätzlich voneinander unabhängig. Vorhandene Abhängigkeiten werden nur über Vorbedingungen in den Prozessbeschreibungen spezifiziert.

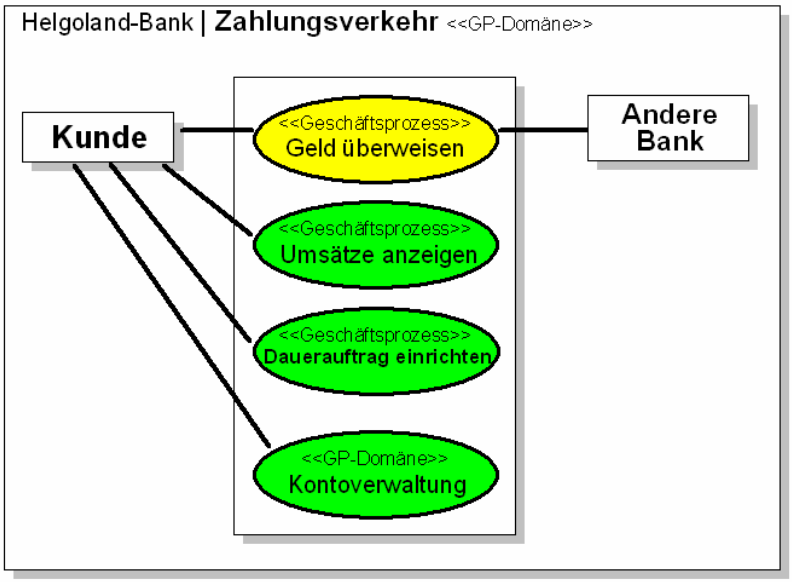

Fig. 2: Prozessmodell „Zahlungsverkehr" der Helgoland-Bank (Ebene 2)

3.3 Datenmodellierung

Wir empfehlen, neben der Prozessmodellierung in der Spezifikation auch schon mit einer groben Datenmodellierung zu beginnen. Damit wird der Informationsbedarf des Systems grob spezifiziert. Als Methodenbausteine verwenden wir UML-Klassendiagramme oder Entity-Relationship- (ER-) Diagramme.

3.4 Ergebnisse der Spezifikation und deren Verwendung

Am Ende der Spezifikation kennen wir die

- die attributiven, qualitativen und prozessualen Anforderungen,
- das Kontextdiagramm als Abgrenzung des Systems mit den [Geschäfts-] Prozessen und den externen Akteuren und
- optional ein Datenmodell.

Diese Informationen reichen aus, um eine Aufwandsschätzung durchzuführen und ein Schneiden des Projektes in Projektstufen (Inkremente) zu ermöglichen.

Durch das Konstrukt einer *<<benutzt>>*-Beziehung im UML-Anwendungsfalldiagramm sind erste Wiederverwendungspotenziale modellierbar.

Wie auch bei der folgenden Analyse werden bei der Spezifikation technologieabhängige Einschränkungen nicht berücksichtigt.

4 Analyse

In der Analyse werden die Schichten des 4-Schichten-Architekturmodells technologieunabhängig modelliert. Technologieunabhängig bedeutet, dass technologische Einschränkungen wie z.B. begrenzter Speicherplatz, Verzögerungen in der Abarbeitung von Prozessen, Warten auf Antworten zur Fortführung von Prozessen usw. nicht berücksichtigt werden.

Technologieunabhängig bedeutet auch, dass in der Analyse noch keine Entscheidungen getroffen werden, ob Menschen oder Maschinen die Systemaufgaben später übernehmen werden. Solche technologieabhängigen Entscheidungen fallen im Design.

4.1 Prozessmodellierung

Das in der Spezifikation bereits erarbeitete Prozessmodell wird bei Bedarf mithilfe von Kontextdiagrammen weiter verfeinert. Je nach Größe des zu analysierenden Systems entsteht in der Regel nach der 2. bis 4. Ebene die Notwendigkeit zum Übergang von einer hierarchischen Zerlegung durch Kontextdiagramme in eine ablauforientierte Modellierung. Hier gibt es verschiedene Modellierungstechniken: Programmablaufpläne (PAP), Ereignisorientierte Prozessketten (EPKs), Diagramme nach der Business Process Modeling Notation (BPMN) und andere. Wir entscheiden uns für UML-Aktivitätendiagramme mit eingeschränktem Modellelemente-Umfang.

