Android – Eine Einführung

Layouts, Views & Adapter

Andreas Wilhelm

Institut für Informatik
Georg-August-Universität Göttingen

www.avedo.net

Menüs

Contents

Allgemeines

- Wichtige grafische Elemente zur Navigation und Arbeit auf Daten
- Android stellt umfangreiche Menü-APIs bereit
- Einheitliche Benutzeroberflächen in verschiedenen Applikationen
- Leider viele Änderungen in den letzten APIs
- Seit Android 3.0 nicht länger zwingend notwendig, dass ein Android-Endgerät speziellen Menü-Button zur Verfügung stellt
- Aktuell vier verschiedene Arten von Menüs unterstützt

Optionsmenü & ActionBar

- Optionsmenü ist das primäre Menü einer Applikation
- Bisher sechs Aktionen am unteren Rand des Bildschirms
- Alternativ fünf Aktionen und weiterer Button um Liste weiterer Aktionen anzuzeigen
- Seit Android 3.0 Integration in Overflow-Menü der ActionBar
- Angebot von Kontext unabhängigen Aktionen und Optionen
- Beispielsweise Suche, Import und Export, sowie Einstellungen der Applikation
- Bis Android 2.3 konnte dieses Menü *immer* über Menü-Button geöffnet werden

Kontextmenü

- Wird in einem Dialog angezeigt
- Öffnen durch langen Klick auf Element der Benutzeroberfläche
- Erlaubt Aktionen auf den selektierten Inhalten
- Beispielsweise Editieren eines Inhalts
- In Android 3.0 wurde dieses Menü teilweise durch den sogenannten *Contextual Action Mode* abgelöst

PopUp-Menü

- Stellt Aktionen in Liste unter aufrufendem View zur Verfügung
- Ausführung Kontext sensitiver Aktionen ohne Inhalte zu ändern
- Typisches Beispiel ist die Text basierte Wörterbuchverwaltung in einem SMS-Client
- Es ist stehts auf die strikte Unterscheidung zwischen Kontextmenüs und PopUp-Menüs zu achten

Menü-Deklaration

- Deklaration im XML-Format
- Beeinflussung des Layouts durch Attribute
- XML-Deklaration wird unter beliebigem Dateinamen im Ordner *res/menu/* hinterlegt

Attribut	Beschreibung
android:id	Eindeutige ID des Menüeintrags
android:icon	Icon des Menüeintrags
android:title	Titel des Menüeintrags
android:showAsAction	Anzeigeverhalten in der ActionBar
android:orderInCategory	Beeinflussung der Reihenfolge der menüeinträge (Aktivitäten & Fragmente)

Menü-Elemente

Element	Beschreibung
<menu>	Erzeugt ein Menü dessen Einträge in *<item>* Umgebungen deklariert werden. *<menu>*-Element muss die Wurzel eines Menübaums sein, der dann beliebige Verschachtelungen von *<menu>*, *<item>* und *<group>* Elementen enthalten kann.
<item>	Repräsentiert einen einzelnen Menüeintrag. Ein in diesem Element verschachteltes *<menu>* Element stellt ein Untermenü dar.
<group>	Ein unsichtbarer Kontainer, der optional dazu genutzt werden kann um gemeinsame Attribute für die darin liegenden *<item>*-Elemente zu definieren.

Untermenüs

Um ein Untermenü nutzen zu können, muss das Menü an entsprechender Stelle mit Hilfe eines *MenuInflaters* geladen werden. Dies gilt auch für normale Menüs, die beispielsweise in der Methode *onCreateOptionMenu()* initialisiert werden sollen.

