OFFLINE-WEB-ARCHITEKTUREN: HERAUSFORDERUNGEN UND LÖSUNGSANSÄTZE

Die Möglichkeit der Offline-Fähigkeit von web-basierten Applikationen ist eine Anforderung, die auch heute – in Zeiten von DSL und UMTS – aus vielen Unternehmen nicht wegzudenken ist. Die Umsetzung von offline-fähigen Web-Anwendungen ist aber nicht trivial: Herausforderungen liegen im Bereich Datensynchronisierung und Speicherung von Daten auf lokalen Endgeräten des Clients. Dieser Artikel identifiziert die Herausforderungen und Schwierigkeiten und zeigt Lösungsansätze auf.

die autoren

Kerstin Maier
[E-Mail: kerstin.maier@widas.de]
arbeitet als Senior IT-Architektin bei der WidasConcepts GmbH, einer IT-Unternehmensberatung mit dem Fokus auf IT-Architekturen und Anwendungsentwicklung. Zu ihren Schwerpunkten gehören JEE-Architekturen, Web-Anwendungen sowie Web-2.0-Themen.

Thomas Widmann
[E-Mail: thomas.widmann@widas.de]
ist Geschäftsführer der WidasConcepts GmbH und beschäftigt sich in erster Linie mit den Themen IT-Strategien, IT-Architekturen und Business-Prozessmanagement.

Auf den ersten Blick scheint die Idee einer Offline-Web-Anwendung paradox. Schließlich zeichnet sich eine Web-Anwendung dadurch aus, dass sie online im Web läuft. Und sicherlich gibt es Anwendungsbereiche, wo eine Offline-Web-Anwendung nicht sinnvoll ist. So wird eine offline-fähige Chat-Anwendung oder Social Community relativ wenig Sinn machen. Andere Anwendungen hingegen möchte der Anwender jederzeit nutzen können. Muss er zum Beispiel unterwegs eine wichtige Aufgabe in seine To-Do-Liste eintragen, möchte er dies sofort tun und nicht erst dann, wenn eine Internet-Verbindung hergestellt ist. Auch bei größeren Web-Anwendungen, die z. B. für Verkaufstätigkeiten von einem mobilen Vertriebsmitarbeiter genutzt werden, spielt Offline-Fähigkeit oft eine große Rolle. Aber was genau muss eine Offline-Web-Anwendung eigentlich können?

Anforderungen und Herausforderungen

Für den Anwender einer offline-fähigen Web-Anwendung steht der Benutzungskomfort im Mittelpunkt. Er möchte möglichst wenig in seiner Arbeit gestört werden, wenn die Anwendung vom Online- in den Offline-Betrieb wechselt. Außerdem darf es keinen übermäßig hohen Installationsaufwand für lokale Komponenten geben.

Wenn die von der Anwendung offline verwalteten Daten auch verändert werden können, kommt ein weiterer schwieriger Aspekt hinzu: Die offline bearbeiteten Daten müssen am Server wieder synchronisiert werden. Dies wird besonders interessant, wenn mehrere Anwender die gleichen Daten bearbeiten können. Ressourcenintensive Synchronisationsprozesse dürfen nicht dazu führen, dass die Anwendung an der Benutzungsoberfläche hakelt oder ständig Fehlermeldungen ausgegeben werden. Deshalb besteht die Anforderung, die Synchronisation möglichst automatisiert im Hintergrund ablaufen zu lassen. Daraus ergibt sich eine ganze Menge an fachlichen und technischen Fragen, die ein Anwendungsarchitekt im Vorfeld analysieren und beantworten muss, um eine offline-fähige Web-Anwendung zu designen.

Was offline?

Die erste Frage lautet: Welche Daten und welche Features der Anwendung müssen offline zur Verfügung stehen?

In der Regel ist es weder sinnvoll noch technisch möglich, alle Daten und Funktionen auch offline zur Verfügung zu stellen. Besteht diese Anforderung dennoch, muss kritisch die Frage gestellt werden, ob die Anwendung überhaupt eine Web-Anwendung sein sollte und nicht eher eine Desktop-Anwendung (die gegebenenfalls über Online-Funktionalitäten verfügt).

