

Tobias Günther

c't

# Anwendungs-Esperanto

Mit OpenSocial MySpace, Orkut, StudiVZ und Co. aufpeppen



Soziale Netzwerke haben sich zu lebendigen Kontaktbörsen und Treffpunkten entwickelt. Der OpenSocial-Standard gibt den Anwendern und Entwicklern die Möglichkeit, den Funktionsumfang der Dienste durch eigene Anwendungen zu erweitern.

Unter den großen Anbietern von sozialen Netzwerk-Diensten hatte Facebook als Erster die Idee, seiner Plattform eine Programmierschnittstelle zu spendieren. Inzwischen veröffentlichen unzählige freie Entwickler und Drittfirmen Erweiterungen dafür – die allerdings nur auf Facebook laufen. Damit nicht jeder Betreiber eines sozialen Netzwerks sein eigenes Süppchen kocht, hat Google gemeinsam mit anderen Diensten OpenSocial entworfen.

Durch einen gemeinsamen Satz von Schnittstellendefinitionen sollen Entwickler leichter Programme schreiben können, die auf einer ganzen Reihe von Netzwerkdien-

ten laufen. Dazu gehören derzeit MySpace, Orkut, hi5, Friendster und LinkedIn; hierzu-lande haben Xing und StudiVZ angekündigt, den Standard umzusetzen. Insgesamt kann man über die beteiligten Netzwerke schon mehr als 500 Millionen Anwender erreichen.

## Orkut-Beispiel

Dieser Artikel stellt das OpenSocial-API anhand eines Beispiels vor, einem einfachen Chat-Client. Den kompletten Code zur Applikation finden Sie über den Link am Ende des Artikels. Wir haben die Beispielanwendung

auf Orkut entworfen und getestet, weil dort die Entwickler-Sandbox schnell und reibungslos funktioniert. Die MySpace-Sandbox arbeitete dagegen recht träge.

Um das Beispiel nachvollziehen zu können, benötigen Sie zwei Accounts bei Orkut – alleine zu chatten macht keinen Spaß. Wenn Sie keinen Kollegen zum Mitmachen bewegen können, sollten Sie sich einen zweiten Zugang einrichten, am besten mit einem anderen Browser. Anschließend müssen Sie zwischen den beiden Accounts noch eine Freundschaftsbeziehung herstellen. Die Sandbox zu nutzen, ist eine freiwillige, kostenlose Option, die Sie einmalig für beide

Testaccounts aktivieren müssen; [1] erklärt, wie das geht.

Die Anwendung benötigt außerdem PHP-fähigen Webspace inklusive einer MySQL-Datenbank, denn Orkut beherbergt nur die Logik der Bedienoberfläche; die Nachrichten lagern in der externen Datenbank. Es gibt zwar auch Programme, die ohne externen Datenspeicher auskommen, etwa kleine Spiele. Diese Client-Server-Architektur dürfte aber die interessantere Variante für Anwendungen auf sozialen Netzwerken sein.

Haben Sie die Accounts aktiviert, können Sie auf [sandbox.orkut.com](http://sandbox.orkut.com) zugreifen. Dort installieren Sie eine neue Anwendung, indem Sie auf den Link „Anwendungen hinzufügen“ klicken und auf der sich öffnenden Seite nach ganz unten scrollen. Wenn Sie dem mit „Hier kannst du deine OpenSocial-Anwendung an dieses Verzeichnis senden“ bezeichneten Verweis folgen, öffnet Orkut ein Formular, in dem Sie die URL eines Projekts angeben können. Eine fertig installierte Version der Beispielanwendung, die Sie probeweise einbinden können, findet sich unter [2].

Das Beispiel ist ein rudimentärer Chat-Client. Beim Besuch eines Profils, auf dem er installiert ist, zeigt er eine Liste aller Kontakte des Besuchers an. Klickt der Besucher auf einen Kontakt, kann er mit diesem loschaten. Komfortmerkmale wie eine Statusinformation, die anzeigt, ob das Gegenüber auch online ist, fehlen. Die Datenbank speichert Nachrichten nur so lange, bis sie einmal aufgerufen wurden; danach löscht sie sie.

