



Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

DEPARTMENT INFORMATION

Bachelorarbeit

Realisierung eines mobilen Informationsdienstes durch eine Web 2.0 Applikation auf Smartphones

vorgelegt von

Martin Knudsen

Studiengang Medien und Information

erster Prüfer: Prof. Dr. Martin Gennis
zweiter Prüfer: Dipl.-Inf. Stefan Mintert

Hamburg, August 2010

Realisierung eines mobilen Informationsdienstes durch eine Web 2.0 Applikation auf Smartphones

von Martin Knudsen

Abstract

Gegenstand dieser Arbeit ist die Planung und Entwicklung einer prototypischen Smartphone-Applikation zur Interaktion mit einem bestehenden Web 2.0 Angebot. Nach einer kurzen Vorstellung der vorhandenen Plattform „Sempr“ wird der geplante Funktionsumfang der zu entwickelnden Software „Sempr-Mobile“ definiert und anhand von UML veranschaulicht. Um den Entwicklungsaufwand einzuschränken wurde ferner eine der am Markt vorhandenen Smartphone-Plattformen ausgewählt und diese Entscheidung anhand verschiedener Kriterien begründet. Anschliessend werden Anwendungsstruktur sowie Implementierung näher erläutert und mit Beispielen veranschaulicht. Ein weiterer Arbeitsschwerpunkt war die Integration des Projektes in den Softwareentwicklungsprozess der Firma Linkwerk GmbH. Im letzten Abschnitt wird eine Bilanz aus den geleisteten Ergebnissen gezogen und die fertige Anwendung kritisch bewertet. Auch werden hier zukünftige Entwicklungsmöglichkeiten sowie Verbesserungspotenziale der Software näher beleuchtet.

Deskriptoren

Softwareentwicklung – Smartphone – Web 2.0 – Programmierung – Planung – Social Software – Android – Java

Inhaltsverzeichnis

1	Einleitung	6
1.1	Sempr: Überblick & Funktionsumfang	6
1.2	Tags, Links und Annotationen	7
1.3	Webseiten als Ressourcen	8
1.4	Normalisiert und Denormalisiert	9
1.5	Von Browsern und Apps	9
2	Konzept Sempr-Mobile	11
2.1	Funktionsumfang	11
2.2	Benutzeroberfläche	13
3	Architektur	15
3.1	Android kontra iPhone	15
3.2	Android-Plattform im Überblick	16
3.2.1	Intents	17
3.2.2	Activities	19
3.2.3	GUI und Layout	19
3.3	Messaging-Konzept über Intents	21
3.4	Verkettung von Activities	21

<i>INHALTSVERZEICHNIS</i>	5
4 Implementierung	22
4.1 Android SDK	22
4.2 Sempr-Mobile Objektmodell	22
4.3 Dependency Injection und Guice	24
4.4 Verwendete Werkzeuge	26
4.5 Die fertige Applikation	27
5 Integrieren & Testen	30
5.1 Bau mit Maven & SDK	30
5.2 Testbarkeit	31
6 Fazit & Ausblick	32
6.1 Allgemein	32
6.2 Veröffentlichung der Anwendung	33
6.3 Verbesserungen in Design, Layout und Benutzerführung	33
6.4 Browserplugin oder eigener Browser	34
6.5 URI weiter gedacht	34
6.6 Performance-Optimierung	35
Abbildungsverzeichnis	36
Literaturverzeichnis	37
Glossar	40
A Codeausschnitte	41
A.1 Kommentierter Code einer Activity	41
A.2 Kommentiertes XML eines Views	45

Kapitel 1

Einleitung

Im Internet genießen „soziale“ Tools mehr und mehr Beliebtheit. Nutzer sind nicht mehr bloße Konsumenten, sondern werden zu Produzenten der Inhalte, die sie selbst konsumieren. Das Paradebeispiel ist seit Langem die Wikipedia. Die selbst ernannte „freie Enzyklopädie“ umfasst mittlerweile mehr als 16 Millionen Artikel und ist in rund 260 Sprachen vertreten (Wikipedia, 2010b). Auch kommerzielle Plattformen wie Flickr und Youtube setzen auf vom Publikum erstellte Inhalte.

Diese Webseiten werden aufgrund ihrer Neuartigkeit in einigen Merkmalen oft von den „alten Medien“ abgegrenzt und kursieren unter dem so unscharfen wie inflationär gebrauchten Begriff „Web 2.0“. Der Benutzer rückt in den Mittelpunkt des Geschehens, wird Autor, Kritiker, Publizist und Regisseur. Das Erstellen, Verarbeiten und Verteilen von Inhalten soll einfacher und schneller werden, als es vorher war.

Die Linkwerk GmbH aus Hamburg ist seit 2003 in der Softwareentwicklung tätig, traditionell für professionelle Content-Hersteller, wie zum Beispiel Fach-Verlage. Seit einiger Zeit wird hier die Idee eines universellen Tools gepflegt, das das Suchen, Finden, Konsumieren und Entdecken von Inhalten im Web einfacher, schneller und letztendlich besser macht. Dieses Tool heißt Sempr.

1.1 Sempr: Überblick & Funktionsumfang

Sempr ist ein Kollaborationstool für das Web. Nutzer können eigene Inhalte einstellen, wie dies bereits andere Angebote erlauben. Im Gegensatz zu diesen werden die

Informationen bei Sempr aber im Kontext von Webseiten, oder allgemeiner, Ressourcen erstellt. Kommentare sind dort aber nicht bloße Texte auf einem Angebot fernab von der kommentierten Webseite, sondern werden anderen Nutzern direkt auf dieser präsentiert.

Dies ist möglich, weil Sempr als Toolbar im Browser auf jeder Webseite verfügbar ist (siehe Abbildung 1.1) und nicht separat besucht werden muss. So kann der Nutzer Webseiten kommentieren, miteinander verlinken und taggen. Dritte bekommen diese Interaktion dann in ihrem Browser quasi-zeitgleich angezeigt und können die Links, die Andere gesetzt haben, verwenden. So lassen sich auf der Webseite eines Restaurants direkt Kritiken hinterlassen oder auf der Webseite eines Kinos Links zu anderen Kinos anbringen. Die verstreuten Informationen im Web sollen in Zukunft so näher zusammenwachsen und einen Mehrwert bieten, den kein einzelnes Portal bieten könnte (vgl. Mintert, 2010).

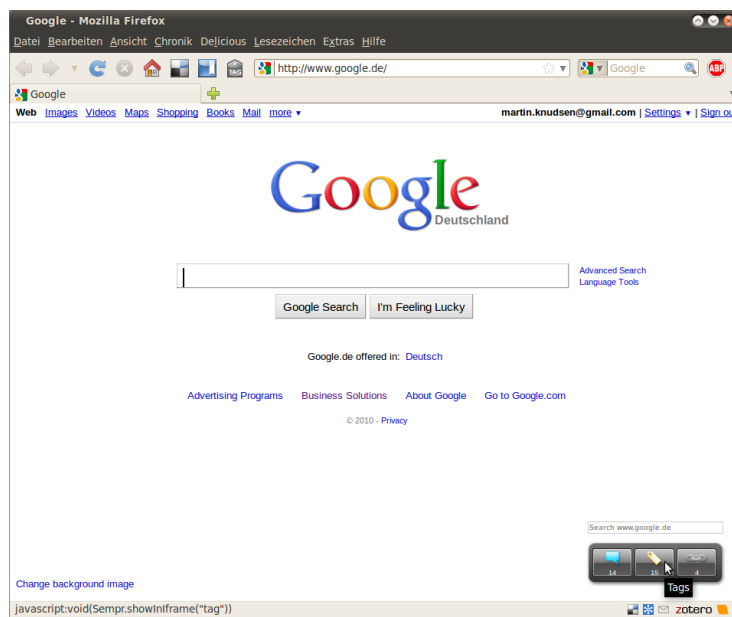


Abbildung 1.1: Sempr-Toolbar (unten rechts) auf der Startseite von Google.

1.2 Tags, Links und Annotationen

Momentan gibt es bei Sempr drei Arten von Informationen: Tags, Links und Annotationen. Diese haben je eine Quell-Webseite, deren Titel und URI erfasst werden. Der Titel ist hierbei vom Nutzer frei wählbar, er muss nicht dem tatsächlichen Titel der Webseite entsprechen, siehe hierzu auch Abbildung 1.1. Ferner haben Links auch

eine Ziel-Webseite, welche ihrerseits mit URI und Titel erfasst wird. Annotationen enthalten zudem einen Annotationstitel sowie einen Annotationstext. Tags bestehen aus einfachem Text und werden durch Kommata getrennt eingegeben. Bei jeder dieser Aktionen kann der Nutzer wählen, ob diese öffentlich oder privat, also nur für ihn selbst, sichtbar sind. Die Ansicht der neuesten Events auf Sempr im *Dashboard* ist in Abbildung 1.2 dargestellt.

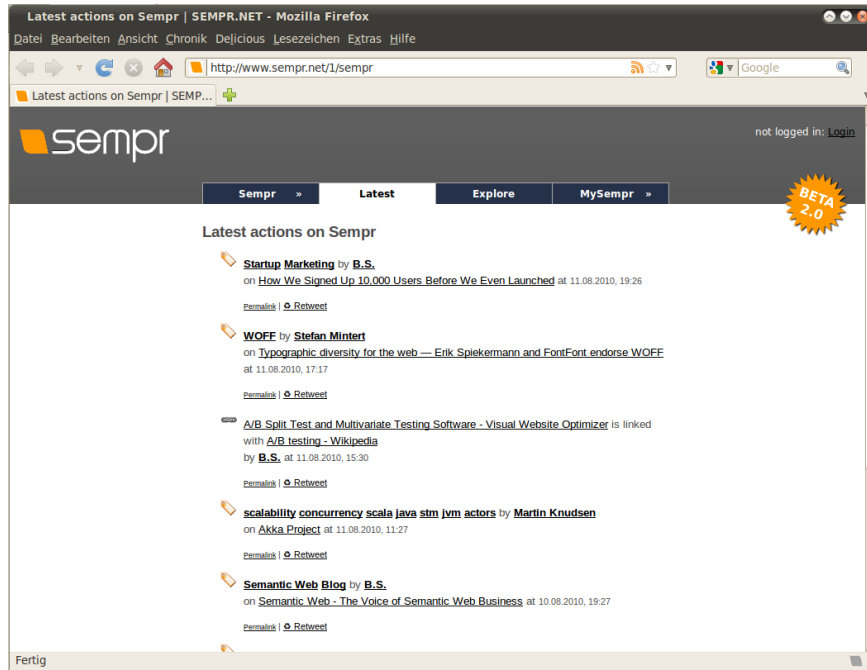


Abbildung 1.2: Das Sempr Dashboard, die Übersicht der aktuellsten Ereignisse auf Sempr.

