FernUniversität in Hagen

Fachbereich Informatik

Lehrgebiet Informationssysteme und Datenbanken

Implementierung der OpenSocial-API in der Communityumgebung für das Fernstudium

Bachelorarbeit

vorgelegt von

Harry Hübner

ha.huebner@nexgo.de

Erklärung

Ich versichere, diese Arbeit selbst verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt und mich auch sonst keiner unerlaubten Hilfsmittel bedient zu haben, sowie Zitate kenntlich gemacht zu haben.

Denzlingen, 11.06.2009

Inhaltsverzeichnis

1 Einleitung

1.1 Ausgangssituation

Eine Communityumgebung für das Fernstudium soll die Kommunikation unter den Studierenden und Betreuern fördern und die Möglichkeit bieten, einfach Ansprechpartner für verschiedene Belange bezüglich des Fernstudiums zu finden.

In den Masterarbeiten [Fuhrmann2008] und [Genise2009] wurde eine Communityumgebung für das Fernstudium erstellt, die bereits wichtige Grundfunktionalitäten einer sozialen Webseite bietet:

- Der Benutzer hat die Möglichkeit, sein Profil anzulegen, in dem persönliche Informationen sowie Informationen zum Studiengang angegeben werden können.
- Es können Gruppen erstellt werden. Gruppen werden anhand eines Typs in freie Gruppen und Kurse unterschieden. Zur Unterstützung der Kommunikation innerhalb einer Gruppe besitzt eine Gruppe ein Forum, einen Chatraum und einen eigenen Kalender.
- Es kann nach Benutzern und Gruppen gesucht werden.
- Innerhalb der Community können Kontakte über eine Buddy-Liste verwaltet werden.
- Es können individuell Nachrichten an andere Community-Teilnehmer gesendet werden.

Die aktuelle Entwicklung von sozialen Netzwerken zeigt, dass immer mehr Netzwerke Techniken anbieten, ihr Angebot dynamisch erweiterbar zu machen. Dies geschieht zum einen über die Integration von Anwendungen, die spezielle Dienste implementieren. Diese können mittels eines Plug & Play-Mechanismus in die Webseite integriert werden. Ein weiterer Ansatz ist die Öffnung des Zugriffs auf das System, um mit anderen sozialen Netzwerken zu kommunizieren, oder aber wie die neueste Entwicklung zeigt, eine Kommunikation über mobile Geräte oder Desktop-Applikationen zu ermöglichen.

Diese Interoperabilität ist eine wichtige Anforderung an Community-Systeme, die [Koch2003] bereits in seiner Arbeit aufgezeigt hat. Da Menschen seit jeher Mitglied in mehreren Communities sind, ist es problematisch, wenn die verschiedenen Plattformen voneinander isoliert sind:

- Benutzer müssen sich explizit in verschiedenen Community-Systemen registrieren und ihre Profilinformationen jedesmal aufs Neue eingeben.
- Der Informationsaustausch untereinander ist nicht möglich. Z.B. Ist es sinnvoll, Informationen automatisch an mehrere Communities zu verteilen oder Informationen automatisch von verschiedenen Communities abzufragen.

Wie weiter oben angedeutet, sind mittlerweile mehrere Lösungsansätze für dieses Problem vorhanden, die im folgenden kurz aufgezeigt werden:

Cobricks-2 ist ein an der TU München entwickelter, modularer und erweiterbarer Baukasten zur Erstellung von Community-Systemen [Cobricks]. Neben einem Web-Interface bietet dieses System auch ein WebService-Interface zum Zugriff auf die Daten und Services des mittels diesem Baukasten erstellten Community-Systems an. Mit dieser Technologie ist es möglich, Community-Systeme zu erstellen, die von

verschiedenen Clients oder Agenten aus ansprechbar sind. Es ist auch möglich, ein solches System mit anderen Applikationen oder Community-Systemen zu integrieren.

Facebook stellt seit Frühjahr 2007 eine Entwicklungs-Schnittstelle zur Verfügung, um eigene Anwendungen in das Facebook-Netz zu integrieren [FbDev]. Das Unternehmen setzt hierbei jedoch auf proprietäre Techniken, wie FBML (Facebook Markup Language) und FQL (Facebook Query Language), um auf Daten des Systems zuzugreifen. Mittlerweile sind auch Anwendungen vorhanden, um über mobile Geräte auf das Netzwerk zuzugreifen.

Der Einsatz von proprietären Techniken hat jedoch den Nachteil, dass Anwendungen nur genau für dieses Netzwerk programmiert werden können. Damit ist die Entwicklergemeinde auf eine Weise eingeschränkt, auch wenn das Netzwerk von den Benutzerzahlen her relativ gross ist.

Den Nachteil der proprietären Schnittstelle für genau ein Netzwerk hat Google mit Einführung der OpenSocial-API umgangen, die auf ähnlichen Konzepten wie die von Facebook basiert, jedoch nicht an eine bestimmte Community gebunden ist [OpenSocial]. Diese API wurde im November 2007 von Google an der Spitze eines Konsortiums kleinerer und grösserer sozialer Netze ins Leben gerufen. Mittlerweile sind ca. 75 Firmen Teil der OpenSocial-Partnerliste.

Mit dieser Schnittstelle ist es möglich, Anwendungen zu erstellen, die über einen einheitlichen, standardisierten Weg auf die Daten eines sozialen Netzwerkes zuzugreifen. Durch Verwendung dieser Standard-Schnittstelle kann eine solche Anwendung einmal entwickelt werden und für verschiedene Systeme bereitgestellt werden ("Write once, run anywhere"). Durch die Verwendung von HTML, JavaScript und CSS ist die Hürde zur Entwicklung eigener Anwendungen relativ gering, da diese Techniken sehr weit verbreitet sind und gängige Entwicklungswerkzeuge genutzt werden können.

Damit ein soziales Netzwerk solche Applikationen ausführen kann, muss es einen entsprechenden Container zur Verfügung stellen, der dieses API unterstützt. Google hat hier bereits eine Referenzimplementierung eines solchen Containers mit Namen Shindig [ApShindig] erstellt und diesen der Apache Software Foundation (ASF) übergeben.

Im Zuge der OpenSocial-Schnittstelle wurde auch die OpenSocial RESTful-API erstellt, die es ermöglicht, auf Community-Daten mittels einer URL zuzugreifen. Diese Technik kann genutzt werden, um auf maschinellem Weg mit dem Netzwerk zu kommunizieren. Somit können Dienste ausserhalb der Seite angeboten werden, die mit dem Netzwerk kommunizieren können.

Eine solche Öffnung eines Systems bietet auch mehr potentielle Angriffsstellen, die vor unerlaubtem Zugriff geschützt werden müssen. Z.B. könnten über die REST-Schnittstelle Daten aus dem System ausgelesen werden, wenn einem Angreifer entsprechende Benutzer-IDs bekannt sind. Hierzu bietet OpenSocial Mechanismen wie die Verwendung eines Security-Tokens zur Authentifizierung und OAuth zur Autorisierung bei plattformübergreifender Kommunikation [OAuth]. Details hierzu werden in nachfolgenden Kapiteln erläutert.

Einige grosse Netzwerke haben die Schnittstelle bereits in ihrer Plattform integriert. Unter ihnen [Xing], [hi5], [Orkut] und [MySpace]. Sun unterstützt diese Schnittstelle mit dem OpenSource-Projekt [SocialSite]. Hiermit wird ein rudimentäres soziales Netzwerk mit OpenSocial-Unterstützung zur Verfügung gestellt wird, das es vor allem kleineren Web-Anwendungen ermöglichen soll, soziale Dienste anzubieten.

1.2 Aufgabenstellung

Die vorangegangene Übersicht zeigt, dass die OpenSocial-API ein enormes Potential bietet, die Forderungen nach Interoperabilität eines Community-Systems zu erfüllen. Die notwendigen Erweiterungen können auch leicht bei bereits bestehenden Systemen vorgenommen werden, was für eine weite Verbreitung dieser Schnittstelle sehr förderlich ist.

Im Rahmen dieser Bachelorarbeit soll die Implementierung der OpenSocial-API in die bereits aus Vorgängerarbeiten vorhandene Communityumgebung für Fernstudenten prototypisch umgesetzt werden. Damit ist es möglich, OpenSocial-Applikationen innerhalb des Systems zu betreiben. So sollen die Benutzer die Möglichkeit haben, eine neue OpenSocial-Applikation in ihrem Profil zu installieren und diese auszuführen. Diese Applikationen stehen auch anderen Nutzern zur Ausführung zur Verfügung, wenn sie das Profil des fremden Nutzers betrachten.

Die OpenSocial-Applikationen erhalten über die OpenSocial-Standardschnittstelle Zugriff auf die Daten der Fernuni-Community. Die Datenintegration verlangt, dass das Community-System Daten-Services für folgende Bereiche anbietet:

- Personendaten sowie Freundesbeziehungen zu weiteren Benutzern der Plattform
- Aktivitäten
- Applikationsdaten

Mittels dem Personendaten-Service kann eine OpenSocial-Applikation die Profildaten der Fernstudenten abfragen und innerhalb der Anwendung nutzen.

Eine Applikationen kann während ihrer Ausführung Aktivitäten protokollieren. Diese werden in der Fernuni-Community abgespeichert und auf Profilebene als Übersicht angezeigt. Über diese Übersicht erhalten andere Benutzer Einsicht über Aktionen eines Nutzers.

Applikationen bieten einen höheren Komfort, wenn sie Daten innerhalb des sozialen Netzwerkes speichern können, in dem sie ausgeführt werden. Mit der API-Implementierung stellt die Fernuni-Community einen Persistenzbereich für Applikationen zur Verfügung, in dem benutzerbezogen zu einer Applikation Daten gespeichert werden können.

Die Öffnung des Community-Systems in dieser Form bietet vielfältige Möglichkeiten zur Interaktion mit dem System. Beispielhaft werden einige Ansätze in dieser Arbeit aufgezeigt:

- Einfache OpenSocial-Applikationen, die eine bestimmte Funktionalität bieten und somit den Funktionsumfang des Community-Systems erweitern.
- Durch die Möglichkeit, aus OpenSocial-Applikationen heraus entfernte Systeme anzusprechen, können bestehende Dienste eingebunden werden. Es kann somit auch eine Kommunikation mit anderen Netzwerken hergestellt werden.
- Die OpenSocial RESTful-API bietet einen Zugriff auf das System von externen Stellen. Es besteht damit die Möglichkeit, Dienste zu erstellen, die eine weitere Sicht auf die Daten der Community bieten. Diese Dienste können in Form von weiteren Web-Applikationen, Desktop-Applikationen oder mobilen Services erstellt werden.

1.3 Aufbau der Arbeit

In Kapitel 2 werden die Konzepte der OpenSocial-API vorgestellt, auf deren Basis die Communityumgebung erweitert wird. Es wird aufgezeigt, welche Funktionalitäten die API bietet, und wie mit diesen Mechanismen eine OpenSocial-Applikation erstellt werden kann.

Um innerhalb eines sozialen Netzwerkes OpenSocial-Applikationen auszuführen, muss dieses Netzwerk als OpenSocial-Container erweitert werden. Die hierfür notwendigen Aspekte werden allgemeingültig in Kapitel 3 aufgezeigt.

Aufbauend auf der allgemeingültigen Beschreibung zur Erstellung eines OpenSocial-Containers beschäftigt sich Kapitel 4 mit dem Entwurf zur Integration der OpenSocial-API in die Fernuni-Community.

Kapitel 5 enthält eine Zusammenfassung der einzelnen Kapitel und zeigt die erweiterten Einsatzmöglichkeiten auf. Abschliessend wird der Ansatz diskutiert.

In Kapitel 6 wird ein Ausblick auf eine tiefergehende Integration der OpenSocial-API gegeben. Es wird aufgezeigt, welche Bereiche mit dieser Arbeit nicht abgedeckt wurden. Es wird auch darauf eingegangen, in welcher Hinsicht die OpenSocial-API Entwicklungspotenzial hat.

2 OpenSocial-API

2.1 Überblick

Ziel von OpenSocial ist es, eine Standard-API für das Umfeld soziale Netzwerke zur Verfügung zu stellen, die innerhalb von verschiedenen Kontexten genutzt werden kann. Mit dieser Technologie können soziale Applikationen mittels Standard-JavaScript und HTML erstellt werden und in Netzwerken betrieben werden, die die OpenSocial-API implementiert haben. Eine solches Netzwerk, auch als OpenSocial-Container bezeichnet, stellt somit eine Laufzeitumgebung für OpenSocial-Applikationen dar.

Die OpenSocial-API stellt Funktionen zur Verfügung, um auf Informationen von Personen (Name, Adresse, Alter, Beruf, ...), Kontaktdaten (Freunde, Beziehungen, ...) und Aktivitätsdaten ("Person X ist gerade...") innerhalb des Kontextes eines Containers zuzugreifen. Die Container-Implementierung ist dafür verantwortlich, die entsprechenden Daten des Containers zu liefern. D.h., wenn eine OpenSocial-Applikation in unterschiedlichen Netzwerken installiert ist und diese eine Suchanfrage nach den Freunden eines bestimmten Nutzers stellt, erhält sie als Ergebnis die Beziehungsdaten der Person innerhalb des jeweiligen Netzwerkes zurück. Dieses Verhalten ist möglich durch Nutzen des Standard-API's, das in jedem Container spezifisch implementiert werden muss.

OpenSocial ist ein Standard, der sich noch in der Entwicklung befindet. Von daher existieren bereits verschiedene Versionen und nicht alle soziale Netzwerke unterstützen die neueste Version. OpenSocial wurde im Dezember 2007 veröffentlicht und seit dem stark weiterentwickelt. Die folgende Übersicht zeigt die bisher verfügbaren Versionen:

- Version 0.5, veröffentlicht im Dezember 2007.
- Version 0.6, veröffentlicht im Dezember 2007.
- Version 0.7, veröffentlicht im Februar 2008. Diese wurde von vielen als erste richtig funktionierende Version angesehen.
- Version 0.8, veröffentlicht im Mai 2008.
- Version 0.9, Teile bereits freigegeben.

Diese Arbeit basiert auf der Version 0.8.

2.1.1 Soziale Applikationen

OpenSocial stellt Funktionalitäten zur Verfügung, um soziale Applikationen zu erstellen. Dies sind Applikationen, die innerhalb eines sozialen Netzwerkes erstellt und betrieben werden. Oft werden diese Applikationen nicht durch den Hersteller des jeweiligen Netzwerkes erstellt, sondern durch Drittpersonen. Eine Applikation wird als sozial bezeichnet, wenn sie Interaktionen zwischen Freunden des Benutzers oder anderen Benutzern unterstützt [Grewe2009]. Die folgende Abbildung zeigt eine in Orkut installierte Applikation zur Bewertung von Büchern.

Abbildung 1: In Orkut installierte Applikation weRead

Mit dieser Applikation ist es möglich, sich ein virtuelles Bücherregal mit einer eigenen Bücherauswahl zu erstellen und diese Informationen mit mit anderen Benutzern zu teilen. Die Applikation ist als OpenSocial-Applikation sowie als Facebook-Applikation verfügbar. Mit Installation in dem eigenen Profil erhält die Applikation Zugriff auf die eigenen Daten innerhalb des Netzwerkes. Eine weitere Anmeldung an diesem Dienst ist nicht notwendig, da diese automatisch mit der Installation durchgeführt wird.

2.1.2 Kern-Konzepte der OpenSocial-API

Die OpenSocial-API stellt eine standardisierte Sicht auf den sozialen Graphen eines sozialen Netzwerkes zur Verfügung. Diese Sicht teilt sich in folgende Bereiche auf:

- Die Daten einer Person und die Beziehungen dieser zu anderen Personen.
- Die Aktivitäten, die eine Person im Kontext der Webseite durchgeführt hat.
- Datenspeicher für OpenSocial-Applikationen. Hierzu wird eine Schnittstelle zur Verfügung gestellt, über die OpenSocial-Applikationen Daten innerhalb des Containers persistieren können.
- API zur Darstellung der OpenSocial-Applikationen innerhalb des Containers.
- API mit dessen Hilfe Nutzer von WebServices, mobilen Geräten und Desktop-Anwendungen auf die Daten des Containers zugreifen können.

Personendaten

Personendaten stellen einen fundamentalen Teil von sozialen Web-Seiten dar. Diese werden in OpenSocial

über die Person-Klasse zur Verfügung gestellt. Üblicherweise sind diese Informationen innerhalb des Containers in den Nutzer-Profilen abgelegt.

Die folgenden Liste bietet einen Auszug der Attribute der Person-Klasse:

- ID (alphanumerisch. Diese ID muss die Person eindeutig innerhalb des Containers identifizieren)
- Name (`opensocial.Name` Objekt)
- Displayname (Name, der innerhalb des Netzwerkes für diese Person angezeigt wird)
- Addresses (Adressen der Person, Array von `opensocial.Address` Objekten)
- Age (Alter)
- Gender (Geschlecht. `opensocial.Enum` Objekt)
- Date of Birth (Geburtsdatum)
- Emails (Email-Adressen der Person. Array von `opensocial.Email` Objekten)
- Interests (Interessen oder Hobbies der Person. Array von Strings)
- Books (Lieblingsbücher der Person. Array von Strings)
- ...

Eine Person wird üblicherweise über ihre ID referenziert. Auf zwei Personen-Objekte kann aber direkt zugegriffen werden, ohne die ID zu kennen. Dies sind der „Viewer“ und der „Owner“. Der Betrachter (Viewer) ist der eingeloggte Benutzer auf dem System und der Eigentümer (Owner) ist derjenige, dem das Profil gehört, auf dem gerade eine Aktion ausgeführt wird. Eigentümer und Betrachter können die selbe Person sein, wenn der eingeloggte Benutzer gerade sein Profil geöffnet hat.

Relationen

Neben den reinen Daten einer Person sind in einem sozialen Netzwerk die Beziehungen, die eine Person zu anderen Personen hat, ein sehr wichtiger Aspekt.

Innerhalb OpenSocial kann auf zwei Mengen von Personen zugegriffen werden. Die Freunde des Betrachters (Viewer Friends) und die Freunde des Besitzers (Owner Friends).

Eine Abfrage auf „Viewer Friends“ liefert eine Liste von Freunden des aktuell angemeldeten Benutzers. Die Abfrage auf „Owner Friends“ entsprechend eine Liste von Freunden des Seiten-Besitzers. Hier gilt auch wieder beim Navigieren auf der eigenen Seite, dass „Owner Friends“ und „Viewer Friends“ ein identisches Ergebnis liefern.

Die Abfrage nach Freunden kann auch mit einem Entfernungsgrad-Parameter erweitert werden, um z.B. die Freunde von Freunden zu selektieren.

Aktivitäten

OpenSocial stellt Möglichkeiten zur Verfügung, um Aktivitäten zu erstellen. Aktivitäten protokollieren Aktionen, die ein Benutzer im Kontext eines Containers durchgeführt hat. Dies kann eine Interaktion mit

dem Container selbst, wie z.B. der Beitritt in eine Gruppe oder aber auch eine Aktion mit einer OpenSocial-Applikation sein. Die Aktivitätenliste eines Benutzers zeigt somit, wie dieser mit einer OpenSocial-Applikation oder dem Container arbeitet. Für andere Benutzer kann dies eine Hilfe sein, um neue Funktionen des Netzwerkes oder den Umgang mit verschiedenen Applikationen kennenzulernen.

Die folgende Liste bietet einen Auszug der Attribute der Activity-Klasse:

- ID (eindeutige ID einer Aktivität)
- Title (Titel, der den Haupttext der Aktivität repräsentiert)
- Body (optionale, erweiterte Darstellung der Aktivität)
- AppId (Applikation, zu der die Aktivität zugeordnet ist)
- URL (URL, die dieser Aktivität zugeordnet ist)
- MediaItems (Fotos, Videos oder Bilder, die dieser Aktivität zugeordnet sind. Array von opensocial.MediaItem Objekten)
- ...

Persistenz

Applikationen können einen höheren Nutzungskomfort bieten, wenn sie Daten innerhalb des Containers abspeichern können. Diese Möglichkeit bietet OpenSocial. Es stehen Funktionen zur Verfügung um benutzerbezogen Daten zu speichern und wieder zu lesen. Die Daten werden in der Form „Key-Value-Pairs" persistiert. Der Container muss mit der Implementierung der OpenSocial-API hierfür eine Speichermöglichkeit anbieten.

Es ist hierbei zu beachten, dass die API keinerlei Reglementierungen vorgibt, wie viele Daten eine Applikation speichern darf. Dies kann im Extremfall zu unkontrollierten Datenmengen führen. Somit ist es sinnvoll, eine Begrenzung der Datenmenge durch den Container vorzugeben.