Hier in Fortführung des Beispiels aus der Spezifikation eine weitere Prozessverfeinerung mit einem Kontextdiagramm:

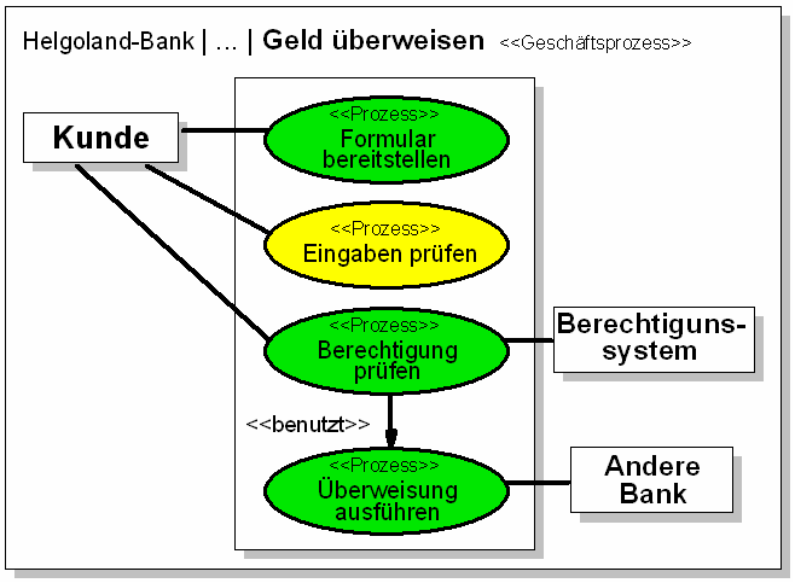

Fig. 3: Prozessmodell „Geld überweisen“ der Helgoland-Bank (Ebene 3)

Der Ablauf des Prozesses “Eingaben prüfen” wird mit einem UML-Aktivitätendiagramm ablauforientiert modelliert:

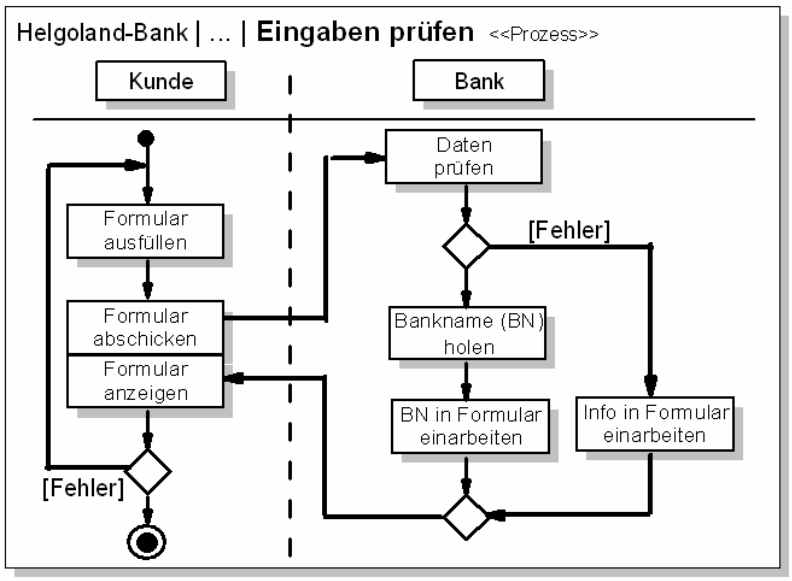

Fig. 4: Prozessmodell „Eingaben prüfen“ der Helgoland-Bank (Ebene 4)

Die Aktivitäten im UML-Aktivitätendiagramm nennen wir Funktionen. Funktionen realisieren die Schicht ‘Transformation’ im 4-Schichten-Architekturmodell.

