Dipl.-Wirt.-Inf. (BA) Jochen Christ
(jochen.christ@senacor.com)
ist Technical Lead bei der Senacor Technologies AG in Nürnberg. Als Spezialist für moderne Java-Technologien arbeitet er gerne an eleganten Lösungen mit innovativen Architekturkonzepten. Für paydirekt hat er wesentliche Teile der Service-Architektur konzipiert und mit umgesetzt. Ihm ist dabei immer wichtig, dass er mehr in der IDE als mit Powerpoint arbeitet. Jochen Christ ist außerdem als Lehrbeauftragter an der Technischen Hochschule Ingolstadt tätig.

paydirekt: Microservices machen die Organisation agil

Das Online-Bezahlsystem paydirekt der deutschen Banken und Sparkassen setzt auf eine Microservice-Architektur, um schnell auf Marktanforderungen reagieren zu können. Der IT-Spezialist Senacor Technologies zeigt, wie mit kleinen Deployment-Einheiten neue Funktionen innerhalb weniger Wochen und mit geringem Risiko produktiv gesetzt werden können, wie die IT-Architektur damit auf die gesamte Organisation ausstrahlt und welche ganz neuen Problemstellungen dabei gelöst werden mussten.

Mit paydirekt haben die deutschen Banken und Sparkassen ein gemeinsames Online-Bezahlsystem gestartet, um ihren Kunden einfaches und sicheres Bezahlen für Online-Einkäufe zu ermöglichen.

Eine besondere Herausforderung für die Umsetzung und den Betrieb stellt dabei die enge Integration mit den verschiedenen IT-Systemen der teilnehmenden Banken und deren Rechenzentren dar. Hinter den über 1000 Instituten der privaten Banken, der Genossenschaftlichen Finanzgruppe und der Sparkassen stehen über 20 verschiedene IT-Organisationen mit eigenen Rechenzentren und individuellen spezifischen Anforderungen.

Das Bezahlsystem basiert auf dem Girokonto und es wird für jeden Bezahlvorgang das Guthaben in Echtzeit geprüft, um dem Händler eine sofortige verbindliche Zahlungsgarantie auszusprechen. Dazu werden kontospezifische und bankenindividuelle Sicherheitsfeatures genutzt und integriert.

Der IT-Spezialist Senacor Technologies, der das Paydirekt-Kernsystem konzipiert und umgesetzt hat, setzt für die IT-Plattform auf eine Microservice-Architektur, die im Wesentlichen auf den von Martin Fowler in [FuL14] beschriebenen Charakteristika beruht.

Eigenständige fachliche Services

In der Microservice-Architektur werden fachliche Funktionen als Services angeboten. Klar definierte Funktionen können über eine eindeutig definierte Schnittstelle von anderen Konsumenten aufgerufen werden. Die Services sind typischerweise so geschnitten, dass ein Service ein Geschäftsobjekt verwaltet oder einen definierten technischen Aspekt realisiert. Jeder Service wird als eigenständige Komponente umgesetzt und unabhän-

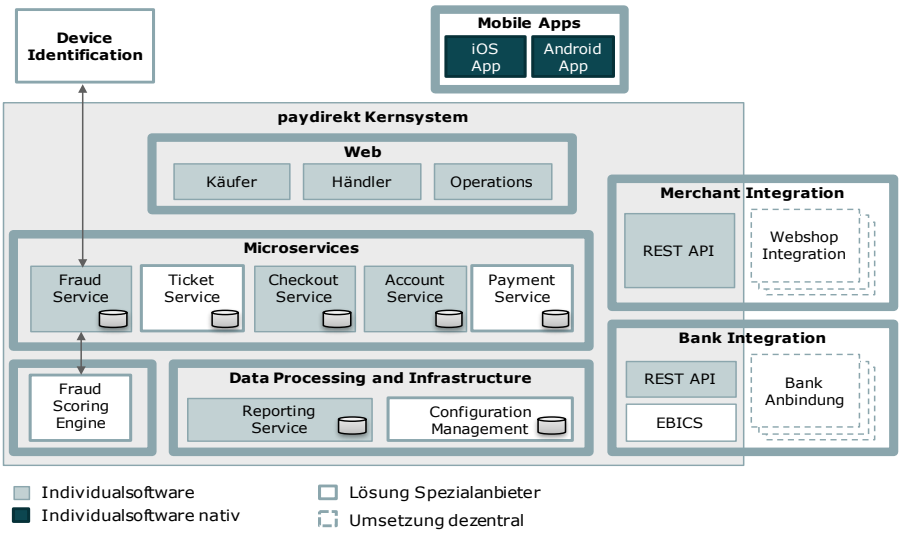

Abb. 1: In der paydirekt-IT-Architektur hat jeder Microservice seine eigene Datenbank

gig voneinander als eigener Prozess ausgeführt (siehe auch Abbildung 1).

