

Zukunftsfähiges Datenmanagement durch hybride Lösungen – Ein Entwurfsmusterkatalog zur Integration von SQL- und NoSQL-Datenbanken

*Jens Nimis/Matthias Armbruster/Martin Kammerer**

Abstract

Die Akzeptanz neuer Technologien hängt häufig von der Existenz und Verbreitung sogenannter Success Stories und Best Practices ab. Sie illustrieren das Potenzial der Neuentwicklung und helfen zugleich ein Verständnis zu entwickeln, das für eine Übertragung auf die eigenen Aufgaben erforderlich ist. Ähnlich verhält es sich im Feld der Datenbanken, in dem zunehmend erfolgreiche Anwendungen der relativ neuartigen NoSQL-Datenbanken – insbesondere in den Bereichen Web und Big Data – bekannt werden. Für viele Anwendungsszenarien in Unternehmen mit umfangreichen existierenden IT-Systemen kann jedoch nicht von einer isolierten Existenz von Anwendungen auf NoSQL-Basis ausgegangen werden. Es bedarf häufig vielmehr der Integration von SQL- und NoSQL-Datenbanken zu einer hybriden Datenhaltung. Einige der wenigen auf diesem Gebiet zugänglichen Erfahrungen werden in der vorliegenden Arbeit zu einem initialen Entwurfsmusterkatalog verallgemeinert und somit erschlossen für eine Wiederverwendung in künftigen Anwendungen. Dies erfolgt auch mit dem Ziel, indirekt die Akzeptanz der NoSQL-Datenbanken in klassischen Unternehmensbereichen zu erhöhen.

* Prof. Dr. Jens Nimis, Fakultät für Wirtschaftswissenschaften, Hochschule Karlsruhe – Technik und Wirtschaft. Kontakt: jens.nimis@hs-karlsruhe.de

Matthias Armbruster, wissenschaftlicher Mitarbeiter, Hochschule Karlsruhe – Technik und Wirtschaft. Kontakt: matthias.armbruster@hs-karlsruhe.de

Martin Kammerer, wissenschaftlicher Mitarbeiter, Hochschule Karlsruhe – Technik und Wirtschaft. Kontakt: martin.kammerer@hs-karlsruhe.de

1. Einleitung und Motivation

Ein zentraler Punkt jeder Anwendungsentwicklung im Unternehmen ist die Gestaltung der Datenhaltung, denn durch sie werden die Leistungsfähigkeit und Zuverlässigkeit von Anwendungen in großem Maße mitbestimmt. Zu dieser Gestaltung zählt die zielgerichtete Auswahl von Datenmodell und Datenbank ebenso wie deren effizienter Einsatz. Die Freiheitsgrade der Gestaltung lagen in den letzten drei Jahrzehnten vor allem im Bereich des Einsatzes, weil für die weitaus überwiegende Zahl der Anwendungsfälle in Unternehmen das relationale Datenmodell als gegeben betrachtet wurde und die Zahl der Datenbankhersteller mit den entsprechenden professionellen SQL-Datenbanken überschaubar war.

Durch das Aufkommen der großen Web-Anwendungen, wie sie z.B. durch Google, ebay, amazon, facebook und andere repräsentiert werden, haben sich die Anforderungen jedoch derart verändert, dass eine neue Generation von Datenbanken erforderlich wurde. SQL-Datenbanken stoßen bei den dort anzutreffenden Benutzerzahlen, Transaktionsraten und Flexibilitätsanforderungen an ihre Grenzen. Initiiert aus dem Kreis derartiger Datenbankanwender entstand folgerichtig die NoSQL-Bewegung (Edlich et al. 2011). Sie rüttelt an den Grundfesten der SQL-Datenbanken, indem sie deren relationales Datenmodell und das dort meist anzutreffende transaktionale ACID-Paradigma (Kemper und Eickler 2009) infrage stellt und ersetzt durch vereinfachte Datenmodelle und abgeschwächte Konsistenzgarantien.