Bei den offline bereit zu stellenden Daten muss zwischen statischen Darstellungsinhalten (zum Beispiel *Cascading Style Sheets (CSS)*, Bilder, HTML) und dynamischen Darstellungsinhalten (zum Beispiel serverseitige Skripte zur dynamischen Generierung von Benutzungsoberflächeninhalten) unterschieden werden. Außerdem verwaltet eine Web-Anwendung Geschäftsobjekte (Datenobjekte, die fachliche Daten speichern, z. B. Personendaten, Antragsdaten oder Inhalte einer To-Do-Liste). Statische Ressourcen lassen sich am einfachsten speichern. Sie haben eine überschaubare Datenmenge und keine dynamische Funktionalität. Geschäftsobjekte sind in der Speicherung anspruchsvoller, da sie in der Regel größere Datenmengen aufweisen. Serverseitige Skripte brauchen eine lokale Ausführungsumgebung oder müssen technologisch auf dem Client anders umgesetzt werden. Damit ist das Bereitstellen dieser Inhalte am aufwändigsten.

Wann offline, wann online?

Erst wenn die erste Frage geklärt ist, kann überlegt werden, wann die Anwendung auf den Server zugreift und wann sie offline läuft. Wenn keine Internet-Verbindung vorhanden ist, muss sie offline laufen – das ist offensichtlich. Im Gegenzug muss sie aber nicht automatisch online arbeiten, wenn eine Online-Verbindung existiert.

Wann und wie synchronisieren?

Bei einer offline-fähigen Web-Anwendung verlieren die Daten zwischen lokalem Speicher und Server ständig an Synchronität. Bei der Synchronisation stellen sich in der Regel zwei Aufgaben:

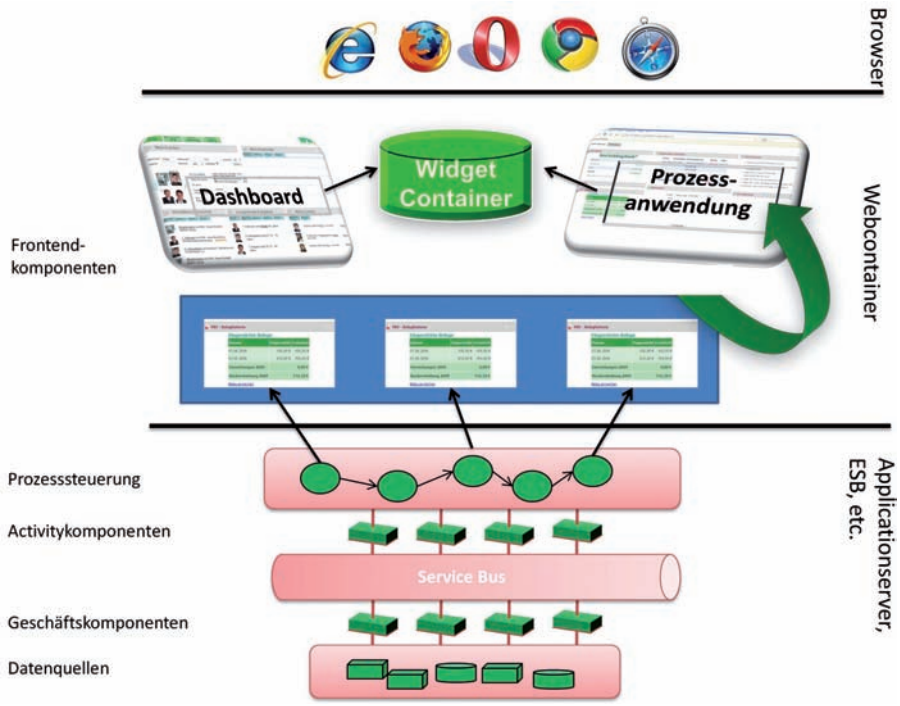

Abb. 1: Bestehende Web-Architektur ohne Offline-Funktionalität.

1. Neuere Versionen von Ressourcen, Geschäftsobjekten und dynamischen Skripten müssen online vom Server gezogen werden.
2. Offline veränderte Geschäftsobjekte müssen an den Server übergeben und dort gespeichert werden.

Die Komplexität der Synchronisationsaufgaben kann – je nach Anforderung – von „einfach" zu „sehr schwierig" variieren. Die einfachste Variante ist es, den lokalen Speicher nur als flüchtigen *Read-Only Cache* vorzuhalten und bei der Synchronisation wegzuwerfen. Dies reicht aber nur in den wenigsten Fällen aus. Schwieriger wird es, sobald Daten offline verändert oder hinzugefügt werden können. Werden die Objekte nur von einem Anwender bearbeitet, ist dieser Vorgang noch unproblematisch. Eine besondere Herausforderung stellt sich, wenn mehrere Anwender offline das gleiche Objekt bearbeiten können. Hierfür müssen Regeln für die Konflikterkennung und -auflösung definiert und umgesetzt werden.