## Google Gadgets als Rahmen

Den Rahmen für eine OpenSocial-Applikation bildet ein sogenanntes Google Gadget, was der eine oder andere Entwickler vielleicht schon aus der Programmierung etwa für iGoogle kennt. Ein OpenSocial-Gadget kann in sozialen Netzwerken an verschiede-

nen Stellen erscheinen, je nachdem, wo es eingebunden wurde und wo sich ein User gerade befindet. Das Gadget kann die aktuelle Ansicht abfragen und je nach Ansicht unterschiedliche Funktionen anbieten.

Grundsätzlich sind vier verschiedene View-Typen in OpenSocial definiert: In der Profile-View befindet sich das Gadget, wenn es auf der Profseite des Nutzers eingebunden wurde. Die Home-View ist dann aktiv, wenn sich ein Gadget auf der persönlichen Homepage des Nutzers befindet. Die Canvas-View ist eine Ansicht, in der das Gadget mehr oder weniger allein auf der Seite ist und entsprechend viel Platz zur Verfügung hat. Die Preview-Ansicht schließlich hat eingeschränkte Zugriffe und ist für Demo-Zwecke gedacht.

Das Beispiel-Gadget macht von Orkuts Standardvorgaben Gebrauch. Ruft man es über die Profseite eines Benutzers auf, bettet es sich in die mittlere Spalte des Layouts ein; ruft der Benutzer es über seine Startseite auf, erscheint es in der Canvas-Ansicht.

Ein Gadget ist ein XML-Dokument, das folgendermaßen aufgebaut ist:

```
<?xml version="1.0" encoding="UTF-8" ?>
<Module>
  <ModulePrefs title="Chat Tutorial">
    <Require feature="opensocial-0.8" />
  </ModulePrefs>
  <Content type="html">
    <![CDATA[
      Hier folgt der Applikations-Code.
    ]]>
  </Content>
</Module>
```

Der eigentliche Applikations-Code steht im <Content>-Knoten und besteht im Beispiel lediglich aus HTML und JavaScript. Die Einbettung von Flash- oder Silverlight-Objekten wäre aber ebenfalls möglich. Alle folgenden

JavaScript-Code-Ausschnitte liegen in diesem <Content>.

Da das Beispiel-Gadget mit JavaScript arbeitet, muss es als Erstes warten, bis die Seite komplett geladen ist, um die Applikation sauber initialisieren zu können. Aus JavaScript-Frameworks wie jQuery oder Prototype kennt man Funktionen, die der Anwendung Bescheid geben, sobald das DOM aufgebaut ist und sie mit der Initialisierung beginnen kann. Auch das Gadget-API verfügt über solch eine Hilfsfunktion:

```
gadgets.util.registerOnLoadHandler(function(){
  load_friends();
});
```