1.3 Webseiten als Ressourcen

Ein zentrales Konzept von Sempr ist, dass es nicht auf Webseiten beschränkt ist, sondern abstrakt „Ressourcen“ beschreiben will. Mittelfristig sollen hierfür aber auch andere Ressourcen als Webseiten - wie ISBN, Produkte, Telefonnummern, Personen oder Orte - infrage kommen. Dies würde ermöglichen, das Web mit der realen Welt zu verknüpfen, zum Beispiel ein Produkt mit seiner Rezension im Internet zu verlinken, oder einen Ort mit einem Kommentar zu versehen. Systemintern werden alle Ressourcen als URI formuliert, einer der Gründe, warum Sempr sich derzeit auf Webseiten konzentriert, die standardmäßig mit einem URI versehen sind.

1.4 Normalisiert und Denormalisiert

Im Backend von Sempr werden diese Relationen in einer RDF-Datenbank gespeichert und stark normalisiert. So ist zum Beispiel der Vorgang des Taggens einer Webseite aufgeteilt in die einzelnen Tags, die jeder für sich genommen eine Resource darstellen. Nur die Texte von abgegebenen Kommentaren liegen hierbei nicht in der RDF-Datenbank vor, sondern werden nur als Referenz vorgehalten und extern gespeichert.

Um diese stark normalisierten Informationen in einen zeitlichen Kontext zu setzen, wurde die denormalisierte Entität des *Events* eingeführt. Ein Event bezeichnet den Vorgang des Kommentierens, Linkens oder Taggens einer Webseite, den ein bestimmter Nutzer zu einem definierten Zeitpunkt getätigt hat. Das Event enthält ebenfalls die jeweiligen Links, Kommentare oder Tags, fasst diese aber etwas zusammen. So können in Atom-Feeds die einzelnen Aktionen der Nutzer beschrieben werden, ohne jeden verwendeten Tag als eigenen Eintrag dort abzubilden. Demnach ist im Datenbestand von Sempr einerseits die Sichtweise auf Ressourcen wie Webseiten oder Tags, andererseits auf Aktionen gewährleistet.

1.5 Von Browsern und Apps

Das Hauptproblem, mit dem sich diese Arbeit befasst ist, dass sich die Verwendung von Web-Anwendungen auf Smartphones mehr und mehr vom Browser in eigene Apps verlagert. Die Gründe hierfür sind nicht bekannt. Eine Vermutung hierzu ist die gefühlte schnellere Reaktion von lokal installierten Applikationen im Vergleich zu Webseiten. Eine Weitere die teils sehr schlechte Anpassung vorhandener Webangebote auf mobile Endgeräte mit ihren vergleichsweise kleinen Bildschirmen. Was immer der Grund hierfür sein mag, Größen wie Facebook, Twitter oder StudiVZ bieten seit geraumer Zeit eigene Applikationen (kurz Apps) für Smartphones an.

Der technische Weg, Sempr „in“ den Browser des Nutzers zu bringen, ist momentan ein angepasster HTTP-Proxy. Die Einrichtung eines solchen ist zwar prinzipiell auf Smartphones nicht unmöglich, wird aber beispielsweise auf Apples iPhone immer im Kontext eines bestimmten Netzwerk-Zugangspunktes (also eines WLAN-Access-Points) erwartet. Die Proxy-Nutzung zur Regel zu machen, ist also nicht vorgesehen.

Des Weiteren ist die Sempr-Toolbar, die die Funktionen von Sempr im Browser auf dem Desktop zur Verfügung stellt, bereits auf große Anzeigen optimiert. Zur Anzeige

auf Smartphones wäre also eine größere Anpassung in der Benutzeroberfläche des Sempr Web-Frontends nötig.

Dies sind die Hauptgründe, die zur Entscheidung, eine eigene Applikation zu entwickeln, beigetragen haben.

Kapitel 2

Konzept Sempr-Mobile

2.1 Funktionsumfang

Der Funktionsumfang der in dieser Arbeit entwickelten Software soll dem entsprechen, was derzeit auf der Webversion von Sempr möglich ist. Das heißt konkret Nutzern soll es erlaubt sein, Tags, Links und Annotationen für Webseiten anzulegen. Dies sollte gleichzeitig komfortabel und schnell sein, also das Laden einer separaten Webseite nach Möglichkeit entfallen.

Weiterhin sollen Nutzer alle Informationen, die Sempr über Webseiten bereitstellt, abrufen können. Zudem sollen auch soziale Funktionen einfließen, beispielsweise das Weiterleiten eines interessanten Kommentars an einen Freund oder das Veröffentlichen von Events auf Twitter.

Der gewünschte Funktionsumfang wurde hierfür zuerst in UML, genauer gesagt UML Use-Case-Diagrammen, ausgedrückt. Hierbei wurde versucht, weitestgehend ähnliche Anwendungsfälle zu gruppieren, um idealerweise später direkt Potenzial für die Wiederverwendung von Programmteilen abzuleiten. Der Systemüberblick in Abbildung 2.1 zeigt, stark vereinfacht, wozu Sempr-Mobile imstande sein soll. Die einzelnen Use-Cases sind in den weiteren Abbildungen genauer aufgeschlüsselt.

Das Anzeigen von Events, zu sehen in Abbildung 2.2, findet in verschiedenen Ausprägungen statt. Mal werden alle aktuellen Events dargestellt, mal alle Events, die einen bestimmten Tag enthalten. Die Gemeinsamkeit dieser Anwendungsfälle besteht darin, dem Nutzer eine Menge von Events darzustellen. Hierbei bleibt zu bedenken, dass für die Aufgaben mitunter verschiedene Eingabeparameter benötigt werden. So wird

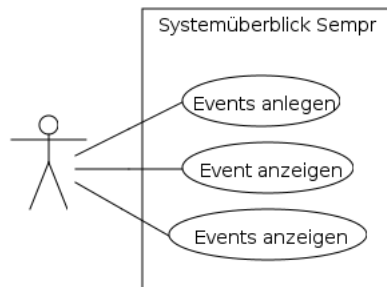


Abbildung 2.1: Systemüberblick Sempr

zum Beispiel für das Anzeigen aktueller Events nichts, für das Anzeigen aller Events mit Tag „X“ eben dieser benötigt.



Abbildung 2.2: Detailansicht Events (als Listen) anzeigen

Das Anlegen von Events, dargestellt in Abbildung 2.3, konkretisiert sich entsprechend der drei momentan vorhandenen Arten von Events: Links, Tags und Annotationen.

Einen zusätzlichen Use-Case für das Anzeigen stellt, wie bereits erwähnt, das Weiterleiten von Events zu Twitter dar. Hierbei wird ein Link auf das betreffende Event in einen Tweet eingefügt, sodass Nutzer von Twitter auf interessante Inhalte auf Sempr hingewiesen werden. In Abbildung 2.4 ist dies als Erweiterung des Anwendungsfalles *Event anzeigen* realisiert, also als Option zu verstehen.

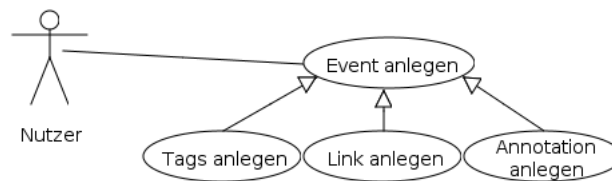


Abbildung 2.3: Detailansicht Event anlegen

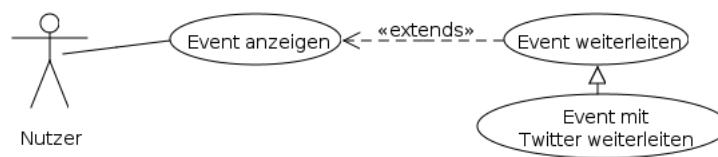


Abbildung 2.4: Detailansicht Event anzeigen

2.2 Benutzeroberfläche

Die Benutzeroberfläche der Anwendung sollte in erster Entwicklungsstufe schlicht und funktional sein. Es geht hierbei mehr um einen Proof-of-Concept als um eine endgültige Software. Wird dieser erfolgreich erbracht, werden im Anschluss an diese Arbeit Design- und Benutzerführungs-Aspekte stärker berücksichtigt. Insgesamt sollte die Oberfläche eine bessere Bedienbarkeit bieten, als die momentane Sempr-Website. Das bedeutet geringe Wartezeiten, möglichst direktes Feedback bei Eingabefehlern und ein schnelles, unkompliziertes Abrufen von Informationen. Hierbei dienen gängige Anwendungsstrukturen als Vorbild - ein Hauptmenü zeigt beim direkten Aufruf der Anwendung die verfügbaren Aktionen, eine Eingabe wird vor dem Speichern auf Fehler geprüft, mehrseitige Suchergebnisse erlauben das seitenweise Vor- und Zurücknavigieren. Einige Mockups für das Layout sind in Abbildung 2.5 zu sehen.