Offline-Architektur

Hat man die Anforderungen an die offlinefähige Web-Applikation definiert, muss die Offline-Architektur in die Gesamtarchitektur integriert werden. Meist gibt es bestehende Web-Anwendungen, die teilweise offline bereitstehen müssen. Häufig wurden diese Anwendungen ursprünglich nicht darauf ausgerichtet, auch Offline-Funktionalität zu bieten. Als Beispiel für die Erweiterung einer bestehenden Unternehmensarchitektur dient eine mehrschichtige Architektur, bestehend aus einem komponentenbasiertem Frontend, Prozesssteuerung, Aktivitätskomponenten und Geschäftskomponenten, die über Services nutzbar sind (siehe Abbildung 1).

Wie lässt sich in eine derartige Architektur die Anforderung der Offline-Fähigkeit integrieren? Oft muss der gesamte Architektur-Stack am Client zur Verfügung stehen, um die notwendigen Aktivitäten abbilden zu können:

- Präsentation
- Prozesssteuerung
- Geschäftslogik
- Persistenz

Hier gibt es zwei unterschiedliche Ansätze, die sich vor allem in ihrer Ablaufumgebung unterscheiden. Welcher Ansatz im Einzelfall gewählt wird, hängt stark davon ab, wie die bisherige Online-Web-Architektur aussieht, auf welchen Web-Entwicklungs-Frameworks sie basiert und wie viel Funktionalität offline benötigt wird.

Bei einer reinen Online-Web-Architektur erfolgt auf dem Client nur die Darstellung. Die Businesslogik erfolgt auf dem Web-Server bzw. in den Aktivitäts- und Geschäftskomponenten. Abhängig vom gewählten Web-Entwicklungsframework gilt das auch für die Präsentationslogik. *JSF*, *Wicket* oder *Tapestry* – komponentenorientierte Web-Frameworks, die den Status auf dem Server halten und Logik in Java implementieren – benötigen für jeden *Web-Request* einen Server-Roundtrip: vom Client zum Web-Server und zurück. Clientseitige Frameworks, wie *GWT*, *Flex* oder *JavaFx*, führen die Präsentationslogik auf dem Client aus. Im Folgenden stellen wir zwei Ansätze für die Integration von Offline-Fähigkeit vor und zeigen, wann sich welcher Ansatz eignet.

ALTERNATIVE 1: Einsatz von browser-basierten Offline-Frameworks

Browser-basierte Offline-Frameworks ermöglichen es, Web-Anwendungen oder Teile davon lokal auf dem Client auszuführen, auch wenn keine Internet-Verbindung besteht. Meist basieren sie auf *Ajax*- und *JavaScript*-Technologien und bieten Funktionen zum lokalen Speichern von Daten, zur Reaktion auf Online/Offline-Ereignisse und zur Unterstützung von Synchronisationsarbeiten.

Offline-Frameworks verlagern die Persistenz und mindestens die für den Offline-Betrieb benötigten Teile der Geschäftslogik in den Browser. Bei der Umsetzung der Anforderung nach Offline-Fähigkeit im Browser stellen sich bei Alternative 1 zwei Herausforderungen:

- Mehrfache Umsetzung der Geschäftslogik
- Integration in die Online-Architektur

Im Browser steht als Programmiersprache nur JavaScript zur Verfügung. Die im Backend genutzte Sprache (z. B. Java) stellt meist keine Option dar. Das führt zur mehrfachen Entwicklung der Logik in verschiedenen Umgebungen. Web-Entwicklungs-Frameworks, die den Server bei jeder Anfrage benötigen, um die Präsentationslogik z. B. in Java umzusetzen, unterstützen den Entwickler kaum bei der Umsetzung von Offline-Applikationen. Zwar können Offline-Komponenten in diesen Frame-

