„Offline Strategie“-Patterns für mobile SOA Prozesse

Thomas Ritz, Jakob Strauch

Fachbereich Elektrotechnik und Informationstechnik
FH Aachen
Eupenerstr. 70
52070 Aachen
ritz@fh-aachen.de, strauch@fh-aachen.de

Abstract: Service-orientierte Architekturen, kurz SOA, haben die Phase überzogener Erwartungen überstanden und gewinnen immer mehr Bedeutung bei der Integration von Legacy Systemen und der Wiederverwendung von Geschäftslogik. Die unzuverlässige Datenverbindung bei mobilen Anwendungen verhindert hingegen eine kontinuierliche Nutzung bzw. eine garantierte Verfügbarkeit solcher Services. Im Rahmen dieses Beitrags lassen sich zwei Kompensations-Strategien ableiten, die sich lokationstransparent implementieren lassen. Die verwendeten Kommunikations- und Dienstarten bestimmen die Anwendbarkeit der Lösungen. Diese Lösungen werden im Folgenden durch Patterns beschrieben bzw. aus bereits existierenden Patterns abgeleitet und können somit als generalisierter Bauplan für Implementierungen dienen.

1 Motivation

Die Einbindung von Außendienstmitarbeitern in die innerbetriebliche IT wird mit zunehmend leistungsfähigerer mobiler Infrastruktur (Endgeräte, Netzwerke etc.) immer interessanter. Im betrieblichen Kontext wird dies als mobile Unternehmenssoftware bezeichnet (vergl. [Rit2007]). Parallel entdeckt die IT Branche das Serviceparadigma für sich (vergl. [LeKlNi2009]) und nutzt entsprechende Technologien (SOA) oder Vertriebsmodelle (*Software as a Service*, kurz SaaS).

Die unzuverlässige (drahtlose) Konnektivität (vergl. [B'Fi2005]) mobiler Clients führt allerdings dazu, dass externe Services zeitweise nicht erreichbar sind. Dem wird in der Regel mit selbst konstruierten und implementierten Mechanismen entgegengewirkt. Dies lenkt aber das Augenmerk von den eigentlichen Geschäftsprozessen weg und ist zudem wenig methodisch. Um keine weitere Individual-Lösung zu schaffen, sondern eine wiederverwendbare und technologieneutrale Lösung darzustellen, bietet sich die Verwendung sogenannter *Patterns* an. Die resultierenden Patterns beschreiben die Kompensations-Strategien der „Offline Situation“ so, dass diese in den Hintergrund der eigentlichen Geschäftslogik tritt. Im nachfolgenden Abschnitt erläutern wir in aller Kürze den Pattern-Begriff und stellen die Zusammenhänge zwischen Nachrichtenaustausch, Service Klassifikation und der möglichen Kompensations-Strategien vor. Auf der Basis bereits etablierter Patterns sowie des zuvor ermittelnden Kontextes beschreiben wir im Anschluss die „Offline Strategie“ Patterns.

2 Stand der Technik

2.1 Patterns

Patterns beschreiben ein *wiederkehrendes* Problem und beschreiben den *Kern* einer Lösung, sodass die Lösung an unterschiedliche Anwendungs-Kontexte angepasst werden kann. Man nutzt dazu eine semiformale Notation, um das bewährte Lösungswissen zu dokumentieren (vergl. [Bus2009]). Patterns werden nebst ausdrucksvollen *Namen*[1] mindestens durch die Beschreibung eines *Problems*, des zugrunde liegenden *Kontextes*, der *Lösung* sowie dessen *Konsequenzen* charakterisiert (vergl. [BuHe2007b]).

2.2 Nachrichtenaustausch

***M**essage **E**xchange **P**atterns*[2] beschreiben (z.B. mit UML Sequenzdiagrammen) die *Sequenz* sowie *Richtung* und ggf. *Kardinalität* auszutauschender Nachrichten (vergl. [Jos2008]) in einem verteilten System. Das *Request/Response* MEP stellt die Standard Vorgehensweise beim Konsumieren eines *lesenden* Services (vergl. 3.1 Informationsfluss) dar. Strikt *schreibende* Service-Operationen können das *One-Way* Pattern verwenden, bei dem der Konsument seine Nachricht abschickt und nicht auf eine Antwort wartet. Komplexere MEPs, z.B. das asynchrone *Request/Callback*, bauen im Wesentlichen auf den Basis-Pattern *Request/Response* oder *One-Way* auf, sodass eine weiterführende Betrachtung keine Relevanz zur Ermittlung des Kompensationspattern-Kontextes haben.