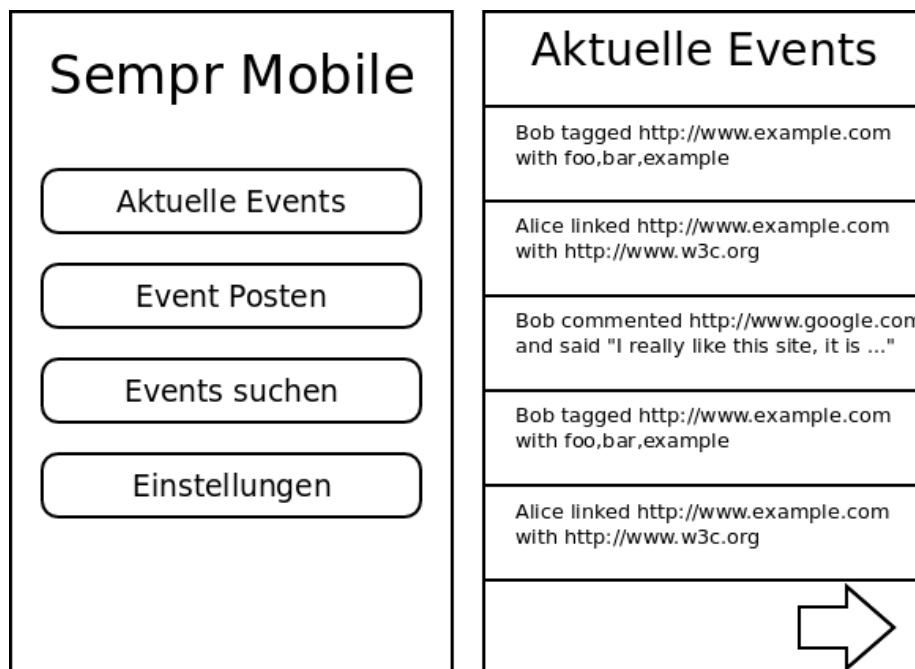


Abbildung 2.5: Mockups von Hauptmenü sowie Listen-Ansicht mehrerer Events.

Kapitel 3

Architektur

3.1 Android kontra iPhone

Im Laufe dieser Arbeit musste, vor allem aus Zeitgründen, die Entscheidung getroffen werden, für welche Plattform Sempr-Mobile zunächst entwickelt würde. Die Vorauswahl fiel, aufgrund derzeitiger Marktanteile, auf Apples iPhone sowie die von Google mitentwickelte Plattform Android. Die Gründe hierfür werden im Folgenden dargestellt.

Ein Argument gegen die Entwicklung für das iPhone war die kostspielige Entwicklung, denn Apples eigenes SDK ist ausschließlich für die Benutzung unter Mac OS geeignet. Dementsprechend hätte vor Beginn der Entwicklung ein hierfür geeigneter Computer angeschafft werden müssen. Hinzu kommt die sogenannte Developer-Subscription, die benötigt wird, um die eigene Software auf einem iPhone oder iPod zu testen.

Apple stellt zudem restriktivere Anforderungen an die Entwickler, so darf Software im App Store nur in den Sprachen Objective-C und JavaScript programmiert sein (Quelle: Ihlenfeld, 2010). Dies wäre kein Problem, wenn nicht der App Store für handelsübliche iPhones die einzige Möglichkeit wäre, überhaupt Programme auf dem Gerät zu installieren (siehe Sullivan, 2010). Des Weiteren ist das Einstellen von Anwendungen in den App-Store nicht trivial. So gibt es eine Reihe von Beschränkungen, was den Inhalt der Software angeht und Entwickler haben keinerlei Rechtsanspruch, ihre Software von Apple dort platzieren zu lassen.

Alle Argumente, die hier gegen die iPhone-Plattform sprechen, treffen auf Android nicht zu. Android ist als Plattform bedeutend offener als Apples iPhone. Sie ist prinzipiell kostenlos und plattformübergreifend nutzbar und es gibt eine Vielzahl verschiedener Hardwarehersteller, die Android-Geräte auf den Markt bringen, von denen die wenigsten restriktiv in der Installation von Software auf den Geräten sind.

Zudem gab es auch technisch-pragmatische Gründe für die Entscheidung. So liegt der Schwerpunkt der Firma Linkwerk im Java-Umfeld, sodass mit einem Umstieg auf iPhones Objective-C ein nicht unerheblicher Aufwand verbunden wäre. Die meisten Komponenten von Sempr auf der Serverseite sind zudem in Java entwickelt. Auf dem iPhone ist das Ausführen von Java-Applikationen derzeit nicht vorgesehen.

Zuletzt sei erwähnt, dass das Marktforschungsunternehmen Gartner bereits Ende letzten Jahres eine Marktführerschaft für Android-Telefone ab 2012 prognostizierte (laut Kirsch, 2009).

3.2 Android-Plattform im Überblick

Vor einer Erläuterung der Architekturmerkmale dieser Anwendung ist es sinnvoll, einige Konzepte der Softwareentwicklung unter Android kennenzulernen. Für sich genommen sind diese Konzepte nicht neu, in ihrer Kombination innerhalb der Plattform für Mobilgeräte aber innovativ. Alles hier Beschriebene lehnt sich an die offizielle Android Dokumentation¹ an, soweit es nicht Dinge behandelt, die spezifisch auf das Verhalten dieser Anwendung eingehen. Ein Großteil von Android-Applikationen besteht aus Intents und Activities. Grob gesagt stellen Intents hierbei Aufgaben dar, Activities arbeiten diese ab. Details zu diesen Elementen sollen Gegenstand dieses Abschnittes sein.

Den Ablauf der Interaktion verschiedener Activities und Intents kann man sich, stark vereinfacht, wie folgt vorstellen:

1. Activity A erstellt Intent Z
2. Activity A sendet Intent Z an das System
3. Das System stellt Intent Z zu
 - (a) Extraktion der Eigenschaften von Intent Z (Action, Data)
 - (b) Abgleich mit den bekannten Activities der installierten Applikationen

¹Die Dokumentation ist als Archiv dieser Arbeit beigelegt.

- (c) Gegebenenfalls Auswahl für den Nutzer, welche Activity er für den Intent verwenden möchte
 - (d) Starten der Gewünschten Activity und zustellen des Intents
4. Activity Y wird gestartet, liest Daten aus Intent Z und arbeitet diesen ab

3.2.1 Intents

Ein Intent² ist eines der Hauptkonzepte innerhalb der Android-Plattform und stellt das zentrale Nachrichtenformat dar. In diesem werden die Absichten des Nutzers oder einer Anwendung formuliert. Intents werden zur programminternen sowie zur programmübergreifenden Kommunikation genutzt. So kann ein Entwickler sowohl Komponenten seiner eigenen Software als auch einer anderen auf dem Gerät installierten über einen Intent aufrufen. Diese Universalität wird durch eine Reihe simpler Bausteine erreicht, aus denen ein Intent besteht.

Intents enthalten prinzipiell ein Daten-Feld, das die zu verarbeitenden Daten anhand eines URI beschreibt. Beim Intent zum Öffnen einer Webseite wäre dies die URL. Der Intent zum Bearbeiten eines Bildes würde entsprechend eine eindeutige Referenz auf das Bild beinhalten, beispielsweise `images:///jpeg/1233`.

Das zweite zwingend erforderliche Feld eines Intents ist das Aktions-Feld (oder Action-Feld). Hier kann ein beliebiger String enthalten sein, der die zu verrichtende Aufgabe artikuliert. Im Kontext der eigenen Applikation spricht also nichts dagegen, eigene Aktionen zu definieren. Jedoch ist es im Sinne der anwendungsübergreifenden Interoperabilität sinnvoll, eine der von der Plattform vorgegebenen Aktionen zu nutzen. So sind in der Klasse Intent eine Vielzahl von Aktionen vordefiniert, unter anderem `ACTION_VIEW`, `ACTION_EDIT` sowie `ACTION_SEND`. Tabelle 3.1 zeigt einen Intent, wie er auch in dieser Software benutzt wird.

Data	<code>sempr://www.sempr.net/1/tag/public/to/hamburg</code>
Action	<code>ACTION_VIEW</code>

Tabelle 3.1: Aufbau eines exemplarischen Intents aus der entwickelten Anwendung. Dieser Intent zeigt alle Events an, die mit dem „hamburg“ getaggt sind.

Optional können Intents noch sogenannte *Extras* enthalten. Dies sind pro Intent beliebig viele Felder, die durch einen String identifiziert werden. Diese Daten können (fast) beliebigen Typs sein und zusätzliche Informationen zur Verarbeitung des Intents enthalten. So kann zum Beispiel bei der Anzeige einer Liste von Dingen

²Grob übersetzt etwa „Absicht“, „Vorhaben“ oder „Ziel“.

eben diese bereits mitgesendet werden, anstatt sie lediglich über einen URI zu referenzieren. Die Klassen der so angehängten Objekte müssen hierfür wahlweise das Interface *Serializable* oder *Parcelable* implementieren³. Die Ursache hierfür liegt vor allem im Speichermodell von Android. Prinzipiell laufen Programme in einem für sie reservierten Speicherbereich. Andere Programme können nicht direkt auf den Zustand einer anderen Applikation zugreifen. Daher wird nicht das eigentliche Objekt, sondern eine Kopie an den Empfänger übertragen.

Sender und Empfänger von Intents sind sich hierbei nicht bekannt. Vielmehr registrieren sich Empfänger von Intents über einen Eintrag im sogenannten *Android Manifest* auf die Intents, die sie zu verarbeiten wissen. Ein Browser würde sich so auf Intents mit der Aktion `ACTION_VIEW` und einem Datenfeld beginnend mit `http` registrieren, ein Mailprogramm hingegen auf Intents mit der Aktion `ACTION_SEND` mit einem Textanhang. Der in Tabelle 3.1 dargestellte Intent wird in der Anwendung durch den im Listing 3.1 dargestellten Eintrag aus dem Manifest an die Activity *ShowEventsForTags* weitergeleitet.

Listing 3.1: Exemplarischer Eintrag im Android-Manifest für eine Activity, die den in Tabelle 3.1 gezeigten Intent verarbeiten kann.

```
<!--
    Das Attribut android:name im Element activity
    bezeichnet den Paket und Klassennamen der auf-
    zurufenden Activity.
-->

<activity android:name=".activities.eventlists.
    ShowEventsForTags">
    <intent-filter>
        <category android:name="android.intent.category.DEFAULT"
            />
        <action android:name="android.intent.action.VIEW" />
        <data android:scheme="sempr" />
        <data android:scheme="http" />
        <data android:host="www.sempr.net" />
        <data android:pathPattern="/1/rel/tag/public/to/.*" />
    </intent-filter>
</activity>
```

³Das Interface *Serializable* ist Teil des offiziellen Java-API, *Parcelable* ein Android-eigenes Interface. Warum Android ein eigenes Interface für die Serialisierung von Klassen definiert hat, wurde im Rahmen dieser Arbeit nicht näher untersucht und im Code auf das Interface *Serializable* gesetzt.