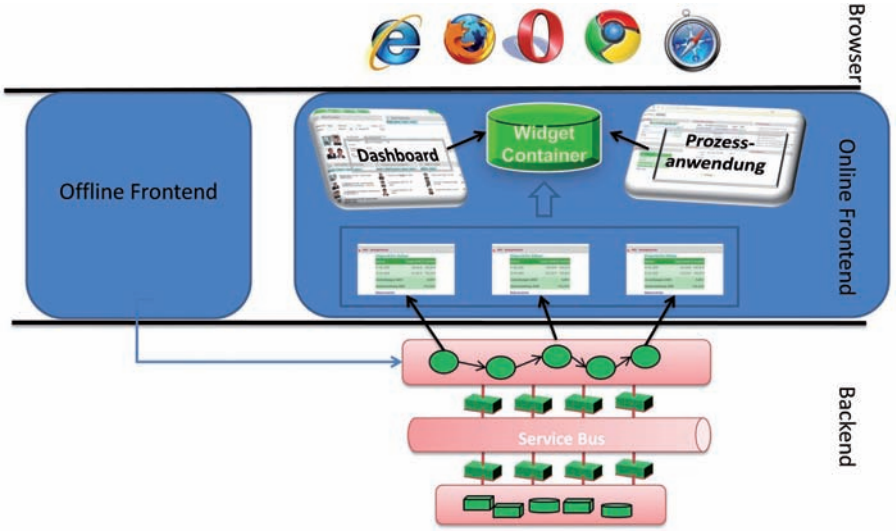

Abb. 2: Alternativer Zugriff auf das Backend beim Offline-Betrieb.

works gekapselt werden, jedoch muss die Präsentationslogik doppelt implementiert werden: für den Online- und den Offline-Fall. Eine Verlagerung der Logik zum Client im Online- und Offline-Betrieb untergräbt das Konzept dieser Frameworks. Abbildung 2 zeigt diesen Fall.

Bei einigen Web-Frameworks, wie z. B. dem *Google Web Toolkit (GWT)* und vor allem den client-seitigen Frameworks *Silverlight, Flex* und *JavaFX*, läuft die Präsentationslogik im Client ab. Im Offline-Fall kann die Präsentationslogik weiter auf dem Client ausgeführt werden. Der Zugriff auf Services und Aktivitätskomponenten wird bei der Verfügbarkeit des Backends über meist synchrone *Remote*-Schnittstellen (*Remote Procedure Calls*) implementiert. Steht das Backend nicht zur Verfügung, muss dieser Zugriff gekapselt werden. Die Implementierung erfolgt über JavaScript oder eine andere Skriptsprache, die im Client zur Verfügung steht. Eine Mehrfachimplementierung der Geschäftslogik ist dabei nicht vermeidbar. Die sinnvollen Integrationsoptionen hängen also stark von der verwendeten Frontend-Technologie ab. Bei Frameworks, die die Präsentationslogik im Client abbilden, wird beim Zugriff auf die Remote-Services ein Proxy verwendet, der im Offline-Fall lokal ausgeführt wird. Das Frontend muss nicht ausgetauscht werden. Anders sieht es bei serverorientierten Web-Frameworks wie *Wicket* und *JSF* aus. Die Präsentationslogik muss ausgetauscht werden, da eine client-seitige Ausführung nicht möglich ist. Entscheidet man sich dafür, die Offline-Funktionalität einer Web-Anwendung mit einem browser-basierten Offline-Framework umzusetzen, muss ein passendes Framework gefunden werden. Die aktuellen Entwicklungen im Bereich der Offline-Frameworks stellen wir im Folgenden kurz vor.

Offline-Frameworks und HTML5

Die HTML5-Spezifikation (siehe Kasten 1) definiert zur Unterstützung von Offline-Funktionalität bei Web-Anwendungen als zentrales Feature einen *Offline Application Cache*. Außerdem gibt es einige ergänzende Spezifikationen, die weitere Offline-Funktionalität zur Verfügung stellen. Die Offline-Features von HTML5 bestehen aus den folgenden Komponenten:

- die *Offline Application Caching API*
- die Spezifikation der *Web-Storage*, bestehend aus einer Programmierschnittstelle zur client-seitigen Speicherung von session-spezifischen Daten und einer zur Speicherung von session-übergreifenden Daten
- einer client-seitigen JavaScript-Datenbank mit dem Namen *Web SQL Database*
- *Web Workers* zur Ausführung von Background-Prozessen

HTML5 ist eine Spezifikation des W3C-Konsortiums. Nach der finalen Fertigstellung soll sie die bestehenden Standards „HTML 4.01", „XHTML 1.0" und „DOM HTML Level 2" ersetzen und erweitern. Ziel ist es, die Unterschiede zwischen den bisherigen Sprachausprägungen zu verringern und eine bessere Zusammenarbeit der Komponenten zu erreichen. Außerdem bietet HTML5 neue Funktionalität, um Web-2.0-Inhalte und dynamische Inhalte besser zu unterstützen.

Kasten 1: HTML5 in Kürze.

HTML5 ApplicationCache

Der HTML5 ApplicationCache ermöglicht die Definition einer Manifest-Datei, in der alle Dateien angegeben werden, die vom Browser offline gecacht werden sollen. In die Datei werden alle Ressourcen eingetragen, die für den Offline-Betrieb benötigt werden.

