

Tobias Günther

Offline-Getriebe

Mit Gears programmierte Web-Anwendungen laufen auch ohne Netzzugang

Googles Browser-Plug-in Gears vereinfacht die Entwicklung von Web-Anwendungen, die auch ohne Netzverbindung funktionieren. Nichtsdestotrotz muss sich der Entwickler mit ein paar neuen Konzepten auseinandersetzen.

Ohne Nabelschnur ins Internet funktionieren Web-Anwendungen normalerweise nicht, weil sie keine Daten mehr mit dem Server austauschen können. Mit Google Gears können Entwickler ihre Web-Dienste jetzt auch autark offline ablaufen lassen [1]. Gears bildet dazu die Server-Umgebung lokal nach, fängt Abfragen ins Web ab und bedient sie selbst. Dazu verfügt die Browser-Erweiterung über einen eigenen Speicher mit einer SQLite-Datenbank.

Der Weg, den Gears auf diese Weise für die Entwicklung offline-fähiger Online-Anwendungen vorgibt, dürfte für viele Programmierer der des geringsten Widerstands sein. Statt sich in Adobe AIR oder die .NET-Entwicklung einarbeiten zu müssen, können sie mit gewohnten Techniken sofort loslegen: JavaScript gehört zu ihrem aktiven Wortschatz und auch SQLite als relationale Datenbank bringt keinen Web-Entwickler ins Schwitzen.

Auf Benutzerseite dürfte Gears – auf deutsch: Getriebe – ebenfalls schnell Anklang finden. Gears-Anwendungen laufen wie gewohnt im Browser ab; Anwender können also die vertrauten Oberflächen und Funktionen (weiter) nutzen. Um Gears einsetzen zu können, müssen sie nur ein 710 KByte kleines Plug-in für den Browser installieren (siehe Soft-Link). Gears läuft mit dem Internet Explorer ab Version 6.0 oder Firefox ab Version 1.5 und damit unter Linux, Windows XP/Vista und Mac OS X. Safari- und Opera-Versionen sind in Vorbereitung. Gears wird kostenlos als Open Source unter der New-BSD-Lizenz verteilt.

Wie sich fertige Gears-Anwendungen anfühlen, zeigt Googles eigener FeedReader [2]. Damit darf man Feeds auf den Rechner laden und im Browser auch ohne Internet-Verbindung mit ihnen hantieren. Ein zweiter Dienst, der Gears bereits einsetzt, ist der Web-2.0-Aufgabenverwalter Remember The Milk [3]. Ob der online oder offline arbeitet, bemerkt man nur an der Statusanzeige. Die Entwicklergemeinschaft der populären JavaScript-Bibliothek Dojo [4] will Gears als Basis ihres Offline-Toolkits verwenden. Es dürfte also nur eine Frage der Zeit sein, bis weitere Anwendungen Google Gears nutzen.

Drei Zahnräder

Google Gears besteht aus drei unabhängigen Komponenten, dem Mini-Webserver

„LocalServer“ der „Database“ und dem „WorkerPool“. Der LocalServer speichert Ressourcen (Bilder, CSS-Files, HTML-Seiten, Javascript-Dateien ...) auf dem PC des Anwenders. URLs, die auf solche Ressourcen verweisen, werden im Offline-Modus beim Aufruf abgefangen und aus diesem Cache bedient. Das Database-Modul lagert Anwendungsdaten auf dem PC des Anwen-

ders, sodass sie auch einen Neustart überleben. Dazu enthält Google Gears die Datenbank SQLite, eine vollwertige relationale Datenbank. Sie wird sogar mit fts2-Extension ausgeliefert, was Volltext-Suchen in den Datensätzen ermöglicht.

Je raffinierter die Anwendung, desto häufiger sind zeitintensive Operationen notwendig. Sie blockieren den Browser und beschwören



mitunter sogar Warn-Dialoge wie das berühmte „Nicht antwortendes Skript“ herauf, wenn ihre Verarbeitung zu viel Zeit beansprucht. Hier kommt der WorkerPool ins Spiel. Er verlagert Operationen in den Hintergrund und sorgt dafür, dass der Browser für weitere Interaktionen zur Verfügung steht.