Mit Hilfe von Intents kann der Entwickler Funktionalität in seiner Anwendung nutzen, ohne diese selbst zu implementieren. Wollte man eine Nachricht zu Twitter senden, müsste man nur herausfinden, in welcher Form gängige Twitter-Clients einen solchen Intent erwarten. Der Aufruf von verschiedenen Anwendungs-Komponenten über Intents bedeutet für die Gesamtarchitektur eine gewisse Entkopplung, die sich bei Tests positiv auswirkt. Die einzelnen Aktivitäten rufen sich nicht direkt auf und können so als Einzelkomponenten getestet oder ausgetauscht werden. Zu Intents und ihren Filtern im Android-Manifest siehe auch (Google, 2010b).

3.2.2 Activities

Activities bezeichnen die tatsächlichen Tätigkeiten, die ein Nutzer auf Basis eines Intents verrichtet. Hierbei sind Activities als Klassen innerhalb der Android-Plattform modelliert und haben Zugriff auf eine Vielzahl von Schnittstellen, mit deren Hilfe diese Tätigkeiten verrichtet werden können. Im Regelfall instantiiieren und steuern Activities die hierfür nötigen Benutzeroberflächen, beispielsweise Formulare oder Listen.

Sie können hierbei eng mit der Anzeige verknüpft werden und zum Beispiel als Listener an ein Textfeld gekoppelt werden, um bei einer Eingabe sofort zu reagieren, wodurch eine umgehende Verarbeitung erfolgen kann.

Activities werden immer über die vorher beschriebenen Intents gestartet. Da selbst das Starten einer Anwendung vom System als ein Intent formuliert wird, müssen Entwickler im *Android Manifest* definieren, welche ihrer Activities als Einstiegspunkt in die Anwendung dienen soll (siehe hierzu auch Google, 2010a).

Wie alle Tätigkeiten unter Android sind auch das Versenden einer E-Mail oder SMS, das Öffnen einer Webseite oder das Editieren eines Kontakts als Intent formuliert. Dies ermöglicht Entwicklern, mit wenig Aufwand zentrale Komponenten des Systems durch eigene zu ersetzen. Hierzu muss lediglich bekannt sein, wie die jeweils verwendeten Intents aufgebaut sind, also beispielsweise, welche Daten in welchen Feldern mitgesendet werden müssen (oder können).

3.2.3 GUI und Layout

Android bietet von Haus aus zwei Arten, die grafischen Benutzeroberflächen innerhalb einer Applikation zu definieren. Deklarativ, über XML-Dateien oder im Code,

also direkt an den Objekten der jeweiligen Ein- und Ausgabeelemente. Letzteres hat den Vorteil, auch komplexere Layouts möglich und mitunter besser wiederverwendbar zu machen, weil gleichartige Elemente und Elementgruppen über Vererbung und Komposition elegant abbildbar sind.

Die Definition des Layouts über XML-Dateien besitzt den Hauptvorteil, dass es Code und Layoutdefinition stärker trennt. Des Weiteren können Designer ohne Programmierkenntnisse das Layout mit visuellen Editoren anpassen. Hierbei ist zu erwähnen, dass die Layouts voll XML-konform und gegen ein Android-eigenes Schema validierbar sind. Eine Bearbeitung dieser Layout-Definitionen über XML-Editoren ist demnach möglich. Auch das SDK von Google bietet hierfür verschiedene Plugins und Tools an. So gibt es ein Plugin für die freie Entwicklungsumgebung Eclipse, das eine Vorschau der entworfenen XML-Layouts ohne Starten der Anwendung ermöglicht.

Alle Elemente, aus denen ein solches Layout besteht, werden in der Terminologie der Plattform als *Views* bezeichnet. Hierzu zählen sowohl Elemente mit eigener Darstellung (wie beispielsweise *ImageView* oder *TextView*) als auch solche, die nur das Layout der enthaltenen Elemente beeinflussen (wie *RelativeLayout* oder *TableLayout*). Alle Views können mit einer ID versehen, um aus dem Code heraus aufgerufen zu werden.

Die Layoutdefinition über XML ist in Listing 3.2 exemplarisch dargestellt. Dieselbe Ausgabe wird in Listing 3.3 direkt aus Java heraus erzeugt.

Listing 3.2: Definition eines View-Elements in XML.

```
<!-- Anstatt über Setter-Methoden werden hier die Werte über
      XML-Attribute gesetzt -->
<TextView
  android:text="Hallo_Welt!"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  android:id="@+id/some_string_value" >
</TextView >
```

Listing 3.3: Definition eines View-Elements aus Java.

```
TextView textview = new TextView();
textview.setText("Hallo_Welt!");
textview.setHeight()
```

3.3 Messaging-Konzept über Intents

Wie bereits erwähnt, sind URLs ein zentrales Konzept, sowohl von Sempr als auch von Android. Innerhalb dieser Applikation wird jede Aktion, die ein Nutzer vornehmen will, in einem Intent formuliert.

Die meisten Aktionen beschränken sich hierbei auf die Ansicht von Daten, also *ACTION_VIEW*. Die URLs aller applikationsweiten Intents orientieren sich an den Adressen der jeweiligen Informationen auf dem Server von Sempr. Will der Nutzer also alle Events, die mit „foo“ getaggt sind, ansehen, wird der dazugehörige Intent mit der Aktion *ACTION_VIEW* sowie dem URI *sempr://www.sempr.net/1/rel/tag/public/to/foo* formuliert.

Hierbei fällt auf, dass das Schema für diesen URI nicht „http“ wie auf der Webseite, sondern „sempr“ lautet. Dies liegt vor allem an der Funktionsweise des Browsers unter Android. Dieser ist Listener auf alle Intents, deren Daten eine Web-URL und deren Aktion *ACTION_VIEW* enthalten. Unser Intent würde also ohne diese Unterscheidung einem Intent zum Öffnen einer Webseite gleichkommen.

3.4 Verkettung von Activities

Zu den Konzepten dieser Software gehört, dass innerhalb der Applikation nicht jede Aktivität eine eigene Anzeige definiert. Einige Aktivitäten übernehmen die Funktion eines Routers, indem sie allgemeine Intents, die von anderen Programmen erzeugt wurden (beispielsweise vom Browser), in konkrete, anwendungsinterne Intents übersetzen. Andere Aktivitäten reichern Intents lediglich mit Daten an, damit dieser Code klarer von der Anzeigenlogik getrennt ist.

Für den „Fluss“ der Applikation ist es wichtig, dass solche „Transit“-Aktivitäten sich nach dem Weiterleiten selbst aus dem Aktivitäts-Stack entfernen, indem der Entwickler nach Absenden des nächsten Intents der Kette *finish()*⁴ aufruft.

⁴Diese Methode ist Teil des Android-eigenen Lifecycle-Modells. Einige Anmerkungen hierzu finden sich im Abschnitt 6.6.

Kapitel 4

Implementierung

4.1 Android SDK

Für die Entwicklung einer Android-Applikation wird das *Software Development Kit* (SDK) von Google benötigt. Dieses steht auf <http://developer.android.com/sdk> kostenlos zum Download bereit und bietet eine Vielzahl von Werkzeugen für die Anwendungsentwicklung innerhalb der Plattform an. Unter anderem enthält es einen Emulator, der ein komplettes Android-Gerät auf Hardwareebene simuliert und es ermöglicht, die entwickelte Anwendung auf einem normalen Desktop-Rechner zu testen. Der Kauf eines entsprechenden Endgerätes ist also nicht zwingend erforderlich.

Auch das Android-API wird im SDK mitgeliefert. Wichtig ist, dass die Android-eigenen Bibliotheken, also alles außerhalb des Standard Java-API, nur als sogenannte Stubs vorliegen. Diese bilden die tatsächlich auf einem Android-Gerät verfügbaren Bibliotheken zwar nach, enthalten aber keinen funktionierenden Code. Hierdurch erschwert sich das Testen auf dem Entwicklungsrechner, da das Programm nicht direkt auf diesem ausgeführt werden kann. Um die Software auszuführen, muss der Entwickler sie also auf einem Gerät oder Emulator installieren.

4.2 Sempr-Mobile Objektmodell

Das (Domänen-)Objektmodell in dieser Software ist so einfach wie möglich gehalten, einige Sachzwänge der Plattform sowie Vereinfachungen für die spätere Nutzung wurden hierbei berücksichtigt. Aus bereits genannten Gründen implementieren

alle SemprEvent-Klassen das Interface Serializable. Des Weiteren wurden die Setter der Datenobjekte so implementiert, dass diese jeweils das Objekt zurückgeben, an dem sie aufgerufen werden. Dies implementiert sogenanntes *Method-Chaining*, ermöglicht also den Programmierfluss etwas übersichtlicher zu gestalten (vgl. Ruiz u. Bay, 2008, Abschnitt 1). Ein einfaches Beispiel für Code mit und ohne dieses Pattern findet sich in Listing 4.1. Abbildung 4.1 zeigt eine grafische Darstellung der Interfaces, ihrer Methoden und Rückgabewerte sowie der implementierenden Klassen einschließlich ihrer Vererbungshierarchie.

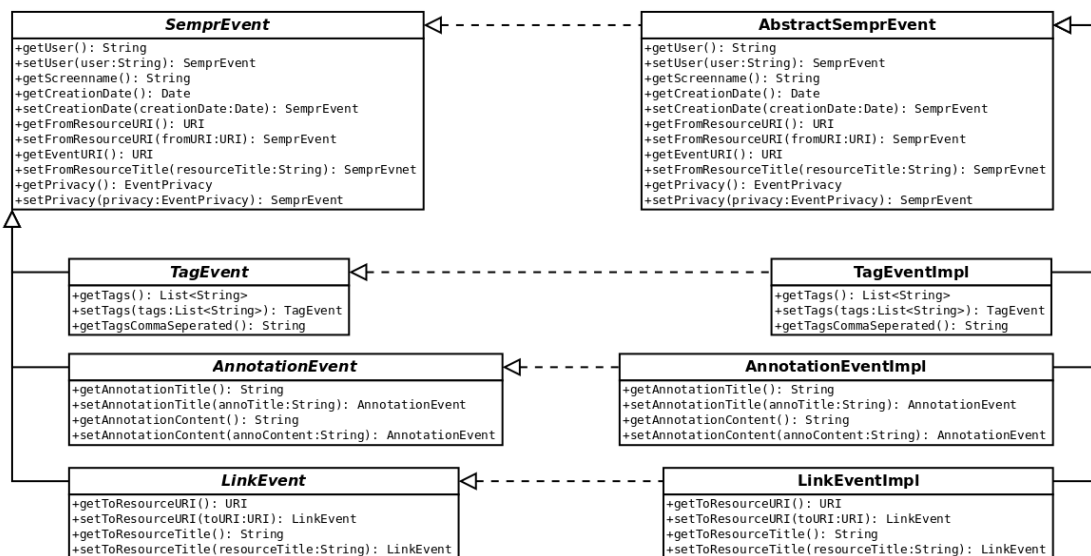


Abbildung 4.1: Interfaces und Implementierende Klassen des Objektmodells. Das Interface SemprEvent implementiert erbt außerdem vom Interface Serializable, was hier der Übersichtlichkeit halber ausgelassen wurde.