Ruft der Anwender eine Seite auf, für die ein Cache definiert wurde, versucht der Browser, die Cache-Datei zu aktualisieren. Er lädt sich zunächst eine Kopie des Manifests. Beim ersten Aufruf lädt er alle definierten Ressourcen in den Cache. Wird die Seite erneut aufgerufen, überprüft er, ob sich das Manifest verändert hat. Falls ja, werden alle Ressourcen erneut heruntergeladen. Es gibt eine JavaScript-Programmierschnittstelle, um Veränderungen am Cache manuell herbeizuführen.

HTML5 Web Storage

Die HTML5-Web-Storage-Spezifikation definiert zwei *Key Value Stores* zur Speicherung von Daten auf dem lokalen Endgerät des Anwenders. Die beiden zentralen JavaScript-Objekte sind localStorage und sessionStorage. Der localStorage ist für eine langfristige Speicherung über mehrere Browser-Fenster hinweg gedacht. Der sessionStorage speichert kurzfristige Daten, die sich auf ein Browser-Fenster beziehen. Das Konzept ähnelt dem vom *Cookies*, ist aber für größere Datenmengen angelegt.

HTML5 Web SQL Datenbank

Die HTML5-Web-SQL-Datenbank-Spezifikation definiert eine relationale JavaScript-Datenbank. Diese ist auch für größere Datenmengen geeignet als der Application Cache oder der Web Storage. Die Programmierschnittstelle enthält Methoden zum Schreiben und Lesen von Daten sowie zur Reaktion auf erfolgreiche und fehlgeschlagene Zugriffe.

	ApplicationCache	Web Storage	Web Database	Web Workers
Internet Explorer	nein	ab 8.0	nein	nein
Mozilla Firefox	ab 3.5	ab 3.0	nein	ab 3.5
Safari	ab 4.0	ab 4.0	ab 3.2	ab 4.0
Chrome	ab 4.0	ab 3.0	ab 3.0	ab 3.0
Opera	nein	ab 10.5	ab 10.5	nein

Tabelle 1: HTML5-Offline-Unterstützung von Browsern.

Web Workers

Web Workers ist ebenfalls nicht Teil von HTML5, sondern eine eigene Spezifikation. Die Unterstützung von Offline-Funktionen ist nicht das Hauptziel von Web Workers, die Spezifikation ist aber für die Umsetzung von Synchronisationsarbeiten definitiv interessant. Synchronisationsaufgaben sollten im Browser immer im Hintergrund ablaufen, damit die Web-Seite für den Anwender nicht blockiert und Benutzereingaben weiterhin möglich sind. Web Workers stellt einen Standard zur Verfügung, um JavaScript im Browser im Hintergrund parallel auszuführen.

Die HTML5-Spezifikation befindet sich derzeit im Entwurfsstadium. Tabelle 1 zeigt, welche Spezifikationen von den aktuellen Browsern schon unterstützt werden.

Architektonische Lösungsansätze der Offline-Frameworks

Wie sieht bei den vom W3C propagierten Standards und bei bestehenden Offline-Frameworks (siehe Kasten 2) nun das architektonische Zusammenspiel der Komponenten aus? Hier zeigt sich, dass die Standards dazu tendieren, die Anwendung als MVC-Architektur umzusetzen

- *Model:* Kapselung aller Daten in *Model*-Objekten, Data-Switch auf *Model*-Ebene
- *View:* client-seitige Generierung
- *Controller:* Event-Handler der *Views*

Model

Greift eine offline-fähige Web-Anwendung auf Daten zu oder soll sie diese abspeichern, muss in jedem Fall immer eine Entscheidung getroffen werden, ob die Anwendung auf den lokalen Speicher zugreift oder auf den Server. Für die Benutzungsoberfläche sollte diese Entscheidung transparent sein. Für den Zugriff auf Daten empfiehlt es sich deshalb, einen *Single Point Of Contact* vorzusehen. Dieser dient in Form einer Datenschicht mit vorgeschaltetem Switch dazu, alle Speicherungs- und Lade-*Requests* zentral abzuhandeln.

> Google Gears ist eine im Mai 2007 veröffentlichte Browser-Erweiterung für Firefox und Internet Explorer, die es möglich macht, Web-Seiten offline zu benutzen. Die Offline-Funktionalität von GoogleMail ist zum Beispiel mit Google Gears umgesetzt. Wie im Februar 2010 verkündet, wird Google Gears nicht mehr aktiv weiterentwickelt, da das Entwicklerteam sich darauf konzentrieren will, alle Gears-Features in Standards wie HTML5 einzubringen. Viele Elemente des Gears-Ansatzes sind in die HTML5-Spezifikation eingeflossen.