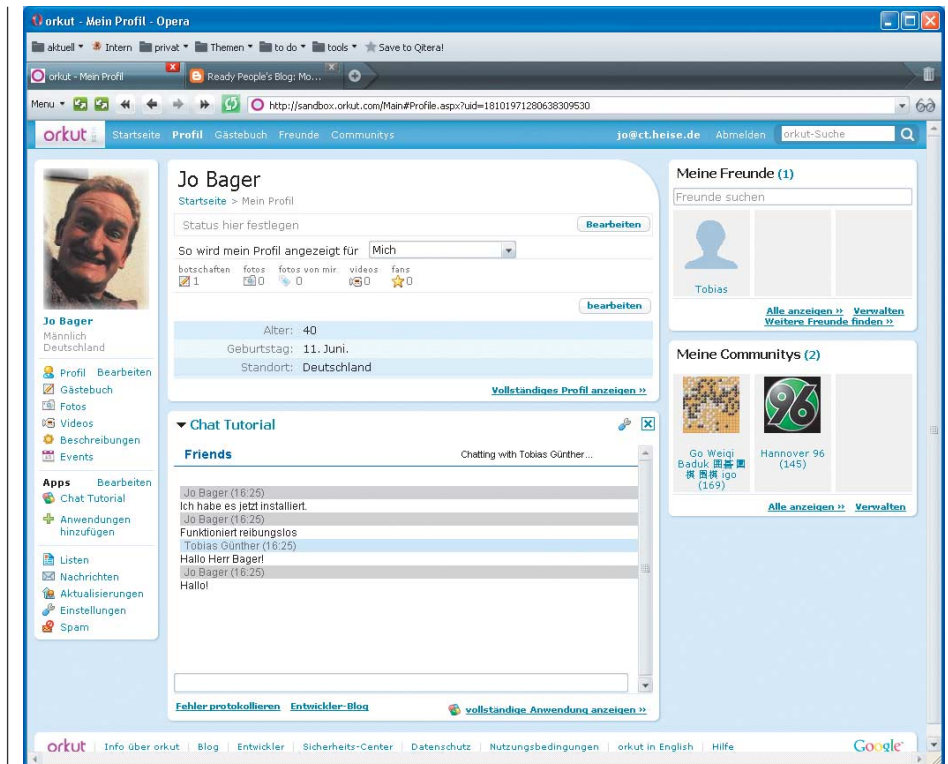
Ein wichtiger erster Bestandteil der Chat-Applikation ist eine Liste mit allen Freunden. Die kommt vom Social-Network-Container – im Beispiel Orkut. Die zugehörige JavaScript-Funktion `load_friends`, die bei der Initialisierung aufgerufen wird, lädt sie:

```
function load_friends(){
  var req = opensocial.newDataRequest();

  req.add(req.newFetchPersonRequest(opensocial.IdSpec.PersonId.VIEWER), 'viewer');
  var friends = opensocial.newIdSpec({ "userId" :
    "VIEWER", "groupId" : "FRIENDS" });
  req.add(req.newFetchPeopleRequest(friends, {}),
    'viewer_friends');
  req.send(on_load_friends);
}
```

Die erste Zeile in der Funktion erzeugt ein `DataRequest`-Objekt. An dieses werden in der Folge alle Anfragen angehängt, die beantwortet werden sollen; `req.add()` fügt zuerst eine Abfrage nach den Profildaten des `VIEWER` und dann nach dessen Freunden hinzu. Erst in der letzten Zeile schickt der Befehl `req.send()` sämtliche Anfragen in einem Rutsch ab – anstatt mehrere Ajax-Requests zu benötigen

Anzeige



Die Chat-Anwendung bettet sich in die mittlere Spalte des Orkut-Layouts ein, wenn Besucher es über eine Profilseite aufrufen.

Anzeige

und damit mehrfach Traffic und Wartezeit zu erzeugen. Dieses Verfahren der stapelweise und in einem Rutsch bearbeiteten Anfragen nennt sich „batch requesting“.

Gleichzeitig verdeutlicht der Code-Ausschnitt, dass sich in OpenSocial alles um Personendaten und -beziehungen dreht. Vor allem zwei Personentypen stehen im Mittelpunkt des Interesses, weshalb das API diese als Konstanten definiert: der VIEWER als (eingeloggter) Benutzer, der die Applikation gerade sieht und der OWNER, also der Besitzer der Profilseite, auf der sich das Gadget befindet. Es kommt durchaus vor, dass Owner und Viewer identisch sind – dann nämlich, wenn der Besitzer gerade die Anwendung auf seiner eigenen Profilseite nutzt. Außer Viewer und Owner kann per User-ID auch ein anderer Benutzer abgefragt werden. Die Beispielanwendung kann jeder nutzen, der ein Profil aufruft, auf dem sie installiert ist.