Sehr wichtige Elemente bei der ablauforientierten Prozessmodellierung sind die Schwimmbahnen (engl. Swimlanes), dargestellt durch gestrichelte Linien Mit ihnen werden Verantwortlichkeiten für die Abarbeitung der Aktivitäten festgelegt.

4.2 Funktionenmodellierung

Komplexere Funktionen können erneut durch UML-Aktivitätendiagramme detaillierter spezifiziert werden. Das sollte aber die Ausnahme bleiben.

Sollte der Funktionsname die Funktion nicht ausreichend genau beschreiben, wird eine zusätzliche Funktionsbeschreibung benötigt.

4.3 Datenmodellierung

Mit der Datenmodellierung wird der Informationsbedarf des Systems beschrieben. Klassen- bzw. Entity-Relationship-Modellierung ist hinreichend bekannt, so dass an dieser Stelle keine detaillierten Erläuterungen erfolgen.

Wichtig ist, dass die Datentöpfe (Entitäten bzw. Klassen) benannt, die Abhängigkeiten zwischen ihnen (Beziehungen) modelliert und die Datenfelder (Attribute) beschrieben sind.

Für SOA ist es zusätzlich notwendig, die inhaltlich nahe stehenden Datentöpfe zu einer Gruppe zusammenzustellen (zu "clustern"). Diese Datencluster bilden später die Basis für den Zuschnitt der SOA-Domänen, z.B. Partnerverwaltung und Kontoverwaltung.

Hier ein Beispiel eines Datenmodells mit Cluster (gestricheltes Rechteck mit Namen):

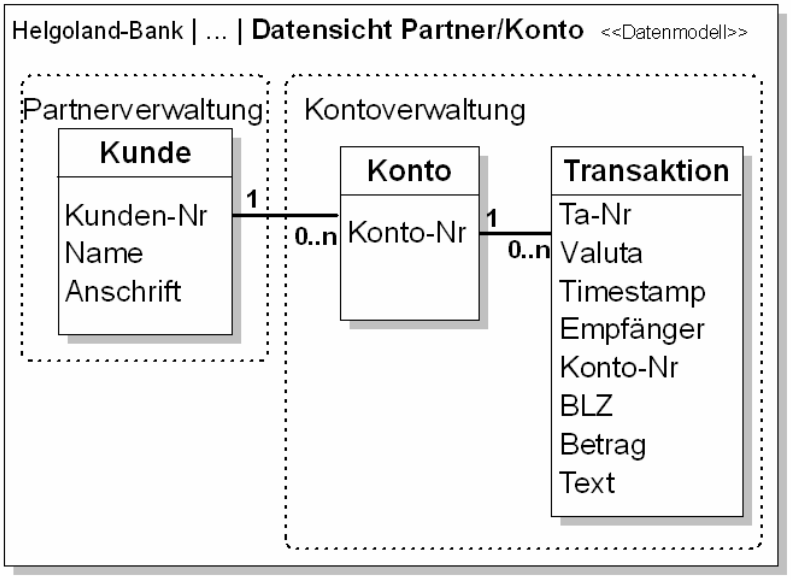

Fig. 5: Datenmodell-Ausschnitt der Helgoland-Bank – Datensicht Partner/Konto

4.4 Schnittstellenmodellierung

In den Prozessmodellen sind die Schnittstellen schnell erkennbar: im Kontextdiagramm gibt es Schnittstellen beim Überschreiten der Systemgrenze; im Aktivitätendiagramm beim Wechseln einer Schwimmbahn.

Mensch-Maschine Schnittstellen

Mensch-Maschine-Schnittstellen sind in den meisten Fällen GUIs (Graphical User Interfaces) oder Reports / Berichte. Hier gibt es keine SOA-spezifischen Aspekte zu beachten.

Maschine-Maschine Schnittstellen

Die Datenflüsse dieser Schnittstellen sind technologieunabhängig zu beschreiben. Im Design wird entschieden, wie die technologieabhängige Abbildung aussehen wird.

4.5 Ergebnisse der Analyse und Verwendung

Mit der Analyse erhalten wir technologieunabhängige Beschreibungen unseres Systems. Da diese Beschreibungen technologieunabhängig sind, spielte SOA bisher auch kaum eine Rolle. Durch die Art und Weise der vorgestellten Modellierung haben wir aber bereits die Grundlagen für einen sinnvollen Servicezuschnitt („Granularität“) gelegt.