Kasten 2: Google Gears in Kürze.

Der Switch entscheidet, ob die Daten vom Server oder vom lokalen Store bzw. eine Kombination von beiden geladen werden. Einen Switch vorzusehen, ist recht einfach, wenn man die Möglichkeit hat, auf der „grünen Wiese" anzufangen. Soll eine bestehende Web-Anwendung auf eine Offline-Fähigkeit umgestellt werden, stellt sich jedoch alles etwas komplizierter dar.

Bei bestehenden Anwendungen ist häufig keine Datenschicht vorhanden. Stattdessen verteilen sich Ajax-Aufrufe über den ganzen Code. Dann müssen alle Stellen identifiziert werden, die offline benötigt werden und an denen Daten vom Server geladen oder an den Server gesendet werden. Wenn das erledigt ist, muss entschieden werden, ob alle Datenzugriffe über einen zentralen Switch laufen sollen. Werden nur sehr wenige Daten offline benötigt, ist es eventuell aus wirtschaftlicher Sicht sinnvoller, nur die betroffenen Stellen anzupassen und auf eine sauberere und schönere Lösung zu verzichten.

View

Bei der Generierung der View in einer offline-fähigen Web-Anwendung ist es zwingend notwendig, dass diese im Offline-Betrieb auch komplett offline generiert werden kann. Der einfachste Ansatz, um das zu erreichen, ist es, die Views möglich statisch zu halten und vor allem keine vom Server berechneten HTML- Codefragmente zu verwenden. Die HTML-Spezifikation präferiert diesen Ansatz.

Controller

Je mehr Zustand und Verhalten sich bereits im Client befinden, desto einfacher ist es, eine bestehende Web-Anwendung auf Offline-Fähigkeit umzustellen. So ist ein *Rich Client*, bei dem das Meiste im Client abläuft, einfacher offline zu nehmen als ein *Thin Client*, bei dem der Server nur noch einfaches HTML an den Client schickt.

Bei einer neuen Web-Anwendung sollte darauf verzichtet werden, Logik im Client und Server redundant auszuführen. Bei der Integration von Offline-Fähigkeit in eine bestehende Web-Anwendung lässt sich das jedoch meistens nicht vermeiden. Für die Synchronisation empfiehlt sich auf jeden Fall eine eigene Synchronisations-Engine, die im Hintergrund ablaufen muss und die den Browser nicht blockieren darf. Die Oberfläche bekommt höchstens in Form von Statusmeldungen Informationen über Synchronisationsarbeiten. Bestehende Frameworks und Anwendungen, wie das in Kasten 3 beschriebene dojo.off oder „Remember the Milk" (eine web-basierte Aufgabenliste), verwenden zur Durchführung der Synchronisation einen *Action-Log*-Ansatz. Bei dieser Vorgehensweise werden alle Benutzereingaben des Anwenders im Offline-Betrieb in einem loka-

Dojo.off war früher unter dem Namen „Dojo Offline Toolkit" ein eigenständiges proprietäres Framework für Offline-Ajax. Dann wurde es unter dem Namen „dojo.off" Teil von „Dojox" (den Dojo-Erweiterungen) und kapselte und erweiterte die Funktionalität von Google Gears. Im Vergleich zu Google Gears bietet es einige zusätzliche Funktionen, zum Beispiel ein Framework für die Umsetzung der Synchronisationsarbeiten.
Zu beachten ist, dass Dojox.off in der aktuellenVersion von Dojo (1.4) wegen der Google-Aufgabe von Google Gears (siehe Kasten 2) nicht mehr enthalten ist. Laut Aussage der Dojo-Entwickler soll es weiterentwickelt und dann als Framework, das die neuen HTML5-Spezifikationen unterstützt, neu herausgebracht werden.

Kasten 3: Dojo.off in Kürze.

len *Action-Log* gespeichert. Dabei wird jede Offline-Aktion als *Action* persistiert. Dementsprechend müssen alle zur Ausführung der *Action* notwendigen Daten lokal gespeichert werden. Beim Upload werden die *Actions* in der richtigen Reihenfolge sequenziell gegen den Server ausgeführt.