Aktuell steht man also bei einem neu umzusetzenden Anwendungsszenario vor der Entscheidung, ob eine SQL- oder NoSQL-Datenbank zur Adressierung der spezifischen Anforderungen die bessere Alternative darstellt. Dabei gibt es in einigen Fällen bekannte, dominante Anwendungsparameter, die direkt zur Eliminierung respektive Wahl einer der beiden Alternativen führen. Viele moderne Anwendungsszenarien indizieren jedoch mit ihrem Anforderungsprofil nicht klar die eine oder andere Variante und führen bei der Entscheidungsfindung zu Unsicherheiten.¹

1 An dem beschriebenen Entscheidungsproblem setzt das LEADS-Projekt mit seinem Leitfaden zur erfolgreichen Auswahl innovativer Datenbank-Systeme an: in einer empirischen Studie werden vier Anwendungsszenarien mit grenzwertigen Anforderungen jeweils in einer SQL- und einer NoSQL-Variante implementiert und die Leistungsfähigkeit der Varianten experimentell evaluiert. Auf Basis dieser Erfahrungen wird ein Entscheidungsverfahren entwickelt, das den Anwender ausgehend von einer systematischen Analyse der Merkmale des Anwendungsszenarios zur Auswahl eines geeigneten Datenbanksystems führt. LEADS adressiert im Kern also den neugewonnenen Freiheitsgrad, ob für ein gegebenes Anwendungsszenario NoSQL zum Einsatz kommen sollte und ggf. auch welche der NoSQL-Spielarten Column Store, Document Store, Graph Database und Key/Value-Store (siehe Edlich et al. 2011) dann infrage kommen. Wird die Frage nach NoSQL bejaht, dann schließt sich direkt der bisher aus der

Die meisten Neuanwendungen in Unternehmen entstehen jedoch nicht auf der grünen Wiese, sondern müssen mit einer bestehenden System- und Datenbank-Landschaft integriert werden. Und auch innerhalb alleinstehender Neuanwendungen zeigt sich oft, dass bestimmte Teilaufgaben der Datenhaltung besser mit NoSQL- und andere mit SQL-Datenbanken abgedeckt werden. In der NoSQL-Bewegung haben derartige Beobachtungen dazu geführt, dass das Label „NoSQL“ nicht mehr mit „No SQL“ sondern mit „Not Only SQL“ aufgelöst wird. Zu der Frage der Integration von NoSQL- und SQL-Datenbanken zu hybriden Datenhaltungssystemen ist noch kein umfangreicher Literaturbestand verfügbar, was zum einen in der Neuigkeit und damit geringen Marktdurchdringung der NoSQL-Systeme begründet ist und zum anderen in der Fokussierung der NoSQL-Anbieter auf die eigenen Paradigmen und Produkte. Auf Anbieterseite finden sich so zumeist gut ausgearbeitete Einführungsbeispiele, die die Anwendung der jeweiligen NoSQL-Datenbank illustrieren und damit erleichtern sollen. Auf Anwenderseite sind einige wenige Systeme dokumentiert, die beide Datenbank-Typen in Kombination einsetzen und dabei die bekannten Grundregeln aus (relationalem) Datenbankeinsatz und Software Engineering einhalten. Darüber hinaus gibt es Erkenntnisse von den Werkzeugentwicklern, die z.B. Zugriffs-Bibliotheken zum transparenten Zugriff auf Datenbanken beliebigen Typs anbieten und die zugrundeliegenden Systeme auf einen gemeinsamen Kern abbilden.

Hier gilt es also aus den wenigen zugänglichen Erfahrungen für die Zukunft zu lernen, d.h. existierende Beispiele zu analysieren und so aufzubereiten, dass sie als Vorlage für weitere Anwendungsentwicklungen dienen können. Einen entsprechenden Vorstoß unternimmt der hier vorgelegte Entwurfsmusterkatalog zur Integration von SQL- und NoSQL-Datenbanken, der sowohl auf unseren eigenen Erfahrungen bei der Entwicklung von NoSQL-basierten Neuanwendungen als auch auf öffentlich verfügbaren Umsetzungsbeispielen und -werkzeugen von Anbietern und Anwendern beruht.