2.3 Service Klassifikation

Quelle	Aspekt	Klassifizierungen		
N. Josuttis et. al	Informationsfluss	*Reading*	*Writing*	
N. Josuttis et. al	Granularität/Zustand	*Basic (Data/Logic)*	*Process*	*Composed*
D. Krafzig et. al	Zielgruppe/Sicherheit	*Internal*	*Public*	
N. Josuttis	Verwendungszweck (grob)	*Technical (Infrastructure)*	*Business*	
T. Erl	Verwendungszweck (fein)	*Business*	*Process*	*Application*
		Integration	*Proxy*	*Controller*
P. Allan	Wertschöpfung	*Commodity*	*Territory*	*Value-added*

Tabelle 1 - Morphologischer Kasten „Service Klassifikationen“

[1] Teils metaphorisch, wie „Beobachter“, „Erbauer“ oder „Fassade“, um die Kommunikation zu fördern
[2] Der hier verwendete Pattern-Begriff, bezieht sich *nicht* auf die eingangs eingeführten „Lösungsmuster“

Tabelle 1 führt mögliche Klassifikationen für Dienstarten auf (vergl. [Jos2008]). Diese Auflistung erhebt keinen Anspruch auf Vollständigkeit, stellt jedoch die wesentlichen Blickwinkel dar. Um den Kontext des jeweiligen Patterns einzugrenzen, werden in den nachfolgenden Abschnitten die technisch orientierten Klassifikationen sowie die MEPs näher betrachtet. Zuvor werden jedoch noch die generellen Ansätze zur Kompensation eines Netzausfalles vorgestellt.

2.4 Kompensationsstrategien

Um der Forderung der losen Kopplung u.a. gerecht zu werden, kommunizieren Anwendungen und Dienste in einer SOA, wie bereits erwähnt, über Nachrichten (vergl. [Erl2009]). Sind der Übermittlungskanal oder der Dienst nicht verfügbar, oder ist die Funk-Verbindung von geringer Qualität, so kann die Anfrage ggf. zurückgestellt und zu einem späteren Zeitpunkt erneut gesendet werden[3]. Eine alternative Vorgehensweise ist es, die Anfrage an einen anderen, erreichbaren Dienst mit semantisch äquivalenter Funktionalität zu stellen, insbesondere einer lokal verfügbaren Version[4]. Wiederum können manche Dienstleistungen weder aufgeschoben noch an eine (lokal) verfügbare Version umgeleitet werden, da Informationen ad hoc benötigt werden oder Dienste nicht replizierbar sind. Offensichtlich ist die Auswahl dieser Strategien von

- der *Art der Kommunikation (MEPs)* sowie
- der *Art des Dienstes (Service Klassifikation)*

abhängig. Der folgende Abschnitt stellt diese Abhängigkeiten detaillierter dar.

3 Service Klassifikation und Kommunikation

3.1 Informationsfluss

Strikt *schreibende* Services können ein *One-Way* MEP verwenden, d.h. dass beide Strategien prinzipiell in Frage kommen können. Digital ausgefüllte Formulare oder Unterschriften lassen sich beispielsweise aus Sicht des Geschäftsprozesses problemlos zu einem späteren Zeitpunkt versenden, wenn eine Datenverbindung wieder vorhanden ist. Auch eine Replikation ist denkbar, sodass die Ergebnisse des schreibenden Serviceaufrufes auf einer darunterliegenden Persistenz-Schicht temporär gespeichert werden, die im Verbindungsfall synchronisiert wird. *Lesende* Services nutzen das *Request/Response* MEP, da sie naturgemäß ein Ergebnis erwarten. Da letztendlich der Endnutzer zeitnah auf Informationen aus dem Backend angewiesen ist, ist das Zurückstellen von *lesenden* Anfragen in der Regel nicht zielführend. Nur in wenigen (Anwendungs-) Fällen kann der Benutzer über eine verspätete Antwort benachrichtigt werden (*Request/Callback*).