API zur Erstellung von OpenSocial-Applikationen

Mit der OpenSocial JavaScript-API werden Funktionen zur Verfügung gestellt, mittels denen OpenSocial-Applikationen mit JavaScript, HTML und CSS erstellt werden können. Die Applikation muss zur Ausführung innerhalb des jeweiligen Netzwerkes installiert werden. Der Zugriff auf die oben genannten Daten des sozialen Netzwerkes erfolgt rein mit JavaScript-Funktionen. Abbildung 2 zeigt ein solches Modell. In den Ablauf sind nur das soziale Netzwerk sowie die Applikation selbst involviert.

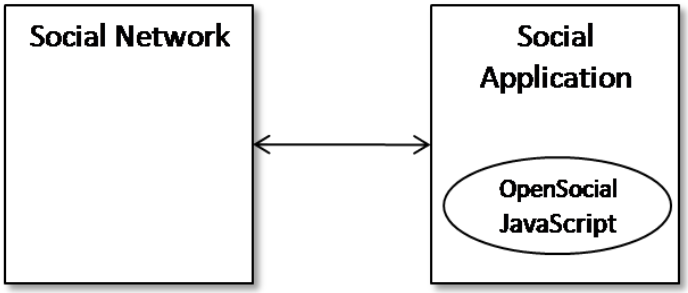

Abbildung 2: OpenSocial-Applikation mit JavaScript

Die OpenSocial-API bietet mit Ihrer Persistenz-Schnittstelle Funktionen, um Daten einer Applikation innerhalb des sozialen Netzwerkes zu speichern, in dem die Applikation installiert ist. Ist dies für die Applikationsentwickler nicht ausreichend, oder wird der Zugriff auf entfernte Ressourcen benötigt, so kann von der Applikation aus auf einen externen Server zugegriffen werden.

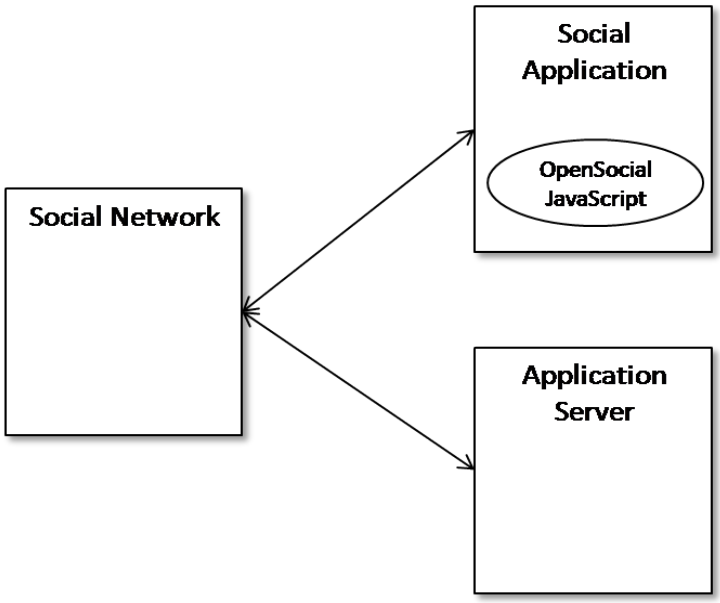

Abbildung 3: OpenSocial-Applikation mit Zugriff auf entfernte Ressourcen

Der Zugriff auf den Server muss über die JavaScript-Methode `gadgets.io.makeRequest` durchgeführt werden und erfolgt mit diesem Aufruf nicht direkt, sondern über das soziale Netzwerk.

API zur server-seitigen Entwicklung mit OpenSocial

Mittels der OpenSocial RESTful-API ist es möglich, Applikationen mit anderen Web-Technologien als JavaScript zu erstellen. Diese Applikationen werden auf einem anderen Server ausgeführt und können über die RESTful-API mit dem sozialen Netzwerk kommunizieren.

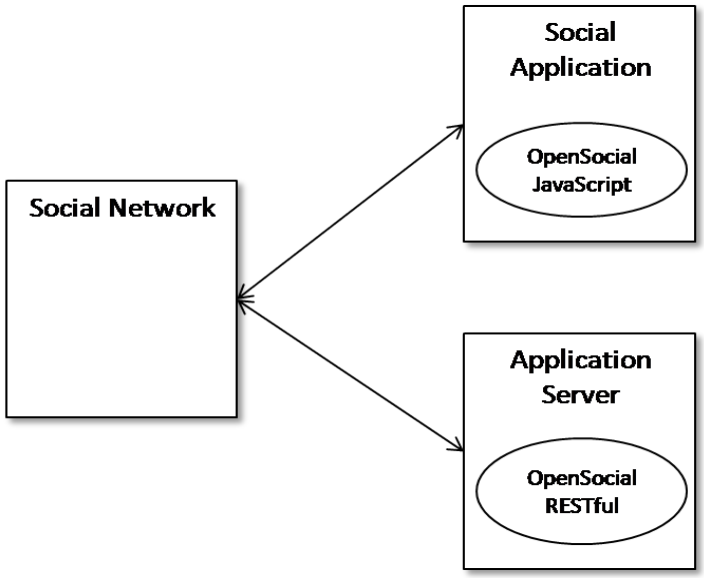

Abbildung 4: OpenSocial-Application mit server-seitigem Zugriff auf das Netzwerk

Mit diesem Modell kann innerhalb des Servers via RESTful-API auf die Daten des sozialen Netzwerkes zugegriffen werden. Zum Beispiel könnte die Server-Applikation auf eine Benutzeraktion hin, wie die Bewertung eines Buches, direkt einen Eintrag gemäss dem Muster „Benutzer x hat Buch y bewertet“ in die Aktivitätsliste des Benutzers vornehmen. Bei diesem Szenario existiert also eine OpenSocial-Applikation, die innerhalb des sozialen Netzwerk installiert ist. Der Benutzer interagiert direkt mit dieser Applikation und diese kommuniziert im Hintergrund mit einer eigenen Server-Applikation, welche mittels der RESTful-API wiederum auf das Netzwerk zugreift.

Mit der RESTful-API können aber auch Anwendungen erstellt werden, wo keine Applikation im jeweiligen Netzwerk installiert werden muss. Hierunter fallen Anwendungen für mobile Endgeräte oder Batch-Verarbeitungen.

2.2 OpenSocial JavaScript-API

Google hat mit der OpenSocial JavaScript-API keine neue Technologie erfunden, sondern vorhandene Techniken konsequent weiterentwickelt. So basiert OpenSocial auf dem Google Gadget-API und setzt damit komplett auf den Google Gadgets auf.

Gadgets werden in XML beschrieben und stellen kleine, eigenständige Webseiten dar, die eine bestimmte Funktionalität realisieren. Sie können problemlos in eine eigene Webseite integriert werden.

2.2.1 Anatomie eines Gadgets

Folgendes Listing stellt das Grundgerüst eines Gadgets dar.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
  <ModulePrefs title="Sample"
               author="..."
               author_email="..."
```

```
    [...] />
  <UserPref name=" "
            datatype="..."
            default_value="..."
            display_name="..." />
    [...]
  </UserPref>
  <Content type="...">
    <![CDATA[
      [...]
    ]]>
  </Content>
</Module>
```

Ein Gadget besteht aus folgenden drei Bereichen:

- Abschnitt ModulePrefs mit Modul(Gadget)-Einstellungen: In diesem Bereich werden sämtliche charakteristische Eigenschaften eines Gadgets definiert, wie z.B. Titel oder Beschreibung des Gadgets. Hierzu gehören auch Grösse des Gadgets, Vorschau-Thumbnail und ein Screenshot. Die Einstellungen sind nur über den Ersteller änderbar.
- Abschnitt UserPrefs mit Benutzer-Einstellungen: In diesem Bereich werden Einstellungen definiert, die direkt vom Benutzer änderbar sind. Hierdurch können durch den Autor Einstellungsmöglichkeiten vorgegeben werden (z.B. Anzahl der Einträge pro Seite), die durch den Benutzer angepasst werden können.
- Abschnitt Content: Dieser Abschnitt stellt die eigentliche Implementierung eines Gadgets dar. Der Code muss in dem Bereich CDATA [] angegeben werden.

2.2.2 Google Gadget-API

Die API unterscheidet zwei Typen von Gadgets. HTML-Gadgets und URL-Gadgets. Die Unterscheidung wird im Content-Bereich angegeben. Bei HTML-Gadgets ist die Implementierung vollständig im Gadget selbst enthalten. Bei URL-Gadgets ist die Implementierung üblicherweise in einer Web-Anwendung vorgenommen und das Gadget enthält einen Link auf die entsprechende URL.

Im Content-Bereich des HTML-Gadgets ist der eigentliche Inhalt der Anwendung vorhanden. Dieser kann aus HTML, CSS und JavaScript-Code bestehen, der entweder lokal innerhalb des Content-Bereiches definiert ist, oder über einen externen Link eingebunden wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
  <ModulePrefs title="Sample" />
  <Content type="html">
    <![CDATA[
      Hello World
    ]]>
  </Content>
</Module>
```

Bei URL-Gadgets verweist die URL auf eine Web-Anwendung jeglicher Art. Wichtig ist jedoch, dass diese Anwendung als Ausgabe HTML zurückgibt, welches in dem Darstellungsbereich der Applikation dargestellt werden kann.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
  <ModulePrefs title="Sample" />
  <Content type="url" href="http://www.example.com/site.jsp"/>
</Module>
```

Dank der Verwendung von XML als Beschreibungssprache für Gadgets kann ein Gadget mit einem einfachen Texteditor erstellt werden. Komfortabler geht es mit Googles eigenem Gadget Editor [GGE], der neben einfachem Syntax-Highlighting auch eine direkte Vorschau des Gadgets anbietet. Weitere Möglichkeiten, ein Gadget zu programmieren, werden in Kapitel 3.3 beschrieben.

Alle Objekte der Gadget-API starten mit dem gadgets-Objekt. Eine detaillierte Beschreibung der API befindet sich unter [GadgetSpec].

Die Gadget-API unterteilt sich in folgende Teilbereiche:

- Core Gadget-API
- Feature-spezifische Gadget-APIs

Core Gadget-API

Der Kern der API beinhaltet folgende Hauptobjekte:

- gadgets.io – Unterstützung des Aufrufs von entfernten Ressourcen.
- gadgets.json – Unterstützung des JSON-Formates
- gadgets.util – Hilfsfunktionen
- gadgets.prefs – Informationen zum Benutzer wie Ländereinstellung

Feature-spezifische Gadget-APIs

Um feature-spezifische APIs in einem Gadget zu nutzen, müssen diese mit dem `<Require>` - Element eingebunden werden. Folgende Features sind Teil der OpenSocial-Spezifikation und können in diesem Zusammenhang genutzt werden:

- gadgets.views – Behandlung von unterschiedlichen Views.
- gadgets.window – Operationen zur Anpassung des Gadget-Titels oder der Höhe.
- gadgets.skins – Einstellung von Farben.
- gadgets.flash – Ermöglicht Flash-Inhalte in Gadgets.
- gadgets.MiniMessage – Zur Erzeugung von Benutzer-Meldungen innerhalb des Gadgets.
- gadgets.TabSet – Unterstützung von Tab-Reitern
- gadgets.Tab – Eigenschaften eines Tab-Reiters
- gadgets.pubsub – Unterstützung zur Veröffentlichung von Meldungen
- gadgets.rpc – Unterstützung von RemoteProcedureCalls.

Unterschiedliche Views

Ein Container kann verschiedene Bereiche in der Benutzerschnittstelle definieren, wo Applikationen angezeigt werden. Je nach Bereich soll die Applikation unter Umständen in einer anderen Detaillierungsstufe oder Form angezeigt werden. Diese Form der Unterscheidung bietet OpenSocial mit dem Konzept der „Views“ an. Die Applikation kann hierzu den Container fragen, auf welcher View sie gerade angezeigt wird und abhängig vom View-Typ eine entsprechende Darstellung anbieten. Umgekehrt kann aber auch das Gadget als Reaktion auf eine Benutzeraktion, den Container anfragen, auf eine andere View zu navigieren. Dies geschieht mit der Methode `requestNavigateTo`. Zum Beispiel kann ein Gadget in einer Voransicht dargestellt werden und hier nur eine zusammenfassende Übersicht bieten. Beim Klick auf einen bestimmten Bereich wird das Gadget zur Bearbeitung in einer Detailansicht geöffnet.

In OpenSocial sind folgende Standard-Views definiert:

- Canvas – Dies ist quasi die Vollansicht eines Gadgets, wo der grösste Teil des User-Interfaces durch das Gadget selbst ausgefüllt wird.
- Home – Dient zur Darstellung des Gadgets auf der Home-Seite. Dies ist typischerweise die Ansicht, die nach dem Einloggen des Benutzers als erstes angezeigt wird. Die Darstellung zeigt oft nur einen Teilbereich gegenüber der Canvas-Darstellung, sie hat aber den Vorteil, dass man sofort nach dem Einloggen Zugriff auf die Applikation hat.
- Profile – Dient zur Ansicht mit anderen Applikationen. Z.B. als Auflistung aller Applikationen eines Benutzers. Die Darstellung ist wie bei der Home-Darstellung kleiner als die Canvas-Darstellung.
- Preview – Voransicht. Hier ist kein Zugriff auf die Daten des Besitzers oder Betrachters möglich. Der Hauptzweck dieser Ansicht ist die Demonstration von Funktionen für potentielle Nutzer.

Remote-Zugriffe

Der Funktionsumfang von Gadgets kann wesentlich erweitert werden, wenn es möglich ist, auf Daten eines RemoteServers zuzugreifen. Dieser Zugriff erfolgt mittels der Methode `gadgets.io.makeRequest` in der Gadgets-API. Vor Aufruf der Methode muss ein entsprechender Content-Typ festgelegt werden. Aktuell werden die Typen DOM (DOM-Objekt), FEED (JSON-Repräsentation eines RSS- oder ATOM-Feeds), JSON (JSON-Objekt) und Text (Textdateien).

```
...
var url = http://example.com/resource
var params = {};
params[gadgets.io.RequestParameters.CONTENT_TYPE] = gadgets.io.ContentType.JSON;
gadgets.io.makeRequest(url, callback, params);

function callback(data) {
  ...
}
```

In diesem Beispiel wird mittels der Operation `makeRequest` ein Aufruf auf einen anderen Server durchgeführt. Das Ergebnis des Aufrufes ist über eine callback-Methode verwendbar.

2.2.3 OpenSocial-Gadgets

Um dem eigentlichen Thema „OpenSocial” gerecht zu werden, muss das Gadget noch „sozialisiert" werden.

Dies geschieht über die Angabe der zu verwendenden OpenSocial-Version. Mit

```
<Require feature="opensocical-0.8"/>
```

wird die OpenSocial-API in der Version 0.8 eingebunden. Somit ist der volle Zugriff auf die OpenSocial-Funktionalität möglich.

Beispiel-Applikation

In diesem Abschnitt wird die OpenSocial JavaScript-API anhand eines einfachen Beispiels vorgestellt. Nachfolgend werden weitere Konzepte der API vorgestellt.

Die folgende Beispiel-Applikation listet die Freunde des eingeloggten Benutzers auf.

```
<?xml version="1.0" encoding="UTF-8"?>
<Module>
  <ModulePrefs title="FriendList"
    description="Listet die Freunde des angemeldeten Benutzers auf.">
    <Require feature="opensocial-0.8" />
  </ModulePrefs>
  <Content type="html">
  <![CDATA[
  <script type="text/javascript">

  function loadFriends() {
    var req = opensocial.newDataRequest();
    req.add(req.newFetchPersonRequest(opensocial.IdSpec.PersonId.VIEWER), 'viewer');

    var viewerFriends = opensocial.newIdSpec({ "userId" : "VIEWER", "groupId" :
       "FRIENDS" });
    req.add(req.newFetchPeopleRequest(viewerFriends), 'viewerFriends');

    req.send(onLoadFriends);
  }

  function onLoadFriends(data) {
    var viewer = data.get('viewer').getData();
    var viewerFriends = data.get('viewerFriends').getData();

    html = new Array();
    html.push('Friends of ' + viewer.getDisplayName());
    html.push(':<br><ul>');
    viewerFriends.each(function(person) {
      if (person.getId()) {
        html.push('<li>', person.getDisplayName(), '</li>');
      }
    });
    html.push('</ul>');
    document.getElementById('friends').innerHTML = html.join('');
  }

  function init() {
    loadFriends();
  }

  gadgets.util.registerOnLoadHandler(init);