2. Entwurfsmuster: Best Practices der Software-Entwicklung

Entwurfsmuster stammen ursprünglich aus der objektorientierten Software-Entwicklung und gelten dort als probates Mittel zur Übertragung erfolgreich erprobter Strukturen von einer Anwendung auf eine andere, neue Anwendung. Durch Abstraktion und Verallgemeinerung von Erfahrungen werden für wiederkehrende Problemklassen allgemeingültige Lösungsansätze in einem sogenannten Entwurfsmusterkatalog bereitgestellt.

SQL-Welt bekannte Freiheitsgrad an, wie sich die NoSQL-Datenbank am erfolgversprechendsten einsetzen lässt.

Die Entwurfsmuster eines Kataloges weisen eine einheitliche Struktur auf, die dem Anwender hilft, eine konkrete Problemstellung abstrakt zu analysieren und ein entsprechendes – ebenfalls abstraktes – Muster zur Lösung auszuwählen. Die Beispiele, die für jedes Entwurfsmuster bereits existierende oder hypothetisch mögliche Einsatzszenarien illustrieren, dienen sowohl der Unterstützung des Verständnisses als auch als Hinweis auf die konkrete Umsetzung zur Lösung der eigenen Problemstellung.

Die hier vorgestellten Entwurfsmuster kategorisieren die verschiedenen Möglichkeiten der Integration von nicht-relationalen Datenbanken in relationalen Datenbankumgebungen auf einer eher abstrakten statt technischen Ebene, um die vorgeschlagenen Lösungen für möglichst viele unterschiedliche Datenbanken anwendbar zu machen. Ziel der Entwurfsmuster ist es also nicht, einen fertigen Programmcode zur Implementierung zu liefern, sondern vielmehr, verschiedene Implementierungsansätze zu beschreiben, um die jeweiligen Vor- und Nachteile der Alternativen situationsabhängig bewerten und damit eine geeignete Alternative auswählen zu können.

2.1 Beschreibungsraster

Ein Entwurfsmuster benennt, abstrahiert und identifiziert die relevanten Aspekte einer allgemeinen Entwurfsstruktur. Jedes Entwurfsmuster konzentriert sich auf eine bestimmte Art von Problemstellung und beschreibt, wann und wie es einsetzbar ist und wann es unter bestimmten Gegebenheiten verwendet werden kann (vgl. Gamma et al. 1994). Das einheitliche Beschreibungsraster der Entwurfsmuster in der vorliegenden Arbeit orientiert sich an diesen Anforderungen. Die beschreibenden Elemente wurden dafür soweit sinnvoll übertragbar aus den Entwurfsmustern der objektorientierten Software-Entwicklung nach (Gamma et al. 1994) ausgewählt und wo erforderlich um die spezifischen Belange der SQL/NoSQL-Integration ergänzt:

Name

Der Mustername (inklusive eventuell existierender Aliase) dient der Identifizierung des Entwurfsmusters und beschreibt kurz und prägnant die charakteristischen Merkmale des Musters.

Überblick und Zweck

Der Überblick enthält die wesentlichen Aufgaben des Musters bei der Problemlösung sowie eine kurze Funktionsbeschreibung, die das grundlegende Wirkprinzip des Musters vermitteln soll.

Beschreibung und Struktur

Im Abschnitt Struktur werden ausführlich die technischen Bestandteile des Musters, ihre Abhängigkeiten und Interaktionen veranschaulicht.

Konsequenzen

Dieser Abschnitt diskutiert die Vor- und Nachteile des Mustereinsatzes in Bezug auf die Lösung des adressierten Anwendungsproblems.

Beispiele und bekannte Verwendungen

Hier werden ggf. bereits verfügbare Anwendungen und Werkzeuge vorgestellt, die das vorliegende Muster realisieren. Zusätzlich werden anhand von Beispielen Szenarien aufgezeigt, in denen das Muster bereits verwendet wird oder eingesetzt werden könnte.