[3] Im Folgendem „Anfragewarteschlange“ genannt
[4] Im Folgendem „Replikation“ genannt

3.2 Verwendungszweck und Granularität

SOA Services werden primär zur einfachen Wiederverwendung geschäftlicher bzw. fachlicher Dienste eingesetzt (sogenannte *Business Services* nach Josuttis/Erl). Dennoch werden auch technische oder infrastrukturelle Services eingesetzt. Lediglich *Business Services* unterschiedlichster Granularität sind Bestandteil mobiler Anwendungen. Die Business Services mit der kleinsten Granularität implementieren die fundamentalsten Dienstleistungen einer SOA und sind nicht abhängig von weiteren Diensten. Diese Basisdienste werden in *Basic Data Services* [5] und *Basic Logic Services* unterteilt (vergl. [Jos2008], [KrBaSl2008]). Erstere erfüllen fundamentale CRUD[6] Operationen auf Geschäftsentitäten, jedoch aus fachlicher Sicht, d.h. dass insbesondere keine Details über die Datenzugriffs-Schicht bekannt sein sollten. In Bezug auf die Replikation der Dienstleistung muss die darunterliegende Datenschicht mit gängigen Synchronisationsmechanismen[7] (ggf. nur teilweise) lokal persistent gehalten werden.

Dies ist auf mobilen Geräten jedoch nicht immer praktikabel, da der (Daten-)Kontext zu umfangreich für die Datenübertragung und -haltung sein kann (vergl. [B'Fi2005]). Die Replikation des eigentlichen Dienstes ist zwar technisch einfach, da sich lediglich die Zugriffsart und -Lokation ändert, jedoch ist die Replikation abhängig von den *Mengen- und Zeitgerüsten* der Daten (vergl. [Rit2007]). Ist vor allem die Aktualität der Daten für den mobilen Prozess von signifikanter Bedeutung, so ist diese Vorgehensweise nicht praxistauglich. Praktikabel ist diese Kompensations-Strategie daher bei Daten, die sich nicht häufig ändern.

Services höher Granularität erfüllen nach [Jos2008] eher integrative Dienste oder werden mit Hilfe der Service Choerografie zu langläufigen, zustandsbehafteten Diensten komponiert. Abhängige Services, Zustände und Systeme müssten mobil replizierbar sein, was sich in den meisten Fällen nicht realisieren lässt oder zumindest unwirtschaftlich ist. Lediglich die initiale Anfrage an einen derartigen Service (sozusagen als auslösendes Ereignis) lässt sich ggf. zurückstellen (*One-Way* MEP). Die Unterschrift eines Kunden kann beispielsweise ein Auslöser für einen Fakturierungs-Prozess auf „Serverseite“ sein.

4 Offline Strategie Patterns

XML/SOAP Webservices haben sich als de facto Standard zur Implementierung von Service-orientierten Architekturen etabliert. Diese Popularität liegt u.a. in der breiten Standardisierungs- und Toolunterstützung begründet (vergl. [MeEb2008], [Erl2009], [Jos2008]). Eine Referenzimplementierung des *Request Queue Pattern* (zweite Variante) wurde mit Hilfe der .NET *SOAP Extensions* realisiert[8]. Das *Smart Service* Pattern kann in jeder Webservice-fähigen objektorientierten Sprache einfach umgesetzt werden.

---

[5] Auch als Business Entity - oder Entity-centric Business Services bezeichnet (vergl. [Erl2008])
[6] **C**reate, **R**ead, **U**pdate, **D**elete
[7] SyncML, Microsoft RDA / Sync Services, Open Mobile Sync
[8] http://msdn.microsoft.com/en-us/library/esw638yk(VS.85).aspx

4.1 Smart Service Pattern

Smart Service ***Wie kann man die mobile Nicht-Verfügbarkeit eines entfernten Dienstes kompensieren?***	
Kontext	Mobile Clients wollen *Basis* Dienste mit *replizierbaren Kontext* nutzen
Problem	Der entfernte Dienst ist zeitweise nicht verfügbar
Lösung	(s.u.)
Konsequenzen	- Der Dienstaufruf bleibt lokationstransparent - Der Service-Kontext muss ebenfalls lokal verfügbar sein
Beziehung zu	Strategy, Facade (vergl. [Gam2008])