  </script>

  <div id='main'>
    <div id='friends'></div>
  </div>
  ]]>
```

```
  </Content>
</Module>
```

Der Abaluf lässt sich grafisch wie folgt darstellen:

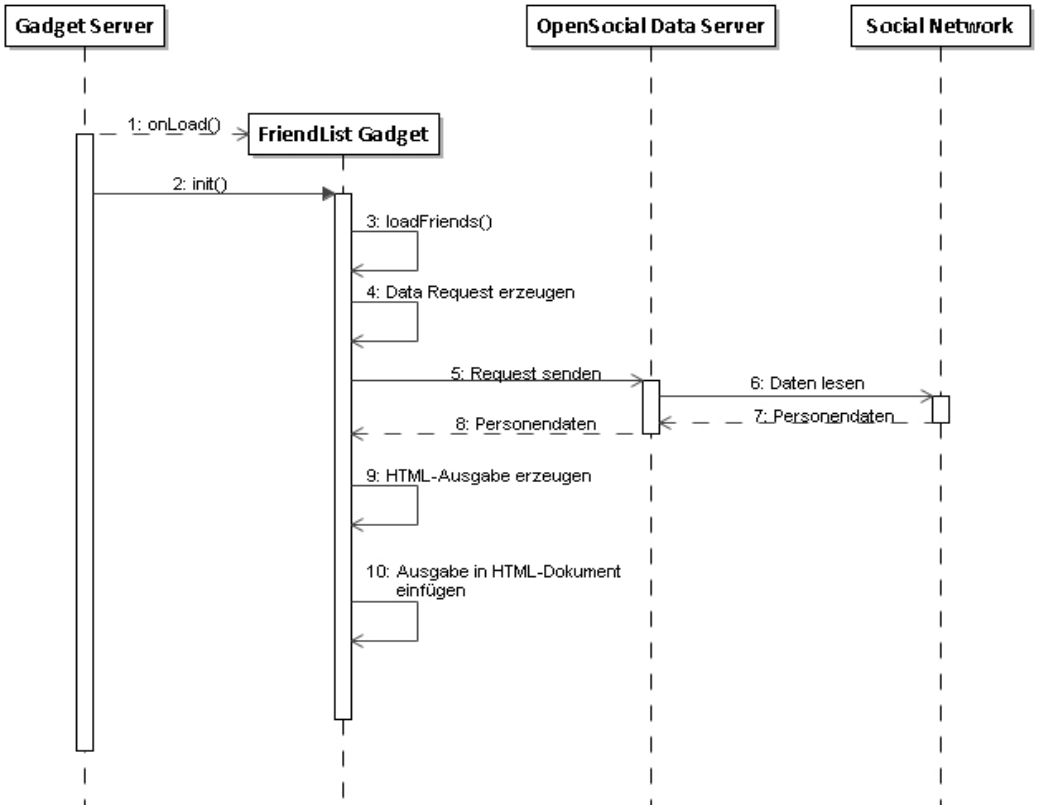

Abbildung 5: Ablauf FriendList-Applikation

Mittels `registerOnLoadHandler` wird eine callback-Funktion registriert, die ausgeführt wird, wenn das Gadget geladen wird. In diesem Beispiel ist dies die `init()`-Methode, die die Methode `loadFriends()` ausführt. In dieser Methode wird zu Beginn ein neues DataRequest-Objekt erzeugt, über das in OpenSocial Datenabfragen vorgenommen. Zur konkreten Abfrage von Daten müssen diesem Objekt FetchRequest-Objekte mitgegeben werden. Innerhalb diesem Beispiel wird ein FetchPersonRequest-Objekt zur Ermittlung der Daten des eingeloggten Benutzers (VIEWER) und ein FetchPeopleRequest-Objekt zur Ermittlung der Freunde dieser Person spezifiziert.

Der Request wird nach seiner Zusammenstellung per `send()`-Methode abgeschickt. Diese Methode enthält als Parameter eine Callback-Funktion (`onLoadFriends`), die nach der Ausführung der Datenabfrage aufgerufen wird. Innerhalb dieser Methode erfolgt die Verarbeitung der zurückgegebenen Daten.

Die Daten des eingeloggten Benutzers werden aus dem übergebenen Viewer-Objekt ausgelesen und innerhalb der HTML-Ausgabe dargestellt. Die Freundesdaten sind in dem Viewerfriends-Objekt als Array von Personendaten enthalten und werden als HTML-Liste ausgegeben.

Der Aufruf der `send()`-Methode wird in ein Aufruf des Containers umgewandelt, der die Daten im Format JavaScript Object Notation (JSON) zurückliefert. Die Daten in diesem Format werden via OpenSocial in ein

JavaScript-Objekt DataResponse umgewandelt, dass über die `onLoadFriends`-Methode ausgewertet werden kann. Auf dieser Art der asynchronen Verarbeitung basieren die meisten Aufrufe innerhalb der OpenSocial JavaScript-API.

In diesem Beispiel werden mit einem DataRequest-Objekt mehrere Aufrufe gebündelt. Dieser Mechanismus ist aus Performance-Gründen wichtig, da man sich dadurch viele kleine Server-Aufrufe erspart.

Die folgende Abbildung zeigt die Ausgabe der Beispiel-Applikation innerhalb des Community-Systems der Fernuni:

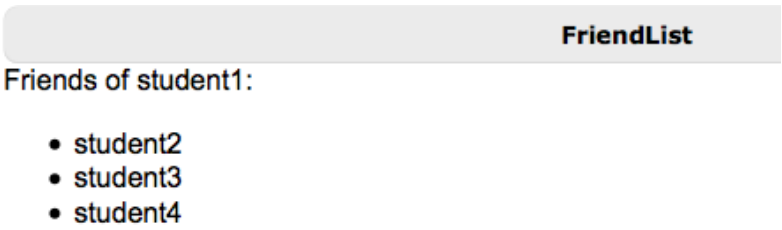

Abbildung 6: Ausgabe der Beispielapplikation

Der Prozess zum Auslesen von Daten aus dem Container via OpenSocial-JavaScript, kann allgemein wie folgt dargestellt werden:

- Datenabfrage erstellen
  - Erzeuge DataRequest-Objekt mittels Aufruf `opensocial.newDataRequest`.
  - Jeder Request, der durchgeführt werden soll, muss mittels `opensocial.new`*-Methoden erzeugt werden.
  - Dieser Request wird mittels `DataRequest.add(request)` hinzugefügt
- Callback-Methode erstellen, um Daten auszulesen
  - Nachdem der Container die Daten-Abfrage bearbeitet hat, wird die Callback-Methode aufgerufen. Diese erhält als Parameter `opensocial.DataResponse`-Objekte, die das Ergebnis des Aufrufs beinhalten.
  - Die Daten werden aus dem Response-Objekt ausgelesen und weiterverarbeitet.

Erweiterung um eigene Attribute

Im obigen Beispiel ist der Anzeigename der Person innerhalb des Netzwerkes über die Methode `getDisplayName()` ausgelesen worden. Weitere Felder der Person können über die Methode `getField()` ausgelesen werden. Als Parameter muss das entsprechende Feld angegeben werden. Z.B. `opensocial.Person.Field.DATE_OF_BIRTH` für das Geburtsdatum der Person. In der Klasse `Person` sind bereits eine Vielzahl an Person-Attributen enthalten, die einen grossen Bereich an Personenangaben abdecken. Da aber die benötigten Felder von Container zu Container differieren, besteht in OpenSocial die Möglichkeit die Person-Klasse um eigene Felder zu erweitern. Hierbei müssen Namenskonventionen

beachtet werden. Will z.B. Netzwerk „xy“ ein neues Attribut bereitstellen, so muss auf dieses Attribut mit `xy.Person.Field.new_Fieldname` benannt werden. Die Möglichkeit, einem Netzwerk eigene Attribute zu erlauben, ist auch der Grund, warum in der Klasse `Person` keine hartcodierten „getter” und „setter” für die Personen-Attribute vorhanden sind. Diese müssten sonst bei einer Erweiterung um neue Felder auch ausgebaut werden.

ActionRequests und Zugriffsrechte

In manchen Fällen kann es sinnvoll sein, dass der Zugriff zur Durchführung einer Action durch den Benutzer oder den Container blockiert wird.

OpenSocial definiert hierfür request-Methoden, die die Entscheidung zur Durchführung einer Aktion an den Container delegieren. Hierfür stehen Methoden wie `opensocial.requestCreateActivity` und `opensocial.requestPermission` zur Verfügung, die es dem Container erlauben, eigene Kontrollmechanismen einzubinden.

2.3 OpenSocial RESTful-API

Die OpenSocial RESTful-API ist die server-basierte Alternative zu der OpenSocial JavaScript-API. Diese basiert auf dem REST-Prinzip. Da mittels REST externe Systeme Zugriff auf die Daten erlangen, müssen Sicherheitsmechanismen zur Autorisierung und Authentifizierung bereitgestellt werden. Hierzu wird OAuth verwendet.

2.3.1 REST

REST steht für Representational State Transfer und bezeichnet einen Softwarearchitekturstil für verteilte Informationssysteme wie das World Wide Web. Der Begriff stammt aus der Dissertation von Roy Fielding aus dem Jahre 2000 [Fielding2000]. REST verfolgt 5 Grundprinzipien:

- Adressierbarkeit
- Zustandslosigkeit
- Wohldefinierte Operationen
- Verwendung von Hypermedia
- Unterschiedliche Repräsentationen

Adressierbarkeit

Ein Service, der seine Daten gemäss REST zur Verfügung stellt, ordnet jeder Ressource eine spezifische URI zu, über die auf die Ressource zugegriffen werden kann. Ein System, dass Kunden verwaltet kann hiermit den Zugriff auf die Daten eines Kunden ermöglichen. Mit der URI http://example.com/customer/1234 werden die Daten des Kunden mit der Kennung 1234 zurückgegeben. Dieser Ansatz bietet den Vorteil, dass man einen Link auf jeden einzelnen Kunden verschicken kann.

Zustandslosigkeit

REST setzt auf ein zustandsloses Client/Server-Protokoll. Dabei sind in jeder HTTP-Botschaft alle

Informationen enthalten, die notwendig sind, die Nachricht zu verarbeiten. Deshalb muss weder der Server, noch der Client Zustandsinformationen zwischen zwei Nachrichten speichern.

Wohldefinierte Operationen

Es existiert eine Menge wohldefinierter Operationen, die auf alle Ressourcen angewendet werden können. Bei HTTP sind dies die Operationen GET, POST, PUT und DELETE.

- Mit GET werden Daten vom Server gelesen.
- PUT legt neue Daten auf dem Server ab.
- POST aktualisiert vorhandene Daten auf dem Server oder ergänzt untergeordnete Ressourcen.
- DELETE löscht Daten auf dem Server.

Verwendung von Hypermedia

Innerhalb von REST wird sowohl für Anwendungsinformationen als auch für Zustandsveränderungen Hypermedia benutzt. Die Idee hinter Hypermedia ist das Konzept von Verknüpfungen (Links), wie aus HTML bekannt. D.h., ein Eintrag kann wiederum Links auf weitere Sub-Einträge enthalten. Ein Konsument dieses Services kann diesen Links „folgen", um an weitere Informationen zu gelangen.

Unterschiedliche Repräsentationen

Stellt ein Service seine Daten in unterschiedlichen Repräsentationen zur Verfügung, so kann dieser auch für unterschiedliche Anwendungszwecke genutzt werden.

2.3.2 OAuth

Wenn innerhalb einer OpenSocial REST-Applikation Zugriff auf Daten des sozialen Netzwerkes benötigt wird, ist ein Autorisierungs- und Authentifizierungsmechanismus notwendig, um den Zugriff aus Sicherheitsgründen zu steuern.

Autorisierung ist der Prozess, um einem Benutzer Zugriff auf Ressourcen zu erlauben. Mittels Authentifizierung wird die Identität eines Benutzers geprüft. Zur Steuerung dieser beiden Mechanismen wurde OAuth entwickelt, dass anhand dem folgenden Anwendungsszenario näher erläutert wird:

Es existiert eine allgemeine Web-Applikation zum Ausdruck von Bildern eines Online-Fotoalbums. Die Seite erlaubt dem Benutzer, die Fotos aus dem Fotoalbum eines sozialen Netzwerkes auszuwählen, anstatt die Bilder einzeln hochzuladen. Hierzu benötigt die Applikation Zugriff auf die Fotos des Benutzers innerhalb des sozialen Netzwerkes. Dies könnte erreicht werden, in dem der Benutzer innerhalb der Druck-Applikation seine Benutzerangaben des sozialen Netzwerkes, wie Benutzername und Passwort angibt. Die Druck-Applikation führt mit diesen Informationen selbständig einen Zugriff auf das Netzwerk durch, um die entsprechenden Fotos von dieser Seite auszulesen. Dieser Ablauf wäre höchst fahrlässig, da private Informationen wie das Passwort an Dritte weitergegeben werden müssen. Um dies zu verhindern, wurde OAuth entwickelt. Anstelle das Passwort auf der Druck-Seite anzugeben, wird der Benutzer auf die Netzwerk-Seite weitergeleitet, wo er sich einloggen muss und bestätigen muss, dass der Zugriff von der Druck-Seite aus erlaubt ist. Die Netzwerk-Seite erzeugt daraufhin einen OAuth-Token und gibt diesen zurück an die Druck-Seite, die wiederum diesen Token für jeden Request an die Netzwerk-Seite benutzt. Mit

diesem Ablauf wird also erreicht, dass kein Passwort an Dritte weitergegeben werden muss, aber trotzdem ein beschränkter Zugriff auf die eigenen Ressourcen erteilt werden kann.

Dieses Verfahren lässt sich am einfachsten mit der Metapher „Valet-Parking“, zu Deutsch „Das Parken eines Fahrzeuges durch einen Angestellten“, verdeutlichen [OAuthExpl2007]. Einige Autohersteller bieten mittlerweile einen Zweitschlüssel mit einer limitierten Funktion an, den man einer Person zum Parken des Autos aushändigen kann. Dieser Schlüssel erlaubt z.B. nur das Fahren des Autos für 2-3 km, und verbietet die Öffnung des Kofferraums und des Handschuhfachs. Der Diebstahl des Fahrzeuges kann somit erschwert werden. Der „Valet-Key“ ist also ein Schlüssel, der die Nutzung einer Ressource in einem beschränkten Masse erlaubt. Übertragen auf Internet-Seiten will der Benutzer einer fremden Anwendung auch nur Zugang zu bestimmten Ressourcen geben und nicht seine ganzen Daten freigeben.

In OAuth wird folgende Terminologie verwendet [OAuthBegGuide2007], die im weiteren in diesem Dokument genutzt wird.

- Service Provider – Dies ist die Web-Seite, wo die freizugebenden Ressourcen abgelegt sind. Dies ist in OpenSocial typischerweise das soziale Netzwerk.
- Consumer – Dies ist die Web-Seite oder die Applikation, die Zugriff auf die Ressourcen benötigt.
- User – Der Benutzer, dessen Ressourcen geschützt werden sollen. Dieser besitzt Ressourcen, die er nicht allen öffentlich zugänglich machen will.
- Tokens – Diese werden anstelle von Benutzerinformationen wie Passwort genutzt, um den Zugriff auf Ressourcen zu steuern. Ein solcher Token ist allgemein gesagt, ein eindeutiger String, bestehend aus Buchstaben und Zahlen, kombiniert mit einem Geheimschlüssel, um den Token vor Missbrauch zu schützen.

Innerhalb OpenSocial existieren für OAuth drei verschiedene Anwendungsszenarien, die im folgenden aufgezeigt werden:

OAuth-Signierung

Ein JavaScript-Gadget greift mittels `makeRequest` auf einen Application Server zu. Der Request des Gadgets wird mittels OAuth signiert, um es dem Backend Server zu erlauben, die Gültigkeit des Containers, der Applikations-URL und der ID des Viewers und des Owners zu verifizieren.

2-legged OAuth

Ein JavaScript-Gadget besitzt eine Verarbeitung auf einem Application Server, die wiederum auf die Daten des sozialen Netzwerkes zugreift.

Die Applikation zur Bewertung von Büchern aus dem Beginn von Kapitel 2 stellt ein Beispiel für diesen Fall dar. Schreibt die Verarbeitung auf dem Server im Hintergrund Einträge in die Aktivitätenliste des Benutzers, so geschieht dies über die RESTful-API. Der Application Server benötigt hierfür Zugriff auf die sozialen Daten des Nutzers, der die Applikation innerhalb des Netzwerkes installiert hat. Das Szenario wird als 2-legged bezeichnet, da in den Ablauf zwei Parteien, der Consumer und der Service Provider eingebunden sind.

3-legged OAuth

Eine Web-Seite benötigt Zugriff auf Daten eines sozialen Netzwerkes, ohne dass ein Gadget installiert ist.

Ein Beispiel wäre eine allgemeine Nachrichten-Seite. Bewertet ein Benutzer einen für ihn interessanten Artikel, wird im Hintergrund ein neuer Eintrag in die Aktivitätenliste des Benutzers vorgenommen. Hierzu benötigt die Nachrichten-Seite Zugriff auf die Daten des jeweiligen sozialen Netzwerkes. Die hierfür notwendige Autorisierung wird durch die Verlinkung des Benutzer-Konto´s auf der Nachrichten-Seite mit dem entsprechenden Benutzer-Konto in dem sozialen Netzwerk durchgeführt. Der Benutzer wird bei diesem Vorgang auf die Netzwerk-Seite weitergeleitet um den Zugriff zu gewähren. Dieses Szenario wird als 3-legged bezeichnet, da neben dem Consumer und Service Provider auch der User in den Ablauf beteiligt ist.

2.3.3 OpenSocial REST-Spezifikation

Die OpenSocial RESTful-API stellt im Wesentlichen die gleichen Möglichkeiten wie die JavaScript-API zur Verfügung. Für jeden Teilbereich von OpenSocial wie Personendaten, Aktivitäten und Applikationsdaten können Requests konstruiert werden. Ein solcher Request baut sich aus einer Basis-URL und einem API-spezifischen Bereich zusammen.

Die Basis-URL ist die URL des Netzwerkes, über das die OpenSocial REST-Aufrufe durchgeführt werden können. Für das Netzwerk hi5 lautet diese URL z.B. http://api.hi5.com/social/rest.

Der API-spezifische Teil stellt die Erweiterung dar, um den jeweiligen REST-API-Aufruf anzugeben. Um z.B. die Personendaten einer bestimmten Person auszulesen, lautet der spezifische Teil `/people/{guid}/@self`, wobei `{guid}` einen eindeutigen Bezeichner für eine Person innerhalb des Containers darstellt. Der Wert von `{guid}` kann eine konkrete ID oder das Schlüsselwort `@me` sein. `@me` steht für den Benutzer der Applikation, also der Benutzer, der mittels OAuth authentifiziert wurde. Die komplette URL zum Lesen der Daten einer Person lautet somit `http://api.hi5.com/social/rest/people/{guid}/@self` . Ist die Person spezifiziert, muss mittels einem weiteren Symbol angegeben werden, ob es sich um diese Person oder eine Gruppe von Benutzern handelt, die zu der Person in Beziehung stehen.

- `@self` – Dieses Symbol gibt den Benutzer an.
- `@all` – Mittels diesem Symbol werden alle Personen, die zu dem angegebenen Benutzer in Kontakt stehen, identifiziert.
- `@friends` – Dieses Symbol identifiziert von allen zu dem angegebenen Benutzer in Kontakt stehenden Personen nur diese, die als Freund zugeordnet sind.

Innerhalb der REST-API wird dieses Konstrukt zum Lesen von Personendaten, sowie Aktivitäten und Applikationsdaten genutzt.

Personendaten

Die folgende Übersicht zeigt Beispiele um Personendaten auszulesen:

- `/people/@me/@self` – Liefert Informationen über den Aufrufer.
- `/people/{guid}/@friends` – Listet alle Freunde der mit `{guid}` angegebenen Person.

Aktivitäten

Zum Lesen von Aktivitätsdaten wird GET verwendet. Neue Aktivitäten können mit POST angelegt werden. In dieser Übersicht sind Beispiele zum Lesen von Aktivitätendaten angegeben:

- `/activities/{guid}/@self` – Aktivitäten des angegebenen Benutzers.
- `/activities/{guid}/@friends` – Aktivitäten der Freunde des angegebenen Benutzers.

Applikationsdaten

Diese Übersicht zeigt Beispiele zum Lesen von Applikationsdaten:

- `/appdata/{guid}/@self/{appid}` – Liefert alle Applikationsdaten der mit `{appid}` angegebenen Applikation, die zu dem angegebenen Benutzer gespeichert sind.
- `/appdata/{guid}/@self/@app` – Alle Applikationsdaten der Applikation, die diesen Aufruf durchgeführt hat und zu dem angegebenen Benutzer gespeichert sind.

Datenformate

OpenSocial unterstützt die Datenformate JSON, XML und Atom. Die gewünschte Art kann beim Aufruf spezifiziert werden.

JSON steht für JavaScript Object Notation und ist ein leichtgewichtiges Datenformat um Daten auszutauschen. Ein minimaler Personendatensatz sieht in JSON folgendermassen aus:

```
{
  "id" : "example.org:34KJDCSKJN2HHF0DW20394",
  "displayName" : "Janey",
  "name" : {"unstructured" : "Jane Doe"},
  "gender" : "female"
}
```

Entsprechende XML-Repräsentation:

```
<person xmlns="http://ns.opensocial.org/2008/opensocial">
  <id></id>
  <displayName>Janey</displayName>
  <name>
    <unstructured>Jane Doe</unstructured>
  </name>
  <gender>female</gender>
</person>
```

Format in Atom+XML:

```
<entry xmlns="http://www.w3.org/2005/Atom">
  <content type="application/xml">
    <person xmlns="http://ns.opensocial.org/2008/opensocial">
      <name>
        <unstructured>Jane Doe</unstructured>
      </name>
      <gender>female</gender>
    </person>
  </content>
  <title/>
  <updated>2003-12-13T18:30:02Z</updated>
```

```
  <author/>
  <id>urn:guid:example.org:34KJDCSKJN2HHF0DW20394</id>
</entry>
```

OpenSocial RPC-Protokoll

Das RPC-Protokoll definiert eine Struktur, die den gleichen Funktionsumfang wie die REST-API bietet, aber auf Remote Procedure Calls basiert. Diese hat gegenüber REST den Vorteil, dass sie für Batch-Verarbeitungen besser geeignet ist, da in einem Aufruf mehrere Datenabfragen durchgeführt werden können. Mit der REST-API wären hierfür mehrere Aufrufe notwendig.

Nachfolgend ein Beispiel zum Auslesen von Informationen zu einem Benutzer. Mit der REST-API sieht der Aufruf wie folgt aus:

```
http://container.com/people/@me/@self
```

Der gleiche Aufruf mit RPC:

```
http://container.com/rpc?method=people.get&id=myself@userid=@me&groupId=@self
```

Innerhalb RPC wird die Art des Aufrufs (Datenabfrage, spezifiziert mit `people.get`) in die Abfragedaten mit aufgenommen, während in REST die Datenabfrage durch HTTP GET spezifiziert wird.

2.4 Weitere OpenSocial-Technologien

Im Zuge der Entwicklung der OpenSocial-API sind noch einige zusätzliche Technologien entstanden, die die Entwicklung von OpenSocial-Applikationen vereinfachen sollen. Diese Technologien werden in diesem Kapitel nur ansatzweise vorgestellt, um einen groben Überblick zu geben. Die Erwähnung hier verdeutlicht aber, dass in diesem Bereich momentan eine rege Entwicklung stattfindet, um den OpenSocial-Standard weiter voranzutreiben.

2.4.1 OpenSocial ClientLibraries

In einem weiteren Projekt wurden auf Basis der RESTful-API die OpenSocial ClientLibraries erstellt. Diese Bibliothek bietet Klassen für verschiedene Programmiersprachen an, die eine komfortablere Nutzung der oben gezeigten Technologie ermöglichen sollen [OSClientLib]. Implementierungen sind mittlerweile für verschiedene Programmiersprachen und Umgebungen verfügbar, wie z.B. Java, Phyton, Ruby und Google Android.

Ein Zugriff auf das Profil eines Benutzers sieht mit der Java-ClientLibrary folgendermassen aus:

```
OpenSocialClient c = new OpenSocialClient("myhost.com");
c.setProperty(OpenSocialClient.Properties.REST_BASE_URL,
       "http://localhost:8080/social/rest");
try {
  OpenSocialPerson p = c.fetchPerson("john.doe");
  System.out.println(p.getDisplayName());

  Collection<OpenSocialPerson> friends = c.fetchFriends("john.doe");
  for (OpenSocialPerson friend: friends) {
    System.out.println(friend.getDisplayName());
  }
```