3. Der Entwurfsmusterkatalog**3.1 Entwurfsmuster 1: Referenztabelle***Name*

Referenztabelle

Überblick und Zweck

Bei einer hybriden Datenhaltung stehen Daten in einer relationalen Datenbank in einer Beziehung zu Daten aus einer nichtrelationalen Datenbank. Die inhaltliche Beziehung auf Anwendungsebene muss technisch in dem hybriden System abgebildet werden. Referenztabellen beinhalten die Primärschlüssel bzw. identifizierenden Merkmale sowohl der relationalen Datensätze als auch der nichtrelationalen Datenelemente, sodass die Daten getrennt gespeichert und über die in der Tabelle hinterlegten Referenzen einander zugeordnet werden können. Die Anwendung übernimmt die Zuordnung der jeweils identifizierenden Merkmale in der Tabelle.

Beschreibung und Struktur

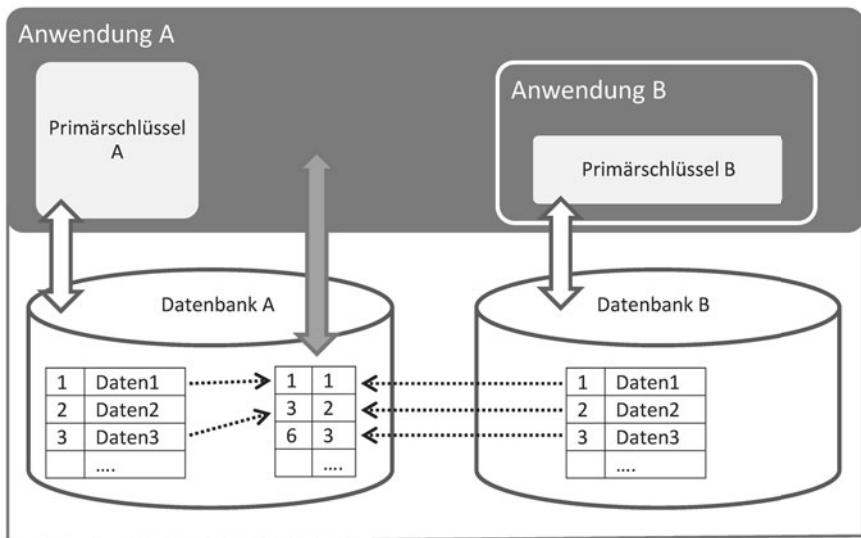
Die Referenztabelle wird als semantische Schnittstelle zwischen einer relationalen und nicht-relationalen Datenbank verwendet. Dabei können die Daten funktional getrennt in verschiedenen Datenbanktypen gespeichert werden, um das jeweilige passende Datenmodell einer Datenbank zu nutzen. Da bei beiden Datenbank-Konzepten (relational und nicht-relational) identifizierende Merkmale für alle Datenelemente vorhanden sind, können die Daten über die Referenztabelle verknüpft werden.

Die Tabelle enthält so viele Spalten wie Datenbanken im hybriden System enthalten sind. Werden Daten beispielsweise in einem Document Store und einer

relationalen Datenbank aufgeteilt, so enthält die Tabelle lediglich zwei Spalten, in der die jeweiligen Kennungen der Datensätze gespeichert werden. In welcher der beteiligten Datenbanken die Tabelle selbst gehalten wird, ist nicht festgelegt, allerdings sollte hier die Anwendung näher betrachtet werden, die die Daten verarbeitet. Die Tabelle sollte sinngemäß in der Datenbank gehalten werden, welche die übergeordneten Daten enthält. Wenn also beispielsweise die Adresse eines Kunden in einer relationalen Datenbank gespeichert wird und das Kundenbeziehungsmanagement mit einer Graph-Datenbank verwaltet wird, dann sollte die Referenztable in der relationalen Datenbank gehalten werden, da für das Beziehungsmanagement des Kunden zuerst die Adresse des Kunden angelegt werden muss. Wichtig hierbei ist, dass die Anwendung die jeweiligen Kennungen aus den Datenbanken auslesen können muss, da die Kennungen ausschließlich über die Anwendung in die Referenztable geschrieben werden.

