Mobil mit OSGi und DBF

SOA-basierte NoSQL-Lösung im Mobile-Umfeld

Michael Hüttermann, Daniel Schneller

Dieser Artikel zeigt, wie Bewährtes zielgerichtet mit Modernem kombiniert werden kann, ohne einer buzzword-getriebenen Entwicklung oder einer unnötig komplexen Lösung durch den Einsatz zu vieler Frameworks zu erliegen.

Eine mobile Applikation zur Datenerfassung baut auf der Plattform Windows CE6 und IBM J9 Java VM auf. An die Lösung existieren insbesondere die folgenden Anforderungen:

- *Netzwerkunabhängigkeit:* Alle Daten müssen auf dem Gerät vorgehalten werden, um auch ohne Netzwerkverbindung einsatzfähig zu bleiben.
- *Robustheit:* Bei unerwartetem Neustart muss jederzeit ein konsistenter und möglichst aktueller Datenbestand gewährleistet sein.
- *Performance:* Die Lösung muss so schnell sein, dass eine im Akkord stattfindende Erfassung von Daten nicht behindert wird.
- *Ressourcen:* Die Lösung muss auch gemäß der eingeschränkten Ressourcen im mobilen Umfeld (Rechenleistung, Arbeitsspeicher) tragfähig sein.

Von SQL zu NoSQL

Es existieren SQL-Datenbanken, wie Apache Derby Embedded oder Microsoft SQL Server Compact, die unter JavaME einsetzbar sind. Die Wenigsten verfügen dabei aber über einen zu JavaME kompatiblem JDBC-Treiber [JSR169]. Trotz theoretischer ACID*-Kompatibilität lassen sich bei diesen Datenbanken auf einem mobilen Endgerät zudem operative Probleme mit Daten nicht vollständig ausschließen, was insbesondere bei einer großen Zahl geografisch weit verstreuter Geräte eine gegebenenfalls technisch anspruchsvolle Fehlerbehebung unpraktikabel macht.

Eine leichtgewichtige Alternative zu einer SQL-Datenbank sind strukturierte Datendateien. Zum Einsatz kann dabei die unter der LGPL verfügbare xBaseJ-Bibliothek [xBaseJ] kommen. Diese bietet die nötige Funktionalität, um Dateien im dBase-Format (DBF-Dateien, [dBase]) mit fester Satzstruktur sowie zugehörige Indizes anzulegen, zu durchsuchen und zu modifizieren. Mehr zur Entstehung und zu den Hintergründen von xBaseJ findet sich unter [xBaseJBuch].

> Alle Implementierungen der in Abbildung 1 dargestellten Schnittstellen sind über OSGi Declarative Services realisiert, werden zur Laufzeit u. a. abhängig vom konkreten Hardwaretyp aufgelöst, und könnten – zumindest theoretisch – ebenso zur Laufzeit ausgewechselt werden.

* ACID (Atomicity, Consistency, Isolation, Durability): Atomarität, Konsistenzerhaltung, Isolation und Dauerhaftigkeit. Steht im Gegensatz zu BASE (Basically Available, Soft-State, Eventual Consistency), siehe [Brew00].

Architektur und Hotspots der Umsetzung

Das vorliegende Gesamtsystem skizziert ein Produkt, das aus einer Mischung aus grundsätzlichen architektonischen Erwägungen („horizontal slicing“) und anwendungsfallgetriebener Vorgehensweise („vertical slicing“) evolutionär konstruiert werden kann. Das Gesamtsystem lässt sich in verschiedene Teilsysteme zerlegen, von denen das mobile Endgerät eines ist und hier näher betrachtet werden soll (s. Abb. 1).

Das Teilsystem „Mobile“ besteht im Wesentlichen aus einem umfangreichen Softwaresegment und den DBF-Dateien, in denen Stamm- und Bewegungsdaten vorgehalten werden. Das Softwaresegment setzt sich zusammen aus verschiedenen Softwareeinheiten, die ihrerseits aus wiederverwendbaren Komponenten bestehen.

Die Softwareeinheit „DBF Unit“ kapselt die externe Bibliothek „xbasej“ als OSGi-Bundle. Ein technischer Wrapper nutzt dessen API und kapselt essenzielle Zugriffsoperationen auf DBF-Dateien, wie das Öffnen und Schließen sowie das Vor- und Zurücknavigieren innerhalb von Dateien. Die von diesem technischen Wrapper bereitgestellte Schnittstelle wird von einer O/DBF-Mapping-Schicht genutzt, die konkrete DBF-Dateien sowie deren „Spalten“, Indizes und Wertebereiche definiert. Für jede DBF-Datei existiert ein Pendant in der O/DBF-Schicht, das die Zugriffe isoliert und kapselt. Das Mapping wird von statuslosen Zwischenservices angesprochen.