Beispiel

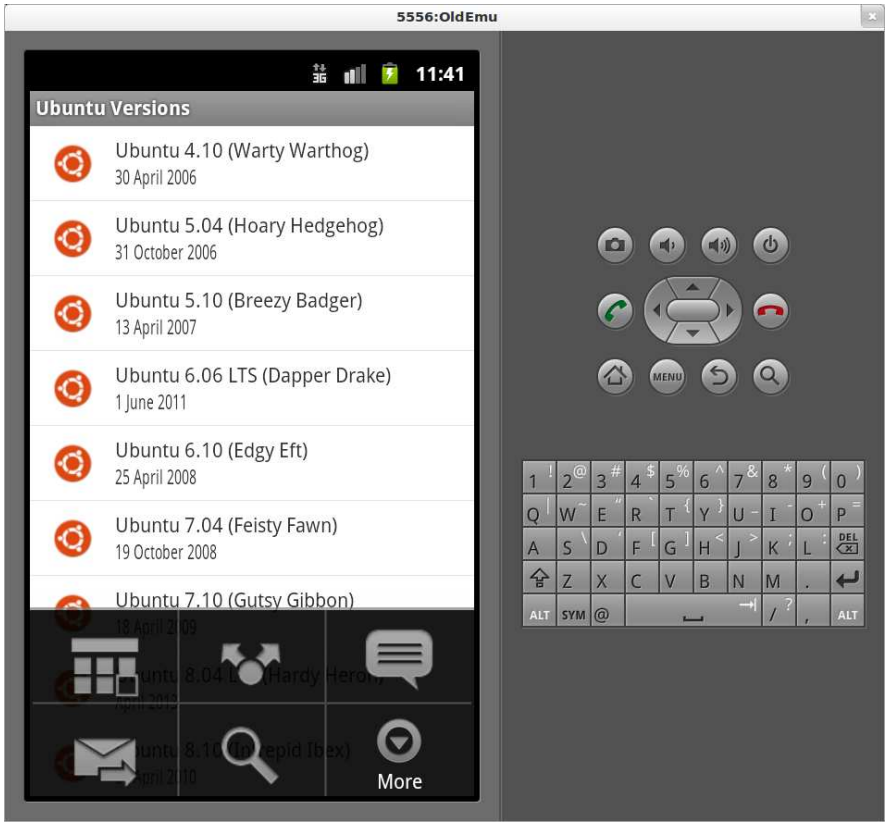
```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_calendar"
        android:title="@string/menu_calendar"
        android:icon="@drawable/menu_icon_calendar"
        android:showAsAction="ifRoom" />
    <item
        android:id="@+id/menu_share"
        android:title="@string/menu_share"
        android:icon="@drawable/menu_icon_share"
        android:showAsAction="ifRoom" />
    <item
        android:id="@+id/menu_chat"
        android:title="@string/menu_chat"
        android:icon="@drawable/menu_icon_chat" />
    <item
        android:id="@+id/menu_send"
        android:title="@string/menu_send"
        android:icon="@drawable/menu_icon_send" />
    <item
        android:id="@+id/menu_search"
        android:title="@string/menu_search"
        android:icon="@drawable/menu_icon_search"
        android:showAsAction="ifRoom" />
    <item
        android:id="@+id/menu_settings"
        android:title="@string/menu_settings"
        android:showAsAction="never"
        android:icon="@drawable/menu_icon_settings" />
</menu>
```