Der Aufbau und das Funktionsprinzip für eine Referenztable ist in Abbildung 1 gezeigt, in der Anwendung A und Anwendung B zwei mehr oder weniger eng gekoppelte Bestandteile einer integrierten Anwendung darstellen. Konkret sei die Anwendung B in der Anwendung A integriert, um die Kennungen der zugehörigen Datenbank B an die Anwendung A zurückgeben zu können. Die Referenztable ist daher in der Datenbank A untergebracht, damit die zugehörige Anwendung A die Referenzen einfach verwalten kann.

Abbildung 1: Struktur des Entwurfsmusters "Referenztable"



Auf die Referenztabelle könnte verzichtet werden, wenn sich die Kennungen der Datenbank A direkt als Kennungen in der Datenbank B erzwingen lassen würden. Dies hätte den Vorteil, dass die Synchronisierung der Tabelle entfallen könnte und somit weniger Zeit und Rechenleistung für Pflege und Anfragen benötigt werden würde. Allerdings ist dies nur möglich, wenn das Szenario konfigurierbare Schlüssel in mindestens einer der beiden beteiligten Datenbanken erlaubt.

Konsequenzen

Der Vorteil dieses Musters liegt in der unkomplizierten Art der Datenbankverwendung. Die Datenbanken können ohne tiefgreifende Anpassungen unabhängig installiert und verwendet werden, da die Verknüpfung der Daten in der Anwendung geschieht. Die Struktur kann auch bei bereits existierenden Datenbeständen eingesetzt werden, wenn die Anwendungen bzw. Anwendungsbestandteile technisch einen Datenaustausch ermöglichen. Die Daten können durch die verschiedenen Anwendungen mit den jeweiligen Anforderungen an Konsistenz, Verfügbarkeit usw. verarbeitet werden, wodurch die Vorteile der jeweiligen Datenmodelle ohne Kompromisse genutzt werden können.

Nachteilig ist die Erfordernis, dass eine der beiden Anwendungen mit den verschiedenen Datenmodellen umgehen können muss bzw. verschiedene Anwendungen aufeinander angepasst werden müssen, sodass die Kennungen übergeben werden können. Außerdem liegt die gesamte Verknüpfungsinformation in einer Tabelle, die damit eine kritische Komponente im Hinblick auf Performanz und Stabilität darstellen kann.

Beispiele und bekannte Verwendungen

Das Muster eignet sich besonders für Data-Warehouse-Anwendungen, bei denen die Daten für Analysen in eine separate Datenbank ausgelagert werden, falls dennoch ein Bezug zu operativen Daten (beispielsweise für Aktualisierungen) erhalten bleiben soll. Konkret könnten die Daten zur Analyse beispielsweise mit dem Massendatenwerkzeug Apache Sqoop (Sqoop 2012) von einer relationalen Datenbank in eine Hadoop-Instanz (Hadoop 2012) kopiert werden.

Als Beispiel kann hier die Trennung in Adress-Stammdaten und Kundendaten genannt werden. Die Stammdaten eines Kunden (Name, Adresse, Funktion etc.) werden dabei in einer relationalen Datenbank gehalten und sind in den meisten Fällen bereits vorhanden. Spezifische Kundendaten wie Kundenkontakte, Ansprechpartner, Beziehungen untereinander, zu eigenen Mitarbeitern oder zu anderen Kunden können beispielsweise in einer dafür besser geeigneten Graphdatenbank gespeichert werden, um komplexe Vernetzungen schnell und effizient zu erfassen und aufzurufen. Die Stammdaten der Adresse werden mit

einem Primärschlüssel versehen, über den die Daten eindeutig zugeordnet werden können.

Von diesem Datensatz aus kann dann die Anwendung für die Kundendaten gestartet werden, sodass der Primärschlüssel der zugrunde liegenden Adress-Stammdaten bekannt ist. Beim Anlegen von Kundendaten wird dann ein Knoten in der Graphdatenbank erzeugt, der entweder eine eigene Kennung erhält oder den Schlüsselwert der Adress-Stammdaten als solche übernimmt. Diese beiden identifizierenden Merkmale können dann in die Referenztabelle geschrieben werden, sodass bei zukünftigen Aufrufen von Kundendaten direkt zu dem entsprechenden Knoten in der Graphdatenbank traversiert werden kann.