Gears kann derzeit nicht überprüfen, ob der PC mit dem Internet verbunden ist oder nicht. Google stellt auf seiner Homepage zwei Möglichkeiten vor, damit umzugehen [4]: Bei einer sogenannten nichtmodalen Anwendungsarchitektur geht die Applikation davon aus, dass sie offline ist oder jederzeit die Netzverbindung verlieren kann. Sie benutzt daher den lokalen Speicher so häufig wie möglich und versucht im Hintergrund kleine Datensynchronisierungen mit dem Server zu erledigen, wann immer eine Online-Verbindung existiert.

Bei der modalen Anwendungsarchitektur dagegen stellt das Programm einen Mechanismus bereit, mit dem der Benutzer signalisieren kann, ob es lokal laufen soll – etwa einen Schalter wie beim Google Reader. Befindet sich das Programm im Offline-Modus, greift es auf die Ressourcen aus dem LocalServer zurück, auch wenn eine Netzwerkverbindung besteht. Greift der Benutzer ohne Netzwerkverbindung, aber im Online-Modus auf die Anwendung zu, gibt es einen Fehler, den das Programm abfangen muss.

Als weitere Möglichkeit ist auch eine ebenfalls modale Anwendung denkbar, bei der der Benutzer nicht manuell den Modus wechseln muss. Das Dojo Offline Toolkit zum Beispiel sendet alle fünf Sekunden eine Anfrage ins Netz. Je nachdem, ob diese eine Fehlermeldung produziert, oder nicht, weiß die Dojo-Anwendung, ob sie online oder offline ist.

Die einfachste Architektur für den Entwickler dürfte die rein modale sein. Besonders elegant ist sie nicht, da sie dem Benutzer Raum lässt für Fehler: Wer im ICE sitzend feststellt, dass er zu Hause den entscheidenden Klick auf „Offline-Modus“ vergessen hat, wird die modale Anwendung zum Teufel wünschen – Einfachheit für den Entwickler hin oder her.

Erster Schaltversuch

Ein simpler, in JavaScript realisierter Adressverwalter veranschaulicht die Funktionsweise von Google-Gears-Anwendungen. Er steht unter dem Soft-Link zum Herunterladen bereit. Um die Anwendung zu installieren, muss man das Archiv nur entpacken und auf seinen Webspaces hochladen. Statt in einer Datenbank speichert der Adressmanager die Daten in einer einfachen Datei, `javascripts/data.js`, auf dem Server. Achten Sie darauf, dass Sie diese Datei mit ausreichenden Dateirechten (775) versehen, damit das Skript die Änderungen auch schreiben kann. Das Programm setzt nur PHP5 voraus. Sollte auf Ihrem Server kein PHP5 laufen, können Sie mit ein paar Handgriffen den

PHP5-Zweizeiler `save_online.php`, der für das Befüllen der Datei `data.js` zuständig ist, auf Ihre Wunsch-Sprache portieren. Anschließend müssten Sie noch die Aufrufe des Skripts in `application.js` anpassen. Die Serverseite wäre für ein wirkliches Programm natürlich umfangreicher. Für dieses Beispiel haben wir sie auf ein Minimum gestutzt, da sie nicht der Punkt ist.

Gears kommt ins Spiel, sobald die Anwendung auf Knopfdruck in den Offline-Modus geht: Sie lädt sich ins Gears-Plug-in herunter und steht dann uneingeschränkt zur Arbeit bereit. Reload oder Browserneustart können ihr nichts anhaben. In diesem Zustand verlassen keine Daten den Browser – das PHP-Skript wird nicht ausgeführt und `data.js` nicht verändert. Schaltet der Benutzer die Anwendung wieder auf „online“, sendet sie den lokalen Datenbestand an den Server, was ein Blick in `data.js` beweist.