Listing : Die menü Deklaration

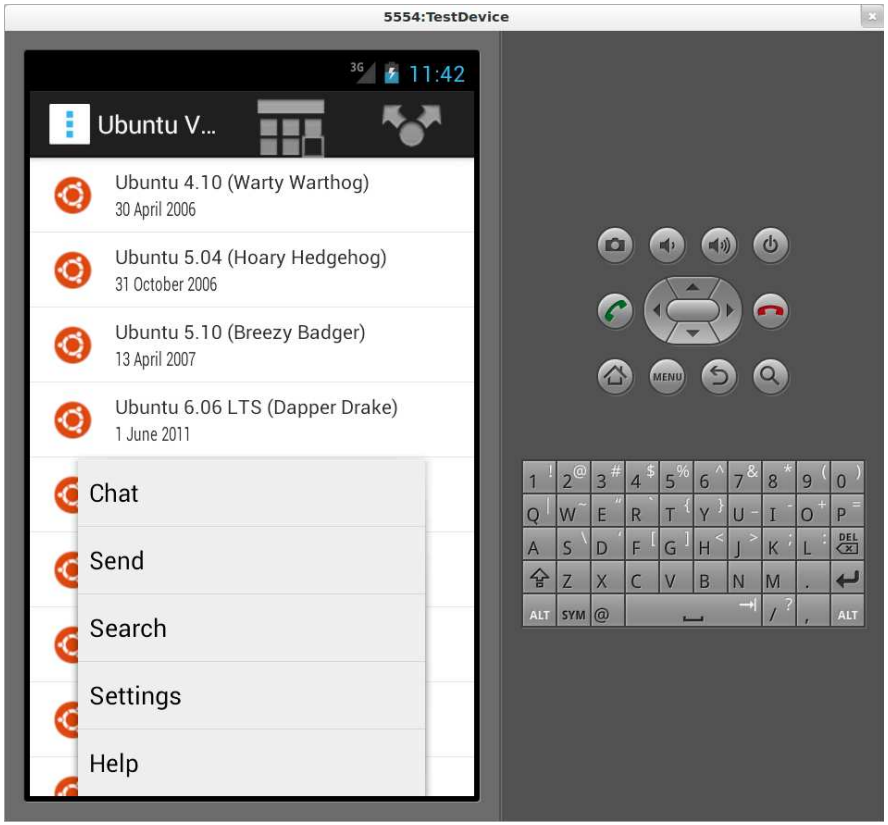
Allgemeines

- Stellt Kontext unabhängige Aktionen bereit
- Typische Aktionen Suchen, Ändern der Einstellungen und Exportieren von Daten
- Bis Android 2.3.x am unteren Rand des Bildschirms positioniert
- Normalerweise enthält es sechs Aktionen
- Alternativ fünf Aktionen und weiterer Button um Liste weiterer Aktionen anzuzeigen
- Seit Android 3.0 anzeige der Einträge im Overflow-Menü der ActionBar
- Overflow-Menü kann über Overflow-Icon am rechten Rand der ActionBar oder Menü-Button geöffnet werden
- Anzeige Overflow-Menü als vertikale Liste am unteren Bildschirmrand
- Aktionen können mit *android:showAsAction* Attribut in ActionBar verschoben werden

Optionsmenü bis 2.3 und ab 3.0



(a) Optionsmenü bis Android 2.3.x



(b) Optionsmenü ab Android 3.0

Abbildung: Ubuntu Release Menü

Hinweise

Dartsellung von Optionsmenüs

Durch Änderung der Darstellung seit Android 3.0 sehen Optionsmenüs unter älteren Systemen anders aus als unter neuen.

Es ist möglich eine eigene ActionBar zu implementieren und diese in älteren Versionen von Android zu nutzen. Ein Beispiel findet man in den mit dem Android-SDK mitgelieferten Beispielen.

Android-SDK Beispiele

Das Android-SDK bringt viele Beispiele mit sich, die Anregungen für Ansätze zu Problemlösung liefern, mit sich.