Wir haben im Aktivitätendiagramm eine Schichtentrennung zwischen Transformation (repräsentiert durch Aktivitäten = Funktionen) und Steuerung (= Ablauf = Abfolge der Aktivitäten = Prozess) erreicht.

Wir haben Prozesse über die UML-Anwendungsfallmodellierung geschnitten. Getriggert werden die UML-Anwendungsfälle (= [Geschäfts-]Prozesse) durch die externen Akteure. Im Kontext eines Unternehmens sprechen wir von Geschäftsprozessmodellierung.

Das Datenmodell mit Clustering antizipiert den Zuschnitt der SOA-Domänen. Diesen Begriff werden wir später noch näher erläutern.

5 Design

Im Design treffen wir technologieabhängige Entscheidungen. Wir entscheiden, was manuell und was maschinell ausgeführt werden soll. Bei den maschinellen Systemteilen entscheiden wir über Technologien und Architekturen, die zum Einsatz kommen sollen.

Im Fortgang dieses Artikels entscheiden wir uns für Service-Orientierte Architekturen (SOA).

5.1 Der Übergang von der Analyse zum Design

Technologische Restriktionen

Es gibt vielfältige technologische Restriktionen, die beim Übergang von der Analyse zum Design berücksichtigt werden müssen:

- Begrenzte Leistungsfähigkeit der Implementierungsplattformen, z.B. begrenzte Performance und begrenzter Speicher
- Einschränkungen beim Datenbanksystem, z.B. läßt eine relationale Datenbank keine n:m-Beziehungen zu
- Unterbrechungen in der Verarbeitung
- Reihenfolgen von Datenfeldern, ... usw.

Zusätzliche technische Prozesse und Funktionen

Immer werden auch zusätzliche technische Prozesse und Funktionen benötigt, die im Design – da technologieabhängig – zu modellieren sind:

- Logging- und Protokollierungsfunktionen
- Authorisierungsprüfungen (Ist ein Service-Nehmer berechtigt, einen Service zu nutzen?)
- usw.

Technologieschichten

Ein gutes Design zeichnet sich u.a. dadurch aus, dass die technologieunabhängigen Systemteile (aus der Analyse) und die technologieabhängigen voneinander getrennt bleiben. Wir empfehlen eine "Zwiebel-" Schichtenarchitektur, bei der im Kern die technologieunabhängigen (fachlichen) Komponenten stehen und die technologieabhängigen (technischen) Komponenten einen Ring darum bilden.

5.2 Methodenbausteine des Designs

Im Design werden dieselben Modellierungsarten verwendet wie in der Analyse: Prozess-, Daten-, Funktionen- und Schnittstellenmodellierung. Diesmal werden aber technische Aspekte berücksichtigt, z.B. die Reihenfolge und Positionen von Datenfeldern in Übergabeschnittstellen und Bildschirmmasken, zu verwendete Protokolle, usw.

5.3 Prozesse/Funktionen werden Service-Prozesse/ Service-Funktionen

In der Analyse haben wir bereits Prozesse und Funktionen modelliert. Prozesse spezifizieren den Ablauf der Verarbeitung (Steuerung) und rufen die Funktionen, aber auch andere Prozesse auf. Funktionen verarbeiten die Daten (Transformation). Funktionen dürfen nach unseren Überführungsregeln keine Prozesse aufrufen.

Prozesse aus dem Analyse-Prozessmodell überführen wir 1:1 in Service-Prozesse des Design-Prozessmodells. Bei den Funktionen sieht das anders aus. Hier kann es sein, dass eine fachliche Funktion durch genau eine Service-Funktion implementiert wird, aber auch durch mehrere. Das folgende Metamodell zeigt diese Zusammenhänge mit Hilfe eines UML-Klassendiagramms: (siehe Fig. 6:).

Die Möglichkeit, Service-Prozesse zu größeren Service-Prozessen zusammenzubinden, wird hier nicht weiter thematisiert.