Blick ins Getriebe

Die ersten Zeilen aller Gears-Listings dürften sich ähneln. Denn als ersten Schritt muss eine solche Anwendung prüfen, ob auf dem Client denn überhaupt schon die Gears-Run-time installiert ist. Andernfalls sollte das Skript den Benutzer zur Installationsseite weiterleiten:

```
if (!window.google || !google.gears){
    location.href =
        "http://gears.google.com/?action=install&
        message=Bitte zuerst Gears installieren!";
}
```

Klickt der Benutzer auf den Knopf „In Offline-Modus wechseln“, legt unsere Beispielanwendung eine Datenbank auf dem Rechner des Anwenders an (Listing verkürzt):

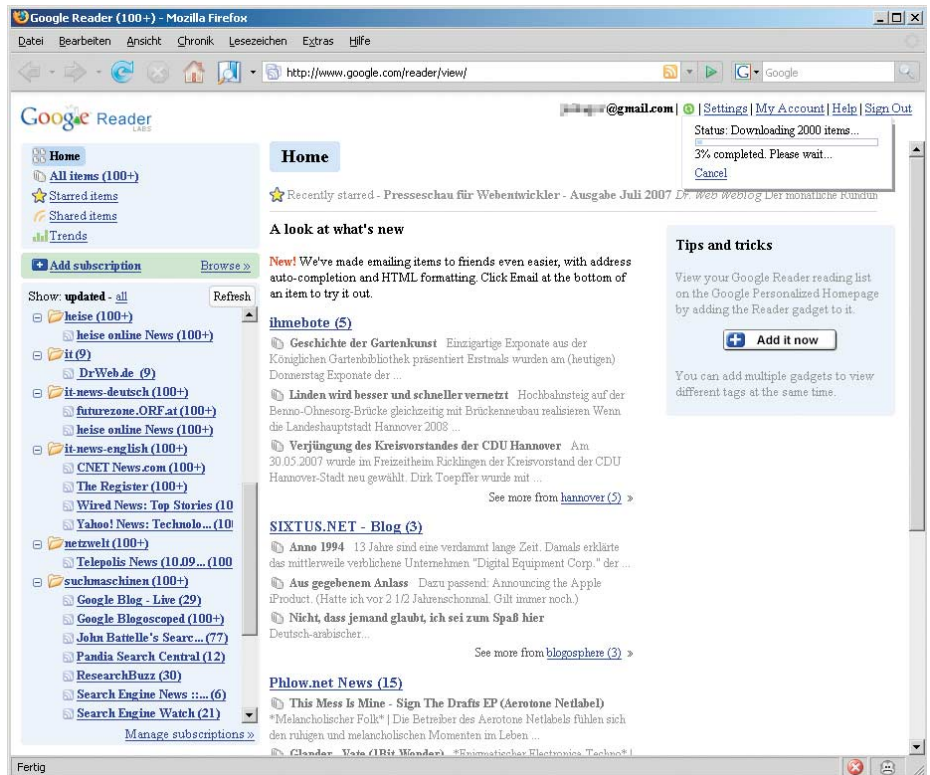
```
db = google.gears.factory.create('beta.database', '1.0');
db.open('addressmanager');
db.execute('create table if not exists addresses ('+
'id integer not null primary key autoincrement,' +
'name varchar(255),' +
'street varchar(255),' +
'description text)');
```

Die `execute`-Methode des Database-API ist für die Ausführung von SQL-Statements zuständig. Gears benutzt sie auch, um einzelne Datensätze abzuspeichern:

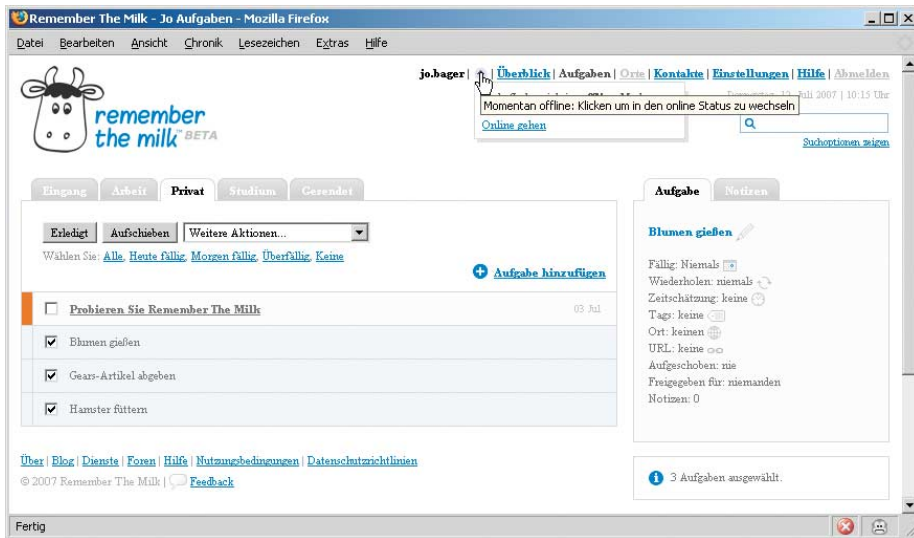
```
db.execute('INSERT INTO addresses
(name, street, description)
VALUES (?, ?, ?)');
[ name_val, street_val, description_val];
```