Tabelle 2 - Steckbrief Smart Service Pattern

Die Schnittstelle der (i.d.R. generierten) Proxyklasse *(RemoteService)*, die die Nachrichtenvermittlungs-Schicht verbirgt und den Zugriff auf einen entfernten Dienst lokationstransparent hält, dient als Vorlage für ein zu erstellendes Interface[9] *(IService)*. Die lokale Version *(LocalService)* implementiert diese Schnittstelle mit äquivalenter Logik. Ein Stellvertreter *(ServiceFacade)* implementiert ebenfalls das Interface und dient als dynamische (transparente) Fassade, d.h. dass es die Anfragen (durch Aufruf einer Service-Operation) an den *LocalService* oder den *RemoteService* weiterleitet.

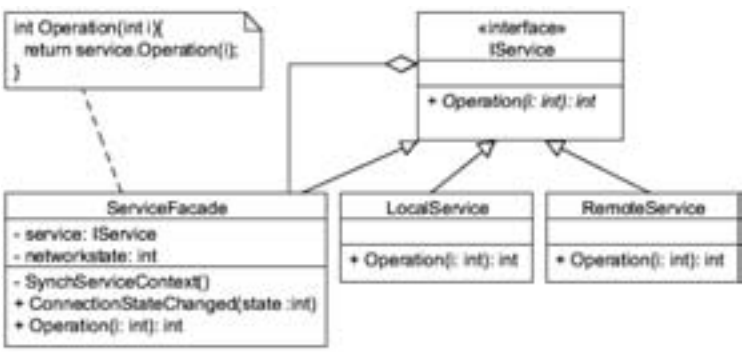

Abbildung 1 - Smart-Service Pattern

Über ein Ereignis *(ConnectionStateChanged)* kann die Fassade informiert werden, dass sich der Status der Datenverbindung geändert hat. Infolgedessen kann die Referenz auf den konkreten Service „umgebogen" werden. Abhängig vom Verbindungszustand und -Qualität kann der Service-Kontext (etwa benötigte Daten) entsprechend synchronisiert werden *(SynchServiceContext)*, um möglichst aktuell zu bleiben. Adäquate Mechanismen sind Teil aktueller Forschungsbemühungen, z.B. die Verwendung einer ressourcenschonenden Synchronisation auf Attributbasis.

[9] Die gängigen Entwicklungsumgebungen erstellen bereits automatisiert passende Stub-Interfaces beim Einlesen der Servicemetadaten. Alternativ unterstützen Refactoring Mechanismen diesen Prozess.

4.2 Request Queue Pattern

Anfragewarteschlange (engl. Request Queue Pattern) ***Wie kann man die mobile Nicht-Verfügbarkeit eines entfernten Dienstes kompensieren?***	
Kontext	Mobile Clients wollen Dienst(-Operationen) nutzen, die auf dem *One-Way* MEP basieren. Das schließt auch komplexere darauf aufbauende MEPs ein (Bsp. *Request/Callback*)
Problem	Der entfernte Dienst ist zeitweise nicht verfügbar
Lösung	Aufschieben von Anfragen (Warteschlange)
Konsequenzen	- Verspätet eintreffende Nachrichten können veraltet sein - Ein Konfliktmanagement im Backend ist i.d.R. notwendig
Varianten	- Der Aufrufkontext (OOP) wird zwischengespeichert - Die (XML) Nachrichten werden „abgefangen“ und gesichert

Tabelle 3 - Steckbrief Request Queue Pattern

Eine *ServiceFacade* implementiert die gleiche Schnittstelle wie der *RemoteService* (vergl. *Smart Service* Pattern). Zusätzlich aggregiert die Fassade eine (clientseitige) Warteschlange, indem ggf. Anfragen zurückgestellt werden können. Abbildung 2 zeigt dies exemplarisch durch das Zwischenspeichern der Methodensignatur. In der Praxis sind jedoch plattformspezifische Mechanismen wie Delegation, Introspektion oder dynamischer Invokation notwendig. Bei erneut hergestellter Konnektivität, findet der eigentliche Aufruf der entfernten Methode statt, indem die *ServiceFacade* ereignisbasiert über die Statusänderung informiert wird.