Listing 4.1: Beispielcode mit und ohne Method-Chaining

```

// Mit Method-Chaining: Setter geben Referenz auf aktuelles
// Objekt zurück.
final TagEvent someTagEvent = new TagEventImpl().setUser("
    Max").setFromResourceTitle("Irgendein Titel").
    setCreationDate(new Date());

// Ohne Method-Chaining: Setter geben nichts (void) zurück.
final TagEvent someTagEvent = new TagEventImpl();
someTagEvent.setUser("Max");
someTagEvent.setFromResourceTitle("Irgendein Titel");
someTagEvent.setCreationDate(new Date());
  
```

4.3 Dependency Injection und Guice

Sempr-Mobile wurde in einigen Aspekten nach dem *Inversion of Control* oder *Dependency Injection* Pattern entwickelt (vgl. Fowler, 2004). Dieses Pattern empfiehlt, dass sich eine Komponente in einer Software nicht selbst um die Instantiierung ihrer Abhängigkeiten kümmert, sondern diese einfach deklariert. Dies hat vielerlei Vorteile, da es unter anderem eine komplette Trennung zwischen Interface und Implementierung erlaubt, aber auch eine klarere Trennung verschiedener Aufgabenbereiche vereinfacht und damit die Wiederverwendbarkeit des geschriebenen Codes verbessert¹. Ein Framework zur Realisierung dieses Patterns ist *Google Guice*², welches in diesem Projekt Verwendung findet.

Mit diesem Framework ist es nichtmehr nötig, in jeder Activity Boilerplate-Code zum Finden verschiedener Views (wie Knöpfen oder Eingabefeldern) zu schreiben. Stattdessen injiziert das IoC-Framework diese in die hierfür annotierten Felder. Die Listings 4.2 und 4.3 zeigen exemplarisch den (gekürzten) Code einer Activity, einmal mit und einmal ohne Guice-Annotationen.

¹Weitere Implikationen von DI / IoC auf die Testbarkeit und Wartbarkeit werden im Aufsatz „Big Ball of Mud“ (Foote u. Yoder, 2000) näher erläutert.

²Google Guice ist unter <http://code.google.com/p/google-guice/> erhältlich. In diesem Projekt wurde ebenfalls der Android-Integrationslayer *RoboGuice* verwendet, welcher unter <http://code.google.com/p/roboguice/> zu finden ist.

Listing 4.2: Beispielcode einer Activity mit Guice

```
public final class PostEvent extends GuiceActivity
    implements OnCheckedChangeListener, OnClickListener {

    @InjectView(R.id.sendbutton)
    private Button sendButton;
    @InjectView(R.id.cancelbutton)
    private Button cancelButton;
    @InjectView(R.id.event_type_selection)
    private RadioGroup eventType;
    @InjectResource(R.string.hello_user)
    private String helloUser;

    /* weitere Felder */

    public PostEvent() {
        super();
    }

    protected void onCreate(final Bundle
        savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.postevent);
        eventType.setOnCheckedChangeListener(this);
        sendButton.setOnClickListener(this);
        cancelButton.setOnClickListener(this);
    }

    /* ... weitere Methoden ... */
}
```

Listing 4.3: Beispielcode einer Activity ohne Guice

```
public final class PostEvent extends GuiceActivity
    implements OnCheckedChangeListener, OnClickListener {

    private Button sendButton;
    private Button cancelButton;
    private RadioGroup eventType;
    private String helloUser;

    /* weitere Felder */

    public PostEvent() {
        super();
    }

    protected void onCreate(final Bundle savedInstanceState) {

        super.onCreate(savedInstanceState);
        setContentView(R.layout.postevent);

        // zusätzlich benötigter Code
        sendButton = (Button) this.getViewById(R.id.sendbutton);
        cancelButton = (Button) this.getViewById(R.id.cancelbutton
        );
        eventType = (RadioGroup) this.getViewById(R.id.
            event_type_selection);
        helloUser = this.getStringResource(R.string.hello_user)

        eventType.setOnCheckedChangeListener(this);
        sendButton.setOnClickListener(this);
        cancelButton.setOnClickListener(this);
    }

        /* ... weitere Methoden ... */
}
```

4.4 Verwendete Werkzeuge

Bei der Erstellung dieser Software waren weiterhin eine Vielzahl weiterer Tools behilflich, die hier kurz Erwähnung finden sollen. Als Entwicklungsumgebung diente die freie IDE Eclipse. Dabei war vor allem das sogenannte ADT-Plugin (kurz für *An-*

droid Development Tools) für Eclipse nützlich. Dieses erleichterte im hohen Maße das Entwickeln der Layout-Definitionen in XML, da unter anderem eine Vorschau-funktion enthalten ist und das entwickelte Layout ohne erneutes Bauen, Installieren und Ausführen angesehen werden kann.

Für die Layout-Entwicklung wurde ebenfalls das Werkzeug *Droid Draw*³ benutzt, welches eine vollständig grafische Komposition der verschiedenen Elemente erlaubt und das Layout anschließend in XML exportieren kann. Droid Draw befindet sich noch in einer frühen Entwicklungsphase, sodass ein manuelles Bearbeiten der XML-Dateien im Laufe dieser Arbeit nicht ganz erspart blieb.

4.5 Die fertige Applikation

In diesem Abschnitt werden kurz einige Impressionen der fertigen Software gegeben. Die Software selbst befindet sich auf dem Datenträger im Anhang dieser Arbeit. Abbildung 4.2 zeigt das Hauptmenü und die Übersicht aktueller Events nebeneinander. In Abbildung 4.3 sieht man die „Sempr-Option“ im Browser des Android-Systems sowie das Eingabeformular für neue Events.

³Erhältlich ist das Programm unter <http://www.droiddraw.org/>.

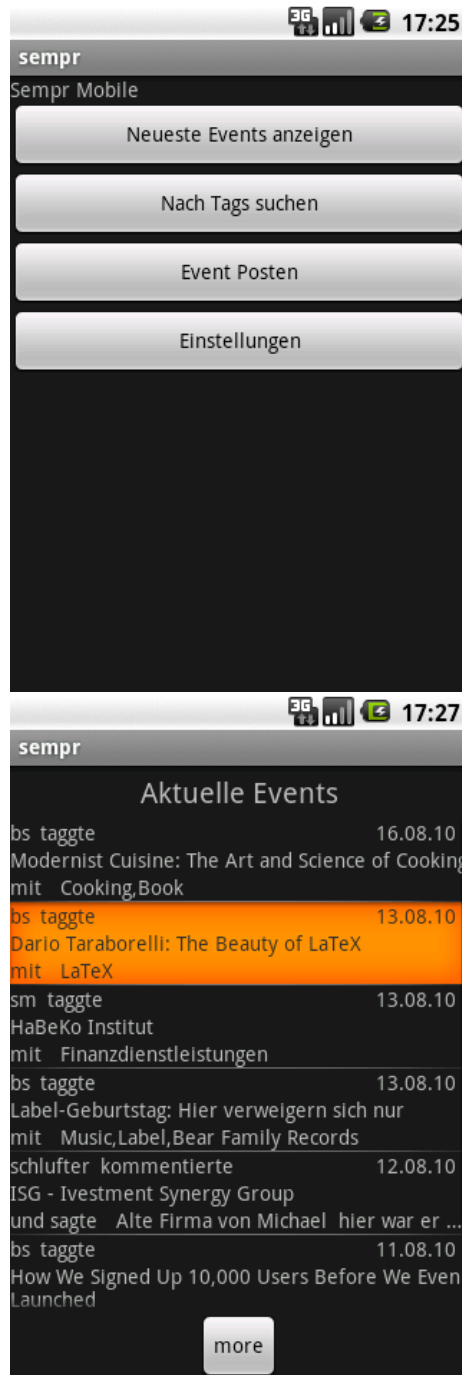


Abbildung 4.2: Hauptmenü (oben) und Übersicht aktueller Events in der fertigen Applikation (unten)

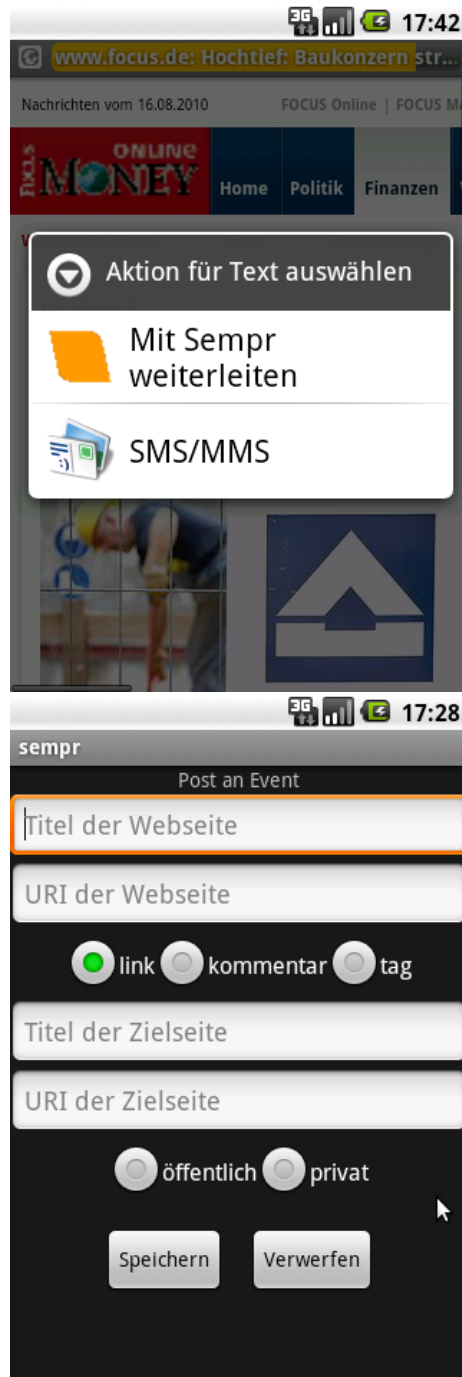


Abbildung 4.3: Möglichkeit URLs aus dem Browser an die Applikation zu senden (oben) sowie Eingabeformular für neue Events (unten)

Kapitel 5

Integrieren & Testen

5.1 Bau mit Maven & SDK

Innerhalb der Linkwerk GmbH wird der Großteil der Softwareprojekte mit dem Build-Tool Maven gebaut. Dies gewährleistet einen systematischen und systemunabhängigen Bauprozess und erlaubt eine Vielzahl von Automatisierungen in der Integration und Auslieferung der Produkte. So lassen sich dank Maven automatische Softwaretests anstoßen, sobald eine Änderung im Versionskontrollsystem registriert wird. Auch dieses Projekt wird daher mittels Maven gebaut.