Beim Abschicken des Requests hat die Methode req.send() einen Parameter mitbekommen, nämlich den Verweis auf die Funktion on\_load\_friends. Diese Funktion wird als Callback verwendet und automatisch aufgerufen, wenn der Request beantwortet wurde und Daten zurückgibt. Sie kann also auf die Ergebnisse der Abfrage, die Personendaten, zugreifen:

```
function on_load_friends(data) {
  var viewer = data.get('viewer').getData();
  var friends = data.get('viewer_friends').getData();
  ...
}
```

Der Zugriff auf die Daten über die Methode get() ist aufgrund des batch requesting notwendig: Beim Zusammenstellen der Abfrage hat req.add() jeweils als letzten Parameter noch die Strings 'viewer' und 'viewer\_friends' mitbekommen. Dies war nötig, da ja mehrere Datenpäckchen auf einmal abgefragt werden sollten – die es am Ende natürlich wieder sauber zu trennen gilt.

Der nächste Schritt gibt die Freunde in einer Liste aus:

```
friends.each(function(friend) {
  friends_html +=
    '<div class="friend_name"> '+
    friend.getDisplayName()+ '</div>';
});
```

Wer schon einmal mit Ruby oder dem JavaScript-Framework Prototype gearbeitet hat, wird die each-Notation kennen, die das übernimmt.

Die Methode getDisplayName() liefert plattformunabhängig einen Anzeigenamen für einen User. Die meisten Datensatz-Felder sind plattformspezifisch. Auf MySpace könnte es zum Beispiel ein Feld namens „Hobbies“ an einem Person-Objekt geben, welches auf StudiVZ nicht existiert. Daher muss man ein wenig Vorsicht walten lassen und das Vorhandensein abprüfen, bevor man auf ein bestimmtes Feld eines Objekts zugreift.

Mit der Zeile

```
opensocial.getEnvironment().supportsField(opensocial.Environment.ObjectType.PERSON, opensocial.Person.Field.ID);
```

lässt sich beispielsweise prüfen, ob innerhalb des aktuellen Social Networks für Objekte

des Typs Person ein Feld namens ID überhaupt verfügbar ist. Wenn das Programm dann weiß, dass das Feld verfügbar ist, kann es über `getField()` darauf zugreifen:

```
friend.getField(opensocial.Person.Field.ID);
```

Die selbst definierte Funktion `send_message()` dient dazu, eine eingegebene Nachricht über einen Ajax-Request an einen Server zu schicken. `send_message()` verwendet das Gadgets-API intensiv. Dies ist nötig, weil es aus Sicherheitsgründen grundsätzlich gar nicht ohne Weiteres möglich ist, per Ajax einen Request an einen anderen als den Ursprungsserver zu senden. Hier springt das Gadgets-API mit der Funktion `gadgets.io.makeRequest` ein:



```
function send_message(message, user_to, user_from){
    var post_data = { action: 'save_message', message:
message, to_user: user_to, from_user: user_from };
    post_data = gadgets.io.encodeValues(post_data);
    var params = {};
    params[gadgets.io.RequestParameters.CONTENT_TYPE]
= gadgets.io.ContentType.TEXT
    params[gadgets.io.RequestParameters.METHOD] =
gadgets.io.MethodType.POST;
    params[gadgets.io.RequestParameters.POST_DATA] =
post_data;
```

```
gadgets.io.makeRequest('http://www.example.com/messa
ge.php', function(){}, params);
}
```

Die Funktionen, die für das Holen und Darstellen der Nachrichten zuständig sind, machen von keinen weiteren OpenSocial-Besonderheiten Gebrauch. Auch der Server-Part enthält nur wenig spannende Datenbankfunktionen und findet hier keine weitere Erwähnung. Sein vollständiger Code findet sich im Listing unter dem Link am Artikelende.

## Sandkastenspiele

Die Entwicklung einer OpenSocial-Applikation beginnt immer in der abgeschirmten Entwickler-Umgebung, der Sandbox, um den Betrieb des Social Networks nicht zu stören. In solchen Sandkasten-Umgebungen kann man ungestört an der eigenen Applikation feilen. Sie ist in dieser Zeit nur für den Entwickler selbst oder andere Entwickler zugänglich – nicht aber öffentlich für normale User.