Statt die Namen der zu speichernden Variablen direkt bei `VALUES` einzusetzen, verwendet das Listing die Substitutionsparameter „?““. Gears ersetzt diese vor der Ausführung der SQL-Abfrage durch die Werte des Arrays, dem zweiten Parameter beim Aufruf von `execute`, und verpackt dabei gleich böse Zeichenkombinationen in den Parametern, so dass sie die Anwendung nicht mit SQL-Injection-Attacken kompromittieren können. Anschließend holt der Adressverwalter den Datenbestand vom Server und legt ihn in der Datenbank ab.

Im nächsten Schritt geht es darum, einen Behälter anzulegen, der die Dateien der An-



Bis der Google Reader seinen Datenstamm für die Offline-Nutzung heruntergeladen hat, können schon mal einige Minuten vergehen.



Bei Remember The Milk kann der Benutzer dank Gears seine To-Do-Listen auch offline verwalten.

wendung offline bereitstellt. Der LocalServer von Gears stellt solche in zwei Geschmacksrichtungen bereit, als ResourceStore und ManagedResourceStore. Gemeinsam ist beiden, dass der Entwickler die URLs der Dateien, die gecacht werden sollen, explizit benennen muss.

Unterschiedlich ist jedoch die Art, wie die Resource Stores mit Änderungen an den Original-Ressourcen umgehen. Der ResourceStore muss explizit dazu bewegt werden, Aktualisierungen durchzuführen. Das kann bei Online-Projekten, die schon mal ihr Aussehen oder ihre Funktionen ändern, schnell umständlich werden. Der ResourceStore eignet sich somit also eher für Dinge, die sich nicht ändern, wie CSS – oder für sehr simple Projekte oder Prototypen.

Beim ManagedResourceStore dagegen ist genau festgelegt, wann er welche Ressourcen neu laden soll. In einer auf dem Server hinterlegten, sogenannten Manifest-Datei definiert der Entwickler dazu, welche Dateien zu einem ResourceStore gehören. Außerdem enthält die Manifest-Datei eine Versionsnummer. Der LocalServer überprüft regelmäßig, ob sich die Version geändert hat und lädt gegebenenfalls die enthaltenen Ressourcen neu herunter.

Ein ManagedResourceStore eignet sich für den sauberen Entwurf also besser und ist auch Basis des Adressverwalters:

```
localServer =
    google.gears.factory.create("beta.localserver", "1.0");
store =
    localServer.createManagedStore('mein_lokaler_server');
store.manifestUrl = 'manifest.json';
store.checkForUpdate();
```

Die Datei manifest.json definiert alle Dateien, deren Auslieferung der LocalServer übernehmen soll. Hier müssen sämtliche benötigten Dateien aufgelistet werden – von HTML-Dokumenten über Bilder bis zu CSS und JavaScripts. Beim Adressmanager handelt es sich um nur drei Dateien:

```
{
  "betaManifestVersion": 1,
  "version": "meine_version 0.1",
  "entries": [ { "url": "index.html"},
               { "url": "images/stripes.png"},
               { "url": "javascripts/application.js"}
            ]
}
```

Um auf die in der Datenbank gespeicherten Datensätze zuzugreifen – etwa, um sie aufzulisten – bedient man sich wieder der execute-Methode des Database-API:

```
var result = db.execute('SELECT * FROM addresses');
while (result.isValidRow()) {
    HTML_output += result.fieldByName('street');
    result.next();
}
result.close();
```

Die Methode isValidRow() gibt Bescheid, wenn das Ende der Ergebniszeilen erreicht ist. fieldByName('<feldname>') extrahiert die gewünschten Daten; next() wandert zum nächsten Abfrage-Ergebnis. Momentan ist es noch

notwendig, ein Resultset nach beendeter Arbeit explizit über close() zu schließen. Dieses umständliche Vorgehen will Google in Zukunft aber vereinfachen.