Das bereits in der Analyse erstellte Prozessmodell (siehe Fig. 4:) soll in das Design überführt werden. Aus der Spezifikation wissen wir, dass eine Web-Anwendung benötigt wird. Wir identifizieren einen Service-Prozess, der den Ablauf kapselt und mehrere Service-Funktionen, die die Daten-Transformationen übernehmen.

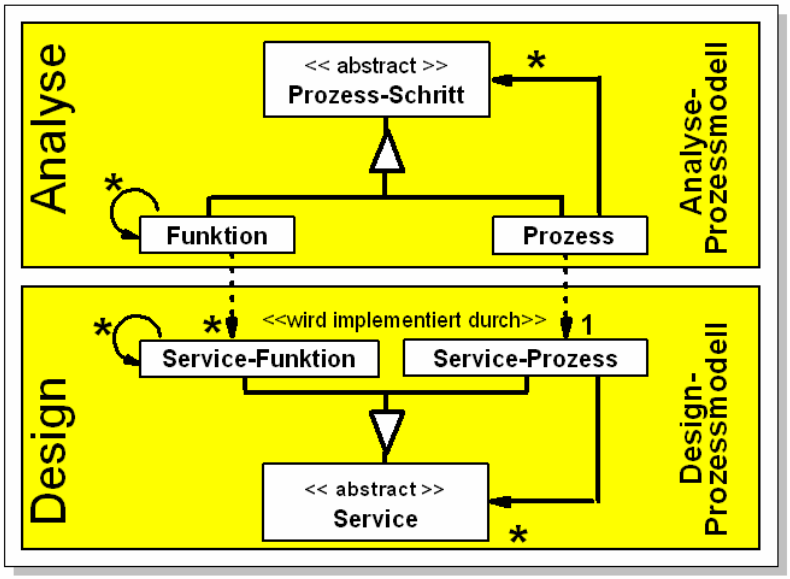

Fig. 6: Metamodell des Prozessmodells – Übergang Analyse zu Design

Wir entscheiden uns, zwei Funktionen aus dem Analysemodell zu einer Service-Funktion zusammenzufassen (gestricheltes Rechteck):

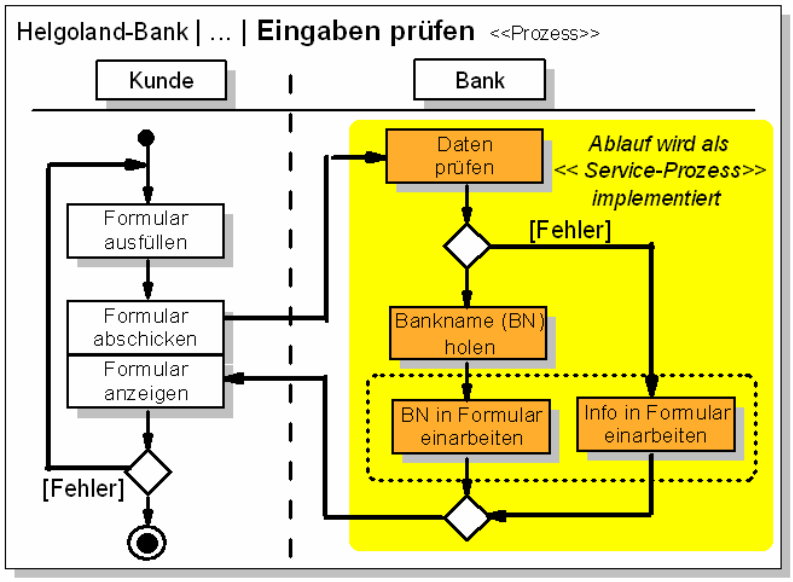

Fig. 7: Überführung Analyse-Prozessmodell in das Design-Prozessmodell

Aus der Funktionsbeschreibung zu „Daten prüfen“ wissen wir, dass folgende Aufgaben zu erfüllen sind:

- Plausibilität der Eingaben prüfen
- Sperrvermerk prüfen
- Dispo- und Wochenlimit prüfen

Für die Funktionen „Sperrvermerk prüfen“ und „Dispo- und Wochenlimit prüfen“ gibt es bereits Service-Funktionen.