ALTERNATIVE 2:
Lokale Server-Infrastruktur

Werden sehr viele Funktionen im Offline-Fall benötigt, haben Offline-Frameworks ihre Grenzen. Die Alternative ist die Installation eines lokalen Servers. Dabei werden die benötigten Backend-Komponenten lokal installiert. Die verwendete Technologie hängt von der des Backends ab. Der Client verfügt über deutlich weniger Ressourcen als der Server. Deswegen sollten zum einen die Funktionalität begrenzt und zum anderen leichtgewichtige Server-Komponenten verwendet werden. Prädestiniert für den lokalen Einsatz sind OSGi-Container, die ihren Ursprung im Bereich eingebetteter Systeme haben, oder leichtgewichtige Web-Container.

Bei Nichtverfügbarkeit des Servers wird – für den Anwender transparent – die URL zur aufgerufenen Ressource gewechselt. Die lokale Prozesssteuerung speichert die Ergebnisse der Aktivitäten lokal und synchronisiert die Ergebnisse beim Online-Gang mit den zentralen Komponenten. Der Vorteil gegenüber den Offline-Frameworks ist die Verfügbarkeit der im Backend verwendeten Programmierumgebung. Die Aktivitätskomponenten und die Präsentationslogik können auf dem Client verwendet werden. Dabei muss die Anbindung der Aktivität an die Backend-Komponenten angepasst werden. Hier bietet sich das Proxy-Muster an, wobei die Schnittstelle der Backend-Services gegen eine lokale Cache-Implementierung ausgetauscht wird. Eine Mehrfachentwicklung entfällt. Auch bei der Verwendung einer lokalen Server-Umgebung gilt es, Herausforderungen zu meistern:

- Statusübergang bei Nichtverfügbarkeit
- Deployment der lokalen Infrastruktur

Sollen laufende Aktivitäten bei einer unerwarteten Nichtverfügbarkeit des Backends lokal übernommen werden, kommt man an einer andauernden Synchronisation des Status nicht vorbei. Häufig rechtfertigt der Aufwand dieser Anforderung nicht den Nutzen. Benötigt der Anwender jedoch diese Funktionalität, werden die lokal zur Verfügung stehenden Komponenten auch im Online-Fall genutzt. Ein Proxy leitet die Anfragen an das zentrale Backend weiter. Nur im Offline-Fall werden die Daten lokal gespeichert und bei Wiederverfügbarkeit synchronisiert.

Neben der Verwaltung laufender Anwendungen ist das Deployment der lokalen Infrastruktur eine Herausforderung. Die seit Jahren verfügbare und stabile Technologie *JavaWebStart* bietet hier eine Lösung. Der Servlet-Container sowie benötigte Deployment-Artefakte werden bei Bedarf lokal installiert. Lediglich die Installationszeit, die je nach der zur Verfügung stehenden Bandbreite zwischen einigen Sekunden und mehreren Minuten liegt, muss in Kauf genommen werden. Diese fällt jedoch nur bei der initialen Installation bzw. beim Update der Programmversion an.

Fazit

Zur Umsetzung der Anforderung nach Offline-Fähigkeit bieten sich die zwei vorgestellten Architekturvarianten an. Soll ein großer Ausschnitt der Funktionalität offline verwendet werden, sollte auf die lokale Server-Infrastruktur zurückgegriffen werden. Ebenso verhält es sich bei bestehenden Web-Anwendungen, bei denen im Frontend ein server-zentriertes Web-Framework, wie beispielsweise JSF, SpringMVC oder Wicket, verwendet wird. In diesen Fällen wäre der Aufwand zur Umsetzung der Offline-Fähigkeit mit im Browser lauffähigen Technologien zu zeitaufwändig und würde zu viel redundanter Funktionalität in Client und Server führen.