Zu Ihren Diensten: Services

Die Zwischenservices fungieren als Gateway und Adapter zwischen den heterogenen Technologien und Persistenz-Varianten, und liefern Fassaden, um die darunter liegende technische Komplexität zu verstecken und feingranulare Aufrufe zu sinnvollen Operationen zu aggregieren. Da DBF-Dateien intern mit Zeigern auf Datensätze arbeiten, die dort verharren, wo sie zuletzt positioniert wurden, ist für die auf die Persistenz zugrei-

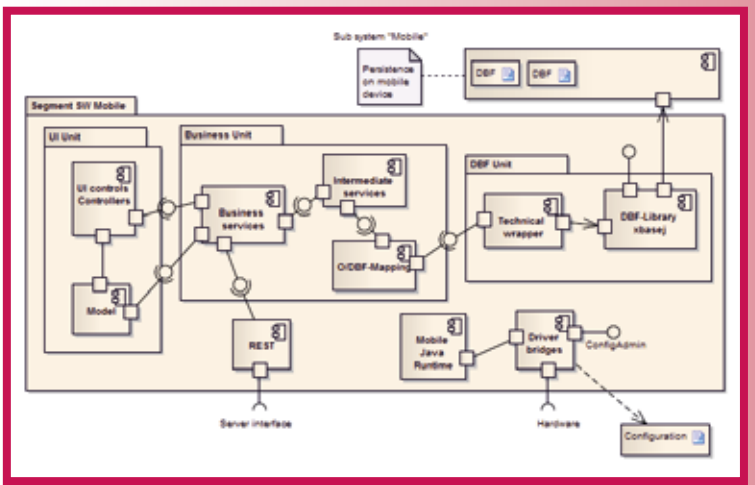

Abb. 1: Dekomposition des Systems: Das Teilsystem „Mobile“ wird zerlegt in Softwareeinheiten und Komponenten

fenden Services Zustandslosigkeit obligatorisch – neben der architektonischen Erwägung, dass derartige Services grundsätzlich zustandslos sein sollten. Auf Ebene der Implementierung ist Zugriffsschutz durch Monitore Pflicht („two-phase locking"), um Datenzugriffe zu isolieren und nicht-funktionale Anforderungen an die Datenhaltung (wie Dauerhaftigkeit, Isoliertheit und Konsistenzerhaltung) zu bedienen.

Den fachlichen Kern bilden die „Business services". Diese bedienen sich der Zwischenservices (und nur dieser), kombinieren deren Funktionen zu fachlich-konsistenten Operationen und bilden so Geschäftslogik ab. Fachliche Services sind von Details der Implementierung der Datenhaltung entkoppelt. Die Komponenten UI (mit Controls, Navigationslogik, Geschäftslogik) und Webservices greifen ausschließlich über die Services auf die Datenhaltung zu. Ein REST-basierter Webservice übermittelt nebenläufig Bewegungsdaten an einen Server und lädt aktuelle Stammdaten nach.

Services erwarten und liefern Geschäftsobjekte. Das UI behandelt folglich ausschließlich Geschäftsobjekte und stellt diese dar. Die über das UI durchgeführten Änderungen (auch Neuanlagen und Löschungen) werden von Services persistiert, die in umgekehrter Richtung auch eingehende Änderungen an der Datenhaltung an das UI propagieren. Eine klare Unterscheidung zwischen konsumierenden und produzierenden Services hilft beim Entwurf einer serviceorientierten Architektur (SOA).

Auch wenn bereits Änderungen an der „Business Unit" erhöhte Ansprüche an die Disziplin der Anwendungsentwickler stellen, können durch die klare Trennung der Zuständigkeiten (Separation of Concerns) recht einfach, flexibel und wartbar neue Anwendungsfälle umgesetzt und bestehende Komponenten wiederverwendet und adaptiert werden.

Eine von der Fachanwendung unabhängige Plattform kapselt Basisfunktionalität, wie beispielsweise die Datenhaltungsschicht (die „DBF Unit") und technische Services (wie die Implementierung eines Rollensystems). Diese robuste Plattform liefert frühe Rückkopplungen über Anomalien sowohl zur Build- als auch zur Laufzeit („fail fast"-Ansatz). Ebenfalls enthaltene technische Wartungsdienste, wie Recovery-Mechanismen für defekte Dateien und Wiedererstellung von Indizes, sind auf dem mobilen Endgerät unerlässlich und können rollenbasiert als Teil der ausgelieferten Anwendung zur Verfügung gestellt werden.

Transaktionsprotokoll

Wichtige Herausforderung beim Einsatz einer per se nicht vollständig den ACID-Prinzipien folgenden Lösung ist die Sicherstellung konsistenter Datenzustände, insbesondere angesichts der zentralen Bedeutung von Schreibzugriffen in einer relativ unzuverlässigen Systemumgebung; ein alternativer Lösungsansatz wie BASE ist im vorliegenden Fall nicht ausreichend.