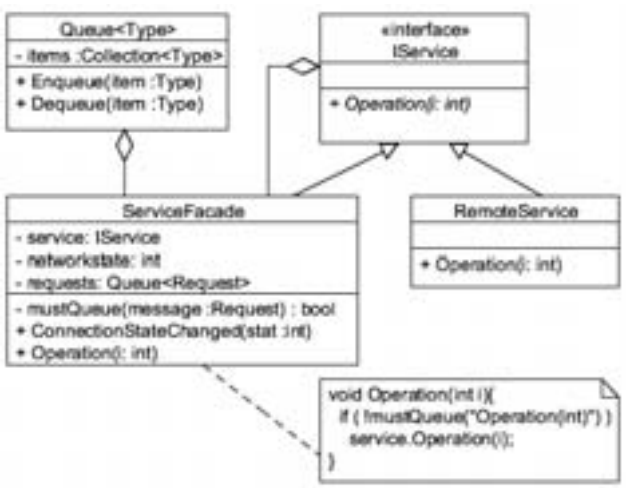

Abbildung 2 - Request Queue Pattern

5 Fazit und Ausblick

Aus dem Blickwinkel der Software-Architektur führt die Anwendung der beschriebenen Patterns zu dem bereits bekannten *Fat Hybrid Client*, d.h. eine Mischung aus Online/Offline Client. Dieser wird nun nicht individuell entwickelt, sondern basierend auf getesteten Entwurfsmustern „komponiert“. Ein gewisser Mehraufwand bleibt, da Teile der entfernten Businesslogik (Services) als auch die darunter liegende Datenschicht oder andere Abhängigkeiten repliziert werden müssen. Die vorgestellten Arbeiten sind Teil einer Pattern Language für mobile Unternehmenssoftware, die zurzeit im BMBF geförderten Forschungsprojekt „Mobile Patterns as a Service“ erarbeitet wird. Ziel ist es, gängige Problemstellungen der mobilen Software-Entwicklung im Bereich „Service und Instandhaltung“ in Form von Patterns zu lösen und entsprechende Werkzeuge zur Konfiguration (=Patternanwendung) zu entwickeln. Beispiel-Implementierungen der Patterns wurden bereits mit .NET Technologien realisiert. Nächster Schritt ist u.a. eine passende Werkzeug-Unterstützung zu entwickeln, die diese Kompensationsstrategien in der Entwurfsphase mit einbeziehen können. Im Hinblick auf den Mehraufwand, der durch die hier vorgestellte Replikation von Service-Funktionalität verursacht wird, bieten sich desweiteren modellgetriebene Ansätze zur Realisierung an.

Literaturverzeichnis

[B'Fi2005] B'Far, R.; Fielding, R. T.: Mobile computing principles. Designing and developing mobile applications with UML and XML. Cambridge Univ. Press, Cambridge, 2005.

[BuHe2007b] Buschmann, F.; Henney, K.; Schmidt, D. C.: On patterns and pattern languages. Wiley, Chichester, 2007.

[Bus2009] Buschmann, F.: A system of patterns. Wiley, Chichester, 2009.

[Erl2009] Erl, T.: SOA design patterns. Prentice Hall, Upper Saddler River, NJ, 2009.

[Gam2008] Gamma, E.: Design patterns. Elements of reusable object-oriented software. Addison-Wesley, Boston, 2008.

[Jos2008] Josuttis, N. M.: SOA in practice. [the art of distributed system design]. O'Reilly, Beijing, 2008.

[KrBaSl2008] Krafzig, D.; Banke, K.; Slama, D.: Enterprise SOA. Service-oriented architecture best practices. Prentice-Hall, Upper Saddle River, NJ, 2008.

[MeEb2008] Melzer, I.; Eberhard, S.: Service-orientierte Architekturen mit Web Services. Konzepte - Standards - Praxis. Spektrum Akad. Verl., Heidelberg, 2008.

[LeKlNi2009] Lenk, A. et al.: What´s inside the Cloud. https://wiki.gridx1.ca/twiki/pub/Main/VirtualizationProjectHome/An_Architectural_Map_of_the_Cloud_Landscape.PDF.

[Rit2007] Ritz, T.: Die benutzerzentrierte Entwicklung mobiler Unternehmenssoftware. In (Gesellschaft für Informatik Hrsg.): MMS 2007: Mobilität und mobile Informationssysteme. 2nd conference of GI-Fachgruppe MMS, 2007.