Zur besseren Übersichtlichkeit wird mit jeder API-Version ein eigenes Paket an Beispielen geschnürrt, dass mit dem Android-SDK-Manager heruntergeladen werden kann.

Verfügbar sind die Beispiele im Unterordner *samples* des Installationsverzeichnisses des Android-SDKs von wo aus sie einfach geladen und ausgeführt werden können.

Implementierung

- Optionsmenüs können in Aktivitäten oder Fragmenten eingebunden werden
- Überschreiben der Methode *onCreateOptionsMenu()*
- Seit Android 3.0 wird die Methode nicht mehr beim ersten Verwenden des Menüs aufgerufen, sondern eim Starten der Aktivität bzw. des Fragments
- Menüs können mit Hilfe des *MenuInflaters* geladen werden
- *Menu*-Objekt ermöglicht hinzufügen von Einträgen mit *add()* und Laden von Einträgen mit *findItem()*

Reihenfolge von Menüeinträgen

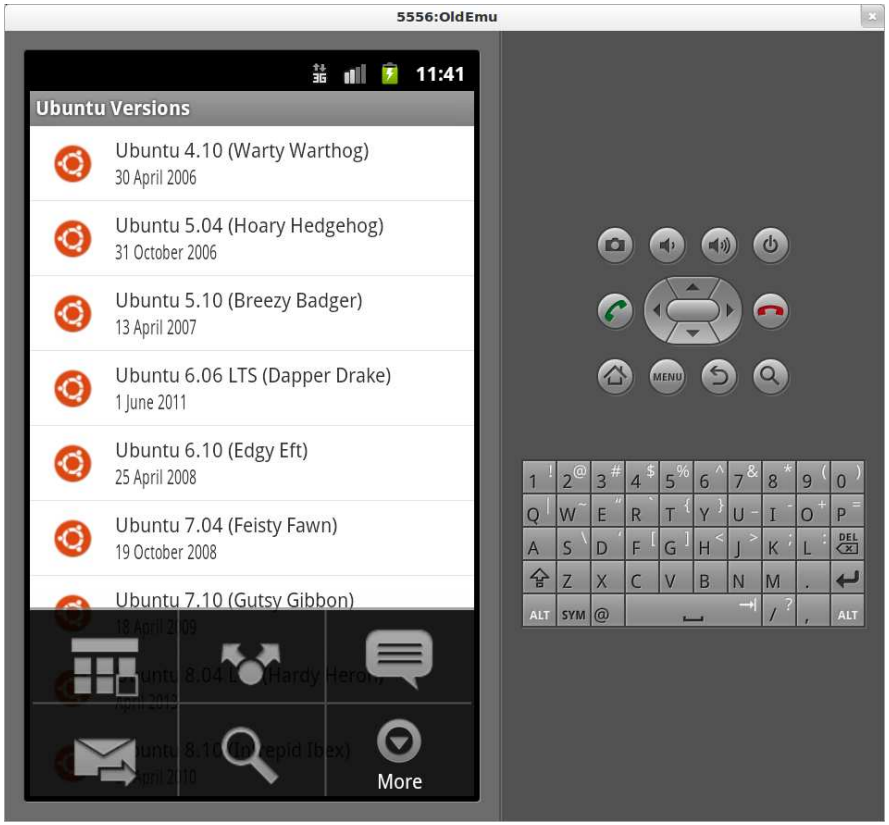
Aktivitäten und Fragmente können Optionsmenüs implementieren. Sollten beide ein Optionsmenü einbinden, so werden diese von Android zusammengeführt und in einem gemeinsamen Menü angezeigt. Einträge des durch die Aktivität spezifizierten Menüs vor den Einträgen des oder der Fragmente. Reihenfolge der Einträge kann in den <*item*>-Elementen mit dem Attribut *android:orderInCategory* beeinflusst werden.