Das paydirekt-System besteht derzeit aus über 30 eigenständigen und unabhängigen Services. Senacor Technologies hat sich für eine Mischung aus selbst implementierter Individual-Software und den Einsatz von Standard-Software entschieden. Es ist beispielsweise nicht effizient, ein Ticketverwaltungssystem zur Verwaltung von Support-Anfragen selbst zu implementieren. Hier konnte auf ein etabliertes Produkt zurückgegriffen werden, das als eigenständige Anwendung betrieben wird.

Die Kernprozesse des Systems, insbesondere der Bezahlvorgang und die Schnittstellen zu Händler und Banken, sind als Individualsoftware implementiert. Senacor Technologies hat die meisten Services in *Java* in Verbindung mit dem *Spring Framework* umgesetzt. Für die Persistenz wird typischerweise die Dokumentendatenbank *MongoDB* eingesetzt, die sich in der Praxis hinsichtlich der Flexibilität aufgrund der Schema-Freiheit und der Performance sehr bewährt hat.

Mit *Spring Boot* steht eine Technologie für die effiziente Implementierung zur Verfügung, die mittels Classpath-Scannung und durch das „Convention-over-Configuration-Prinzip“ [Mil09] viele sinnvolle Standardeinstellungen bereitstellt. Zudem sind wichtige Management- und Überwachungsfunktionen integriert, die für den produktiven Betrieb notwendig sind.

Neue Features sind damit in kurzer Zeit in einem Microservice als Prototyp umgesetzt und können in einer Testumgebung evaluiert werden. Überzeugt das Feature nicht, wird der Microservice einfach wieder gelöscht. Produktmanager können mit neuen Funktionen experimentieren, Erfahrungen sammeln und Probleme frühzeitig identifizieren, ohne dass dabei das bestehende produktive System verändert oder gefährdet wird.

Für komplexere Anpassungen innerhalb bestehender Komponenten werden Feature-Toggles [Hod16] eingesetzt. Damit können Änderungen umgesetzt werden, die in einer bestimmten Umgebung oder vor einem definierten Zeitpunkt nicht aktiv sein sollen. Somit können auch größere Änderungen über mehrere Sprints hinweg entwickelt werden.

REST-API mit mehrstufigem Versionierungskonzept

Die Microservices des paydirekt-Kernsystems bilden eine verteilte Anwendung. Bei allen Vorteilen bringen solche Architekturen auch Herausforderungen mit sich, wenn man sie mit monolithischen Architekturen vergleicht: Hier sei insbesondere die Kommunikation zwischen den Geschäftsobjekten und Domänen hervorgehoben.

Der Aufruf einer Funktion eines anderen Microservices erfolgt immer als entfernter Funktionsaufruf, was aufgrund der Netzwerkkommunikation und der notwendigen Datenserialisierung länger dauert als ein lokaler Methodenaufruf und mit Netzwerk-Störungen gerechnet werden muss. Fowler schlägt entweder den Einsatz synchroner APIs oder asynchroner nachrichtenorientierter Middleware vor.

Da im paydirekt-System vor allem eine Request-Response-orientierte Kommunikation mit dem Käufer und Banken stattfindet, werden APIs nach dem REST-Prinzip als Schnittstellentechnologie eingesetzt. Durch die konsequente Umsetzung des HATEOAS-Konzepts [Hat] erfolgt der Kommunikationsfluss zwischen den Komponenten ausschließlich über benannte HAL-Links [Kel] und nicht über statische URLs. Über Links kann der Aufrufer der API so weitere Informationen zu einem Geschäftsobjekt aufrufen oder Aktionen durchführen, die ggf. auch in einem anderen Microservice implementiert sind.

Ein mehrstufiges Versionierungskonzept erlaubt die flexible Weiterentwicklung des Systems, was insbesondere bei der Anbindung der vielen Bank-Systeme von großer Bedeutung ist. Werden neue Geschäftsvorfälle umgesetzt, dann sind im Laufe der Zeit auch Veränderungen an den Schnittstellen notwendig.