```
}
catch (Exception e) {
...
}
```

2.4.2 OpenSocial Templates

Der OpenSocial Templates-Standard, der erst vor kurzem ins Leben gerufen worden ist, erlaubt Applikations-Entwicklern allgemeingültige Funktionen und GUI-Elemente zu definieren, die dann in verschiedenen Kontexten genutzt werden können. Funktionalitäten, die zuvor mit JavaScript, DOM-Manipulation und OpenSocial JavaScript-Aufrufen erzeugt wurden, können mit dieser Technologie mittels einer Markup-Language und Template-Skripten einfach bereitgestellt werden.

Der OpenSocial Template-Standard unterteilt sich in drei Bestandteile:

- OpenSocial Templates, Teil 1 – Dieser Teil spezifiziert, wie Templates erstellt werden und wie diese in eine Applikation eingebunden werden können.
- OpenSocial Markup Language, Teil 2 – Hiermit werden Tags definiert, über die Daten abgefragt und GUI-Elemente definiert werden können.
- OpenSocial Data Pipelining, Teil 3 – Dieser Bereich enthält eine deklarative Syntax, um die Daten, die ein Container bieten soll, zu definieren.

Template

Templates bieten die Möglichkeit, eine Funktion allgemeingültig zu implementieren. Beim Aufruf des Templates werden die konkreten Daten mitgegeben, die innerhalb des Templates verarbeitet werden. Somit bieten Templates einen Wiederverwendungsmechanismus für die Entwicklung von OpenSocial-Applikationen.

OpenSocial Makup Language

OSML ist eine Sprache, die aus Tag-Elementen mit vordefinierten Bedeutungen besteht. Hiermit sind OpenSocial-Daten oder häufig verwendete GUI-Elemente vordefiniert. Nachfolgend einige Beispiele vordefinierter Tags zur Verdeutlichung:

Tag	Beschreibung	Beispiel
`<os:Name>`	Liefert den Namen zu einem Person-Objekt.	`Hallo <os:Name person="${Viewer}" />`
`<os:PeopleRequest>`	Liefert Informationen zu einer Gruppe von Personen in Form eines Array's mit OpenSocial-Person-Objekten. Das Rückgabeformat ist JSON.	`<os:PeopleRequest key="friends" userid ="@owner" groupid = "@friends" />`
`<os:ViewerRequest>`	Liefert Informationen über den eingeloggten Benutzer in Form eines OpenSocial-Person-Objektes.	`<os:ViewerRequest key="viewer" fields="name, birthday" />`

Mittels OSML können neben den vordefinierten Tags auch eigene Tags definiert werden.

Data Pipelining

OpenSocial Data Pipelining stellt einen Mechanismus dar, Daten deklarativ von einer Stelle zu laden und diese in Templates zur Verfügung zu stellen. Beispiele hierfür sind `<os:DataRequest key="peopledata" method="people.get" userId="@vieser"/>` zum Zugriff auf OpenSocial-Daten oder `<os:HttpRequest key="mydata" href="http://www.example.com/api"/>` zum Zugriff auf Daten von irgendeiner HTTP-Datenquelle.

3 Integration der OpenSocial-API in ein soziales Netzwerk

Damit ein soziales Netzwerk als OpenSocial-Container betrieben werden kann, muss es Services anbieten, die von den JavaScript-Methoden aus aufgerufen werden können und einen Mechanismus bieten, Gadgets innerhalb der Web-Applikation darzustellen. Diese Implementierung kann komplett selbst vorgenommen werden, ist aber sehr aufwendig. Ein einfacherer Ansatz ist die Integration der OpenSocial-Referenzimplementierung Shindig [ApShindig]. Shindig bietet bereits eine fertige Container-Implementierung, die in verschiedenen Bereichen angepasst werden muss, um mit dem aufrufenden Community-System zu kommunizieren.

Zusätzlich zu der technischen Integration sind in dem Community-System noch Funktionalitäten zur Verwaltung der Applikationen zu berücksichtigen.

Im folgenden wird auf die Integrationsaspekte bei Verwendung der Shindig-Implementierung eingegangen und die Bereiche aufgezeigt, die zur Verwaltung der Applikationen abgedeckt werden sollten.

3.1 Apache Shindig, eine Referenzimplementierung

Shindig ist ein Projekt der Apache Software Foundation und ist eine OpenSource-Implementierung der OpenSocial-Spezifikation[OpenSocialSpec] und der Gadget-Spezifikation[GadgetSpec].

Shindig ist in Java und in PHP verfügbar und besteht aus vier Teil-Komponenten:

- Gadget Container-JavaScript – JavaScript-Implementation für generelle Gadget-Funktionalität (Security, Kommunikation, Darstellung und Möglichkeit der Erweiterung mit Features).
- Gadget Server – Eine OpenSource-Version von gmodules.com. Hierüber wird das Gadget xml in JavaScript und HTML transformiert.
- OpenSocial Container-JavaScript – JavaScript-Implementation auf Basis des Gadget Container-JavaScript. Diese Komponente stellt die spezifischen Erweiterungen für OpenSocial (Profile, Freunde, Aktivitäten, ...) zur Verfügung.
- OpenSocial Data-Server – Eine OpenSource-Version der Server-Interfaces, um container-spezifische Informationen bereitzustellen. Über diese Schnittstelle kann eine Anbindung an die eigenen Backend-Services erfolgen.

Zusammengefasst bietet Shindig das Grundgerüst, um OpenSocial-Applikationen auf irgendeiner Web-Seite integrieren zu können. Mit der Nutzung von Shindig kann eine Web-Seite mit relativ geringem Aufwand zu einem OpenSocial-Container erweitert werden.

Mittlerweile kommt Shindig in mehreren kommerziellen sozialen Netzwerken zum Einsatz. Unter anderem hi5, MySpace und Orkut. Eine Liste der aktuell unterstützten Systeme ist unter [OpenSocialContainer] verfügbar.

3.2 Integration von Shindig

Die Integration erfolgt auf zwei Ebenen. Im Bereich Benutzeroberfläche werden die OpenSocial-Applikationen innerhalb den eigenen Web-Seiten des sozialen Netzwerkes dargestellt und ausgeführt. Auf Datenebene erfolgt der Zugriff auf die Daten des aufrufenden Netzwerkes, die innerhalb des Gadget verwendet werden können.

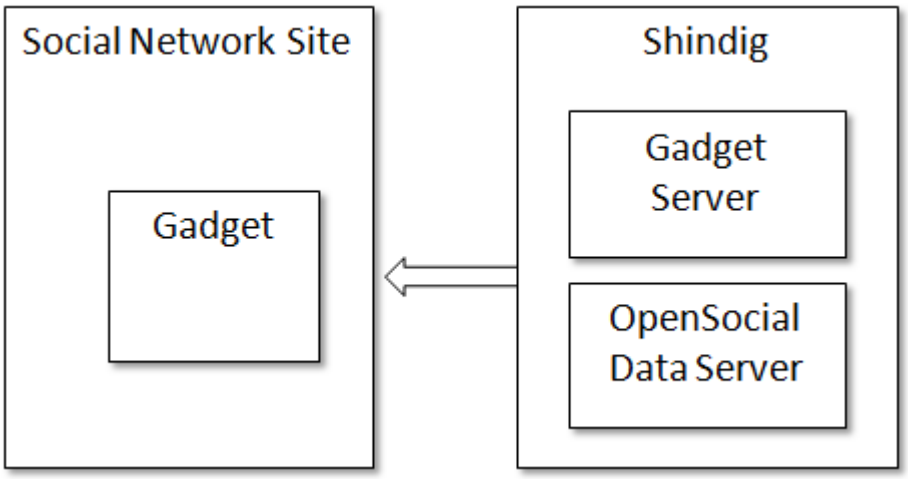

Abbildung 7: Integration von Shindig

Der Gadget Server ist verantwortlich für die Darstellung der Gadgets.

Der Social-Data Server stellt die benötigten Daten des sozialen Netzwerkes zur Verfügung. Hierzu müssen die entsprechenden Daten-Interfaces (`PersonService`, ...) des Shindig-Servers implementiert werden. Der Aufruf dieser Methoden aus der Web-Seite heraus erfolgt über entsprechende JavaScript-Methoden, die dann innerhalb Shindig in Java-Methodenaufrufe weitergeleitet werden.

Um den Datenzugriff für nicht autorisierte Benutzer zu unterbinden, müssen bei der Integration Security-Aspekte berücksichtigt werden.

3.2.1 Datenzugriffs-Schnittstelle

Auf Gadget-Ebene bietet die OpenSocial-API verschiedene JavaScript-Methoden an, um Daten des Netzwerkes zu lesen oder zu schreiben. Z.B. gibt der folgende Code-Abschnitt die Personendaten des eingeloggten Benutzers aus.

```
var req = opensocial.newDataRequest();
req.add(req.newFetchPersonRequest(opensocial.IdSpec.PersonId.VIEWER), 'viewer');
req.send(onLoadPerson);
```

Diese Aufrufe werden an den Social-Data Server in Shindig weitergeleitet, der innerhalb des Servlets eine Transformation vornimmt und eine entsprechende Java-Methode aufruft, die die Daten aus der DB liest und an das Gadget zurückgibt.

Die entsprechende Java-Methode, die durch obigen Request ausgeführt wird, hat folgende Signatur:

```
Future<Person> getPerson(UserId id, Set<String> fields, SecurityToken token)
  throws SocialSpiException;
```

Über den Parameter `id` wird spezifiziert, von welcher Person die Daten gelesen werden sollen. Von einer Person mit einer bestimmten ID oder aber von dem Viewer oder Owner. Bei Viewer oder Owner ist die entsprechende ID aus dem SecurityToken auszulesen, der dem Request als Parameter mitgegeben wird. Der SecurityToken enthält diverse Daten wie den Viewer und den Owner der Applikation. Eine detaillierte Beschreibung des SecurityTokens erfolgt in Kapitel 3.2.3 Security-Aspekte. Über den Parameter `fields` können die zu lesenden Felder des Personen-Records spezifiziert werden.

Die Datenzugriffsschnittstelle besteht aus den drei Interfaces `PersonService`, `ActivityService` und `AppDataService`.

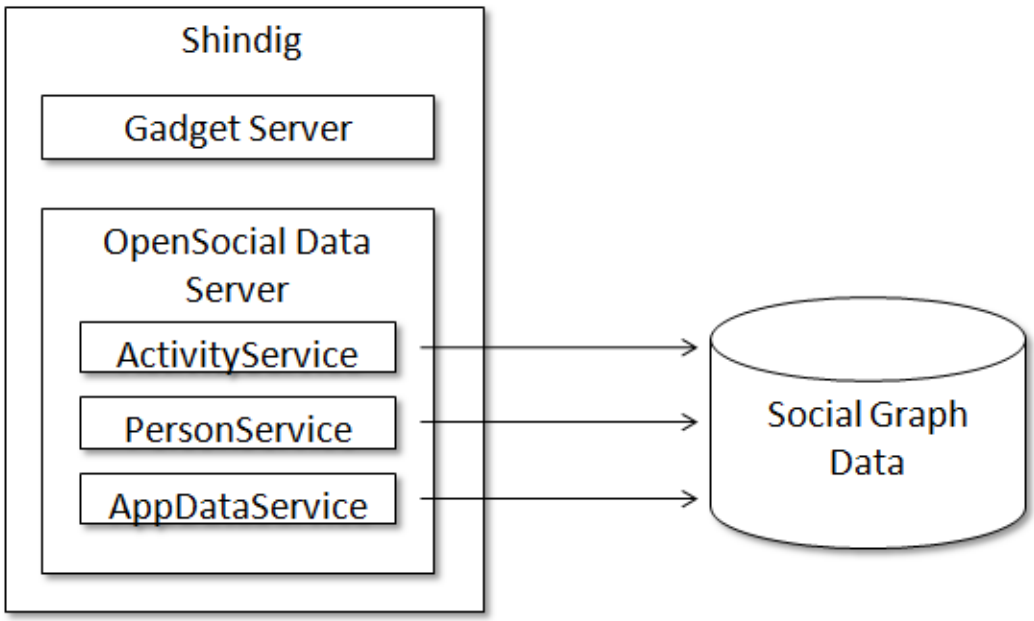

Abbildung 8: Datenzugriffsschnittstelle von Shindig

PersonService

Das Interface enthält Methoden zum Lesen von Personendaten.

- `getPerson` – Liefert die Daten einer Person zurück. Die Person wird über die angegebene UserId spezifiziert.
- `getPeople` – Liefert eine Liste von Personendaten zurück. Die Personen können entweder als Liste von UserIds oder durch Angabe einer GroupId spezifiziert werden. Mit der GroupId können festgelegte Mengen von Personen, wie z.B. die Freunde eines Nutzers, angegeben werden.

ActivityService

Hier sind Methoden zum Lesen, Erstellen und Löschen von Aktivitäten zu einer Person enthalten.

- `getActivities` – Liefert eine Liste von Aktivitäten zu einer Person zurück. Die Aktivitäten werden über eine Liste von ActivityIds spezifiziert.
- `getActivity` – Liefert eine Aktivität zu der angegebenen ActivityId zurück.
- `deleteActivities` – Löscht die angegebenen Aktivitäten
- `createActivity` – Erstellt eine neue Aktivität

AppDataService

Das Interface enthält Methoden zum Lesen, Erstellen und Löschen von Applikationsdaten zu einer Person.

- `getPersonData` – Liefert die Applikationsdaten eines Benutzers zu der angegebenen Applikation zurück.
- `deletePersonData` – Löscht die Applikationsdaten eines Benutzers zu der angegebenen Applikation.
- `updatePersonData` – Legt Applikationsdaten eines Benutzers zu der angegebenen Applikation an. Bestehende Daten werden mit dieser Methode geändert.

Diese Interfaces müssen implementiert werden, um den Zugriff auf die eigene Datenbasis herzustellen. Zur Registrierung dieser Klassen wird ein Guice-Modul erstellt, innerhalb dessen den Interfaces die eigenen Implementierungen zugeordnet werden.

Guice [Raja2007] ist ein leichtgewichtiges Dependency-Injection Framework für Java5. Dependency-Injection wird benötigt, wenn die Erzeugung der Implementierungs-Klassen von Interfaces nicht direkt im Client-Code angegeben werden soll, um z.B. konfigurierbar zu sein. Die Instanzierung der jeweiligen Implementierungsklasse wird in eigenen Modulen mittels einem Binder angegeben, der quasi ein Mapping einer Implementierungsklasse zu seinem Interface darstellt. In Shindig werden mit diesem Mechanismus die eigenen Implementierungsklassen der Service-Interfaces angegeben.

Mit Shindig wird als Default-Implementierung für die drei Services die Klasse `JsonDBOpensocialService` ausgeliefert. Bei dieser Klasse ist zu beachten, dass diese nur zu Testzwecken verwendet werden sollte, da die Daten hierüber in einer Datei im JSON-Format abgelegt werden.

In der nächsten Version der OpenSocial-Spezifikationen [OpenSocialSpec09] ist der Zugriff auf Fotoalben enthalten und eine Möglichkeit, Nachrichten innerhalb des Netzwerkes zu versenden. Die Nachrichten-Funktion beinhaltet öffentliche Profil-Kommentare und private Nachrichten zwischen Benutzern oder innerhalb von Gruppen. Dementsprechend werden die oben angegebenen Interfaces auch erweitert werden.

3.2.2 Benutzer-Schnittstelle

Innerhalb einer HTML-Seite kann ein Gadget an beliebiger Stelle über ein iFrame-Element eingefügt und angezeigt werden. Innerhalb dieses iFrames wird die von dem Gadget Server erzeugte Seite als eigenständige HTML-Seite angezeigt. Mit diesem Ansatz kann ein Gadget nahtlos in die eingebettete Seite eingefügt werden.

Dem iFrame wird als Source eine URL angegeben, die alle Informationen beinhaltet, um das Gadget

anzuzeigen. Die URL entspricht der Adresse des Gadget-Servers, angereichert mit Parametern, die zur Darstellung des Gadgets benötigt werden. Diese URL wird vom Container generiert, da hier alle notwendigen Informationen vorhanden sind.

`http://<SHINDIG>/gadgets/ifr`

- `?container` – Name des eigenen Containers.
- `&viewer` – ID des eingeloggten Benutzers
- `&owner` – ID des Besitzers der Applikation, die gerade ausgeführt wird
- `&aid` – Applikations-ID. Dies kann ein numerischer Wert oder die URL der Applikation sein.
- `&mid` – Modul-Id.
- `&country` – Ländereinstellung.
- `&lang` – Spracheinstellung. Diese Information wird genutzt, wenn das Gadget Mehrsprachigkeit unterstützt.
- `&view` – Name der View, mit der das Gadget angezeigt werden soll. Z.B. default, canvas, home, ...
- `&parent` – URL des Containers
- `&up` – Benutzerparameter. Falls das Gadget Benutzerparameter unterstützt, müssen die entsprechenden Werte im Container gespeichert werden und werden dem Gadget über diesen Weg mitgegeben.
- `&st` – Vom Container generierter SecurityToken. Über diese Zeichenkette werden die Informationen, die zum Zugriff auf den Gadget und Data Server notwendig sind, weitergegeben. Die Zeichenkette kann verschlüsselt werden, um einen unerlaubten Zugriff zu verhindern. Eine detaillierte Erläuterung dieses Mechanismus erfolgt in Kapitel 3.2.3 Security-Aspekte.
- `&url` – URL der XML-Datei der Applikation
- `&nocache` – Angabe, ob der interne Cache von Shindig genutzt werden soll. Mit 1 wird dieser nicht genutzt. 0 schaltet den Cache ein.
- `#rpctoken` – eindeutige per Zufallsgenerator erzeugte Nummer

Neben der iFrame-URL müssen noch diverse JavaScript-Bibliotheken eingebunden werden, die Mechanismen zur Verfügung stellen, über die die eingebettete Gadget-HTML-Seite mit der Container-HTML-Seite kommunizieren kann.

Eine solche Kommunikation ist für folgende Fälle notwendig:

- Anfrage eines Gadgets zur Änderung der Höhe der eingebetteten HTML-Seite. Ein Gadget kann über das Feature dynamicHeight eine dynamische Anpassung der Seiten-Höhe unterstützten. Die Änderung wird über die Funktion `setHeight()` in der Datei gadgets.js durchgeführt.

- Anfrage zur Navigation auf eine andere View. Gadgets, die mehrere Views unterstützen, bieten oft auf einer Übersichts-Seite eine zusammenfassende Darstellung der Daten. Die Detaillierte Ansicht kann durch Betätigung eines Links oder Buttons aktiviert werden. Da sich ein Gadget nicht selbständig auf einer anderen Seite innerhalb des Containers anzeigen kann, muss die Anfrage an den Container weitergeleitet werden. Dieser kann das Gadget auf einer der View entsprechenden Seite darstellen. Ein Beispiel für diesen Mechanismus ist die Anzeige eines Gadgets auf einer Voransicht. Will der Benutzer eine bestimmte Aktion auf der Applikation auslösen, die nur in der Canvas-View erlaubt ist, löst das Gadget mittels der Funktion `requestNavigateTo()` einen entsprechenden Request an den Container aus, der nun das Gadget in der Canvas-Ansicht darstellt.

Die Datei gadgets.js bietet eine Default-Implementierungen seitens Shindig, die aber mit der Container-Implementierung angepasst werden müssen. Z.B. kann ein Container die Höhe des darzustellenden Gadgets limitieren. Dies muss dann in der Funktion `setHeight()` entsprechend implementiert werden. Hierzu muss eine eigene JavaScript-Datei erstellt werden, die die eigenen Implementierungen enthält. Zum Überladen dieser Methoden wird das JavaScript-Framework Prototype [Prototype] verwendet.

Neben dem hier beschriebenen Mechanismus, ein Gadget direkt über eine iFrame-URL darzustellen, bietet gadgets.js auch Features, um die entsprechende URL per JavaScript-Aufrufen zu erstellen. Die Erzeugung über direkte Angabe des HTML-Codes, ist aber wesentlich flexibler.

Die Beispiele in der Shindig-Installation nutzen die JavaScript-Funktionen aus gadgets.js. Der folgende Code stellt einen Auszug aus der Datei Sample1.html dar.