Wieder einkuppeln

Geht der Benutzer online, so liest das Skript alle Daten aus der Datenbank aus und führt einen Ajax-Request aus, der sie zur Speicherung in der serverseitigen Datenbank übermittelt:

```
new Ajax.Request('save_online.php',
{ method:'post',
  postBody:'data='+AM.data_to_string()
});
```

Der Ajax-Request übergibt die in der Gears-Datenbank gespeicherten Datensätze im JSON-Format an das PHP-Skript save_online.php. Dieses wiederum speichert die übergebenen Daten in der Datei data.js. Die Datei haben wir der Einfachheit halber gewählt; genauso gut könnte das PHP-Skript die übergebenen Daten in eine Datenbanktabelle übertragen. Nachdem der Offline-Datenbestand übertragen wurde, leert der Befehl

```
db.execute("DELETE FROM addresses");
```

die Gears-Datenbanktabelle.

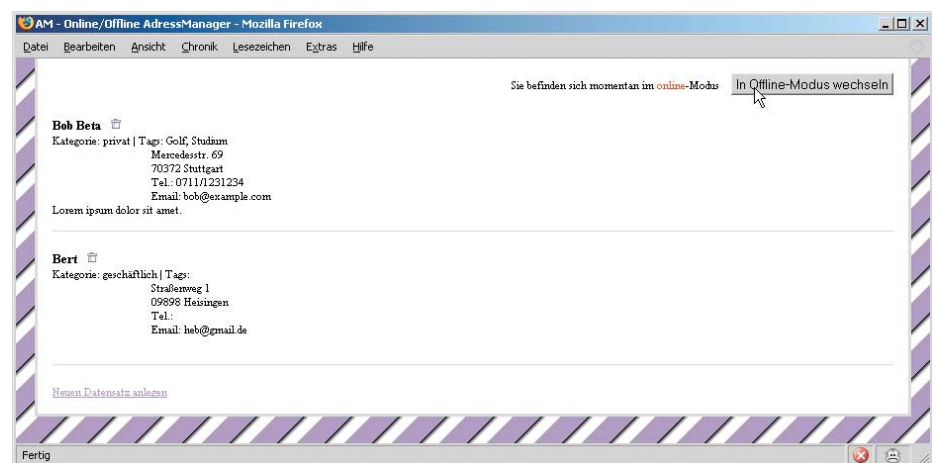
Als letzter Schritt des Übergangs von offline zu online kann es je nach Anwendung notwendig sein, dass das Skript den LocalServer deaktiviert. Hierfür gibt es mehrere Möglichkeiten.

```
localServer.removeManagedStore('offline_docs');
```

löscht den Resource Store mitsamt der gecachten Dateien. Er lässt sich aber auch über sein enabled-Attribut deaktivieren:

```
store.enabled = false;
```

Für unser Beispiel ist es aber nicht notwendig, den LocalServer zu deaktivieren: Im ManagedStore befinden sich nur index.html, stripes.png, apps.js und manifest.json, die „Datenbank“ data.js aber nicht.



Ob der in Gears realisierte Adressverwalter online oder offline arbeitet, sieht der Benutzer nur an der Statusmeldung – ansonsten verhält sich das Programm in beiden Fällen völlig identisch.

Wo bin ich?

Entscheidend für den reibungslosen Ablauf der – modalen – Beispielanwendung ist, dass sie immer genau weiß, ob sie lokal oder online läuft. Immer, wenn sie Daten abfragt, muss sie schließlich wissen, an wen sie diese Abfrage stellen soll: an die lokale Gears-Datenbank oder an den serverseitigen Datenspeicher. Den Modus über eine globale JavaScript-Variable festzuhalten, die bei jedem Klick auf den Online-/Offline-Knopf verändert wird, ist zwar naheliegend, aber problematisch. Dieses Flag fehlt nämlich, wenn der Benutzer die Anwendungsseite über ein Reload neu lädt. Die Beispiel-Applikation prüft den Modus daher anhand des Status der lokalen SQLite-Datenbank:

```
var result =
  db.execute("SELECT COUNT(*) FROM addresses");
if(result.field(0) > 0){
  mode = 'offline';
}else{
  mode = 'online';
}
```

Enthält die Datenbanktabelle Datensätze, so befindet sie sich im Offline-Modus, denn beim Wechsel in den Online-Modus wird sie geleert.