Dabei muss in jedem Fall gewährleistet werden, dass eine Änderung der Schnittstelle abwärtskompatibel ist, ein existierender Verwender der Schnittstelle darf keinen Änderungsaufwand haben. In der REST-Schnittstelle werden dazu zwei sich ergänzende Mechanismen verwendet.

Primär ist das Ziel, Änderungen *abwärtskompatibel* entsprechend einer vereinbarten Konvention zu gestalten. Das Hinzufügen eines neuen Endpoints unter einer neuen URI ist grundsätzlich unproblematisch. Wenn ein bestehender Endpoint verändert werden soll, dann dürfen per Konvention jederzeit neue Felder hinzugefügt oder Enumerationen erweitert werden.

Bestehende Felder dürfen aber nicht geändert bzw. entfernt werden. In der Praxis erweist sich die Schema-Freiheit von REST-APIs hier als sehr hilfreich, die bestehenden Konsumenten der Schnittstelle ignorieren die zusätzlichen Informationen. Ein Anpassungsaufwand ist erst dann notwendig, wenn ein Konsument die neuen Anwendungsfälle fachlich nutzen möchte.

Etwas komplexer ist die Umsetzung *inkompatibler Änderungen*: Die Microservices sind so konzipiert, dass jeder Microservice versioniert ist und gleichzeitig verschiedene Versionen eines Microservices betrieben werden können. Über verschiedene URL-Mappings werden die Services adressiert.

Per Konvention ist im URL-Pfad neben dem Namen des Microservices auch die Zielversion definiert, es kann also unter */<servicename>/v1* die bisherige Version angesprochen werden, während über */<servicename>/v2* auf die neue Version zugegriffen wird (siehe auch Abbildung 2). Nachdem alle Konsumenten auf die neue Version umgestellt haben, wird die alte Version abgeschaltet und der Server ant-

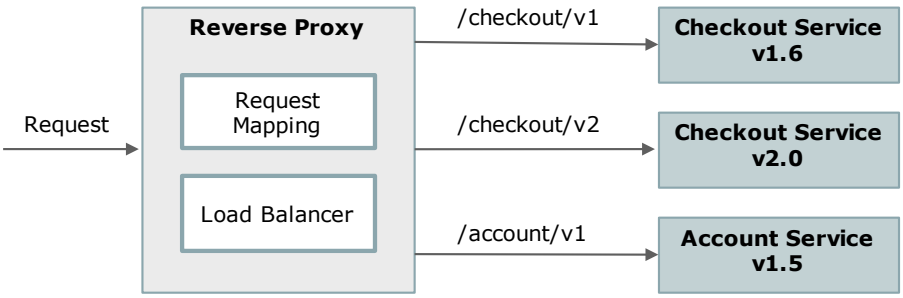

Abb. 2: Es können parallel mehrere Versionen eines Microservices betrieben werden. Das Routing erfolgt über ein einfaches URL-Mapping.

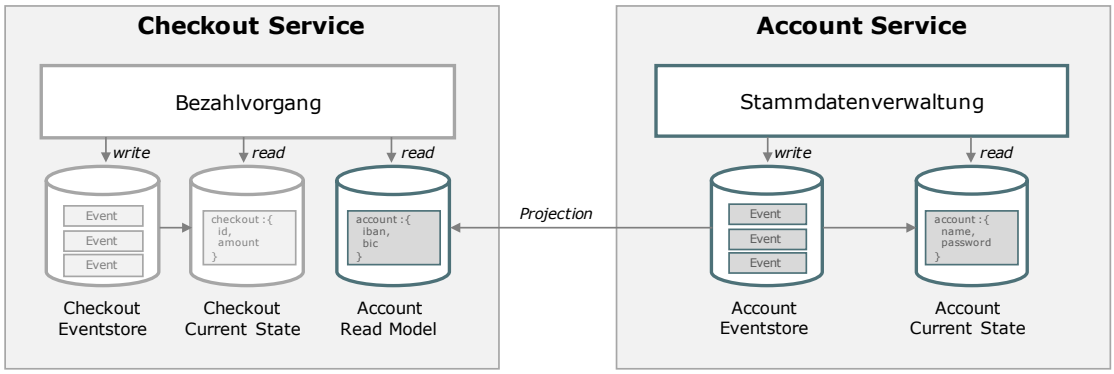

Abb. 3: Relevante Käufer-Stammdaten werden in den Checkout-Service repliziert. Der Bezahlvorgang kann somit durchgeführt werden, auch wenn der Account-Service nicht verfügbar ist.

wortet mit einem entsprechenden http-Status-Code.