Um diese Konsistenz zu gewährleisten, wird für die Bewegungsdaten eine an Transaktionslogs klassischer SQL-Datenbanken angelehnte Lösung eingesetzt. Anstatt beispielsweise eine Stückliste in zwei miteinander über Schlüssel verbundenen Tabellen – jeweils für Kopf- und Postendaten – zu speichern, wird pro Liste nur eine einzige Datei erzeugt, die den Verlauf aller Änderungen in Form chronologisch aufeinander aufbauender Einzelschritte dokumentiert. Auf diese Weise lässt sich das Problem unvollständiger Updates über mehrere Tabellen hinweg vermeiden.

lfdNr	operation	objekt	datenfelder…
0	INS	KOPF	id=0; bearbeiter=Mustermann; datum=2099-xx-xx; …
1	INS	POS	id=1; artikelnr=471123; anzahl=2
2	INS	POS	id=2; artikelnr=635233; anzahl=1; rabatt=2
3	UPD	POS	id=1; artikelnr=471123; anzahl=3
4	INS	POS	id=3; artikelnr=998653; anzahl=1
5	DEL	POS	id=2
6	CLOSE	KOPF	

Tabelle 1: Exemplarische Datendatei als DBF, mit Operationen, Zielobjekt und Datenfeldern, im „ASCII-Fix"-Format

Eine typische Datendatei wird im Dateisystem (vereinfacht) als DBF abgelegt, siehe Tabelle 1. Die Spalte „operation" enthält jeweils eine der möglichen CRUD-Operationen (Create, Read, Update, Delete), ergänzt um einige wenige Meta-Operationen.

Zur Entwicklungszeit werden ACID-kritische Codefragmente mit einer besonders hohen Testabdeckung versehen und sensitive Metriken fortlaufend abgeprüft („Continuous Inspection"). So können potenzielle Fehlersituationen, wie „verlorene Updates" oder das „Phantomproblem", frühzeitig aufgespürt werden.

Logische und physikalische Datenstruktur

Anders als bei der „Java Persistence API" (JPA) sind in dieser Lösung die Strukturen der Daten auf dem Datenhaltungsmedium und innerhalb der Anwendung sehr verschieden. Aufgrund dessen wird die Datei beim ersten Zugriff auf die enthaltene Stückliste komplett sequenziell gelesen und ein entsprechender Objekt-Graph im Speicher aufgebaut. Veränderungen an dieser In-Memory-Darstellung, beispielsweise das Hinzufügen neuer Positionen, werden als entsprechende Sätze an die bestehende Datei angehängt. Im Falle von Abstürzen ist somit höchstens die letzte Änderung verloren, eine Beschädigung anderer Daten ist durch den ausnahmslosen Verzicht auf Modifikationen bereits geschriebener Daten ausgeschlossen.

Neben der Robustheit bringt diese Entkopplung auch eine hohe Geschwindigkeit mit sich, da sich alle relevanten Daten im Speicher befinden und Änderungen sich damit in nahezu konstanter und sehr kurzer Zeit durchführen lassen. Trade-offs, die das Verfahren mit sich bringt, sind die Limitierung der gleichzeitig im RAM zu haltenden Datenmenge, die durch das Wegfallen des Overheads komplexerer Frameworks und die vergleichsweise kleinen Objekte aber im vorliegenden Fall praktisch kein Problem darstellt, sowie die initiale Verzögerung beim Einlesen einer bereits begonnenen Transaktionsdatei.

Fortlaufend automatisch liefern

Eine komponentenbasierte Architektur ist eine Voraussetzung zum Aufbau einer feingranularen Build-/Deployment-Infrastruktur**. Softwareeinheiten und einzelne Komponenten sollten einzeln kompilierbar und paketierbar sein, und zentrale Fragestellungen, beispielsweise zum Abhängigkeits-

** Im Umkehrschluss: Eine monolithische Anwendung wird in aller Regel auch monolithisch gebaut und entsprechend hölzern bereitgestellt.