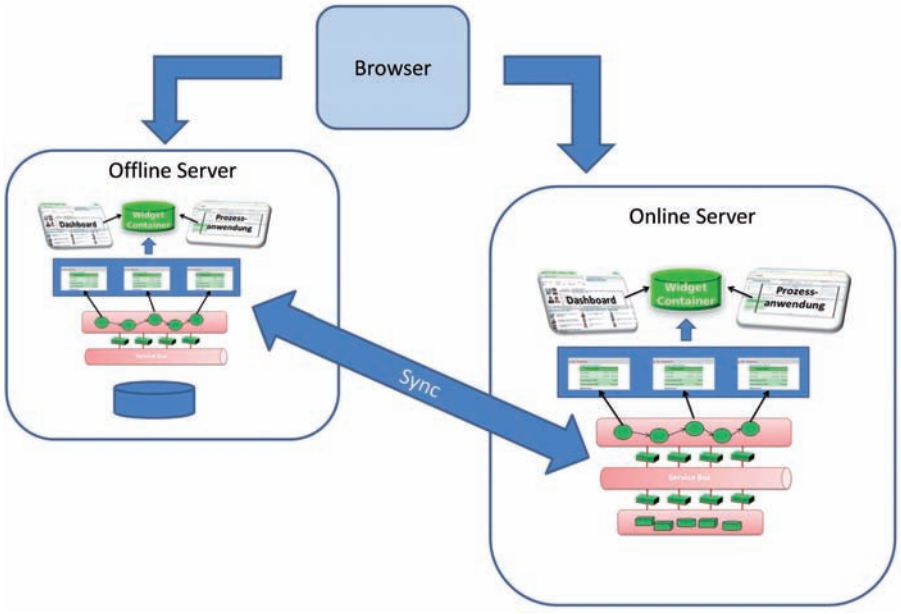

Abb. 3: Zusätzliche lokale Server-Infrastruktur.

Browser-basierte Offline-Frameworks eignen sich insbesondere, wenn client-zentrierte Web-Technologien verwendet werden bzw. wenn nur ein Ausschnitt der Gesamtfunktionalität offline zur Verfügung stehen soll. Ebenso verhält es sich mit der Erstellung einer neuen Anwendung, bei der die Offline-Fähigkeit im Client eine große Rolle spielt. Hier kann die Architektur von vorneherein so gestaltet werden, dass sie diesen Ansatz optimal unterstützt.

Die HTML5-Spezifikation stellt einige Lösungen für die Umsetzung von Offline-Fähigkeit im Browser in Aussicht. Die Spezifikationen befinden sich aber alle noch im Entwurfsstadium. Deshalb ist die Browser-Unterstützung dieser Features aktuell noch sehr uneinheitlich und – je nach Browser – mehr oder weniger eingeschränkt. Vor allem die beiden weit verbreiteten Browser „Internet Explorer" und „Mozilla Firefox" hinken in ihren aktuellen Versionen etwas hinterher. Ebenso hängt die Framework-Unterstützung für Offline-Features aktuell etwas in der Luft. „Google Gears" wurde aufgegeben, um stattdessen die HTML5-Unterstützung voranzutreiben. Auch die Offline-Funktionen von Dojo sollen zukünftig in Richtung HTML5-Unterstützung weiterentwickelt werden. Konkretes gibt es aber in beiden Fällen noch nicht. Trotzdem kann es hilfreich sein, sich die in den älteren Frameworks benutzten Lösungen zunutze zu machen.

Schaut man sich an, was die Frameworks in der Vergangenheit geboten haben, zeigt sich, dass diese auch nur grundlegende Funktionalität bereitstellen können. Sie können mit mehr oder weniger zusätzlicher Eigenentwicklung zu einer Gesamtlösung zusammengesteckt werden. Praktische Beispiele lassen sich vor allem bei kleineren Web-Anwendungen mit Standard-Anwendungsgebieten für Offline-Funktionalität, wie z. B. Mail-Clients und To-Do-Listen, finden. Eine leicht umzusetzende „All-in-One"-Lösung für größere Geschäftsanwendungen, die auf Standards aufbaut, ist aber wohl noch lange nicht in Sicht. Insbesondere im Bereich der Synchronisation wird immer Eigenarbeit notwendig sein.

Die fachliche Anforderungsanalyse spielt beim Erstellen einer offline-fähigen Web-Anwendung eine zentrale Rolle. Es gibt Grenzen für die technische Lösbarkeit der Problemstellungen. Die Entscheidung, welche Datenzugriffe und Anwendungskomponenten in einer Offline-Anwendung zur Verfügung stehen müssen, kann nicht technisch gelöst werden. Genauso wenig ist es möglich, die Synchronisation allein mit technischen Mitteln zu lösen. Die Regeln für die Konfliktauflösung müssen im Vorfeld genau geklärt und definiert werden.

Vor allem die Umstellung einer bestehenden, historisch gewachsenen Web-Anwendung wird immer schwierig bleiben. Dabei ist auch die Wirtschaftlichkeitsrechnung im Auge zu behalten. Eine bestehende Anwendung um Offline-Funktionalität zu erweitern, hat meist eine erhöhte Komplexität bei Entwicklung und Wartung zur Folge. Hier ist deshalb genau zu prüfen, ob das aus ökonomischer Sicht wirklich sinnvoll ist. ■

Links

[Dojo] Dojo Framework, siehe: www.dojotoolkit.org

[Goog] Google Gears, siehe: http://gears.google.com

[HTML5] HTML5 Spezifikation, siehe: http://dev.w3.org/html5/spec/Overview.html

[Rem] Remember the Milk – Offlinefähige Web-Aufgabenliste, siehe: http://www.rememberthemilk.com