Wenn die Anwendung dann einen ausreichenden Reifegrad erreicht hat und bereit ist, die Welt zu bereichern (also anderen Benutzern öffentlich zugänglich gemacht zu werden), kann man sie beim jeweiligen Social Network  die Applikation  zur Prüfung einreichen.

Der Einreichungsprozess verläuft in jedem Social Network zwar etwas anders, dient aber immer dem selben Zweck: Ihre eingereichte Applikation auf Herz und Nieren zu prüfen. Wenn der Anbieter sie dann für gut befindet, nimmt er sie in das Applikationsverzeichnis des Networks auf, von wo sie für alle Nutzer der Plattform installierbar ist.

## Portabilität

Das OpenSocial-API bietet noch weitere interessante Möglichkeiten: Über das Activities-API beispielsweise lassen sich Aktivitäts-Meldungen eines Benutzers hinzufügen oder auslesen. Eine Applikation könnte also von „Martin hat ein neues Profil-Bild“ bis „Martin hat Linda als neue Freundin“ die unterschiedlichsten Aktivitäts-Meldungen sowohl auslesen als auch schreiben.

Ein weiterer Bereich des API, das OpenSocial Persistence API, definiert die dauerhafte Speicherung von Daten. Jedes Social Network kann selbst entscheiden, in welchem Maße es seinen Applikationen erlaubt, Daten für einen Viewer persistent zu speichern. Sogenannte Lifecycle Events helfen dem Entwickler, an entscheidenden Punkten im Lebenszyklus einer Applikation zu reagieren. So besteht hierüber zum Beispiel die Möglichkeit, bei der Deinstallation eines Gadgets benachrichtigt zu werden, um nicht mehr benötigte Datenbankeinträge zu löschen.

In der Entwicklungs-Praxis ist allerdings trotz des vollmundigen Versprechens „Many sites, one API“ noch ein wenig Vorsicht geboten: Die verschiedenen beteiligten Networks implementieren oftmals nicht die komplette Spezifikation oder bleiben teilweise recht lange auf veralteten API-Versionen stehen. Auch gibt es immer wieder kleinere Unterschiede in der Implementierung.

hi5 zum Beispiel verlangt eine etwas erweiterte Form des Gadget-Gerüsts, um fehlerfrei zu laufen. Im OpenSocial-Wiki informiert die Entwicklergemeinde detailliert über die Unterschiede der einzelnen Implementierungen und gibt auch Hinweise darauf, was zu beachten ist, wenn man plattformübergreifende Anwendungen entwickeln will [4].

Trotz aller kleinen Unterschiede der einzelnen Implementierungen ist es wesentlich einfacher, eine OpenSocial-Anwendung für verschiedene Plattformen leicht abzuändern, als für jeden Dienst quasi von vorne anzufangen. Mehr als 20 Plattformen sind von dem Konzept überzeugt, weshalb sie das Zeug dazu hat, die stärkste Plattform für soziale Anwendungen zu werden. (jo)

## Literatur

- [1] Tutorial zur Nutzung der Orkut-Developer-Sandbox: <http://code.google.com/intl/de/apis/orkut/articles/tutorial/tutorial.html>
- [2] Adresse der Beispielanwendung: [http://pure-media-online.de/projekte/opensocial/chat\\_tutorial/chat\\_tutorial.xml](http://pure-media-online.de/projekte/opensocial/chat_tutorial/chat_tutorial.xml)
- [3] Google Gadgets: <http://code.google.com/intl/de/apis/gadgets>
- [4] Unterschiede der OpenSocial-Implementierungen: [http://wiki.opensocial.org/index.php#Container\\_Information](http://wiki.opensocial.org/index.php#Container_Information)
- [5] OpenSocial bei Google Code: <http://code.google.com/intl/de/apis/opensocial>

[ctmagazin.de/0906208](http://ctmagazin.de/0906208)



Anzeige