3.2 Entwurfsmuster 2: Verteiler

Name

Verteiler

Überblick und Zweck

Die gleichzeitige Verwendung mehrerer verschiedener Datenbanken in einem hybriden System fordert von der Anwendung, verschiedene Datenbank-Schnittstellen zu kennen und diese in den richtigen Situationen zu verwenden. Das Verteiler-Muster führt die verschiedenen Schnittstellen (API, engl. Application Programming Interface) in einer zusammen und verteilt die Anfragen, die von einer Anwendung an eine Datenbanklandschaft gestellt werden. Dabei werden die Anfragen nach dem verwendeten Datenmodell unterschieden und an den jeweils richtigen Adressaten weitergeleitet.

Beschreibung und Struktur

Dieses Muster kann als Datenbank-unabhängige Schnittstelle angesehen werden, die verschiedenartige Anfragen von Clients entgegennimmt und anhand des verwendeten Datenmodells die Anfrage an die richtige Datenbank weiterleitet. Die zugrundeliegenden Datenmodelle und Daten werden in einer Art Metadaten-Schicht modelliert, welche deren Ausdrücke durch die Verteiler-Schnittstelle analysiert und zu der entsprechenden Datenbank weiterleitet.

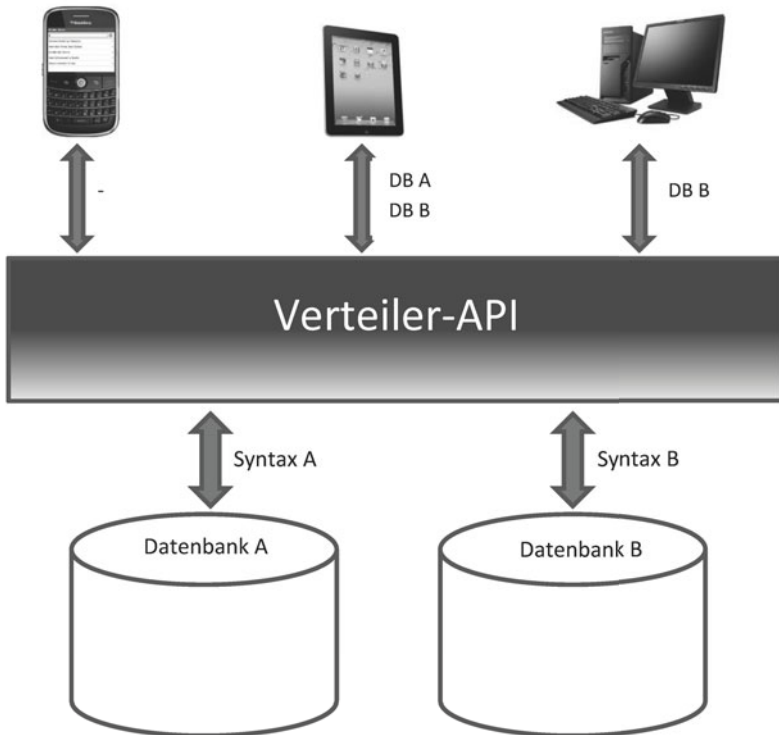
Das Entwurfsmuster sieht vor, dass in den Metadaten eine Festlegung der konkreten Zieldatenbank möglich ist, sodass für bestimmte Fälle auch die Datenbank ohne die Datenmodell-Analyse gewählt werden kann. Ist diese Information nicht vorhanden, kann der Verteiler abhängig von der Komplexität entscheiden, ob durch entsprechende Verfahren eine geeignete Datenbank für das verwendete Datenmodell ermittelt werden soll oder ob die Anfragebearbeitung direkt zurückgewiesen wird.

Durch die dynamische Auswahl der Datenbank anhand des Datenmodells können auch unterschiedliche Client-Anwendungen über dieselbe Schnittstelle

bedient werden, wodurch auf beiden Seiten (Clients und Datenbanken) eine erhöhte Flexibilität erreicht wird. Beispielsweise kann so auch ein ERP-System von beliebigen Client-Anwendungen verwendet werden.