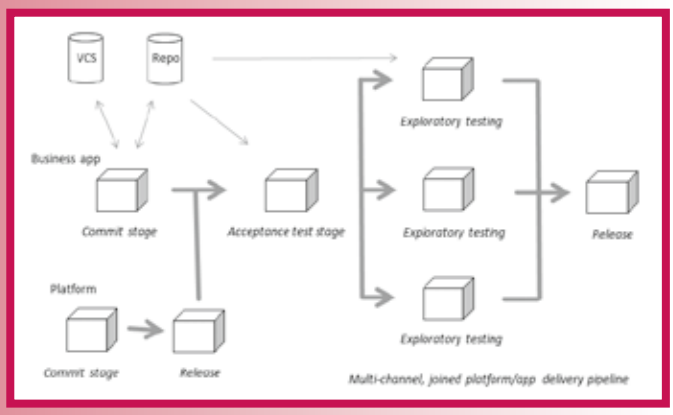

Abb. 2: End2End-Ansatz mit einer multi-channel, joined platform/app delivery pipeline

management, zu Versionierungs- und Branchingstrategien sowie zu effizienten Werkzeugintegrationen, sind zu adressieren (vgl. [AgileALM]). Im vorliegenden Fall liefert eine Delivery-Pipeline kontinuierlich potenzielle Release-Candidates (s. Abb. 2).

Der kontinuierliche Build durchläuft bei jeder Änderung im Versionskontrollsystem die „Commit stage". Diese Stage beinhaltet das Kompilieren und Testen auf Modul-Ebene und stellt bei Erfolg Binärdateien zur Verfügung. Diese sind Ausgangspunkt für die weiteren Schritte. Ein Quality Gate für statische Codeanalyse und Testabdeckung ist ebenfalls eingeflochten. Der Release-Candidate durchläuft automatische Akzeptanztests und wird zudem manuell getestet.

Da die Anwendung auf mehrere Kanäle verteilt werden muss (verschiedene Hardware-Hersteller), wird vom Integrationssystem die Anwendung automatisch für alle möglichen Zielumgebungen paketiert und konfiguriert, um schließlich auf die finale Zielumgebung verteilt zu werden. Verlaufen manuelle Tests auf den jeweiligen Endgeräten erfolgreich, kann aus dem Release-Candidate potenziell ein Release werden. Die Promotion zu einem Release hängt von vielen Faktoren ab, sodass nicht jeder Kandidat an dieser Stelle zwingend zum Release wird.

Das System behandelt Fachanwendung und Plattform unterschiedlich. Im Vergleich zur Fachanwendung werden neue Releases der Plattform in deutlich längerer Taktung bereitgestellt, da Änderungen an zentralen Komponenten eher die Ausnahme sind als die Regel.

Wesentliche Eckpfeiler der *Werkzeugkette* sind Maven/Tycho (wobei OSGi-Manifests die führenden Medien sind), Subversion, Jenkins (mit dem Build-Pipeline-Plug-in) und Nexus (mit der Möglichkeit, P2-Repositories zu halten).

Fazit

Die gegebenen Rahmenbedingungen, insbesondere die im Vergleich zur Server- und Desktop-Java-Entwicklung sehr eingeschränkte Hardware, aber auch die mit JavaME limitierte Laufzeitplattform, die viele häufig eingesetzte Standardframeworks von vornherein ausschließt, machen einige Kreativität erforderlich, um funktionale, aber auch nicht-funktionalen Anforderungen zu erfüllen. Mit einem Stack aus Altbewährtem und Modernem, kombiniert mit einem schlanken Entwicklungsprozess in einem kreativen und agilen Team, kann aber letztlich eine Anwendung bereitgestellt werden, die durch ihre klare Struktur und ihren modularen Aufbau auch für zukünftige Erweiterungen tragfähig ist. Gemäß Martin Fowlers Definition [NoSQL] nutzen NoSQL-Lösungen weder ein relationales Modell noch SQL oder Schemas, sodass die in diesem Artikel skizzierte Lösung sogar mit „NoSQL-enabled" etikettiert werden kann.

Links

[AgileALM] M. Hüttermann, Agile ALM, Manning, 2011
[Brew00] E. A. Brewer, Towards Robust Distributed Systems, PODC Keynote, 19.7.2000,
`http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf`
[dBase] `http://de.wikipedia.org/wiki/DBASE`
[JSR169] Java Specification Request 169:
JDBC Optional Package for CDC/Foundation Profile,
`http://www.jcp.org/aboutJava/communityprocess/final/jsr169`
[NoSQL] `http://martinfowler.com/bliki/NosqlDefinition.html`
[xBaseJ] `http://sourceforge.net/projects/xbasej`
[xBaseJBuch] R. Hughes, The Minimum You Need to Know About Java and xBaseJ, Free-eBook, 2007,
`http://www.theminimumyouneedtoknow.com/xbase_toc.html`

Michael Hüttermann, Oracle Java Champion, ist freiberuflicher Entwickler/Architekt und Delivery Engineer. Er ist Autor der Bücher „Agile Java-Entwicklung in der Praxis", „Fragile Agile" und „Agile ALM". E-Mail: michael@huettermann.net

Daniel Schneller ist Senior Software Consultant und Leiter des Competence Center Mobile Development bei der codecentric AG. Er ist außerdem Autor des „MySQL Admin Cookbook". E-Mail: ds@danielschneller.de