Hilfreich ist hierbei, dass Maven, dank eines Tochterprojekts, bereits über eine Integration für Android-Projekte verfügt. Aufgrund einiger Eigenheiten der Plattform muss jedoch sichergestellt werden, dass das Android-SDK auf allen Systemen verfügbar ist, die später die Software bauen sollen. So ist die Sprache, in denen Anwendungen geschrieben werden, zwar standardkonformes Java, ausgeführt wird aber kein normaler Java-Bytecode. Stattdessen wird hier ein für die Plattform optimiertes Format namens Dalvik Executable (oder kurz Dex) (vgl. Meier, 2008, S. 14) ausgeführt. Die Übersetzung von Java-Classfiles zu Dex-Dateien erledigen Tools im SDK, welche Maven aufzurufen weiß. Durch die Integration dieses Projekts mit Maven wurde weiterhin der automatische Bau der Software mittels Hudson sowie die automatische Prüfung der Codequalität mit Sonar ermöglicht.

5.2 Testbarkeit

Das Ziel, die Software zu jedem Zeitpunkt einfach testbar zu halten, wurde erreicht. Die starke Modularisierung, welche nicht zuletzt durch konsequente Umsetzung des *Inversion of Control*-Patterns erreicht wurde, erleichtert ein abschnittweises Testen der Software.

Hierzu würde ein separates Test-Projekt¹ erstellt werden, das die zu testende Software als Abhängigkeit enthält. Aufgrund der bereits angerissenen Besonderheiten der Android-Plattform ist es nicht möglich, den entwickelten Code direkt auf dem Entwicklungsrechner auszuführen. Stattdessen wird hierfür der im SDK ausgelieferte Android-Emulator benutzt. Das Testprojekt würde dann mittels Maven gebaut, auf einer automatisch startenden Instanz des Emulators installiert und anschließend ausgeführt. Alle hierbei anfallenden Probleme werden registriert und von Hudson automatisch archiviert. Somit ist ein schnelles Feedback bei Programmierfehlern oder destruktiven Änderungen in etwaigen Schnittstellen gewährleistet.

¹Projekt ist hier nicht im Sinne eines organisatorischen Projektes, sondern im Sinne eines Software-Projektes gemeint. Im konkreten Fall heißt dies, das der Quellcode des Testprojektes vom Quellcode des zu testenden Projektes getrennt ist und nicht teil dessen.

Kapitel 6

Fazit & Ausblick

6.1 Allgemein

Die an Sempr-Mobile gestellten Anforderungen wurden erfüllt. Der entwickelte funktionierende Prototyp bietet eine solide Basis für die weitere Entwicklung. Nutzer von Sempr können mit diesem nun schnell und komfortabel auf Informationen zugreifen oder neue Informationen mit Anderen teilen.

So können im einzelnen:

- Events nach Aktualität, Webseite, Tag oder Nutzer abgerufen
- Tags, Links und Annotationen erstellt
- Webseiten vom Browser an Sempr-Mobile weitergeleitet
- Links zu Events an andere Anwendungen, wie Twitter oder Facebook, weitergeleitet werden.

Der Prototyp nutzt die Fähigkeiten der Plattform bereits besser als es eine mobile Webseite könnte und die Möglichkeiten, die sich hieraus ergeben, versprechen in Zukunft spannende Anwendungsfälle abzudecken. Die Testbarkeit ist aufgrund der modularen Struktur voll gewährleistet.

Die Arbeit hat gezeigt, dass es möglich ist, herstellerunabhängig Software für Smartphones zu entwickeln und dass Open-Source-Tools hierbei eine große Unterstützung sind. Insgesamt war die Entwicklung der Software für Entwickler mit Java-Hintergrund gut zu leisten und die Eigenheiten der Plattform erschienen nicht unnötig kompliziert. Weiterhin gewährleistet die Sprachwahl von Android, Java, das Erweitern der Funktionalität der eigenen Programme durch Software aus dem großen Fundus an Open-Source-Projekten sehr einfach.

6.2 Veröffentlichung der Anwendung

Die hier entwickelte Software sollte nach einer Testphase allen Nutzern von Sempr zur Verfügung gestellt werden. Die Plattform Android bietet hierfür mehrere Möglichkeiten. Die Simpelste wäre, das Programm auf der Webseite von Sempr zum Download anzubieten. Die meisten Android-Geräte besitzen die Fähigkeit, Software aus beliebigen Quellen, etwa der Speicherkarte des Telefons oder einer Webseite zu installieren. Es gibt zwei Gründe, die gegen diesen Weg sprechen: Erstens gibt es einige Gerätehersteller, die das Installieren von Applikationen auf dem Telefon auf den „offiziellen“ *Android Market* von Google beschränken, auf dem Entwickler auch kostenlos Programme anbieten können. Zweitens wäre der *Market* eine potenziell bessere Präsentationsfläche für das Programm, da dieser vermutlich für viele Nutzer die erste Anlaufstelle ist, wenn sie neue Software für ihr Gerät suchen.

6.3 Verbesserungen in Design, Layout und Benutzerführung

Ein Ziel dieses Projekts war es, eine für Endbenutzer komfortable Möglichkeit zu schaffen, Sempr auf seinem Mobiltelefon zu verwenden. Die hier entwickelte Software ist in ihrer jetzigen Form zwar voll funktions-, jedoch sicherlich noch ausbaufähig. Ein wichtiger Punkt ist hier die Präsentation, die momentan eher minimalistisch gehalten ist. Das entwickelte System erlaubt es, das Design und weitestgehend das Layout ohne Veränderung der Programmlogik den Wünschen der Nutzer anzupassen. Es gilt jedoch herauszufinden, wie diese Wünsche aussehen und wie sie sich am besten umsetzen lassen.

Ein weiterer Punkt ist die Benutzerführung. In der Architektur wurde berücksichtigt, das System in sich lose zu koppeln. Prinzipiell ist es also möglich, die Navigation

innerhalb der Software vergleichsweise einfach umzugestalten. Zukünftig sollte also evaluiert werden, welche Funktionen sich Nutzer von Sempr an welcher Stelle wünschen und wie sie diese aufrufen möchten.

6.4 Browserplugin oder eigener Browser

Vor Beginn dieser Arbeit wurden verschiedene Wege durchdacht, Sempr auf Android-Geräte zu bringen. Hierzu zählte eine auf Desktop-PCs mittlerweile recht übliche Methode - die Erweiterung der Funktionen des Browsers über ein Plugin. Dies hätte einige Vorteile mit sich gebracht. Sempr ist derzeit primär für das Web entwickelt, eine gewisse Nähe zum Browser hätte unter anderem direkten Zugriff auf die momentan geöffnete Webseite erlaubt. Leider ist nach derzeitigem Stand der Browser von Android nicht durch Plugins erweiterbar. Entsprechend müsste für eigene Funktionen auch ein eigener Browser entwickelt werden, was den Umfang dieser Arbeit deutlich überschritten hätte. Falls sich in Zukunft an der Erweiterbarkeit des Browsers etwas ändert, wäre dies mit Sicherheit ein möglicher Erweiterungspunkt für diese Software.

6.5 URI weiter gedacht

Langfristig sollen die Universalität von Ressourcen und ihren URIs eine höhere Bedeutung in Sempr bekommen. Mit Sempr-Mobile ist ein Grundstein gelegt, um die vorhandenen Dienste mobil nutzen zu können. Diese sollen aber die momentanen Alleinstellungsmerkmale mobiler Plattformen im Detail ausnutzen. Hierzu zählt zum Beispiel die Möglichkeit, den momentanen Standort des Nutzers bis auf wenige Meter genau zu bestimmen. Ein Ort kann dann ebenfalls als Ressource verstanden und in seiner Repräsentation als URI mit anderen Ressourcen verknüpft werden. So ist es denkbar, die Webseite einer Firma mit dem Standort ihrer Niederlassung zu verlinken oder interessante Orte in einer Stadt zu taggen.

Eine andere mögliche Weiterentwicklung wäre die Ressourcensichtweise auf Produkte. Nahezu alle Konsumgüter in Deutschland haben mittlerweile einen eigenen Barcode, in dem eine EAN bzw. IAN encodiert ist (Wikipedia, 2010a). Diese könnte direkt als URI genutzt werden, um anschließend das Annotieren von Produkten zu ermöglichen oder aber Produkte mit dem Webshop zu verlinken, der sie am günstigsten anbietet. Die Möglichkeiten sind also da und es gilt herauszufinden, welche

Anwendungsfälle sich hieraus entwickeln lassen - und nicht zuletzt, welche dieser Anwendungsfälle für den Nutzer einen Mehrwert im Management seiner Informationen bieten.