Dezentrale Datenhaltung via Event Sourcing

Jeder Microservice verwaltet im paydirekt-System seine eigene Datenbank. Der direkte Zugriff eines Microservices auf eine „fremde" Datenbank ist nicht möglich, die Microservices bilden mit ihren Datenbanken eine Einheit. Dies führt zu kleineren Datenbanken, die einfacher zu verwalten und letztlich auch günstiger als große Cluster-Systeme sind.

Die Datenpersistenz der zentralen Geschäftsobjekte erfolgt über Event Sourcing [Akka]. Ein Microservice verwaltet seine Geschäftsobjekte in Form eines Event Stores. Ähnlich zu einem Versionsverwaltungssystem (wie zum Beispiel Git) werden nur Änderungsereignisse persistiert. Der aktuelle Zustand des Geschäftsobjekts ergibt sich, wenn alle Ereignisse nacheinander ausgeführt werden.

Andere Microservices können spezielle Projektionen eines Geschäftsobjekts erstellen, indem sie die Ereignisse und Daten aggregieren, die in ihrem Kontext relevant sind. Beispiel: Während zur Zahlungsabwicklung die IBAN und BIC eines Kunden benötigt werden, ist beim Versand von E-Mails die E-Mail-Adresse und der Name relevant. Diese Projektionen werden entsprechend des Zugriffsmodells des jeweiligen Microservices optimiert, in einer lokalen Datenbank oder in Memory gespeichert und sind so sehr effizient zugreifbar (siehe auch Abbildung 3).

Insbesondere umgeht dieses Prinzip zudem die oben genannte synchrone Kopplung durch entfernte Funktionszugriffe für die lokal gespeicherten Geschäftsobjekte. Die damit einhergehende Datenredundanz und der so erhöhte Ressourcenverbrauch werden in Kauf genommen. Konzeptionell ist diese Redundanz hingegen unkritisch, da schreibende Operationen am Geschäftsobjekt immer nur durch den Microservice erfolgen dürfen, der den Event Store verwaltet.

Entsprechend des CAP-Theorems nach Brewer [Bre04] ist es in einem verteilten System möglich, maximal zwei der drei Eigenschaften Konsistenz, Verfügbarkeit und Partitionstoleranz zu erreichen. In einem verteilten System muss immer damit gerechnet werden, dass Netzwerkfehler auftreten, die Eigenschaft Partitionstoleranz ist daher nach Greiner [Gre14] zwingend.

Innerhalb eines Microservice wird daher „CP" gewählt: Beim Schreiben eines Events in den Event Store wird die Konsistenz gewährleistet, indem gewartet wird, bis das Event in einem zusätzlichen Knoten gespeichert wurde – auf Kosten einer höheren Latenz.

Bei der Projektion in andere Microservices wird dagegen das *Eventually Consistency-Modell* [Vog08] verwendet. Hier wird eine gewisse Zeitspanne akzeptiert, bis die jeweiligen Read-Modelle aktualisiert sind. Die Änderungsereignisse werden im paydirekt-System typischerweise innerhalb einer Sekunde an alle Konsumenten repliziert. Fachlich ist dieser zeitliche Versatz im Allgemeinen unkritisch: Nach einer Stammdatenänderung bis zum nächsten Bezahlvorgang sind einige Klicks notwendig, in der Zwischenzeit sind die Daten längst aktualisiert.

Somit wird ein sehr hoher Grad der Ausfallsicherheit erreicht, die sich in unterschiedlichen Service Level der Geschäftsprozesse abbilden lassen. Das Paydirekt-System ist so in der Lage, Zahlungen durchzuführen, auch wenn andere Services, zum Beispiel zur Kundendatenverwaltung nicht zur Verfügung stehen, da alle für die Zahlung notwendigen Kundendaten in der lokalen Datenbank gespeichert sind.

Ein weiterer fachlicher Vorteil ist die vollständige Verfügbarkeit aller historischer Daten und der Information, wer bestimmte Informationen verändert hat. paydirekt steht damit ein vollständiger Audit-Trail zur Verfügung. Aus den Events können bei Bedarf beliebige Reports neu erstellt werden, was zum Beispiel im Hinblick auf Fraud-Analysen einen wertvollen Mehrwert liefern kann.