In Abbildung 2 ist die Struktur des Verteiler-Musters dargestellt. Verschiedene Anwendungen auf einer Reihe von Endgeräten (z.B. Smartphones, Tablets oder PCs) verwenden unterschiedliche Datenmodelle und stellen Anfragen an die Datenbanklandschaft, die von der Verteiler-Schnittstelle dann zur jeweiligen Datenbank verteilt und die Ergebnisse entsprechend aufbereitet werden. Dabei können den Anfragen auch Metadaten mitgegeben werden, die zur Zuordnung ausgewertet werden. Sind keine Metadaten angegeben (wie in Abbildung 2 bei dem linken Client), wird die Datenbank anhand des Datenmodells von der API bestimmt.

Abbildung 2: Struktur des Entwurfsmusters "Verteiler"



Je nach Anzahl der Client-Anwendungen, Datenmodelle und Datenbanken kann die Komplexität des Verteilers unterschiedlich gestaltet werden. Jedoch kann mit steigender Funktionalität des Verteilers die Flexibilität der Client-Anwendungen gesteigert werden.

Konsequenzen

Der Vorteil dieser Lösung besteht vor allem in der Datenbank-neutralen Entwicklung der Anwendung. Die Anwendung kann daher sehr flexibel angepasst werden und ist für viele unterschiedliche Datenbanken offen, da die Datenbank über das verwendete Datenmodell bestimmt wird. Gerade letztere Eigenschaft wird durch die mobile Datenverarbeitung auch in Unternehmen immer wichtiger, da mobile Endgeräte in der Regel nicht die gleichen Datenformate verwenden wie die Client-Anwendungen auf Desktop-Rechnern, um den Datenaustausch für die Übertragung zu minimieren. Die Datenmodelle der NoSQL-Datenbanken sind meist auf Webanwendungen spezialisiert und können beispielsweise direkt dort anzutreffende Datenformate wie JSON (JavaScript Object Notation) (IETF 2006) verarbeiten.

Hauptnachteil des Musters ist die mögliche Komplexität eines funktionsmächtigen Verteilers. Der Verteiler muss sämtliche Datenmodelle mit deren funktionalen und nicht-funktionalen Eigenschaften verstehen und unterstützen, um die Verteilung sinnvoll durchführen zu können. Dabei wird die Komplexität teilweise von der Anwendung in den Verteiler verlagert, was andererseits der Flexibilität der Anwendung zugutekommt. Bei sehr großen Lasten durch viele Clients kann der Verteiler in Bezug auf die Gesamtleistung zum Flaschenhals der Architektur werden, weshalb eine ausreichende Dimensionierung der Ressourcen zu berücksichtigen ist.

Beispiele und bekannte Verwendungen

Als verfügbare Lösung kann für den Verteiler das Spring Data API (Spring 2012) eingesetzt werden. Allerdings ist hier aktuell nur eine REST-Schnittstelle verfügbar sowie die Integration über Java möglich. Die generische Analyse des Datenmodells ist in dieser Lösung noch nicht eingebaut, kann jedoch für konkrete Datenmodelle selbst nachgerüstet werden, da die Lösung als Open Source Projekt vorliegt.

Verteiler eignen sich besonders für heterogene Anwendungsarchitekturen, die mehrere Datenmodelle einsetzen wollen. Dies kann am Beispiel eines ERP-Systems verdeutlicht werden. Dort könnten beispielsweise die Adressdaten in einem Document Store gespeichert werden, um die immer größer werdende Anzahl an Kontaktmöglichkeiten (Telefon, Mobiltelefon, Email-Adressen, Xing, facebook, Twitter usw.) ohne unnötige Schemaanpassungen zu verwalten. Gleichzeitig muss aber die Finanzbuchhaltung aus Konsistenzgründen in einer

Technologien für digitale Innovationen

Interdisziplinäre Beiträge zur Informationsverarbeitung

Jähnert, J.; Förster, C. (Hrsg.)

2014, VI, 196 S. 46 Abb., Softcover

ISBN: 978-3-658-04744-3