Die beiden Funktionen ‚BN (Bankname) in Formular einarbeiten“ und „Info in Formular einarbeiten“ fassen wir zur Service-Funktion „Infos in F. (Formular) einarbeiten“ zusammen.

Die Funktion „Bankname (BN) holen“ gibt es bereits als Service-Funktion mit dem Namen „Bankname suchen“ in der SOA-Domäne Partnerverwaltung.

Das Ergebnis der Überführung sieht wie folgt aus:

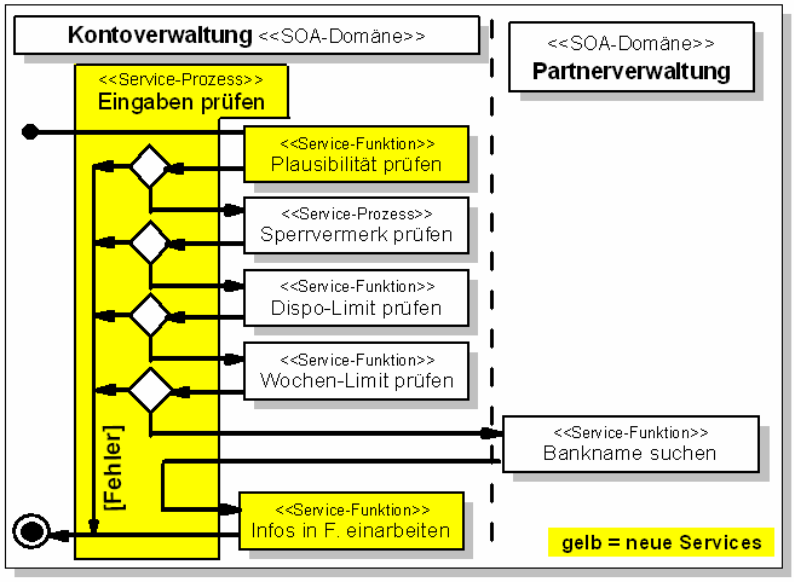

Fig. 8: Design-Prozessmodell

5.4 Bildung von SOA-Domänen

Im Laufe der Zeit werden viele Services entstehen. Es ist daher notwendig, die Services zu ordnen, um insbesondere deren Wiederverwendung und deren Management effizient zu gestalten. Eine sehr gute Strukturierung wird im Rahmen der Analyse mit dem Datenmodell geschaffen. Zusammengehörende Datentöpfe (Entitäten, Klassen)

werden geclustert und bilden dadurch eine eigene SOA-Domäne (z.B. Partnerverwaltung oder Kontoverwaltung).

Die Service-Funktionen werden - sofern möglich - an die Datentöpfe angebunden. Hier kommt das objektorientierte Paradigma zum Tragen, bei dem die Methoden (Funktionen) Teil der Klassen bzw. Objekte sind. Service-Funktionen, die sich nicht an Datentöpfe binden lassen (z.B. 'Bonität beurteilen') werden entweder einer vorhandenen SOA-Domäne zugeordnet oder, wenn das auch nicht sinnvoll ist, einer (ggf. neuen) zentralen SOA-Domäne zugefügt. Das Gleiche gilt für Service-Prozesse.

Damit stoßen zwei Paradigmen aufeinander. SOA ist ein prozessorientiertes Paradigma. Das objektorientierte (OO-) Paradigma ist datenorientiert (Strukturierung über Klassen). Beide Paradigmen werden benötigt und ergänzen sich in dem hier vorgestellten Entwicklungsprozess.

5.5 Orchestrierung

Sichtbares Zeichen einer SOA ist für viele Tool-Hersteller das graphische Modellieren der Prozesse. Mit Graphikeditoren werden die Services zu einem Ablauf zusammengesetzt.

Da bereits in den 90-er-Jahren Werkzeuge zur Modellierung und zum Ausführen von Prozessen entwickelt wurden, wird SOA manchmal auch nicht mit "Service-Oriented Architecture", sondern mit "Same Old Architecture" übersetzt.