```
...
var specUrl0 = 'http://localhost:8080/gadgets/files/container/FriendList.xml';
var specUrl1 = 'http://localhost:8080/gadgets/files/container/HelloWorld.xml';


// This container lays out and renders gadgets itself.

function renderGadgets() {
  var gadget0 = gadgets.container.createGadget({specUrl: specUrl0});
  var gadget1 = gadgets.container.createGadget({specUrl: specUrl1});

  gadgets.container.addGadget(gadget0);
  gadgets.container.addGadget(gadget1);
  gadgets.container.layoutManager.setGadgetChromeIds(
      ['gadget-chrome-x', 'gadget-chrome-y']);

  gadgets.container.renderGadget(gadget0);
  gadgets.container.renderGadget(gadget1);
};
</script>
</head>
<body onLoad="renderGadgets()">
...
```

3.2.3 Security-Aspekte

SecurityToken

Wie zuvor schon angedeutet, wird zur Authentifizierung eines Gadget-Requests ein SecurityToken verwendet. Dieser wird vom Container mittels eines Verschlüsselungsalgorithmus erzeugt und über die iFrame-URL dem Gadget mitgegeben. In jedem Request eines Gadgets zum Server wird der Token

mitgeliefert und innerhalb des Servers vor Verarbeitung des Requests ausgewertet und auf Korrektheit geprüft. Tritt hierbei ein Fehler auf, so wird der Request mit einer entsprechenden Fehlermeldung zurückgewiesen.

Fehler können auftreten, wenn der Token z.B. durch einen Angreifer beschädigt worden ist und somit nicht mehr korrekt entschlüsselt werden kann. Ein Angreifer könnte auch versuchen, selbst einen Token zu generieren und damit einen Aufruf an den Server zu richten. Die Entschlüsselung dieses Tokens wird einen Fehler verursachen, da zur Generierung ein Schlüsselwort notwendig ist, dass im Server hinterlegt ist.

Zu Testzwecken bietet Shindig aber auch die Möglichkeit, den Token unverschlüsselt zu erzeugen. Dies wird über eine Konfigurationseinstellung im Shindig-Server gesteuert.

Innerhalb des SecurityTokens sind verschiedene Informationen enthalten, die zur Verarbeitung eines Requests im Server benötigt werden:

- `viewerid` – ID des eingeloggten Benutzers
- `ownerid` – ID des Besitzers der Applikation
- `moduleid` – Nähere Spezifikation des Applikations-Identifiers
- `appid` – ID der Applikation, die gerade ausgeführt wird
- `appurl` – URL der Applikation, die gerade ausgeführt wird

Die Werte werden zu einem String zusammengesetzt und mit einem Doppelpunkt getrennt, so dass ein Token ein Format gemäss dem Muster `viewerid:ownerid:moduleid:appid:appurl` hat.

Dieser Ansatz genügt zur Authentifizierung innerhalb des Containers. Wird aber auf geschützte Daten von ausserhalb des Containers zugegriffen, so wird ein anderer Mechanismus zum Schutz der Daten benötigt. Hierfür kommt OAuth zum Einsatz.

OAuth

Um OAuth in Shindig vollumfänglich zu nutzen, muss die Basisinstallation von Shindig noch um Mechanismen erweitert werden, um Client-Schlüssel und Autorisierungs-Tokens zu speichern. Die Basisinstallation ist zwar fähig, OAuth-Authenifizierung für Gadget-Requests durchzuführen, kann aber keine Schlüsselwerte speichern.

Erweiterung der Basisinstallation zu Testzwecken

OAuth-Support kann aktiviert werden durch Einträge in der Datei config/oauth.js. Diese Datei beinhaltet eine Liste von Consumer-Geheimnissen, die ein Gadget zur Kommunikation mit einem Service nutzt. Z.B. hat ein Gadget folgende Konfiguration:

```
<OAuth>
  <Service name="MyRemoteServer">
    <Request url="http://some-server.com:9090/oauth-provider/request_token" />
    <Access url="http://some-server.com:9090/oauth-provider/access_token" />
    <Authorization url="http://some-server.com:9090/oauth-provider/access_token" />
  </Service>
</OAuth>
```

Damit ist ein Eintrag gemäss folgendem Muster in der Datei config/oauth.js notwendig.

```
{
  "http://one.author.com/myGadget.xml" : {
    "MyRemoteServer" : {
      "consumer_key : "gadgetConsumer",
      "consumer_secret : "gadgetSecret",
      "key_type" : "HMAC_SYMETRIC"
    }
  }
}
```

Die Daten `comsumer_key`, `consumer_secret` und `key_type` müssen vorab zwischen Gadget-Entwickler und dem Server vereinbart werden. Die Rolle des Containers ist hierbei, die Daten bei der Installation des Gadgets zu erhalten und zu speichern. Mit dieser Konfiguration ist das Gadget in der Lage, OAuth-Daten-Requests zu some-server.com durchzuführen.

Erweiterung für Produktionsumgebungen

Um OAuth in produktiven Umgebungen zu nutzen, muss das Interface `OAuthStore` implementiert werden und die Implementierung in Shindig über ein Guice-Modul registriert werden. Diese Implementierung muss OAuth-Daten (key, secret, key_type) in einem Container-eigenen Speicherbereich persistieren. Shindig wird mit einer Klasse `BasicOAuthStore` ausgeliefert, die aber nur zu Testzwecken genutzt werden soll.

Nutzung von OAuth für die RESTful-API

Zur Nutzung von OAuth beim Zugriff auf den Container über die REST-Schnittstelle muss zusätzlich das Interface `OAuthLookupService` implementiert werden. Hierüber wird geprüft, ob der Server eines Gadgets Zugriffserlaubnis auf Daten eines Benutzers erhält. Für dieses Interface ist in Shindig mit der Klasse `SampleContainerOAuthLookupService` eine Implementierung bereitgestellt, die aber auch nur als Beispiel dient und nicht in produktiven Umgebungen genutzt werden sollte.

3.3 Verwaltung der Applikationen

Mit der Implementierung der OpenSocial-API ist ein Community-System in der Lage, OpenSocial-Applikationen zu betreiben. Neben der rein technischen Anbindung müssen noch weitere Aspekte berücksichtigt werden, die von der OpenSocial-API nicht abgedeckt sind, aber trotzdem wichtig sind zum Betreiben eines OpenSocial-Containers:

- Applikationen installieren und deinstallieren
- Entwickeln von Applikationen
- Positiv- oder Negativlisten
- Zugriffsrechte einschränken

Die Art und Weise, wie diese Aspekte in dem jeweiligen Netzwerk angeboten werden, ist nicht in der OpenSocial-Spezifikation vorgeschrieben. Dies obliegt vollumfänglich dem jeweiligen Netzwerk.

3.3.1 Installationsprozess von Applikationen

Es existieren mehrere Möglichkeiten, Applikationen zu installieren. Die einfachste ist die Bereitstellung eines Applikationsverzeichnis. Dieses Verzeichnis stellt quasi die dem Netzwerk bekannten Applikationen dar. Zur Installation kann der Benutzer die jeweilige Applikation einfach auswählen. Um dem Benutzer mehr Informationen zu Applikationen zu bieten, ist eine Voransicht der Applikation sinnvoll, in der die Hauptfunktionalitäten der Applikation sichtbar sind. Das Verzeichnis kann auch auch mit einem Forum erweitern werden, so das ein Erfahrungsaustausch rund um die jeweiligen Applikationen gefördert wird. In diesem können Benutzer und Entwickler der Applikation Anforderungen an die Applikation besprechen.

Erlaubt ein Netzwerk die Installation von Applikationen, die nicht im eigenen Verzeichnis gelistet sind, so stellt die Eingabe der URL zu der XML-Datei der Applikation die einfachste Art der Installation dar. Beim Installationsprozess werden dann die Metadaten mit Autor, Beschreibung und weiteren Informationen aus den XML-Angaben ausgelesen und übernommen.

Eine weitere Installationsmöglichkeit sind Applikationslisten anderer Benutzer. Durch Navigation auf ein Nutzer-Profil werden dessen installierte Applikationen dargestellt. Mit Auswahl einer Applikation kann diese direkt ausgeführt werden oder in das eigene Profil übernommen werden.

3.3.2 Entwicklung und Veröffentlichung einer Applikation

Einige soziale Netzwerke, insbesondere die populärsten, bieten ihren Nutzern eigene Bereiche an, in denen Anwendungen selbst entwickelt und getestet werden können. Diese Bereiche werden als Sandbox bezeichnet, da innerhalb diesen die Software abgeschirmt vom Rest des Systems entwickelt und getestet werden kann. Der Bereich stellt quasi einen Sandkasten dar, in dem einerseits keine Schäden angerichtet werden können und andererseits die Funktionalität isoliert getestet werden kann.

Nach Fertigstellung kann die Anwendung dann beim jeweiligen sozialen Netzwerk zur Veröffentlichung eingereicht werden, wo sie vorher geprüft werden sollte. Dieser Prozess verläuft bei jedem Netzwerk anders. Der Zweck ist aber überall der gleiche. Die Applikation muss vorher ausgiebig getestet und untersucht werden, um nicht Schadsoftware über diesen Weg in das System einzuschleusen. Wenn für gut befunden, wird die Applikation dann in ein Applikationsverzeichnis aufgenommen, von wo aus sie von allen Benutzern der Plattform installierbar ist.

Zur Editierung des Quellcodes wird im Grunde kein spezieller Editor benötigt. Einige Netzwerke bieten hierfür aber bereits komfortablere Editoren an, die einfaches Syntax-Highlighting unterstützen und über die eine Applikation testweise ausgeführt werden kann.

Mittlerweile ist mit OSDE (OpenSocial Development Environment) ein Eclipse-Plugin bereitgestellt worden, mittels dem Applikationen in Eclipse erstellt werden können [OSDE]. Das besondere an diesem Plugin ist, dass eine Shindig-Version als Server integriert ist und ein Testdatenbestand via integrierter Java H2-Datenbank vorhanden ist. Zur Verwaltung der Daten und des Servers sind diverse Eclipse-Ansichten vorhanden. OpenSocial-Applikationen können somit mit einer populären Entwicklungsumgebung erstellt und getestet werden.

3.3.3 Positiv- oder Negativlisten

Abhängig vom Installationsprozess für OpenSocial-Applikationen können relativ leicht neue Applikationen in das Netzwerk eingebunden werden können. Damit wächst auch die Gefahr, Schadsoftware oder sonstige nicht erwünschte Software in das System einzuschleusen. Um hierauf schnell reagieren zu können, sind Mechanismen notwendig, über die man die Ausführung von Applikationen verhindern kann. Hierzu bieten sich Positiv- oder Negativlisten als Mittel an.

Positivlisten

Eine Steuerung über Positivlisten erlaubt die Ausführung einer Applikation nur, wenn diese innerhalb der Liste enthalten ist. Es ist also für den Benutzer nicht so einfach, eine neue Applikation in das Netzwerk einzufügen, da sie erst durch Aufnahme in die Liste freigegeben werden muss.

Eine Steuerung über diesen Mechanismus eignet sich gut, wenn man die Installation neuer Applikationen stark reglementieren will und z.B. nur Applikationen durch bestimmte Benutzer ins Netzwerk eingebracht werden dürfen.

Negativlisten

Diese Form der Listen werden oft auch als Blacklists bezeichnet und beinhalten die Applikationen, die innerhalb des Containers nicht ausgeführt werden dürfen. D.h., die Installation von neuen Applikationen ist für den Benutzer wenig reglementiert. Bei der Installation wird lediglich überprüft, ob diese Applikation nicht innerhalb dieser Liste gemeldet ist. Es ist sinnvoll, dem Benutzer eine Möglichkeit zu bieten, über die eine Applikation als Schadsoftware oder nicht vertrauenswürdig gemeldet werden kann. Die Applikation kann hierüber nach Prüfung in die Negativliste aufgenommen werden.

Diese Art der Steuerung ist eher für offene Netzwerke geeignet, wo man es dem Benutzer erlauben will, einfach eine neue, dem System noch unbekannte, Applikation zu installieren.

3.3.4 Zugriffsrechte

Mit der OpenSocial-API haben die Applikationen generell die Möglichkeit auf alle Daten, die die Schnittstelle bietetet, entsprechend zuzugreifen. Z.B. kann eine Applikation Einträge in die Aktivitätenliste des Benutzers vornehmen oder auf private Daten des Benutzerprofils zugreifen. Es ist sinnvoll, dem Benutzer verschiedene Einstellungsmöglichkeiten zu Applikationen zu bieten, wo das Verhalten diesbezüglich gesteuert werden kann. Dementsprechend müssen diese Optionen dann in der API-Implementierung berücksichtigt werden.

3.3.5 Speicherbereich für Applikationen

Zur vollständigen Unterstützung der OpenSocial-API muss das Netzwerk einen Speicherbereich anbieten, in dem eine Applikation benutzerbezogen Daten ablegen kann. Die Daten werden immer zu dem angemeldeten Benutzer gespeichert, auch wenn eine Applikation ausgehend von einem fremden Profil ausgeführt wird.

Das grösste Problem hierbei ist der Speicherplatz, der zur Verfügung gestellt wird. Es lässt sich relativ schwer abschätzen, wie viel Speicherplatz eine Applikation benötigt. Von daher ist es sinnvoll, den Speicherplatz von Beginn an zu begrenzen und diesen auch zu überwachen.

3.3.6 Erscheinungsbild einer Applikation innerhalb des Containers

Wo und wie eine Applikation innerhalb eines sozialen Netzwerkes eingebettet wird, ist eine Entscheidung des Herstellers des jeweiligen Netzwerkes. Oft existieren verschiedene Bereiche, in denen Applikationen angezeigt werden. Im Home-Bereich können Applikationen angezeigt werden, die der Benutzer direkt nach dem Einloggen nutzen will. Auch mehrere auf einmal. Werden Applikationen im Profil-Bereich dargestellt, so können andere Benutzer diese auch nutzen, wenn sie ein Profil betrachten. In diesen beiden Ansichten ist es sinnvoll, dem Benutzer Einstellungsmöglichkeiten zu bieten, seine Applikationen nach den eigenen Wünschen anzuordnen. Da im Profil- und Home-Bereich meistens mehrere Applikationen zusammen auf einer Ansicht zusammen dargestellt werden, ist es sinnvoll hierfür eine reduzierte Ansicht zu nutzen. Zur Vollansicht sollte eine eigene Seite zur Verfügung gestellt werden, in der für die Applikation der überwiegende Teil des Bildschirms verfügbar ist.

4 Entwurf Integration in die Fernuni-Umgebung

Die Integration in die Fernuni-Umgebung teilt sich in zwei Bereiche auf. Zum einen muss die fachliche Funktionalität des Community-Systems selbst erweitert werden, so dass der Benutzer eine Applikation auswählen und ausführen kann.

Der zweite Bereich ist die rein technische Integration von Shindig um die eigentliche OpenSocial-Funktionalität anzubieten.

4.1 Erweiterung der fachlichen Funktionalität

Die Anforderungen an das Community-System werden mittels Anwendungsfällen beschrieben. Aus den Anwendungsfällen lassen sich die Domänenklassen ableiten. Die hier angegebenen Diagramme stellen nur die neu hinzugekommenen Elemente wie Anwendungsfälle oder Domänenklassen dar. Die bestehenden Elemente aus der Vorgängerarbeit werden hier nicht mehr aufgeführt.

4.1.1 Anwendungsfälle

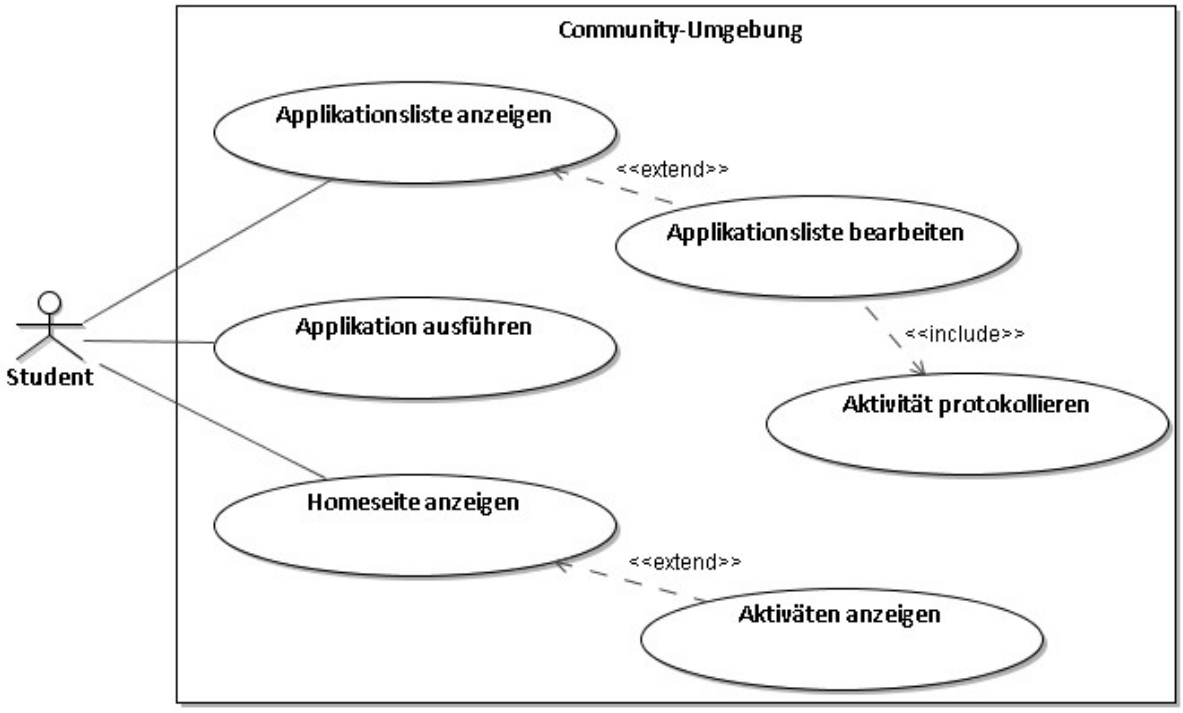

Abbildung 9: Anwendungsfälle der Community-Umgebung

Mittels der Applikationsliste-Funktionalität erhält der Benutzer eine Übersicht seiner bereits installierten Applikationen. Er kann von hier aus über den Anwendungsfall „Applikationsliste bearbeiten“ neue Applikationen in seinem Profil zu installieren, bzw. bereits installierte wieder entfernen. Durch Auswahl einer installierten Applikation aus der Liste kann die Applikation ausgeführt werden. Über den Anwendungsfall „Applikationsliste anzeigen“ können auch die installierten Applikationen eines

anderen Benutzers angezeigt werden. Diese ist erreichbar über die Öffnung des Profils des jeweiligen Benutzers. Eine Applikation kann auch ausgeführt werden, wenn sie nicht im eigenen Profil installiert ist. Falls eine Applikation Daten speichert, werden diese aber immer zu dem eingeloggten Benutzer gespeichert.

Als weitere Funktionalität wird die Möglichkeit angeboten, Aktivitäten für einen Benutzer zu erstellen. Aktivitäten dokumentieren, welche Aktionen ein Benutzer innerhalb des Netzwerkes durchführt. Sie werden nicht aktiv durch den Benutzer erstellt, sondern durch das System als eine Art Protokollierung automatisch erzeugt. Z.B. wird das Installieren einer Applikation in das Benutzerprofil dokumentiert. Der Benutzer erhält auf seinem Startbildschildschirm eine Übersicht der Aktivitäten aus seinem Freundeskreis. Aktivitäten können auch über OpenSocial-Applikationen erstellt werden und sind somit hier auch sichtbar.

4.1.2 Domänenklassen

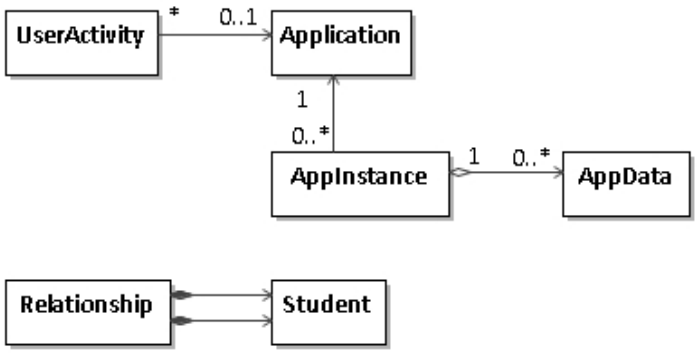

Abbildung 10: Domänenklassen der Community-Umgebung

Aus den Anwendungsfällen können die im Diagramm angegebenen Domänenklassen abgeleitet werden. Eine `Application` beinhaltet die Meta-Information wie Name, Pfad zur XML-Datei zu einer OpenSocial-Applikation. Eine `Application` kann in verschiedenen Profilen installiert werden. Diese Zuordnung wird durch `AppInstance` abgebildet. Speichert eine Applikation Daten innerhalb der Community-Umgebung, so werden diese in `AppData` abgelegt. Hier können mehrere Datensätze zu einer Applikationsinstanz gespeichert werden. UserAcitiviy beinhaltet Akvitäten zu einem Benutzer. Eine Aktivtät kann einer Applikation zugeordnet sein. Freundesbeziehungen werden über `Relationship` abgebildet. Hier sind zwei Beziehungen zu `Student` vorhanden. Eine für den Von-Teil der Beziehung und eine für den zugeordneten Studenten.

4.1.3 Komponenten

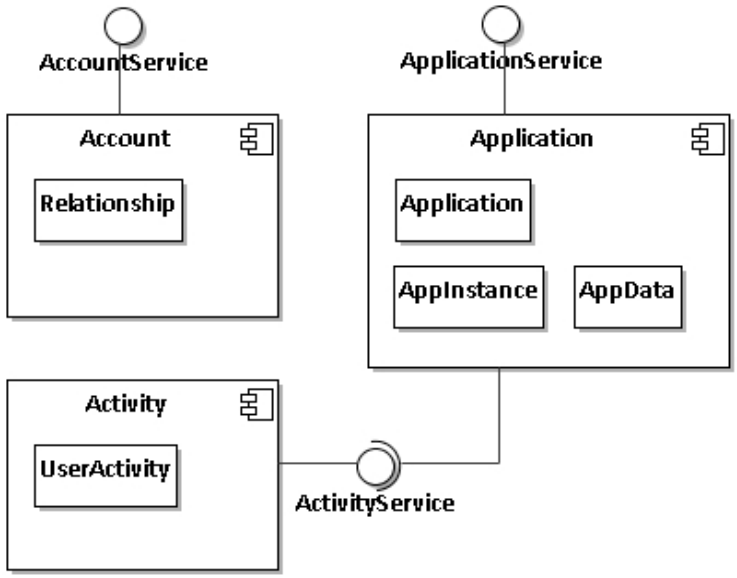

Abbildung 11: Komponenten der Community-Umgebung

Für die Verwaltung und Ausführung der Applikationen steht die `Application`-Komponente zur Verfügung. Sie bietet über die Schnittstelle `ApplicationService` verschiedene Methoden zum Verwaltung der Applikationen, Lesen der Applikationsliste zu einem Benutzer und Ausführen einer Applikation an. Beim Hinzufügen einer Applikation wird durch den `ApplicationService` gleichzeitig eine Aktivität protokolliert.

Die `Activity`-Komponente besitzt eine Schnittstelle `ActivityService` zum Protokollieren und Lesen von Aktivitäten an.

Die `Account`-Komponente ist um die `Relationship`-Funktionalität erweitert worden.

4.1.4 Benutzeroberfläche

Der Aufbau und die Gestaltung der Benutzeroberfläche ist gegenüber dem Vorgängersystem gleich geblieben. Es sind lediglich neue Elemente hinzugekommen.

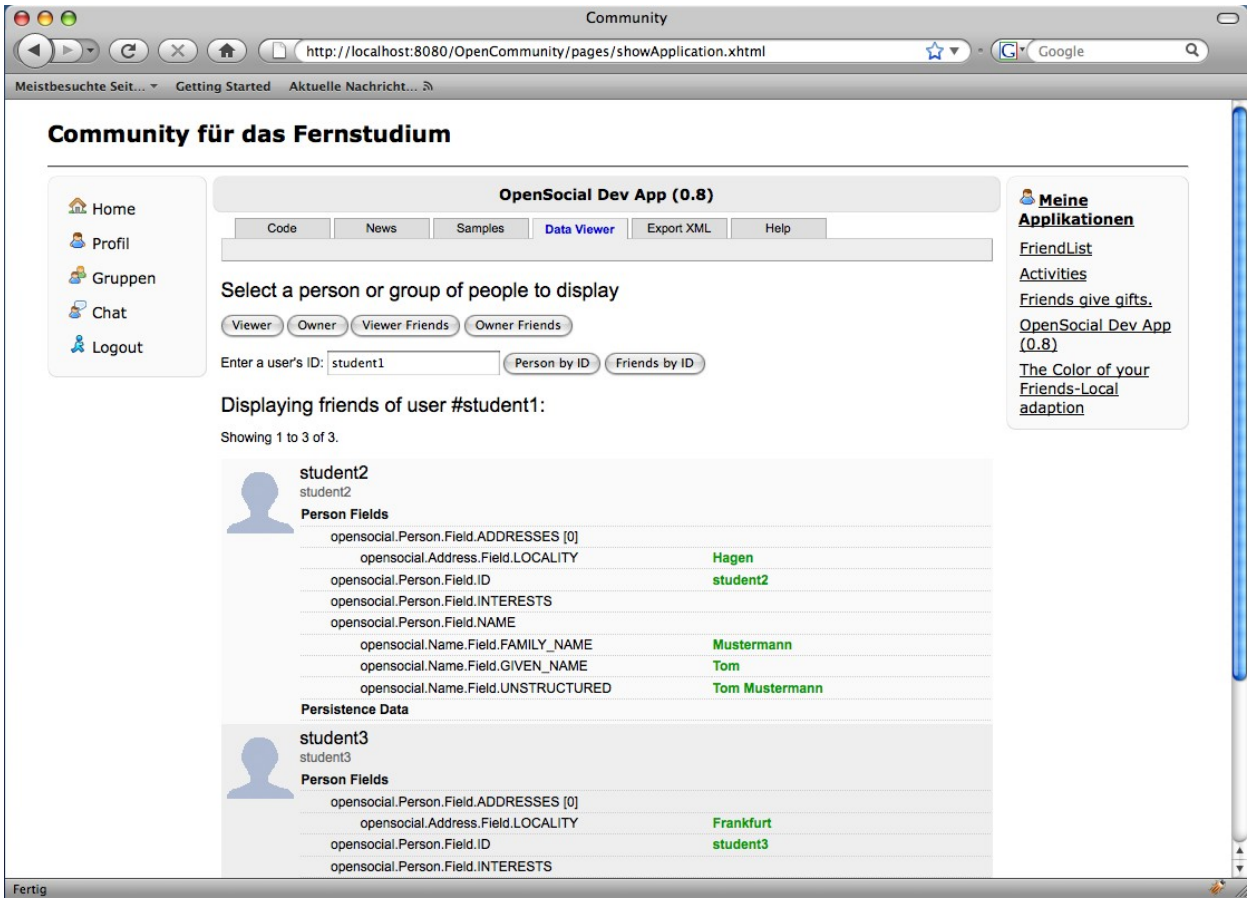

Abbildung 12: Screenshot der Community-Umgebung

Diese Ansicht zeigt die Ausführung der Applikation „OpenSocial Dev App“, über die Quell-Code für OpenSocial-Applikationen getestet werden kann. Auf der dargestellten Seite können Personendaten dargestellt werden. Die Applikation wird im Content-Bereich der Communityumgebung dargestellt.

Die installierten Applikationen eines Benutzers sind direkt im Infobereich aufgelistet. Abhängig davon, ob sich der Benutzer in seinem oder einem fremden Profilbereich befindet, werden die installierten Applikationen der jeweiligen Person aufgelistet.

Im Begrüssungsbildschirm erhält der Benutzer eine Übersicht der letzten Aktivitäten seiner Freunde.

4.2 Shindig-Integration

Die Shindig-Integration in die Fernuni-Community teilt sich ebenfalls in zwei primäre Bereiche auf. Die Datenintegration zum Zugriff der Shindig-Services auf die Community-DB und die GUI-Integration zur Anzeige der Gadgets in den Web-Seiten der Community-Umgebung. Zusätzlich kommen noch weitere Aspekte wie Verteilung der Applikationen und Versions-Upgrade hinzu.

4.2.1 Daten-Integration

Der Zugriff auf die Datenbasis der Community erfolgt über die Integration der Domänen-Klassen und der

entsprechenden Hibernate-Mappings in die Shindig-Services.

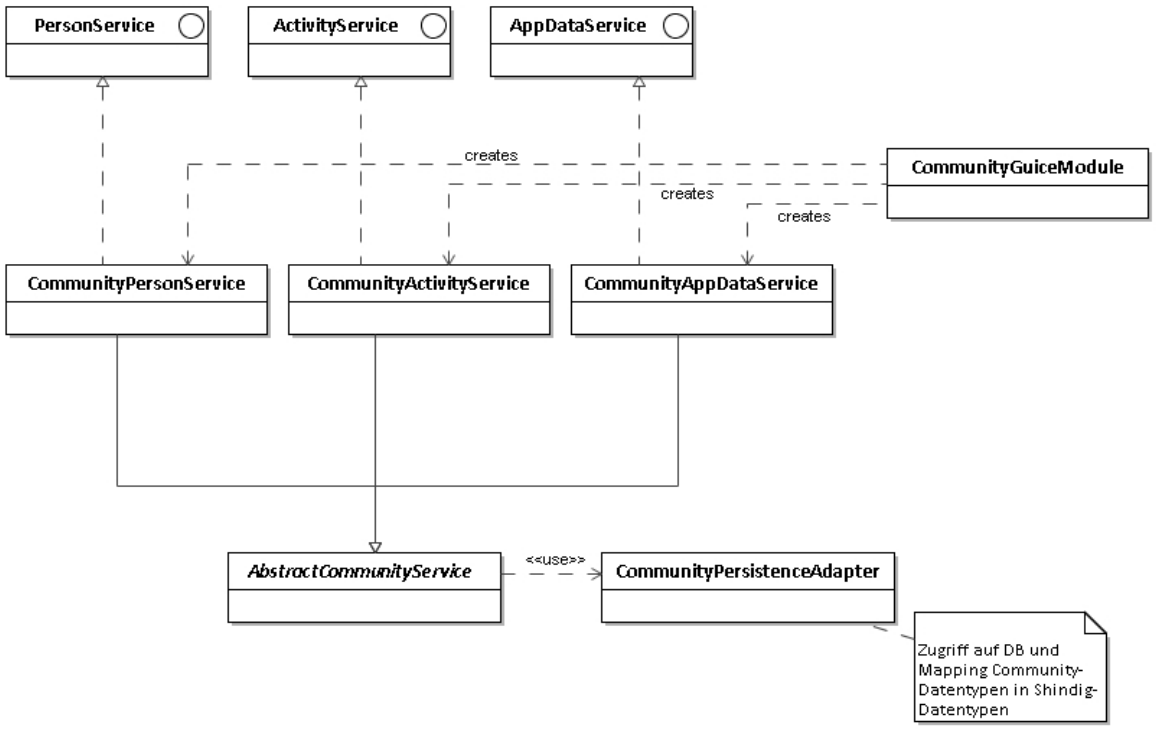

Abbildung 13: Implementierung der Shindig-Service-Interfaces

Je Shindig-Interface wird eine Integrations-Klasse, z.B. `CommunityPersonService` erstellt, die die Methoden des entsprechenden Interfaces implementiert. Der Zugriff auf die Domänen-Klassen erfolgt über `CommunityPersistenceAdapter`. Diese Klasse stellt die Verbindung zwischen Community-Entitäten und Shindig-Services dar. Zur ihren Aufgaben zählt der Datenbankzugriff und das Mapping zwischen den Domänen-Klassen der Community und den Shindig-Klassen.

Folgendes Listing zeigt die Implementierung der Methode `getPerson` in der Klasse `CommunityPersonService`:

```
public Future<Person> getPerson(UserId id, Set<String> fields, SecurityToken token)
throws SocialSpiException {
  initDB();
  openSession();

  if (id != null) {
    try {
      Person currentPerson = db.findPerson(id.getUserId(token));
      if (currentPerson != null) {
        return ImmediateFuture.newInstance(currentPerson);
      }
    }
    catch (Exception e) {
      throw new SocialSpiException(
        ResponseError.INTERNAL_ERROR, "Internal error getting person: " + e);
    }
    finally {
      closeSession();
```

```
    }
  }

  throw new SocialSpiException(ResponseError.BAD_REQUEST, "Person not found");
}
```

Diese Methode nutzt zum Lesen der Personendaten die Methode `findPerson`, wo mittels Hibernate der Zugriff auf die Community-DB durchgeführt wird:

```
public Person findPerson(String id) {
  Query query = session.createQuery(PERSON_BY_USER_ID);
  query.setString("username", id);
  Student student = (Student)query.uniqueResult();

  if (student == null) {
    return null;
  }
  else {
    Person person = mapToPerson(student);
    return person;
  }
}
```

Registriert werden die eigenen Implementierungen über das Guice-Modul `CommunityGuiceModule`.

```
/**
 * Configuration Module to integrate the community data into Shindig.
 */
public class CommunityGuiceModule extends AbstractModule {

  @Override
  protected void configure() {
    bind(PersonService.class).to(CommunityPersonService.class);
    bind(AppDataService.class).to(CommunityAppDataService.class);
    bind(ActivityService.class).to(CommunityActivityService.class);

    ...
  }
}
```

Zur Aktivierung des Guice-Moduls im Shindig-Server wird der Name der Guice-Klasse innerhalb der Konfigurationsdatei `web.xml` angegeben.

```
<context-param>
  <param-name>guice-modules</param-name>
  <param-value>
    de.fernuni.shindigintegration.config.CommunityGuiceModule:
    ...
  </param-value>
...
```

Der Shindig-Server muss schlussendlich mit der neu bereitgestellten Funktionalität neu gebaut und verteilt werden.

4.2.2 GUI-Integration

Innerhalb der Community-Web-Anwendung werden OpenSocial-Applikationen in der Seite showApplication.xhtml angezeigt. Diese beinhaltet das bereits zuvor erwähnte iFrame-Tag, innerhalb dessen

die Applikation angezeigt werden soll.

```
<iframe id="remote_iframe_0" src="#{userApplicationBean.iframeUrl}"
  scrolling="auto" class="gadgets-gadget" style="display: block;"
  frameborder="no" height="400" width="100%" name="remote_iframe_0">
</iframe>
```

Über die ApplicationController-Klasse wird die benötigte URL zusammgestellt und der Webseite zur Verfügung gestellt.

```
private void generateIframeUrl() throws UnsupportedEncodingException {
  // generate SecurityToken
  String viewerId = viewer;
  String ownerId = owner;
  String domain = "default";
  long moduleId = 0;
  String separator = URLEncoder.encode(":", "UTF-8");
  String urlEncoded = encodeUrl(url);
  StringBuilder out = new StringBuilder();
  out.append(URLEncoder.encode(ownerId, "UTF-8")).append(separator)
      .append(URLEncoder.encode(viewerId, "UTF-8")).append(separator)
      .append(Long.toString(appid)).append(separator)
      .append(URLEncoder.encode(domain, "UTF-8")).append(separator)
      .append(urlEncoded).append(separator)
      .append(Long.toString(moduleId));

  securityToken = out.toString();

  // generate iframeUrl
  String serverBase = "http://localhost:8080/gadgets/";
  String container = domain;
  String parent = URLEncoder.encode("http://localhost:8080", "UTF-8");

  iframeUrl = serverBase + "ifr?" + "container=" + container + "&mid=" + moduleId
      + "&nocache=" + "1" + "&country=" + "ALL" + "&lang=" + "ALL"
      + "&view=" + "canvas" + "&parent=" + parent + "&debug="
      + "1" + "&st=" + securityToken + "&url=" + URLEncoder.encode(url, "UTF-8");
  iframeUrl += "#rpctoken=" + getRpcTokenRandom();
}
```

Wie anhand des Code-Beispiels zu sehen, wird innerhalb der Implementierung der Community-Umgebung der SecurityToken noch unverschlüsselt generiert. Zur Generierung eines sicheren Tokens muss die von Shindig zur Verfügung gestellte Klasse `BlobCrypterSecurityToken` verwendet werden. Hierzu sind folgende Einträge in in der Shindig-Konfigurationsdatei /config/container.js notwendig:

```
"gadgets.securityTokenType" : "secure",
"gadgets.securityTokenKeyFile" : "/path/to/key/file.txt",
```

Als Schlüsselwert zur Verschlüsselung des Tokens wird die erste Zeile der unter `gadgets.securityTokenKeyFile` angegebenen Datei gelesen.

Während der Request-Verarbeitung in Shindig wird dieser Token dann mittels `BlobCrypterSecurityDecoder` entschlüsselt und die Werte ausgelesen.

4.2.3 Verteilung

Mit der Integration von Shindig sind für den Betrieb der Communityumgebung zwei Server-Anwendungen notwendig. Weiterhin die Community-Anwendung und zusätzlich der Shindig-Server zum Bereitstellen der

OpenSocial-Funktionalität. Beide Anwendungen werden als Web-Applikation in jeweils eine War-Datei gepackt. Grundsätzlich sollte dabei die Shindig-Anwendung auf einem anderen Web-Server installiert werden als die Community-Anwendung. Durch diese physische Trennung können Sicherheitsrisiken vermieden werden und es sind mehr Möglichkeiten für eine Skalierung des Shindig-Servers gegeben. In der prototypischen Umsetzung zu dieser Arbeit werden aus Einfachheitsgründen beide Anwendungen auf dem gleichen Server betrieben.

Shindig ist standardmässig so konfiguriert, dass die Datei Shindig.war unter dem ROOT-Kontext in Tomcat installiert werden muss. Will man eine andere Domäne, so muss dies in der Shindig-Konfigurationsdatei /java/common/conf/shindig.properties angepasst werden.

Standard-Eintrag zum Betieb von Shindig unter dem ROOT-Kontext:

```
shindig.content-rewrite.concat-url=http://localhost:8080/gadgets/concat?
```

Alternativer Eintrag zum Betrieb unter einer anderen Domäne:

```
shindig.content-rewrite.concat-url=/shindig/gadgets/concat?
```

4.2.4 Shindig Versions-Upgrade

Bei der Integration ist zu beachten, dass Shindig momentan als Code-Paket mittels Subversion von [ShindigCode] geladen werden muss und über Maven gebaut werden muss. Zur Integration sind, wie zuvor aufgezeigt, die Code-Anpassungen und eigene Anpassungen an Konfigurationsdateien notwendig. Will man eine neue Version von Shindig benutzen, muss diese neu geladen und entsprechend wieder angepasst werden. Von daher ist es erforderlich, die Anpassungsstellen möglichst gering zu halten. Die eigenen Code-Anpassungen sollten als isolierte Jar-Datei zur Verfügung gestellt werden und via Maven integriert werden, so dass an dem mitgelieferten Shindig-Code keine Anpassungen notwendig sind. Trotzdem ist zu beachten, dass die Integration einer neueren Version von Shindig immer mit entsprechendem Aufwand in der Konfiguration und Testaufwand verbunden ist.

5 Diskussion

In diesem Kapitel werden zunächst die Einsatzmöglichkeiten der OpenSocial-API aufgeführt. Anschliessend wird eine Bewertung dieses Ansatzes durchgeführt, in dem auch auf Einschränkungen hingewiesen wird.

5.1 Einsatzmöglichkeiten

Wie schon in der Einleitung angedeutet, ergeben sich mit der Implementierung der OpenSocial-API vielfältige Erweiterungsmöglichkeiten für die Community-Umgebung, die im folgenden näher aufgezeigt werden.

5.1.1 Funktionserweiterungen

Die Community-Umgebung kann mittels OpenSocial-Applikationen zur Laufzeit um noch nicht vorhandene Funktionalität erweitert werden. Dies können relativ einfache Anwendungen sein, die z.B. eine einfache Verwaltung der Aktivitätenhistorie eines Benutzers ermöglichen oder eine erweiterte Ansicht der Freunde eines Benutzers anzeigen.

Durch die Möglichkeit, entfernte Ressourcen innerhalb einer solchen Applikation aufzurufen, können aber auch komplexere Anwendungen erstellt werden, wie z.B. eLearning-Module oder Übersetzungsanwendungen. Hier kann man sich verschiedene Szenarien vorstellen. Steht ein Übersetzungsdienst als WebService zur Verfügung, kann man unter Verwendung von HTML und JavaScript eine Oberfläche erstellen, die diesen Dienst direkt nutzt. Ein anderer Ansatz ist die Erstellung der Applikation direkt mit komplexeren Web-Applikations-Mechanismen, wie in [Laird2008] gezeigt.

5.1.2 Zugriff von anderen Endgeräten

Eine Unterstützung der Kommunikation innerhalb des sozialen Netzwerkes sollte nicht nur auf Situationen beschränkt sein, in denen ein internetfähiger PC vorhanden ist. Gerade durch die Bereitstellung eines Zugriffs von mobilen Endgeräten aus kann die „Reichweite“ des Systems enorm erweitert werden. So gibt es viele Situationen, wo es besser ist, eine Information gleich weiterzugeben, und nicht erst, wenn man wieder am Rechner sitzt.

Mit Verwendung der OpenSocial-RESTful API und der OpenSocial ClientLibraries kann diese Form der Kommunikation unterstützt werden. Es können Anwendungen erstellt werden, die es ermöglichen, auf Daten der Community von weiteren Endgeräten aus zuzugreifen. Man kann sich z.B. eine Applikation vorstellen, mit der man von einem mobilen Endgerät aus eine Sicht auf sein Community-Profil erlangt und somit zumindest einen Teil der Funktionalität des Netzwerkes über diese Schnittstelle anwenden kann. Die ClientLibraries stehen für verschiedene Sprachen und Umgebungen zur Verfügung, so dass hier eine hohe Bandbreite an Systemen und Plattformen unterstützt werden kann.

5.1.3 Interoperabilität mit anderen Services

Eine Steigerung der Interoperabilität mit anderen Netzwerken und Anwendungen ergibt sich aber erst, wenn es möglich ist, aus dem eigenen Netzwerk heraus mit anderen Netzwerken oder Applikationen zu

kommunizieren und von der Gegenseite aus auf die eigenen Ressourcen zuzugreifen.

Hierbei lasen sich verschiedene Ansätze unterscheiden.

- Es existieren verschiedene populäre Web 2.0-Applikationen, die sich unter Verwendung von OpenSocial-Techniken mit OpenSocial-Containern integrieren lassen.
- Erstellung von Offline-Diensten zur Datenbearbeitung.

Diese Art der Interoperabilität kann am besten unter Verwendung der OpenSocial RESTful API erreicht werden, die es dem Server der zu integrierenden Applikation ermöglicht, auf Daten des OpenSocial-Containers zuzugreifen. Hierbei werden gemäss der Spezifikation nur HTTP als Protokoll und JSON als Datenformat benötigt.

Interoperabilität von Web-Applikationen mit OpenSocial-Containern

Eine bestehende Web-Anwendung kann mit der RESTful-API Daten aus dem Container lesen, aber auch Daten schreiben und somit eine Integration mit dem Container erreichen. Damit kann die Kernfunktionalität dieser Anwendung durch Zugriff auf Personendaten (z.B. Benutzer des Netzwerkes) und der Möglichkeit Applikationsdaten innerhalb des Containers zu speichern, mit einem sozialen Netzwerk integriert werden.

Ein Beispiel einer solchen Anwendung wäre ein Online Foto-Druck-Service, bei der der Benutzer nicht mehr seine Fotos hochladen muss, sondern angegeben kann, von welchem System aus die Fotos geladen werden sollen. Der Service erstellt dann eine Verbindung zu dem Community-System und kann via REST auf die Fotoalben des Benutzers zugreifen. In [OAuthBegGuide2007] ist ein solcher Ablauf zur Erläuterung von OAuth aufgezeigt. Ein solcher Service muss nicht unbedingt mit OpenSocial-Mitteln erstellt werden. Unterstützt dieser Service aber die OpenSocial RESTful-API, so kann er mit dieser Schnittstelle sehr viele soziale Netzwerke anbinden.

Die Applikation kann aber auch unter Verwendung des Gadget XML als OpenSocial Gadget implementiert werden und mittels Remote-Zugriffen auf Funktionalitäten einer externen Anwendung zugreifen. Mit dieser Art der Integration kann eine Anwendung nahtlos in einem Netzwerk dargestellt werden.

Offline-Dienste zur Datenbearbeitung

Die RESTful-API kann auch zur Durchführung von Offline-Verarbeitungen, wie z.B. einem periodischen Update von Personen- oder Applikationsdaten genutzt werden.

Mit einem solchem Mechanismus können Informationen von verschiedenen sozialen Netzwerken miteinander synchronisiert werden, vorausgesetzt, ein Netzwerk lässt diesen Informationsaustausch zu. Denkbar sind hier z.B. Foren oder übergreifende Newsgroups, die von verschiedenen sozialen Netzwerken aus angesprochen werden können. Benutzer könnten in „ihrem“ Netzwerk angemeldet bleiben, hätten aber trotzdem die die Möglichkeit, Informationen mit Benutzer anderer Netzwerke auszutauschen.

5.2 Bewertung

Soziale Applikationen stellen noch ein recht junges Anwendungsgebiet dar. Momentan sind die meisten der verfügbaren Applikationen in den Bereichen Spiele und Unterhaltung zu finden. Viele dieser Applikationen sind als eher unnützlich zu kennzeichnen und verfolgen keinen bestimmten Zweck. Dieser Umstand wird

auch von vielen Kritikern dieses Anwendungsgebietes als Hauptkritikpunkt aufgeführt, sollte aber nicht darüber hinweg täuschen, dass mit dieser Technologie nicht auch sinnvolle Anwendungen erstellt werden können. Z.B. hat die Uni Osnabrück an der diesjährigen CeBIT eine soziale Applikation zur Vorlesungsaufzeichnung und Wiedergabe dieser als Video-Stream im Browser vorgestellt [VirtPresenter]. Diese Applikation steht momentan nur für Facebook zur Verfügung, eine Erweiterung für OpenSocial ist aber geplant. Mit solchen Anwendungsmöglichkeiten stellt die OpenSocial-API eine sehr gute Grundlage dar, um Inhalte für den eLearning-Bereich bereitzustellen, insbesondere dadurch, dass die Anwendungen durch Nutzung einer Standardschnittstelle für viele Netzwerke zur Verfügung gestellt werden können.

Das vorangegangene Kapitel zu den Einsatzmöglichkeiten der OpenSocial-API zeigt die vielfältigen Möglichkeiten auf, die sich mit Nutzung der RESTful-API ergeben. Es kann von unterschiedlichen Endgeräten, wie z.B. mobilen Geräten, auf das soziale Netzwerk zugegriffen werden. Des weiteren ist die Kommunikation mit anderen Applikationen und sozialen Netzwerken möglich. Mit diesen Mitteln kann die Interoperabilität eines sozialen Netzwerkes wesentlich gesteigert werden. Einschränkungen sind hierbei nur durch den Umfang der API gegeben, da nicht auf alle Daten eines sozialen Netzwerkes über die API zugegriffen werden kann.

Der Grundgedanke von OpenSocial-Applikationen ist die einmalige Erstellung einer Applikation oder eines Dienstes und Nutzung dieser in vielen verschiedenen Netzwerken. Da sich der Standard aktiv in Entwicklung befindet, sind bereits mehrere Versionen vorhanden und nicht alle Netzwerke unterstützen den neuesten Standard. Dieser Umstand erschwert die Zielsetzung von OpenSocial, Applikationen in allen Containern betreiben zu können. Hinzu kommt noch, dass es für einen Container möglich ist, eigene Attribute zu definieren, die von Applikationen genutzt werden können, was wiederum den Nachteil bringt, dass eine solche Applikation unter Umständen in einem anderen Netzwerk nicht lauffähig ist. Von daher sollte man sich so nah wie möglich am Standard orientieren, sei es bei der Entwicklung des Containers sowie auch bei der Erstellung von Applikationen. Hier stellt sich aber die Frage, wie kann ein soziales Netzwerk sich von anderen unterscheiden, wenn der Standard nicht beeinflusst werden sollte. Wie schon im Kapitel 3.3 Verwaltung der Applikationen aufgezeigt, definiert OpenSocial nicht, wie ein Netzwerk die Applikationen verwalten soll. Hier können sich soziale Netzwerke durch unterschiedliche Konzepte voneinander unterscheiden.

Neben den vielen Möglichkeiten, die einem sozialen Netzwerk mit Implementierung der OpenSocial-API zur Verfügung stehen, birgt der Ansatz auch Sicherheitsrisiken, die im folgenden aufgezeigt werden sollen.

Eine OpenSocial-Applikation kann jegliche Art von Web-Funktionalität beinhalten. Somit ist es auch möglich, Schadsoftware oder Trojaner in das Netzwerk einzuschleusen. Wissenschaftler des Institute of Computer Science (ICS) in Griechenland und des Institute for Infocomm Research in Singapur haben im Rahmen einer Forschungsarbeit eine Facebook-Applikation entwickelt, die als Botnet innerhalb des Facebook-Netzwerkes installiert wurde. Das Verhalten wurde untersucht und die Ergebnisse unter [Athan2008] veröffentlicht.

Mittels Botnets können “Denial of Service“-Attacken auf Internetdienste durchgeführt werden. Hierzu müssen möglichst viele Benutzer eine Seite aufrufen, die Links auf eine anzugreifende Seite enthält. Jedes mal, wenn ein Benutzer die Seite besucht, wird im Hintergrund der entsprechende Link ausgeführt und die Daten von dem anzugreifenden Server geladen. Wird die Seite von möglichst vielen Benutzern aufgerufen,

so wird dementsprechend viel Rechenkapazität des Ziel-Servers verbraucht, was zu einem Ausfall oder langsamen Antwortzeitverhalten führen kann.

In dem Versuch wurde eine Applikation namens “Photo of the Day“ erstellt, die dem Benutzer über einen Klick ein Zufalls-Foto von National Geographic präsentierte. Die Applikation wurde so erweitert, dass bei jedem Klick ein spezieller Code im Hintergrund ausgeführt wurde, der einen HTTP-Request zu dem anzugreifendenen Server ausführte, und somit Daten von diesem Server geladen wurden. Die Applikation verbreitete sich so rasch innerhalb des Facebook-Netzwerkes, dass schon nach wenigen Tagen über 300 Requests pro Stunde beobachtet werden konnten.

Der Versuch zeigt, dass Facebook- oder OpenSocial-Applikationen sich besonders gut für solche Angriffe eignen, da der potentielle Nutzerkreis sehr hoch ist und die Applikationen innerhalb des Netzwerkes sehr schnell auf viele Benutzer verbreitet werden können.

Mit Unterstützung der OpenSocial-API sollten diese Gefahren in Betracht gezogen werden und Möglichkeiten untersucht werden, solche Angriffe zu unterbinden. Z.B. könnte die Nutzung bestimmter HTML-Tags innerhalb von Applikationen unterbunden werden. Eine andere Möglichkeit wäre die Einführung von Constraints, um zu verhindern, dass eine Applikation andere Internet-Adressen als die angegebenen kontaktiert. Auf jeden Fall sollte es Stellen geben, die eine Applikation vor Veröffentlichung auf Sicherheitsrisiken untersuchen.

Die Unterstützung von OpenSocial bringt auch für den Software-Betrieb neue Herausforderungen mit sich. Lässt das Netzwerk zu, dass die Applikationen oder Teile dieser auf fremden Servern installiert sind, ist der Schutz des kompletten Systems vor Ausfallsicherheit nicht mehr einfach möglich, da auf die Verfügbarkeit der fremden Systeme kein Einfluss genommen werden kann. Von daher sollten Applikationen, die hoch verfügbar sein sollen, im einen Umfeld betrieben werden, das vom Betreiber des Netwerkes gesteuert werden kann.

6 Schlussbetrachtung

6.1 Zusammenfassung

Ziel dieser Arbeit war die Implementierung der OpenSocial-API in der Fernuni-Community-Umgebung und damit die Steigerung der Interoperabilität dieses sozialen Netzwerkes.

Zur Umsetzung dieser Interoperabilität existieren mehrere Ansätze wie z.B. der Aufbau des Netzwerkes mit dem Cobricks-Framework oder die Definition einer eigenen Schnittstelle, die mittels WebServices zur Verfügung gestellt wird. Die OpenSocial-API wird mittlerweile von sehr vielen Netzwerken unterstützt und lässt sich mit relativ geringem Aufwand in ein bestehendes System integrieren. Für diese Arbeit wurde der OpenSocial-Ansatz gewählt. Mit Integration dieser API kann die Fernuni-Community zur Laufzeit um Applikationen von Drittherstellern erweitert werden. Zudem ist es möglich, auf die Community-Daten via RESTful-Schnittstelle von anderen Diensten aus zuzugreifen.

In Kapitel 2 wurde der Begriff soziale Applikation anhand einem Beispiel näher erläutert. Nachfolgend wurden die Konzepte der OpenSocial-API im Einzelnen dargestellt. Mittels der OpenSocial JavaScript-API können soziale Applikationen erstellt werden, die direkt innerhalb eines sozialen Netzwerkes betrieben werden können. Neben dieser Möglichkeit, soziale Applikationen zu erstellen, können mit der OpenSocial RESTful-API weitere Dienste erstellt werden, die auf die Daten des Netzwerkes zugreifen.

Um als Plattform für soziale Applikationen zu dienen, muss ein soziales Netzwerk an verschiedenen Stellen erweitert werden. Die entsprechend notwendigen Arbeiten wurden in Kapitel 3 vorgestellt. Die Integration teilt sich hauptsächlich in zwei Bereiche auf. Die Integration von Shindig als OpenSocial- und Gadget-Server sowie die Anzeige der Applikationen in den eigenen Web-Seiten. Über Shindig wird die Anbindung an die Datenbasis des jeweiligen sozialen Netzwerkes und die Darstellung der Gadgets durchgeführt. Die Daten-Anbindung ist individuell je Netzwerk zu implementieren. Zur Anzeige der OpenSocial-Applikationen in den eigenen Seiten müssen iFrames genutzt werden. Diese werden mit einer URL auf den Gadget-Server aufgerufen.

Neben diesen technischen Aspekten muss ein soziales Netzwerk noch Funktionalitäten bereitstellen, um die installierten sozialen Applikationen zu verwalten. Dieser Bereich wird von der API nicht vorgegeben und muss individuell je Netzwerk implementiert werden. Nichtsdestotrotz ist dieser Teil nicht unerheblich, da hier die Strategie definiert werden muss, wie ein Netzwerk mit sozialen Applikationen umgeht. Hier reicht die Bandbreite von eher offenen Ansätzen, der es dem Benutzer erlauben, neue Applikationen nach eigenem Ermessen zu installieren, oder eher geschlossene Ansätze, wo die erlaubten Applikationen streng durch das Netzwerk vorgegeben sind.

Anschliessend wurde in Kapitel 4 dargelegt, wie die OpenSocial-API in die Fernuni-Community integriert wurde. Es wurden zunächst die fachlichen Erweiterungen an der Community-Umgebungen aufgezeigt und nachfolgend auf die technischen Implementierungen eingegangen.

In Kapitel 5 wurden die verschiedenen Einsatzmöglichkeiten, die sich für ein soziales Netzwerk mit Implementierung der OpenSocial-API ergeben, aufgezeigt und anschliessend der Ansatz diskutiert.

6.2 Ausblick

In dieser Arbeit wurden die OpenSocial-API und der Umgang mit OpenSocial-Applikationen nur rudimentär implementiert. In Nachfolge-Arbeiten könnte dieser Bereich weiter ausgebaut werden. Folgende Punkte wären hierzu nennen:

- Der Installationsprozess für neue Applikationen ist in dieser Arbeit nur durch Angabe der URL zu der XML-Datei umgesetzt worden. Es könnte dem Benutzer zusätzlich ein Verzeichnis mit verfügbaren Applikationen angezeigt werden, die mittels einer Vorschau-Funktion näher betrachtet werden können. Mit Bestätigung wird die Applikation in die Liste der eigenen Applikationen übernommen. Je nach Strategie, die unterstützt werden soll, sollte die direkte Eingabe der URL unterbunden werden und nur ein Applikationsverzeichnis angeboten werden.
- Momentan werden die Applikationen auf einer eigenen Seite zur Ausführung dargestellt. Für den Benutzer wäre es komfortabel, wenn zusätzlich eine oder mehrere seiner Applikationen auf seiner Home-Seite angezeigt werden. Eine solche Applikation liefert auf dieser Ansicht z.B. einen schnellen Überblick. Bei einer Forums-Applikation könnten dies die letzten 5 Forums-Einträge sein. Es müsste einstellbar sein, welche der installierten Applikationen dort angezeigt werden.
- Des weiteren ist es sinnvoll, dem Benutzer Einstellungsmöglichkeiten zur weiteren Konfiguration seiner Applikationen zu geben, wie z.B. die Angabe, ob eine Applikation auf die privaten Daten des Benutzers zugreifen darf oder nicht. Ebenso existieren für den Administrator der Fernuni-Community in dieser Version noch keine Einstellungsmöglichkeiten, um das Verhalten der Applikation zu steuern. Hier sei als Beispiel nur die Begrenzung des Speicherbereiches für Applikationen erwähnt. Um diese durchzuführen wäre es notwendig, spezielle Ansichten für den Administrator bereitzustellen, in denen verschiedene Kennzahlen, wie Speicherbedarf, Anzahl Installationen und Informationen über die Gebrauchshäufigkeit von Applikationen dargestellt werden.
- In der jetzigen Implementierung ist die OAuth-Unterstützung nicht aktiviert, was Einschränkungen in der Kommunikation mit externen Services zur Folge hat. Hierzu müsste die Shindig-Anbindung gemäss den Angaben bei der Shindig-Integration erweitert werden, so dass es möglich ist, Schlüsselwerte je Applikation zu verwalten.

Diese Arbeit basiert auf der OpenSocial-API v0.8.1. Mittlerweile ist die Spezifikation in der Version 0.9 [OpenSocialSpec09] veröffentlicht und wird auch schon in Shindig unterstützt. Diese Version bietet unter anderem folgende Neuerungen an:

- Unterstützung von Nachrichten. D.h., es ist möglich über die API Nachrichten innerhalb des Netzwerkes zu versenden.
- Unterstützung von Fotoalben. Private Fotoalben, die in einem sozialen Netzwerk hinterlegt sind, können über die API angesprochen werden.

Hier ist zumindest die Umsetzung der Nachrichten-API sinnvoll, da innerhalb der Fernuni-Community bereits Nachrichten versendet werden können. Eine Umsetzung von Fotoalben in der Community-Anwendung ist bisher nicht vorgesehen. In der Shindig-Anbindung müssen die entsprechenden Methoden

aber trotzdem implementiert werden und eine Fehlermeldung liefern, dass der Service nicht unterstützt wird.

Neben diesen Verbesserungen in der neuen Version der Spezifikation gibt es aber noch weitere Punkte, die in eine zukünftige OpenSocial-Version einfliessen könnten.

In Social Networks ist die Möglichkeit, Gruppen zu bilden ein zentrales und bewährtes Konzept. Die OpenSocial-API kennt bis dato keine Gruppen. Es wäre wünschenswert, die Personendaten-Funktionen so zu erweitern, dass die Mitglieder einer Gruppe selektiert werden können. Man könnte auf dieser Basis Applikationen im Kontext einer Gruppe installieren. Die Applikation hätte somit Zugriff auf alle Mitglieder der Gruppe.

Die OpenSocial-API zeigt sehr gut, wie die Gadgets-API von Google für spezielle Bereiche erweitert werden kann. Es ist denkbar, diese Basis für die verschiedensten Bereiche zu erweitern, um damit Web-Applikationen zu erstellen.

Ebay hat z.B. Gadgets als Grundlage für ihre „Selling Manager-Applikationen“ verwendet, wo man es den Benutzern erlauben will, eigene Applikationen für die Ebay-Plattform bereitzustellen [Ebay]. Dabei soll Ebay nicht als Social Network erweitert werden. Es ist vielmehr das Ziel zusätzliche Tools für den Verkaufsprozess bereitzustellen und diesen damit besser zu unterstützen.

6.3 Fazit

Durch diese Arbeit ist aufgezeigt worden, wie die Fernuni-Community durch Implementierung der OpenSocial-API um wichtige Funktionalitäten bzgl. Interoperabilität erweitert werden kann. Die Implementierung ist hier nur rudimentär umgesetzt und stellt eine Basis für zukünftige Erweiterungen dar.

Abschliessend lässt sich sagen, dass die OpenSocial-API einen sehr vielversprechenden Ansatz für diesen Zweck darstellt. Durch Unterstützung dieser API ergeben sich für ein soziales Netzwerk viele neue Möglichkeiten bzgl. Erweiterung der Funktionalität und Kommunikation mit anderen Systemen. Vor einer Implementierung sollte aber unbedingt eine Strategieüberlegung stattfinden, in der festgelegt wird, in welchem Umfang das System geöffnet wird.

7 Literaturverzeichnis

[Koch2003] Koch, Michael, „Community-Unterstützungssysteme - Architektur und Interoperabilität“, http://www.communixx.de/files/Koch2003d.pdf, TU München, Dezember 2003

[Cobricks] Cobricks-Homepabe, http://www.cobricks.de/, 06.04.2009

[Fuhrmann2008] Fuhrmann, Antje, „Communityumgebung für das Fernstudium“, Masterarbeit, FernUni Hagen, 2008

[Genise2009] Genise, Marco, „Erweiterte Communityumgebung für das Fernstudium“, Diplomarbeit, FernUni Hagen, 2009

[Fielding2000] Fielding, Roy, „Architectural Styles and the Design of Network-based Software Architectures“, http://www.ics.uci.edu/~fielding/pubs/dissertation/fielding_dissertation.pdf, Dissertation, University of California, Irvine, 2000

[ApShindig] Apache Shindig Homepage, http://incubator.apache.org/shindig, 06.04.2009

[ShindigCode] Shindig-Code der Version 1.0, http://svn.apache.org/repos/asf/incubator/shindig/branches/1.0.x-incubating/, 06.04.2009

[FbDev] Facebook Developer-Seite, http://developers.facebook.com, 06.04.2009

[OpenSocial] OpenSocial-Startseite, http://www.opensocial.org, 06.04.2009

[OpenSocialSpec] OpenSocial-Spezifikation v0.8.1, http://www.opensocial.org/Technical-Resources/opensocial-spec-v081.html, 06.04.2009

[OpenSocialSpec09] OpenSocial-Spezifikation v0.9, http://www.opensocial.org/Technical-Resources/opensocial-spec-v09/OpenSocial-Specification.html, 06.04.200

[GadgetSpec] Gadget-Spezifikation v0.8, http://www.opensocial.org/Technical-Resources/opensocial-spec-v08/gadget-spec.html, 06.04.2009

[OSClientLib] Wiki-Seite der OpenSocial ClientLibrary, http://wiki.opensocial.org/index.php?title=Client_Libraries, 06.04.2009

[OpenSocialContainer] Übersicht der OpenSocial-Container, http://wiki.opensocial.org/index.php?title=Containers, 06.04.2009

[OSDE] „OSDE-Tutorial – Develop OpenSocial Applications entirely within Eclipse“, http://wiki.opensocial.org/index.php?title=OSDE_Tutorial, 06.04.2009

[Raja2007] Raja, „Introduction To Google Guice“, http://www.javabeat.net/articles/29-introduction-to-google-guice-1.html, August 2007

[OAuth]	OAuth-Homepage, http://oauth.net, 06.04.2009
[OAuthExpl2007]	„Explaining OAuth“, http://www.hueniverse.com/hueniverse/2007/09/explaining-oaut.html, 06.04.2009
[OAuthBegGuide2007]	„Beginner´s Guide to OAuth“, http://www.hueniverse.com/hueniverse/2007/10/beginners-guide.html, 06.04.2009
[GGE]	GoogleGadgetEditor, http://code.google.com/intl/de-DE/apis/gadgets/docs/legacy/gs.html, 06.04.2009
[Laird2008]	Laird, Peter, „Building Dynamic Google Gadgets in Java“, http://wlp.bea.com/blogs/laird_googlegadgets.ppt, 06.04.2009
[hi5]	hi5-Homepage, http://hi5.com, 06.04.2009
[Orkut]	Orkut-Homepage, http://www.orkut.com, 06.04.2009
[MySpace]	MySpace-Homepage, http://www.myspace.com, 06.04.2009
[SocialSite]	SocialSite-Homepage, https://socialsite.dev.java.net, 06.04.2009
[Xing]	Xing-Homepage, http://www.xing.com, 06.04.2009
[VirtPresenter]	Vorlesungsaufzeichnung als Facebook-Applikation – Uni Osnabrück präsentiert auf der CeBIT den „social virtPresenter“, http://idw-online.de/pages/de/news302955, 06.04.2009
[Athan2008]	E. Athanasopoulos, A. Makridakis, S. Antonatos, D. Antoniades, S. Ioannidis, K. G. Anagnostakis, E. P. Markatos, „Antisocial Networks: Turning a Social Network into a Botnet“, http://www.ics.forth.gr/~elathan/publications/facebot.isc08.pdf, 06.04.2009
[Ebay]	Selling Manager Applications – Overview, http://developer.ebay.com/products/selling-manager-applications/overview/default.aspx, 06.04.2009
[Grewe2009]	Grewe, Lynne „OpenSocial-Network Programming“, Wrox-Verlag, 2009
[Prototype]	Prototype-Homepage, http://www.prototypejs.org, 06.04.2009

8 Abbildungsverzeichnis

9 Anhang

9.1 Installationsanleitung

Vorraussetzung für die Installation ist ein JDK 1.5, Apache Tomcat 6 und MySql 5.0.

9.1.1 Datenbank

- MySql-Client auf der Konsole starten und das Datenbank-Skript openCommunityDB_schema.sql ausführen. Die Ausführung erfolgt mit folgendem Kommando: `\. Skriptname.` Das Skript legt die Datenbank opencommunity an. Es wird standardmässig ein User admin mit Passwort admin angelegt. Über diesen können weitere Benutzer für die Plattform angelegt werden. Benutzername und Passwort können im Skript angepasst werden.
- Mit dem Skript openCommunityDB_data.sql können initiale Testdaten importiert werden. Hierbei werden drei User (student1 bis student3) angelegt, mit denen das System getestet werden kann.

9.1.2 Shindig

Die Shindig-Webapplikation ist für die OpenSocial-Erweiterung notwendig und muss als Root in Tomcat installiert werden.

Mit dieser Installation muss die bestehende Root-Applikation aus Tomcat entfernt werden und mit dem mitgelieferten ROOT.war ersetzt werden. Damit kann die Tomcat-Konsole nicht mehr über http://localhost:8080/ aufgerufen werden. Zum Verwalten von Tomcat muss jetzt die Manager-Applikation direkt über die URL http://localhost:8080/manager/index.html aufgerufen werden.

Vorgehen:

- Tomcat stoppen.
- ROOT-Ordner sichern und löschen
- ROOT.war in $TOMCAT_HOME\webapps einfügen
- Tomcat starten. Shindig wird beim Start automatisch deployt.

9.1.3 OpenCommunity

- OpenCommunity.war in $TOMCAT_HOME\webapps einfügen. Die Applikation wird bei laufendem WebServer automatisch deployt oder, wenn der Server nicht gestartet ist, beim nächsten Start.

9.2 Benutzerdokumentation

Die Benutzerdokumentation basiert auf der Vorgängerarbeit von Frau Fuhrmann. Hier werden nur die Erweiterungen zu dieser Version dargestellt.

9.2.1 Start der Web-Anwendung

Mit der URL http://localhost:8080/OpenCommunity/index.xhtml wird die Applikation gestartet. Nach dem Einloggen erscheint die Home-Seite des Benutzers.

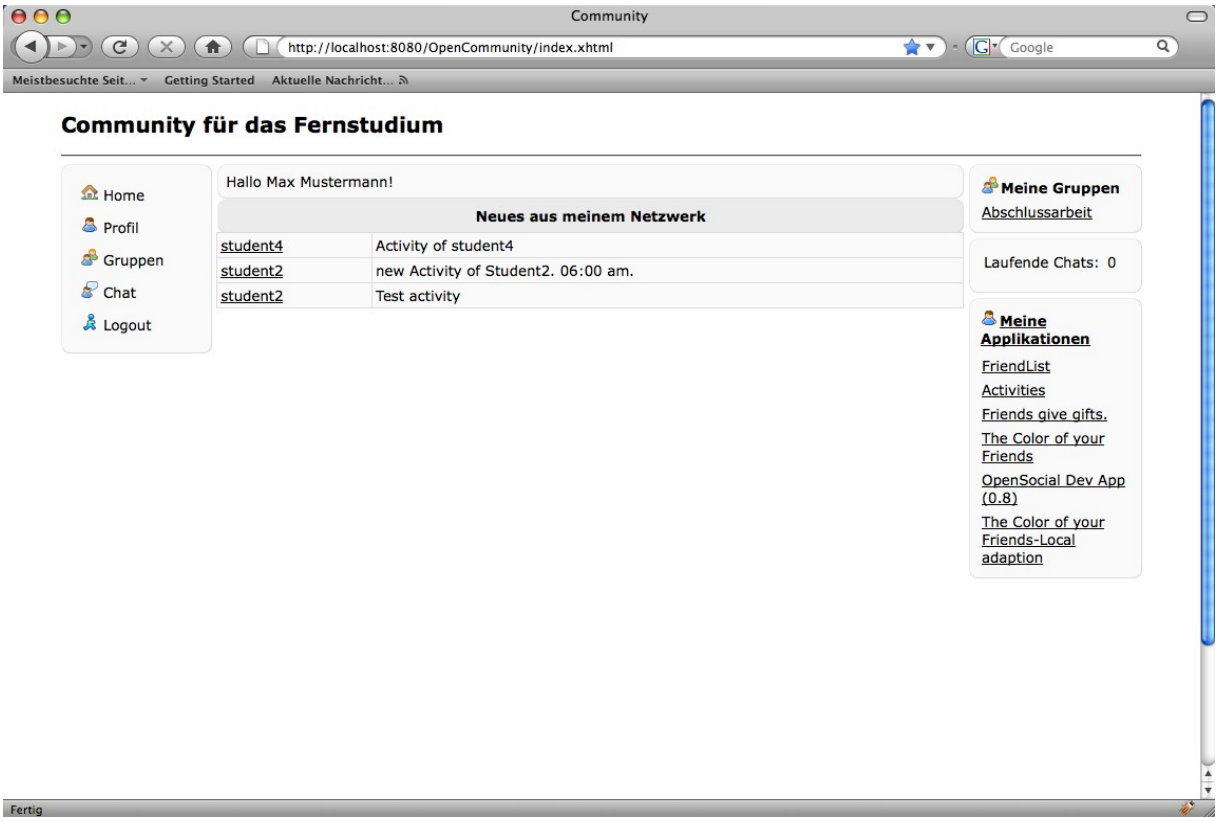

Abbildung 14: Startbildschirm

Im Hauptbereich werden die neuesten Aktivitäten der Freunde des eingeloggten Benutzers angezeigt. Auf der rechten Seite befindet sich eine Übersicht der eigenen installierten Applikationen. Diese Liste kann über den Link "Meine Applikationen" editiert werden.

9.2.2 Applikationen verwalten

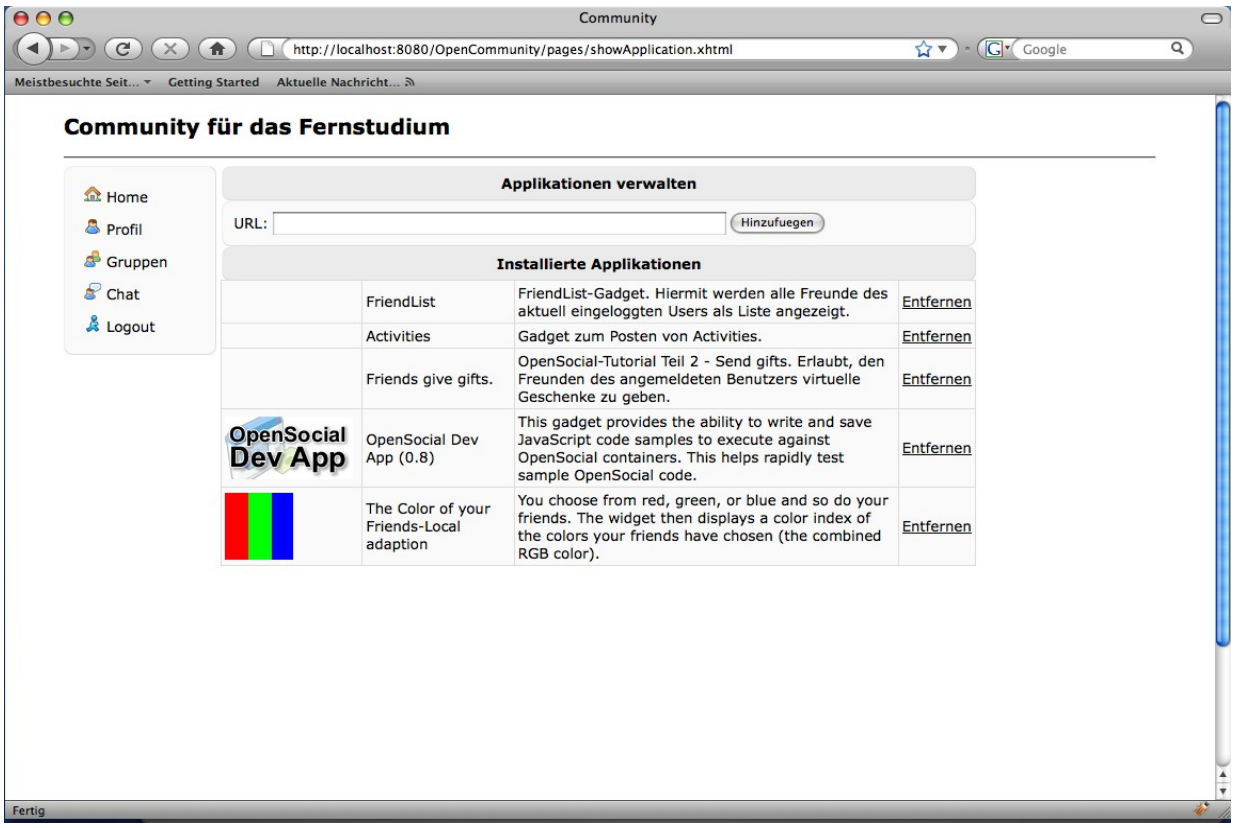

Abbildung 15: Applikationen verwalten

Diese Seite listet die Applikationen, die der angemeldete Benutzer in seinem Profil installiert hat, auf. Über das Eingabefeld und den Button Hinzufügen kann eine neue Applikation installiert werden. Hierzu muss die Adresse der xml-Datei eingegeben werden und mit Hinzufügen bestätigt werden.

In der Installation sind bereits einige Beispiel-Applikationen enthalten, die über die entsprechende URL installiert werden können.

Mit der URL http://localhost:8080/gadgets/files/container/HelloWorld.xml wird ein einfaches HelloWorld-Gadget installiert.

Die Applikationen müssen nicht zwingend auf dem lokalen Rechner installiert sein. Es können auch entfernte Adressen von Applikationen eingegeben werden. Falls der Zugang zum Internet über einen Proxy geregelt wird, ist der Aufruf von entfernten Inhalten mit der momentanen Version nicht möglich.

9.2.3 Applikationen ausführen

Ausgeführt wird eine Applikation über die Applikationsliste auf dem Start-Bildschirm.

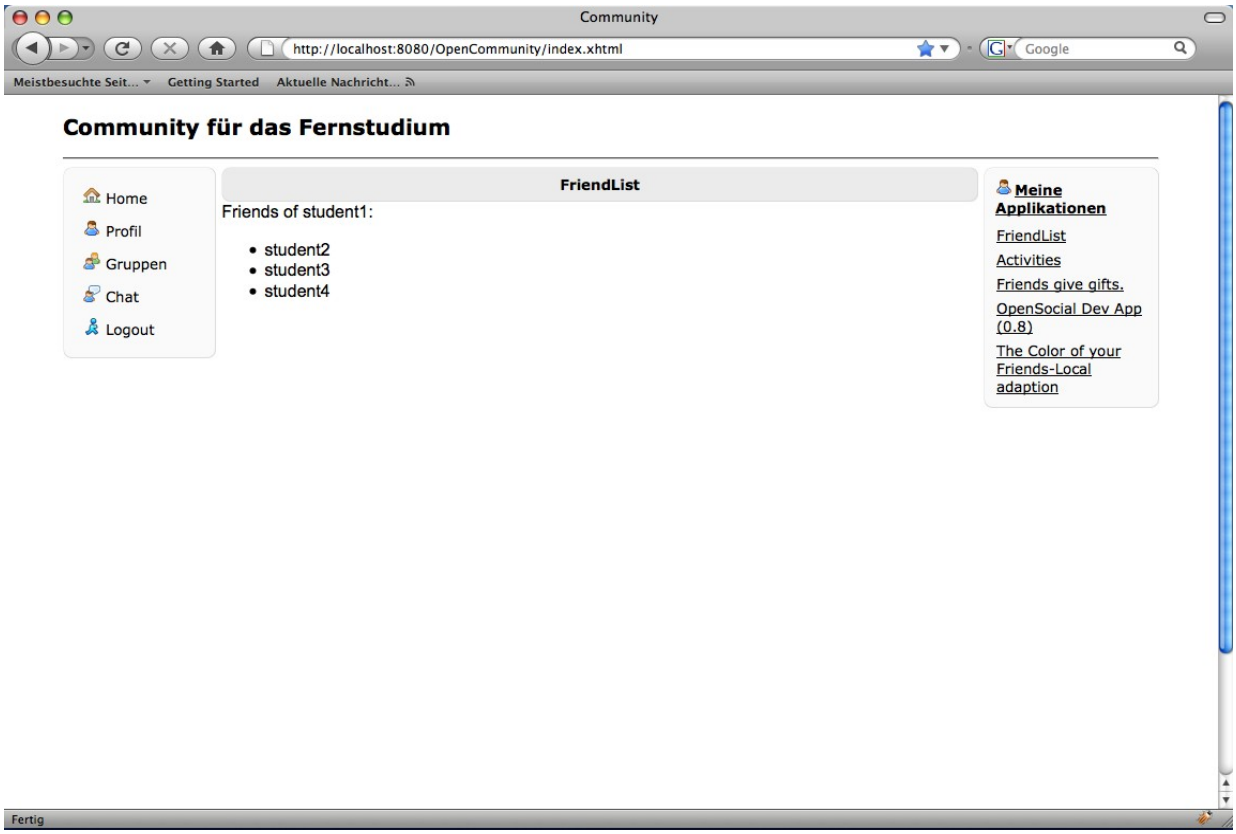

Abbildung 16: Applikation ausführen

Die Applikation FriendList stellt ein einfaches Beispiel für eine OpenSocial-Applikation dar und zeigt die Freunde des aktuell eingeloggten Benutzers (Viewer) an.

9.2.4 Applikationen anderer Benutzer

Navigiert man auf das Profil eines anderen Benutzers werden im Info-Bereich die Applikationen dieses Benutzers angezeigt. Eine Applikation in diesem Bereich kann direkt ausgeführt werden.

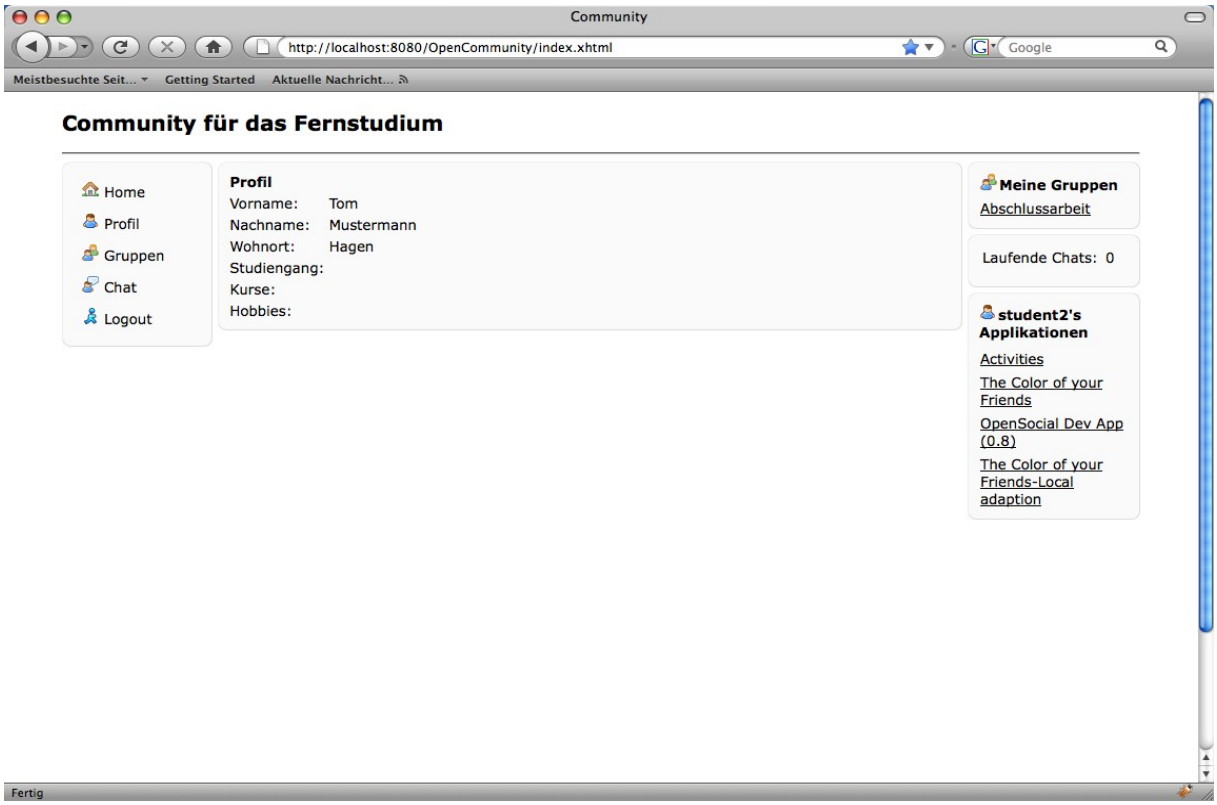

Abbildung 17: Profilseite

9.2.5 Benutzung der REST-API

Mittels dem OpenSocial REST-API kann maschinell auf Daten des OpenCommunity-Systems zugegriffen werden.

Der Aufruf der URL