Beispiel

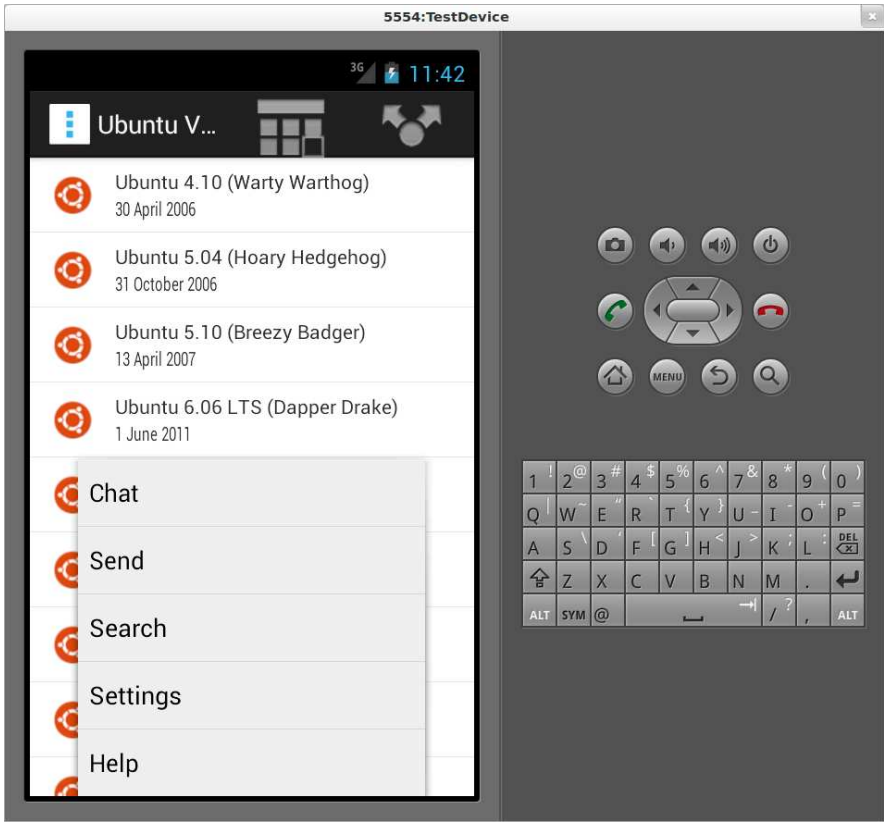
```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.ubuntu_menu, menu);
    return true;
}
```

Listing : Das Optionsmenü

Screenshot



(a) Optionsmenü bis Android 2.3.x



(b) Optionsmenü ab Android 3.0

Abbildung: Ubuntu Release Menü

Interaktion

- Event-Behandlung findet in der Methode*onOptionsItemSelected()* statt
- Methode bekommt Objekt des ausgewählten Eintrags übergeben
- Operation kann dann anhand der ID identifiziert werden
- Falls ausgewählte Aktion ausgeführt werden ist Rückgabewert *true* andernfalls der der super-Methode (*false*)