5.6 Synchronisation Analyse und Design

Idealerweise haben Design-Entscheidungen keinen Einfluss auf die Analyse. In der Praxis lässt sich dieser Einfluss aber nicht gänzlich vermeiden. Stellen wir beispielsweise beim Design fest, dass es bereits Services gibt, die eine bestimmte Funktionalität bereitstellen, sollten diese Services verwendet werden und ggf. die Analyseergebnisse angepasst werden. Unakzeptabel ist es aber, Analyse und Design ohne Trennung gleichzeitig zu betreiben. Die Trennung 'Technologie-Abhängigkeit' und 'Technologie-Unabhängigkeit' ist für die Wartung und für spätere Portierungen auf andere Implementierungsplattformen von sehr hoher Relevanz.

5.7 Ergebnisse des Designs und Verwendung

Nach dem Design liegen alle Ergebnisse vor, die für eine Realisierung notwendig sind. Fachbereiche und IT können mit Hilfe von „Schreibtischtests" schon vor der Realisierung prüfen, ob die fachlichen Anforderungen erfüllt werden.

Bevor neue Services realisiert werden, sollte unbedingt nochmals überprüft werden, ob es bereits bestehende Services gibt, die wiederverwendet werden können. Ein Metadatenmanagement-System mit deskriptiven Metadaten und die bereits erarbeiteten Modelle helfen, eine Wiederverwendung zu identifizieren und die Wiederverwendungsmöglichkeit zu prüfen. Zur Überprüfung der Wiederverwendungsmöglichkeit sind auch Service-Level-Agreements (SLAs) zu berücksichtigen.

6 Realisierung und Tests

Realisierung und Tests unterscheiden sich nicht wesentlich vom klassischen Vorgehen bei modular aufgebauten Systemen.

Bei der Realisierung sind Vorgaben der Implementierungsplattform zu beachten, z.B. die Wahl der Protokolle, Namensräume, die Verfahren zum Suchen und ggf. Binden von Services.

Eine mögliche Implementierungsplattform für Services sind Web Services auf Basis von WSDL, SOAP, XML und HTTP.

Häufig bieten kommerzielle SOA-Implementierungsplattformen einen Graphikeditor an, mit dem der (Service-)Prozess mittels BPMN (Business Process Modeling Notation) orchestriert werden kann. Aus der BPMN-Beschreibung kann BPEL (Business Process Execution Language) generiert werden. BPEL ist eine XML-basierte Sprache, die Workflow-Engines ausführen können. Ein BPEL-Prozess kann selbst wieder als WebService implementiert werden. Durch die graphische Orchestrierung können (Service-)Prozesse schnell erstellt und geändert werden. Die graphische Orchestrierung ist daher der klassischen Programmierung von Prozessen vorzuziehen.

7 Fazit

Der vorgestellte SOA-Entwicklungsprozess beginnt bei den [Geschäfts-]Prozessen und führt „automatisch“ zu Services mit der „richtigen“ Granularität.

Wir unterscheiden zwischen Service-Funktionen und Service-Prozessen. Die Service-Funktionen sollten hinsichtlich der Qualität ihres Zuschnitts mithilfe von Qualitätsmerkmalen (z.B. lose Kopplung, starke Bindung, explizite Schnittstellen) nochmals überprüft werden. Ggf. sind die Service-Funktionen und -Prozesse anders zu schneiden.

Service-Prozesse werden im SOA-Umfeld häufig nicht mehr programmiert sondern orchestriert. Service-Prozesse können durch die Orchestrierung schneller geändert und durch einen Service-Bus überwacht werden. Insgesamt führt das zu einer höheren Flexibilität der Implementierung von fachlichen [Geschäfts-] Prozessen.

Ein wichtiges Argument für den Einsatz einer SOA ist die Wiederverwendung von Services und damit eine Produktivitätssteigerung in Hinblick auf Wartung, Fehlerbehebung, Dokumentation, Test und Entwicklung. Ein Metadatenmanagement-System hilft, Services zu finden.

Neben einer auf Zusammenarbeit zwischen Fachbereich und IT ausgelegten Projektkultur ist das methodische (systematische) Vorgehen erfolgskritisch für SOA-Projekte. Der beschriebene SOA-Entwicklungsprozess ist ein einfaches, praxiserprobtes Verfahren für Projekte jeder Größenordnung.