Stolperfalle Synchronisierung

Die Beispielanwendung arbeitet entweder online oder offline. Sie benutzt immer eine Referenzdatenbank, die die andere überschreibt. Im echten Leben ist das Konsistenthalten von Anwendungsdaten wesentlich komplizierter. So kann es durchaus vorkommen, dass Benutzer sowohl online (etwa bei der Arbeit) als auch offline (zum Beispiel vom Heim-PC aus) auf eine Anwendung zugreifen. Dann ist die hohe Kunst der Synchroni-

sierung gefragt. Wenn etwa ein Datensatz in beiden Versionen verändert wurde, muss die Anwendung entscheiden, welche Änderung maßgeblich ist. Richtig kompliziert kann es werden, wenn mehrere Benutzer auf denselben Datensatz zugreifen können.

Google hat auf seiner Homepage einige Überlegungen zum Datenabgleich zusammengefasst [5]. Letztlich hängt die Wahl der Synchronisierungsstrategie auch von der Anwendung und der Menge der abzugleichenden Daten ab. Bei größeren Synchronisierungen sollte der Einsatz des WorkerPool in Betracht gezogen werden. Damit ließe sich ein größerer Abgleichprozess in mehrere kleine Jobs aufteilen, die im Hintergrund abgearbeitet werden, ohne den normalen Programmfluss zu behindern. Unser Beispiel verzichtet auf den WorkerPool – es hätte den Rahmen dieses Artikels gesprengt, ihn auch noch vorzustellen.

Gears befindet sich im Beta-Stadium. Google rät derzeit deshalb, noch keine damit implementierten Anwendungen auszuliefern. Neben der Unterstützung weiterer Browser werden in den Gears-Foren viele Änderungen angemahnt, die den Entwicklern das Anpassen ihrer Anwendungen leichter machen würden. So ist es derzeit schwierig, Rückmeldung über ein erfolgtes Update des Resource Stores zu bekommen. In der aktuellen Beta-Version enthält die Funktion `checkForUpdate()` keinen Callback, der meldet, wenn ein Update erfolgreich beendet wurde.

Auch bei umfangreichen Datenbank-Operationen muss man derzeit vorsichtig sein. Zum einen sollten Buttons wie „In den Offline-Modus“ nach dem Klicken deaktiviert werden, bis die Operation beendet ist. Wird die Datenbank nämlich überlastet, antwortet sie erstmal mit der Fehlermeldung „Database locked“ und verweigert den Dienst. Im

schlimmsten Fall muss sogar die lokale Datenbanktabelle gelöscht werden – für den Produktiv-Einsatz beim User ist dies aber kein gangbarer Weg [6].

Fazit

Gears könnte den nächsten Evolutionsschub in der Internet-Entwicklung bringen: Web-Dienste, die auf dem PC auch ohne Internet-Verbindung laufen. Im Wettstreit mit den Konkurrenten AIR und Silverlight hat Gears eine gute Startposition. Die Umstellung einer einfachen Applikation mit HTML-Oberfläche dürfte sich mit Gears am simpelsten realisieren lassen; Anwender müssen sich kaum umgewöhnen. Anwendungen programmieren sich freilich auch mit Google Gears nicht von selbst. Mitunter dürften ziemliche Klimmzüge notwendig sein, um größere Applikationen offline-fähig zu machen. (jo)

Literatur

- [1] Homepage von Google Gears: <http://gears.google.com>
- [2] Googles Gears-fähiger FeedReader: <http://reader.google.com>
- [3] Gears-fähiger Aufgabenmanager Remember The Milk: www.rememberthemilk.com
- [4] Herbert Braun, Instant-Ajax, Bibliotheken und Frameworks für die Entwicklung von Ajax-Anwendungen, c't 5/06, S. 160
- [5] Choosing an Offline Application Architecture, Überlegungen zur Anwendungsarchitektur: <http://code.google.com/apis/gears/architecture.html>
- [6] Speicherorte der Datenbank: http://code.google.com/apis/gears/api_database.html#directories