```
http://localhost:8080/social/rest/people/student1/@all
```

liefert die Freunde des Benutzers student1 als JSON-String zurück:

```
{"entry":
 [{
    "addresses":[{"locality":"Hagen"}],
    "interests":[null],
    "name":
    {
      "formatted":"Tom Mustermann",
      "givenName":"Tom",
      "familyName":"Mustermann"
    },
    "id":"student2",
    "isOwner":false,
    "displayName":"student2",
    "isViewer":false
  },
  {
    "addresses":[{"locality":"Frankfurt"}],
    "interests":[null],
```

```
    "name":
    {
      "formatted":"Jenny Mustermann",
      "givenName":"Jenny",
      "familyName":"Mustermann"
    },
    "id":"student3",
    "isOwner":false,
    "displayName":"student3",
    "isViewer":false
  }],
  "totalResults":2,
  "startIndex":0
}
```


Mit folgendem REST-Aufruf werden die Profilinformationen des Benutzers student1 ausgegeben:

```
http://localhost:8080/social/rest/people/student1/@self
```

Ergebnis als JSON-String:


```
{"entry":
 {
  "id":"student1",
  "isViewer":false,
  "interests":[null],
  "name":{"formatted":"Max Mustermann","familyName":"Mustermann","givenName":"Max"},
  "isOwner":false,
  "displayName":"student1",
  "addresses":[{"locality":"Freiburg"}]
 }
}
```