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    // Handle item selection
    Toast.makeText(this, "Selected " + item.getTitle(), Toast.LENGTH_LONG).show();
    return true;
}
```

Listing : Optionsmenü Events

Aktualisierung

- Initialisierung eines Menüs findet in *onCreateOptionsMenu()* statt
- Aufruf nur beim starten der Aktivität bzw. des Fragments
- Dynamische Änderungen der Menüstruktur können in der Methode *onPrepareOptionsMenu* vorgenommen werden
- Methode kann in Aktivitäten und Fragmenten implementiert werden
- Bekommt im Gegensatz zu *onCreateOptionsMenu()* den aktuellen Zustand des Menüs übergeben

Initialisierung und Aktualisierung

Initialisierung und Aktualisierung von Optionsmenüs sollten nicht vermischt werden. Die Initialisierung sollte in *onCreateOptionsMenu()* und die Aktualisierungen in *onPrepareOptionsMenu()* vorgenommen werden.

Hinweise zur Implementierung

Abarbeitung von Events

Sollten sowohl eine Aktivität und Fragmente ein Menü implementiert worden sein, wird erst die *onOptionsItemSelected()* Methode der Aktivität und dann erst die der Fragmente nacheinander aufgerufen bis *true* oder *false* zurückgegeben wird.

Behandlung von onClick-Events

Seit Android 3.0 ist es möglich das XML-Attribut *android:onClick* auch für Menü-Einträge zu deklarieren.

Aktualisierung des Menüs

Da seit Android 3.0 das Optionsmenü in die ActionBar integriert wird, wird nicht jedesmal beim Anzeigen des Menüs die Methode *onPrepareOptionsMenu()* aufgerufen, da die ActionBar während der ganzen Laufzeit sichbar ist. Man muss daher seit Android 3.0 die Methode *invalidateOptionsMenu()* Aufrufen, um das System zu einem Aufruf von *onPrepareOptionsMenu()* zu veranlassen.

Allgemeines

- Aktionen, die auf Inhalten und Elementen der aktuellen grafischen Oberfläche ausgeführt werden sollen
- Kontextmenü ist an ein einzelnes View gebunden
- Es können verschiedene Kontextmenüs an verschiedene Views gebunden werden
- Einsatz von Kontextmenüs macht vorallem bei AdapterViews Sinn
- Menü wird durch einen langen Klick auf das dazugehörige View geöffnet
- Darstellung von Menü-Einträgen in einer scollbaren Liste in einem Dialog
- Alternativ zur Ausgabe als Liste kann eine gesonderte ActionBar geöffnet werden
- Views können sich für Kontextmenü mit *registerForContextMenu()* registrieren
- Initialisierung des Menüs in *onCreateContextMenu()*, die in Aktivitäten und Fragementen implementiert werden kann
- Unterscheidung verschiedener Kontextmenüs anhand der *ContextMenu.ContextMenuInfo*

Beispiel

```
@Override
public void onCreateContextMenu(ContextMenu menu, View v,
                                ContextMenuInfo menuInfo) {
  super.onCreateContextMenu(menu, v, menuInfo);
  MenuInflater inflater = getMenuInflater();
  inflater.inflate(R.menu.ubuntu_context_menu, menu);
}
```

Listing : Initialisierung des Kontextmenüs

- Aufruf von *onContextItemSelected()* beim Klick auf Menü-Eintrag
- Falls Aktion ausgeführt werden kann ist Rückgabe *true*
- Falls Aktion unbekannt ist bzw. nicht behandelt werden soll ist Rückgabewert der der Standard-Implementation (*false*)

```
@Override
public boolean onContextItemSelected(MenuItem item) {
  // Fetch the view item info ...
  AdapterContextMenuInfo info = (AdapterContextMenuInfo) item.getMenuInfo();

  // ... and handle the item selection.
  Toast.makeText(this, "Selected " + item.getTitle() + " for " + info.id,
    Toast.LENGTH_LONG).show();
}
```

Listing : Kontextmenü Events

Allgemeines

- Konext-ActionMode ist der in Android 3.0 hinzugekommene Modus für Kontextmenüs
- Zeigt anstatt eines Overflow-Menüs eine eigene ActionBar an
- Es können mehrere Einträge für eine Aktion ausgewählt werden
- Typisches Beispiel sind Aktionen auf Elementen einer Liste
- ActionBar wird durch Drücken des Erledigt- oder Zurück-Buttons oder Abwählen aller Elemente geschlossen
- Implementierung für ausgewählte Views mit *ActionMode.Callback*-Interface
- Implementierung für Gruppe von Views mit *AbsListView.MultiChoiceModeListener*

Implementierung für ausgewählte Views