6.6 Performance-Optimierung

Zuletzt bleibt ein wichtiger Punkt, der in dieser Arbeit keine explizite Berücksichtigung fand: die Optimierung des Speicher- und Rechenbedarfs der Anwendung. Gerade für Ersteres bietet die Android-Plattform ein umfangreiches Lifecycle-System. Dieses erlaubt es dem Entwickler, die Komponenten seiner Anwendung sehr kleinschrittig in Lebensdauer und Stand-by-Verhalten zu steuern. So können nicht mehr benötigte Objektreferenzen nach der Verwendung explizit gelöscht werden, damit diese von der Garbage-Collection der Plattform aus dem Speicher entfernt werden können. Weiterhin birgt die Verarbeitung der Listen bis jetzt noch Optimierungspotenzial, da beim Anzeigen dieser (aus Effizienzsicht) noch sehr teure Operationen wiederholt durchgeführt werden. Dies ließe sich beispielsweise durch intelligentes Vorhalten bereits bekannter Resultate verbessern. Auch in der Web-Kommunikation der Anwendung herrscht noch Potenzial, das aber einer größeren Abstimmung mit dem Entwicklerteam bei Linkwerk bedarf. Im Internet ist das Caching von Webseiten üblich. Hiervon wird in der jetzigen Entwicklungsstufe von Sempr-Mobile noch keinerlei Gebrauch gemacht. Mitunter werden also bereits bekannte Ergebnisse mehrmals aus dem Netz abgerufen, was gerade in Anbetracht der geringen Bandbreite und hoher Latenz in Mobilfunknetzen nicht wünschenswert ist.

Abbildungsverzeichnis

1.1	Sempr-Toolbar (unten rechts) auf der Startseite von Google.	7
1.2	Das Sempr Dashboard, die Übersicht der aktuellsten Ereignisse auf Sempr.	8
2.1	Systemüberblick Sempr	12
2.2	Detailansicht Events (als Listen) anzeigen	12
2.3	Detailansicht Event anlegen	13
2.4	Detailansicht Event anlegen	13
2.5	Mockups von Hauptmenü sowie Listen-Ansicht mehrerer Events. . .	14
4.1	Interfaces und Implementierende Klassen des Objektmodells. Das Interface SemprEvent implementiert erbt außerdem vom Interface Serializable, was hier der Übersichtlichkeit halber ausgelassen wurde.	23
4.2	Hauptmenü (oben) und Übersicht aktueller Events in der fertigen Applikation (unten)	28
4.3	Möglichkeit URLs aus dem Browser an die Applikation zu senden (oben) sowie Eingabeformular für neue Events (unten)	29

Literaturverzeichnis

- [Foote u. Yoder 2000] FOOTE, Brian ; YODER, Joseph: Big ball of mud. In: *Pattern languages of program design 4* (2000), Nr. 654-692, S. 99
- [Fowler 2004] FOWLER, Martin: *Inversion of Control Containers and the Dependency Injection pattern*. Version: Januar 2004. <http://martinfowler.com/articles/injection.html>, Abruf: 10.07.2010
- [Google 2010a] GOOGLE, Inc: *The AndroidManifest.xml File*. Version: Juli 2010. <http://developer.android.com/guide/topics/manifest/manifest-intro.html>, Abruf: 30.07.2010 (Android Developers Guide)
- [Google 2010b] GOOGLE, Inc: *Intents and Intent Filters*. Version: Juli 2010. <http://developer.android.com/guide/topics/intents/intents-filters.html>, Abruf: 30.07.2010 (Android Developers Guide)
- [Ihlenfeld 2010] IHLENFELD, Jens: *Apple schränkt iPhone-Entwicklerwerkzeuge ein*. Version: April 2010. <http://www.golem.de/1004/74361.html>, Abruf: 30.07.2010
- [Kirsch 2009] KIRSCH, Christian ; HEISE.DE (Hrsg.): *Gartner sieht steilen Aufstieg für Android*. Version: Oktober 2009. <http://www.heise.de/newsticker/meldung/Gartner-sieht-steilen-Aufstieg-fuer-Android-817664.html>, Abruf: 30.07.2010
- [Meier 2008] MEIER, Reto: *Professional Android Application Development*. 1. Auflage. Indianapolis : John Wiley & Sons, 2008. – 409 S. – ISBN 0-470-34471-7
- [Mintert 2010] MINTERT, Stefan ; LINKWERK GMBH (Hrsg.): *SEMPR.NET Introduction*. Version: Dezember 2010. <http://www.sempr.net/introduction/>, Abruf: 15.07.2010

- [Ruiz u. Bay 2008] RUIZ, Alex ; BAY, Jeff: *An Approach to Internal Domain-Specific Languages in Java*. Version: Februar 2008. <http://www.infoq.com/articles/internal-dsls-java>, Abruf: 30.07.2010
- [Sullivan 2010] SULLIVAN, John ; FREE SOFTWARE FOUNDATION, Inc (Hrsg.): *Why free software and Apple's iPhone don't mix*. Version: Mai 2010. <http://www.fsf.org/blogs/community/why-free-software-and-apples-iphone-dont-mix>, Abruf: 30.07.2010
- [Wikipedia 2010a] WIKIPEDIA: *European Article Number*. Version: Juli 2010. http://de.wikipedia.org/w/index.php?title=European_Article_Number&oldid=76370135, Abruf: 20.07.2010. – [Online]
- [Wikipedia 2010b] WIKIPEDIA: *Wikipedia:Sprachen*. Version: Juli 2010. <http://de.wikipedia.org/w/index.php?title=Wikipedia:Sprachen&oldid=77149025>, Abruf: 30.07.2010

Glossar

- Action** ein für Android-Intents zwingend erforderliches Feld, in dem die zu verrichtende Tätigkeit als String beschrieben wird, beispielsweise ACTION_SEND.
- Activities** Konzept und Klasse von Android. Eine Activity verrichtet Aufgaben, wird von Intents gestartet und startet auch die grafische Benutzeroberfläche hierfür.
- Dalvik** Die virtuelle Maschine auf der Android-Plattform. Sie führt die sogenannten Dalvik Executables aus, also den für Android optimierten Bytecode.
- Dex** Abkürzung für Dalvik Executable, siehe Dalvik.
- Extra** unter Android bezeichnen Extras Objekte eines beliebigen (serialisierbaren) Typs, die an einen Intent angehängt werden können.
- HTTP-Proxy** Vom lat. Wort für Stellvertreter abgeleitet, er stellt für Clients die Verbindung zu Webseiten her und liefert den Clients die gewünschten Daten zurück.
- Hudson** ein freier Continuous Integration Server, der Software kontinuierlich aus einem Versionskontrollsystem auschecken und bauen kann. Hierdurch sollen Fehler und Probleme schnell erkannt werden.
- Intent** ein Konzept von Android, das dort als Klasse realisiert ist. Ein Intent bezeichnet eine Absicht, formuliert in einem einfachen Nachrichtenformat.
- Layout (Datei)** unter Android ein XML-Format in dem eine grafische Benutzeroberfläche in XML definiert wird.
- Layout (View)** eine Unterart von Views die keine eigenen Elemente definieren, sondern die Anordnung der Kindelemente steuern. Exemplare hiervon sind die Klassen LinearLayout oder TableLayout.
- Manifest** unter Android eine XML-Datei in der alle Berechtigungen und Aktivitäten einer Anwendung deklariert werden.

- Maven** Build-Tool das primär für das Java-Umfeld entwickelt wurde. Es hilft sowohl beim systematischen Bau als auch beim Verwalten von Abhängigkeiten.
- Parcelable** ein Java-Interface der Android-Plattform das die Serialisierbarkeit von Instanzen implementierender Klassen kennzeichnet.
- Pattern** übersetzt Entwurfsmuster, in der Softwareentwicklung gängige Vorgehensweisen, klar definierte Problemstellungen zu lösen.
- Ressource** Ein Ding, eine Sache, mitunter durch einen URI zu beschreiben. Beispiel: eine Webseite, ein Buch, eine Person
- SDK** Abkürzung für Software Development Kit, meist eine Sammlung von Programmen und Bibliotheken zur Entwicklung von Software für eine bestimmte Plattform.
- Sempr** Kollaborationssoftware für das Web der Firma Linkwerk GmbH
- Serializable** ein Interface der Java-Standardbibliothek, das die Serialisierbarkeit von Instanzen implementierender Objekte kennzeichnet.
- Smartphone** Telefon mit vergleichsweise hoher Rechenkapazität, großer Programmauswahl sowie meist Internetfähigkeit
- Sonar** Eine Software zur Analyse von Code-Qualität anhand verschiedener Metriken.
- Tag** Modebegriff für ein freies vergebenes Schlagwort, siehe auch taggen.
- taggen** freies Verschlagworten von Inhalten. Benutzer haben keine im Vorwege definierte Taxonomie, sondern vergeben Schlagworte so, wie Sie sie subjektiv für sinnvoll erachten.
- Twitter** ein social-networking Dienst, bei dem Nutzer auf 140 Zeichen begrenzte Statusnachrichten veröffentlichen können.
- URI** Uniform Resource Identifier, ein Bezeichner für Ressourcen nach RFC 2376.
- URL** Uniform Resource Locator, ähnlich wie URI, beschreibt aber gleichzeitig den Abrufort der beschriebenen Ressource, siehe hierzu auch RFC 1738.
- View** ein oder mehrere zusammengefasste Elemente einer Benutzeroberfläche unter Android. Beispielsweise ein Eingabefeld oder eine scrollbare Liste.
- WLAN-Access-Point** ein Zugangspunkt zu einem Wireless-LAN Netzwerk, an dem sich Client-Systeme anmelden