Resilienz als fachliche Anforderung

Neben der oben erwähnten Ausfallsicherheit durch lokale Projektionen verwendet das paydirekt-System das Hystrix-Framework von Netflix [Hys]. Durch Circuit-Breaker [Fow14] und Fallback-Strategien wird dem Benutzer ein lauffähiges System präsentiert, auch wenn z. B. ein SMS-Gateway oder ein Bank-System nicht erreichbar ist. Bei über 1000 angebundenen Instituten ist immer damit zu rechnen, dass einzelne Systeme aus unterschiedlichen Gründen nicht erreichbar sind.

Senacor Technologies hat hierzu bei allen externen Schnittstellen mit paydirekt und den teilnehmenden Banken fachliche Fallback-Strategien konzipiert und umgesetzt. Während die Lösung bei einem fehlerhaften SMS-Versand relativ einfach sein kann (ein weiteres SMS-Gateway von einem anderen Anbieter verwenden), müssen im Zahlungsverkehr komplexe fachliche Notfall-Konzepte erstellt werden, indem für technische Parameter wie *Timeouts*, *Request-Volume-Threshold* und *Sleep-Window* konkrete Vorgaben definiert und Fallback-Szenarien ausgearbeitet werden.

Für Käufer und Händler hat dies den Vorteil, dass trotz eines Ausfalls eines Teilsystems Bezahlungen in der Regel trotzdem möglich sind.

Tiefgreifende Testautomatisierung und Continuous Delivery

Bei den teilnehmenden Instituten ist es üblich, pro Jahr etwa zwei bis drei große Produktionsreleases durchzuführen. Diese Releases werden oft wochenlang vorbereitet und dann an einem Wochenende durchgeführt, häufig verbunden mit einer Ausfallzeit des gesamten Systems. Neue Features müssen in der Regel mehrere Monate voraus eingeplant und beauftragt werden.

paydirekt hat sich entschieden, im zweiwöchentlichen Abstand Änderungen direkt in Produktion zu bringen. Damit wird paydirekt den dynamischen Marktanforderungen gerecht, die im E-Commerce notwendig sind. Ein Produktmanager hat so die Möglichkeit Anforderungen dynamisch zu priorisieren und auf das Feedback der Kunden, Banken und Händler zu reagieren.

Die Microservice-Architektur unterstützt dieses Vorgehen, da kleine Einheiten schneller und einfacher deployed werden können. Ein Deployment dauert nur wenige Sekunden und erfolgt rollierend über alle Knoten hinweg, der Kunde bekommt hiervor nichts mit. In der Regel wird in einem Zwei-Wochen-Sprint zudem nur ein kleiner Teil der Services verändert, es muss also nicht das Gesamtsystem neu installiert werden.

Voraussetzung für dieses Vorgehen ist eine umfassende Testautomatisierung. Neben den Unit- und Modultests eines Microservices wird auch das Zusammenspiel der gesamten Zielkonfiguration und der externen Schnittstellen integrativ getestet. Microservices machen diese Tests beherrschbar: Unit- und Modultests stellen den Großteil der Tests dar.

Durch die klar definierte fachliche Funktion eines Services lassen sich gut Testfälle ableiten und umsetzen, eine zu geringe Testabdeckung wird in Tools wie *SonarQube* schnell sichtbar. Kommt es bei Integrationstests zu Fehlern, lassen sich die Ursachen in der Regel auf einen fehlerhaften Service zurückführen und beheben.

Kernstück ist eine vollautomatisierte Deployment-Pipeline. Diese ermöglicht es den Entwicklern, Änderungen qualitätsgesichert über verschiedene Testumgebungen bis in die Produktion zu bringen. Nur wenn in jeder Stufe alle Tests erfolgreich sind (Unit-Tests, Integrationstests, Frontend-Tests, explorative Tests, Smoke-Tests) kann ein Software-Artefakt ausgeliefert werden. Da für ein Deployment keine manuellen Schritte erforderlich sind, Deployments häufig wiederholt und über wenige Klicks ausgeführt werden, ist auch das regelmäßige Produktions-Deployment Alltagsgeschäft.

Docker (noch) nicht im Einsatz

Bei der Infrastruktur hat sich Senacor Technologies vorerst gegen eine Container-basierte Docker-Infrastruktur und für klassische Hypervisoren entschieden. Zum Projektstart Anfang 2015 waren in der Docker-Engine noch signifikante Sicherheitslücken bekannt (vgl. [Wal14]) und das Docker-Ökosystem war und ist bis heute hoch volatil. Für paydirekt ist Sicherheit und Stabilität in der Betriebsinfrastruktur von größter Bedeutung und wichtiger als eine möglichst effiziente Hardware-Auslastung.