```
private ActionMode.Callback actionModeCallback = new ActionMode.Callback() {
    public boolean onCreateActionMode(ActionMode mode, Menu menu) {
        // Initialize the ContextMenu.
        MenuInflater inflater = mode.getMenuInflater();
        inflater.inflate(R.menu.context_menu, menu);
        return true;
    }

    public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
        // Do nothing here.
        return false;
    }

    public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
        switch (item.getItemId()) {
        case R.id.menu_calendar:
            mode.finish();
            return true;
        case R.id.menu_share:
            mode.finish();
            return true;

        ...

        default:
            return false;
        }
    }

    public void onDestroyActionMode(ActionMode mode) {
        // Do nothing!
    }
};
```

Listing : Implementierung des ActionMode.Callback Interfaces

Implementierung für Gruppe von Views

- Implementierung von *AbsListView.MultiChoiceModeListener*
- Zuweisung mit Hilfe von *setMultiChoiceModeListener()*
- Aktivierung der Auswahl mehrerer Elemente mit *setChoiceMode()* und dem Wert *CHOICE_MODE_MULTIPLE_MODAL*

Quellcode

```
ListView listView = getListView();
listView.setChoiceMode(ListView.CHOICE_MODE_MULTIPLE_MODAL);
listView.setMultiChoiceModeListener(
   new MultiChoiceModeListener() {
      public boolean onCreateActionMode(ActionMode mode, Menu menu) {
         MenuInflater inflater = mode.getMenuInflater();
         inflater.inflate(R.menu.context_menu, menu);
         return true;
      }

      public boolean onPrepareActionMode(ActionMode mode, Menu menu) {
         return false;
      }

      public boolean onActionItemClicked(ActionMode mode, MenuItem item) {
         switch (item.getItemId()) {
         case R.id.menu_calendar:
            mode.finish();
            return true;

         ...

         default:
            return false;
         }
      }

      public void onDestroyActionMode(ActionMode mode) {}
      public void onItemCheckedStateChanged(
         ActionMode mode, int position, long id, boolean checked) {}
   }
);
```

Listing : Implementierung des MultiChoiceModeListener Interfaces

Screenshot

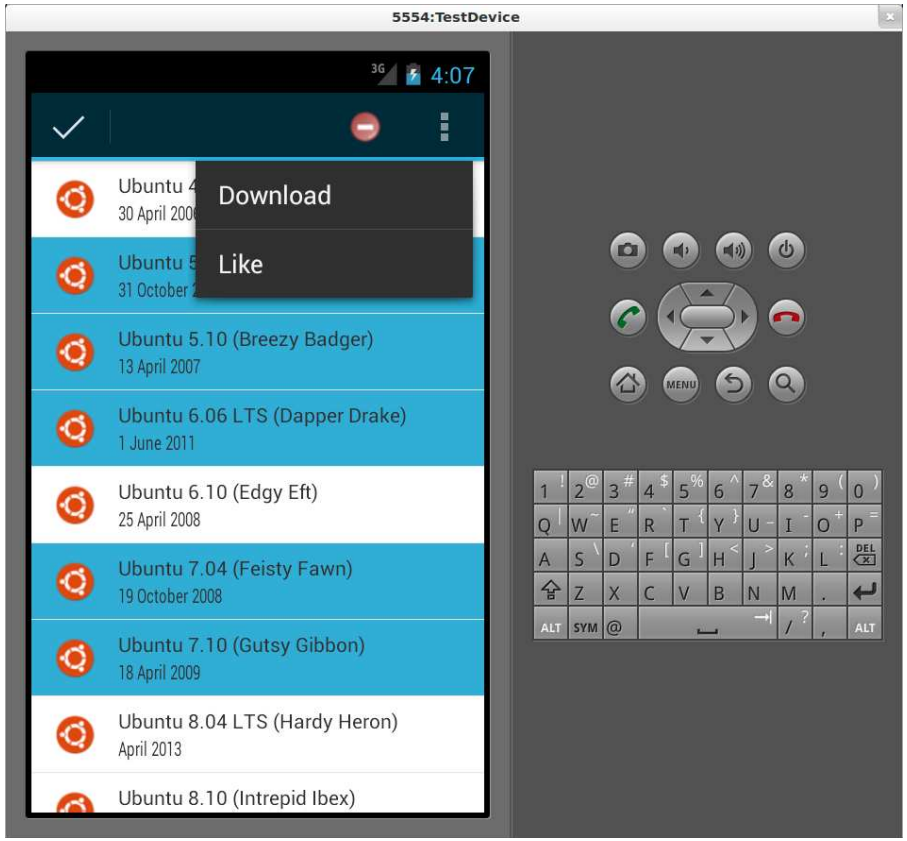


Abbildung: Das Ubuntu Kontextmenü

Allgemeines

- Spezielles Menü, das an einem View verankert wird
- Bietet Kontext sensitive Aktionen
- Anzeige als Overflow-Menü unterhalb des Views
- Falls kein Platz vorhanden Anzeige oberhalb des Views
- Wird geschlossen bei Auswahl oder Klick neben das Menü
- Klick neben das Menü kann mit *OnDismissListener* abgefangen werden
- Verarbeitung von Events mit *onMenuItemClick*-Listener oder mit in *android:onClick* deklarierter Methode
- Typischerweise in SMS-Clients für Wörterbuch genutzt
- Alternativ Nutzung als erweitertes Menü

Unterscheidung PopUp-Menüs und Kontextmenüs

Kontextmenüs bieten Aktionen auf den Inhalten auf die sie sich beziehen und verändern diese. PopUp-Menüs bieten nur Aktionen auf Inhalten an, auf die sie sich beziehen, die sie aber nicht verändern.

Implementierung

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android" >
    <item
        android:id="@+id/menu_share_flattr"
        android:title="@string/share_flattr" />
    <item
        android:id="@+id/menu_share_twitter"
        android:title="@string/share_twitter" />
    <item
        android:id="@+id/menu_share_facebook"
        android:title="@string/share_facebook" />
</menu>
```