Anhang A

Codeausschnitte

A.1 Kommentierter Code einer Activity

Der hier gezeigte Code ist auch im Quellcode-Archiv auf der CD dieser Arbeit unter *src/com/linkwerk/sempr/activities/DisplaySingleEvent.java* zu finden.

```
package com.linkwerk.sempr.activities;

import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.view.View.OnClickListener;
import android.webkit.WebView;
import android.widget.TableLayout;
import android.widget.TableRow;
import android.widget.TextView;
import com.google.inject.internal.Nullable;
import com.linkwerk.sempr.R;
import com.linkwerk.sempr.intents.ShowEventIntent;
import com.linkwerk.sempr.intents.ShowEventsForTagIntent;
import com.linkwerk.sempr.intents.ShowUsersEventsIntent;
import com.linkwerk.sempr.intents.ShowWebpageIntent;
import com.linkwerk.sempr.model.AnnotationEvent;
import com.linkwerk.sempr.model.LinkEvent;
import com.linkwerk.sempr.model.SemprEvent;
import com.linkwerk.sempr.model.TagEvent;
import com.linkwerk.sempr.util.CommentStyler;
import com.linkwerk.sempr.util.DateFormats;
import roboguice.activity.GuiceActivity;
import roboguice.inject.InjectExtra;
import roboguice.inject.InjectView;

import java.util.ArrayList;
```

```

import java.util.List;

import static com.linkwerk.sempr.util.EventDisambiguation.*;

/* Diese Activity hat die Aufgabe, einzelne Events darzustellen sowie die
   Interaktion des Nutzers mit diesen zu steuern */
public final class DisplaySingleEvent extends GuiceActivity implements
    OnClickListener {

    // Ein willkürlich gewählter Wert, der im Folgenden zur Unterscheidung von
    Views benötigt wird.
    private static final String TAG_IDENTIFIER = "IAmATag";
    // Ein simpler Tag der den Logmeldungen aus dieser Activity vorangestellt
    wird.
    private static final String TAG = "DisplaySingleEvent";

    /* Durch die Annotation InjectExtra fügt Guice an dieser Stelle den Inhalt
       des hier genannten Extras aus dem Intent. In diesem Fall also das
       Event, das diese Activity anzeigen soll. */
    @InjectExtra(value = ShowEventIntent.EXTRA_EVENT)
    private SemprEvent event;

    // Die Annotation veranlasst Guice an dieser Stelle den View mit der
    übergebenen ID zu injecten. In diesem Beispiel den Textblock, in dem
    der Titel der Seite dargestellt werden soll.
    @InjectView(R.id.pagetitle)
    private TextView pageTitle;
    @InjectView(R.id.username)
    private TextView userName;
    @InjectView(R.id.pageurl)
    private TextView pageUri;
    @InjectView(R.id.eventdate)
    private TextView eventDate;
    // z.B. "tagged", "linked"
    @InjectView(R.id.event_action1)
    private TextView eventAction1;
    // z.B. "and said", "with"
    @InjectView(R.id.event_action2)
    private TextView eventAction2;

    @InjectView(R.id.toResourceURI) @Nullable
    private TextView toResourceURI;

    @InjectView(R.id.tagtable) @Nullable
    private TableLayout tagTable;

    @InjectView(R.id.toResourceTitle) @Nullable
    private TextView toResourceTitle;

    public DisplaySingleEvent() {
        // Dieser Konstruktor würde auch vom Compiler automatisch erstellt, ist
        hier aber der Vollständigkeit halber enthalten.
        super();
    }

    /* Diese Methode ist der Einstieg für jede Activity. */
    public void onCreate(final Bundle bundle) {
        // Der Aufruf der onCreate-Methode der Superklasse ist vorgeschrieben.
        super.onCreate(bundle);
    }

```

```

Log.d(TAG, "Recieved_event_with_uri"
      + event.getEventURI().toString());

/* Hier beginnt die Fallunterscheidung für das Event, das im Intent
   migesendet wurde. Die statischen Methoden isTagEvent, isLinkEvent
   sowie isAnnoEvent gewährleisten hierbei die Typsicherheit. */
if (isTagEvent(event)) {

    final TagEvent tagEvent = (TagEvent) event;

    // Die Methode setContentView setzt die Anzeige auf die Layout-Datei
    // mit der Übergebenen ID. In diesem Fall entspricht sie der XML-
    // Datei res/layout/show_tagevent.xml.
    this.setContentView(R.layout.show_tagevent);

    // Die IDs die mit R.string beginnen entsprechen Strings in den
    // Dateien res/values/strings.xml bzw. res/values-de/strings.xml (
    // ersteres mit der englischen und letzteres mit der deutschen
    // Übersetzung).
    eventAction1.setText(R.string.tagged);
    eventAction2.setText(R.string.with);

    // Tags werden in einer Tabelle dargestellt mit einer Zeile pro Tag.
    for (final String tag : tagEvent.getTags()) {
        final TableRow row =
            new TableRow(getApplicationContext());
        final TextView text =
            new TextView(getApplicationContext());

        text.setText(tag);
        text.setOnClickListener(this);
        // Dies erlaubt es in der Methode onClick nachzuvollziehen, ob das
        // angeklickte Objekt ein Tag ist.
        text.setTag(TAG_IDENTIFIER);
        row.addView(text);
        tagTable.addView(row);
    }

} else if (isLinkEvent(event)) {

    this.setContentView(R.layout.show_linkevent);

    final LinkEvent linkEvent = (LinkEvent) event;

    eventAction1.setText(R.string.linked);
    eventAction2.setText(R.string.with);

    toResourceURI.setText(linkEvent.getToResourceURI().toString());

    // Dies übergibt diese Activity als Listener an den View und macht ihn
    // in diesem Zuge automatisch klickbar.
    toResourceURI.setOnClickListener(this);
    toResourceTitle.setText(linkEvent.getToResourceTitle());

} else if (isAnnoEvent(event)) {

    this.setContentView(R.layout.show_annotationevent);

```

```

    final AnnotationEvent commentEvent = (AnnotationEvent) event;
    final WebView commentWebView = (WebView) findViewById(R.id.comment);

    eventAction1.setText(R.string.commented_on);
    eventAction2.setText(R.string.and_said);
    eventAction2.setVisibility(View.GONE);
    pageUri.setVisibility(View.GONE);
    pageTitle.setVisibility(View.GONE);

    // Hier wird ein HTML-Fragment (welches in jedem AnnotationsEvent
    // enthalten ist) in ein vollständiges HTML-Dokument verwandelt und
    // anschließend in den WebView geladen. Der WebView zeigt HTML-
    // Dokumente als Teil der Benutzeroberfläche an und erlaubt auch die
    // Interaktion mit der geladenen Webseite.
    commentWebView.loadData(
        CommentStyler.getStyledHTML(commentEvent), "text/html", "utf-8");

} else {
    Log.e("sempr", "Unsupported_event_type" + event.getClass());
}

pageTitle.setText(event.getFromResourceTitle());

eventDate.setText(DateFormats.getShort()
    .format(event.getCreationDate()));

pageUri.setText(event.getFromResourceURI().toString());
pageUri.setOnClickListener(this);

userName.setText(event.getUser());
userName.setOnClickListener(this);
}

/* Die Methode onClick wird immer dann aufgerufen, wenn ein View-Objekt
   dessen Listener diese Activity ist, angeklickt wird. Zur
   Fallunterscheidung wird der View mitgesendet. */
public void onClick(final View view) {

    final int viewId = view.getId();

    // Die ersten beiden Fälle können bei allen Arten von Events auftreten
    // und werden deshalb zuerst geprüft.
    if (viewId == userName.getId()){

        // Klickt der Nutzer auf einen Usernamen werden weitere Events dieses
        // Nutzers angezeigt.
        this.startActivity(new ShowUsersEventsIntent(event.getUser()));

    } else if (viewId == pageUri.getId()) {

        // Der Klick auf die URI ruft einer Webseite ruft diese im Browser auf
        // .
        this.startActivity(new ShowWebpageIntent(event.getFromResourceURI()));

    } else if (isTagEvent(event)) {
        if (view.getTag().equals(TAG_IDENTIFIER)){
            // Wird ein Tag angeklickt wird ein ShowEventsForTagsIntent

```

```

        abgeschickt, dem Nutzer also weitere Events mit diesem Tag
        angezeigt.
        final String clickedTag = ((TextView) view).getText().toString();

        // Dieser Intent wird auch für die Tag-Suche verwendet, unterstützt
        // also prinzipiell auch mehrere Suchbegriffe/Tags, daher die
        // Liste.
        final List<String> tagList = new ArrayList<String>();
        tagList.add(clickedTag);

        this.startActivity(new ShowEventsForTagIntent(tagList));
    }
} else if (isLinkEvent(event)) {
    if (viewId == toResourceURL.getId()) {
        this.startActivity(
            new ShowWebpageIntent(
                ((LinkEvent) event).getToResourceURL());
    }
} else if (isAnnoEvent(event)) {
    // Die interaktiven Funktionen bei Annotations-Events werden durch den
    // WebView abgedeckt, müssen hier also nicht berücksichtigt werden.
} else {
    // Sollte nie vorkommen, der Ordnung halber hier trotzdem eine
    // Logmeldung.
    Log.w(TAG, "Clicked something which we could not identify.");
}
}
}
}

```

A.2 Kommentiertes XML eines Views

Der hier gezeigte Code ist auch im Quellcode-Archiv auf der CD dieser Arbeit unter *res/layout/show_tagevent.xml* zu finden. Dieses Layout wird auch im Codebeispiel aus dem vorigen Abschnitt verwendet, um ein Tag-Event darzustellen.

```

<?xml version="1.0" encoding="utf-8"?>
<!-- Element LinearLayout ordnet die Kind-Views untereinander an,
mit einem View pro Zeile. -->
<LinearLayout android:id="@+id/widget45"
    android:layout_width="fill_parent" android:layout_height="
    fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">

    <!-- ScrollView macht die Kind-Views scrollbar, sobald diese zu
    groß für die Darstellung auf einem Bildschirm sind. -->
    <ScrollView android:layout_height="fill_parent"
        android:layout_width="fill_parent">

        <!-- Verhält sich wie obriges LinearLayout, durch Angabe von
        android:orientation="vertical" wird der Inhalt jedoch nicht

```

```
unter- sondern nebeneinander und in einer Zeile
dargestellt. —>
<LinearLayout android:layout_width="fill_parent"
  android:layout_height="wrap_content" android:orientation="
  vertical">

  <!-- include fügt an dieser Stelle den Inhalt der Datei
  showevent_common.xml ein. Hier sind die Felder enthalten,
  die allen Events gemeinsam sind. Hierzu zählen der
  Nutzer, die Web-URI und der Seitentitel. —>
  <include layout="@layout/showevent_common"
    android:layout_width="fill_parent" android:layout_height="
    wrap_content" />

  <!-- Diese Tabelle wird im Code um eine Zeile pro
  darzustellenden Tag erweitert. —>
  <TableLayout android:layout_width="wrap_content"
    android:id="@+id/tagtable" android:layout_height="
    wrap_content"
    android:layout_marginLeft="25px" android:padding="10px" />
</LinearLayout>
</ScrollView>
</LinearLayout>
```

Eidesstattliche Versicherung

Ich versichere, die vorliegende Arbeit selbstständig ohne fremde Hilfe verfasst und keine anderen Quellen und Hilfsmittel als die angegebenen benutzt zu haben. Die aus anderen Werken wörtlich entnommenen Stellen oder dem Sinn nach entlehnten Passagen sind durch Quellenangabe kenntlich gemacht.

Hamburg, den 30. August 2010 Martin Knudsen