Dennoch bleibt Docker mittelfristig eine Option: Mittlerweile gilt Docker als ausgereift und mit den Cluster-Systemen wie Swarm oder Kubernetes und virtuellen Netzen entwickelten sich interessante neue Features [Fir15]. Wenn sich paydirekt für den Einsatz von Docker entscheidet, dann können einzelne Microservices auf Docker aufgesetzt und getestet werden. Eine Migration bestehender Services ist dann schrittweise möglich.

Entwickler schneller produktiv

Da die Microservices aufgrund der eindeutigen Fachlichkeit und klar definierter Schnittstellen in ihrer Komplexität überschaubar sind, können sich neue Entwickler im Projekt schnell einen Überblick verschaffen und Funktionen in einem Microservice bereits nach wenigen Tagen selbständig umsetzen. Die Einarbeitung ist so praxisnah gestaltet, dass neue Mitarbeiter schnell produktiv und damit auch motiviert werden.

Die Einarbeitung erfolgt immer zusammen mit erfahrenen Entwicklern: Durch die konsequente Implementierung in Feature-Branches und Pull-Requests mit obligatorischer Prüfung durch erfahrene Kollegen sowie einer durchgängigen

Literaturverzeichnis

[FuL14] Fowler, Martin und Lewis, James. Microservices. A definition of this new architectural term. 25. März 2014. http://martinfowler.com/articles/microservices.html.

[Mil09] Miller, Jeremy. MSDN Magazine. Patterns in Practice - Convention Over Configuration. Februar 2009. https://msdn.microsoft.com/en-us/magazine/dd419655.aspx.

[Hod16] Hodgson, Pete. Feature Toggles. 08. Februar 2016. http://martinfowler.com/articles/feature-toggles.html.

[Hat] https://spring.io/understanding/HATEOAS.

[Kel] Kelly, Mike. HAL - Hypertext Application Language . http://stateless.co/hal_specification.html.

[Akka] Typesafe Inc. Akka Persistence. http://doc.akka.io/docs/akka/current/scala/persistence.html#Event_sourcing.

[Vog08] Vogels, Werner. All Things Distributed. Eventually Consistent - Revisited. 22. 12 2008. http://www.allthingsdistributed.com/2008/12/eventually_consistent.html.

[Bre04] Brewer, Eric A. Towards Robust Distributed Systems. 19. Juli 2004. http://www.cs.berkeley.edu/~brewer/cs262b-2004/PODC-keynote.pdf.

[Gre14] Greiner, Robert. CAP Theorem: Revisited. 14. August 2014. http://robertgreiner.com/2014/08/cap-theorem-revisited/.

[Hys] https://github.com/Netflix/Hystrix.

[Fow14] Fowler, Martin. CircuitBreaker. 7. März 2014. http://martinfowler.com/bliki/CircuitBreaker.html.

[Wal14] Walsh, Daniel. Are Docker containers really secure? 22. Juli 2014. https://opensource.com/business/14/7/docker-security-selinux.

[Fir15] Firshman, Ben. Announcing Docker 1.9: Production-ready Swarm and Multi-host Networking. Docker Blog. 3. November 2015. https://blog.docker.com/2015/11/docker-1-9-production-ready-swarm-multi-host-networking/.

Messung der Testabdeckung und weiterer Code-Qualitätsmetriken wird die Qualität der Gesamtanwendung sichergestellt.

Für Senacor Technologies als Entwickler und Betreiber der Anwendung ergibt sich durch die gewählte Architektur somit ein wirtschaftlicher Vorteil durch den effizienten Einarbeitungsprozess und die damit verbundene Flexibilität in der Team-Struktur.

Innovative Plattform für die deutsche Kreditwirtschaft

Die deutsche Kreditwirtschaft hat mit paydirekt eine gemeinsame Plattform geschaffen, um ihren Kunden ein schnelles, einfaches und sicheres Bezahlverfahren im stark wachsenden E-Commerce-Markt zu bieten. Die geschaffene IT-Architektur unterstützt dank flexibler Microservices agile Prozesse in der Organisation und die teilnehmenden Institute sind so in der Lage, das Bezahlsystem in kurzen Release-Zyklen zu erweitern, neue Funktionen zu integrieren und sicher zu betreiben.

Das Bezahlsystem ist in Produktion mit über 30 Microservices sehr stabil angelaufen und zeigt einen hervorragenden Servicelevel hinsichtlich Performance und Verfügbarkeit. Inzwischen ist das zweiwöchentliche Releasing neuer Features zur Normalität geworden. ■