Listing : Share PopUp-Button

```
@Override
protected void onListItemClick(ListView l, View v, int position, long id) {
   PopupMenu popup = new PopupMenu(this, v);
   popup.inflate(R.menu.share_menu);
   popup.show();
}
```

Listing : Die PopUp-Callback-Methode

Aktualisierung des Menüs

Seit API Version 14 ist es übrigens möglich das Laden des Menüs direkt aus der Klasse PopupMenu zu erledigen. Dazu muss man nur *PopupMenu.inflate()* aufrufen.

Screenshot

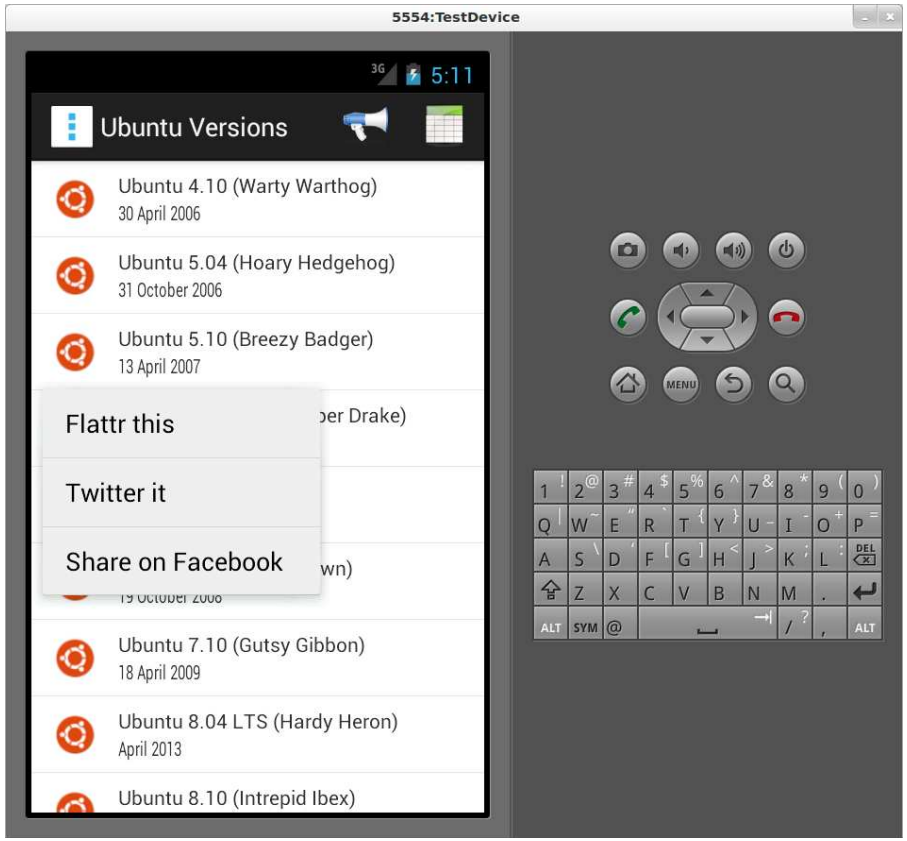


Abbildung: Das PopUp-Menü

Allgemeines

- Eingeführt in Android 3.0
- Bieten häufig genutzte Aktionen und globale Navigation
- Anzeige des Applikationsicons & Titel der Aktivität
- Navigation über Applikationsicon möglich
- Button am rechten Rand öffnet Overflow-Menü (Optionsmenü)
- Tab oder Spinner basierte Navigation und Filterung von Daten
- Einbinden eigener Views als *Action Views*
- Seit Android 3.0 standardmäßig in allen Aktivitäten mit Theme *Theme.Holo*
- Ausblendung der ActionBar mit *@android:style/Theme.Holo.NoActionBar* in Deklaration der Aktivität

ActionBars in älteren Android-Versionen

ActionBars wurden zwar erst in Android 3.0 eingeführt, können jedoch durch eine eigene ActionBar ersetzt werden. Eine Implementierung findet man in den Beispielen des Android-SDKs.

Action Items

- Einträge direkt in ActionBar
- Optionsmenü-Einträge als Action Items (*android:showAsAction*)
- Anzeige des Icons und/oder Titels des Menüeintrags
- Standardmäßig nur Icons in ActionBar $\rightarrow$ *android:withText*

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_share"
        android:icon="@drawable/share"
        android:title="@string/menu_sahre"
        android:showAsAction="ifRoom|withText" />
</menu>
```

Listing : Aktionen in der Actionbar

Erzeugen der ActionBar

- Initialisierung der ActionBar in *onCreateOptionsMenu()*
- Verarbeitung der Menüauswahl in *onOptionsItemSelected()*
- Seit Android 4.0 können mit *android:uiOptions=splitActionBarWhenNarrow* ActionBars geteilt werden
- Ausblenden des Titels der Aktivität mit *setDisplayShowTitleEnabled(false)*
- Ausblenden des Icons mit *setDisplayShowHomeEnabled(false)*

Hinweis zu *onOptionsItemSelected()*

Das System ruft die Methode *onOptionsItemSelected* der aktuellen Aktivität vor der entsprechenden Methode eventuell verwendeter Fragmente auf.

Neue Attribute in älteren Versionen

Auch wenn die minimal unterstützte API Version ein Attribut in einer XML-Deklaration nicht kennt, kann man es ohne bedenken verwenden, denn wenn die Appilkation auf einem älteren System ausgeführt wird, wird es einfach ignoriert. Allerdings muss dafür natürlich die Zielplatform, für die die Applikation kompiliert wird, eine API Version größer oder gleich der benötigten API Version unterstützen.

Applikationsicon

- Standardmäßig auf linker Seite der AktionBar
- Verwendung als ActionItem zur Navigation möglich
- Bei Klick Aufruf von *onOptionsItemSelected()*
- Identifikation über ID *android.R.id.home*
- Üblich sind Navigation zur Start-Aktivität oder durch Applikationshierarchie (Aufsteigen)

Wichtiges Intent-Flag

Da beim Zurückkehren zur Startansicht eigentlich nichts anderes gemacht werden soll, als mehrere Aktivitäten zu schließen, sollte man immer das Flag *FLAG_ACTIVITY_CLEAR_TOP* bei der Deklaration des Intents setzen.

Aktivierung des Navigationsmodus

Seit Android 4.0 muss das Applikationsicon explizit als Action Item mit Hilfe der Methode *setHomeButtonEnabled(true)* angegeben werden.

Klick auf Applikationsicon

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case android.R.id.home:
         Intent intent = new Intent(this, UbuntuList.class);
         intent.addFlags(Intent.FLAG_ACTIVITY_CLEAR_TOP);
         startActivity(intent);
         return true;
      default:
         return super.onOptionsItemSelected(item);
   }
}
```

Listing : Drücken des Applikationsicons

Aufstieg in der Applikationshierarchie

Modus muss explizit mit *setDisplayHomeAsUpEnabled(true)* aktiviert werden.

```
protected void onCreate(Bundle savedInstanceState) {
    ...

    ActionBar actionBar = getActionBar();
    actionBar.setDisplayHomeAsUpEnabled(true);
}
```

Listing : Höhersteigen in der Hierarchie

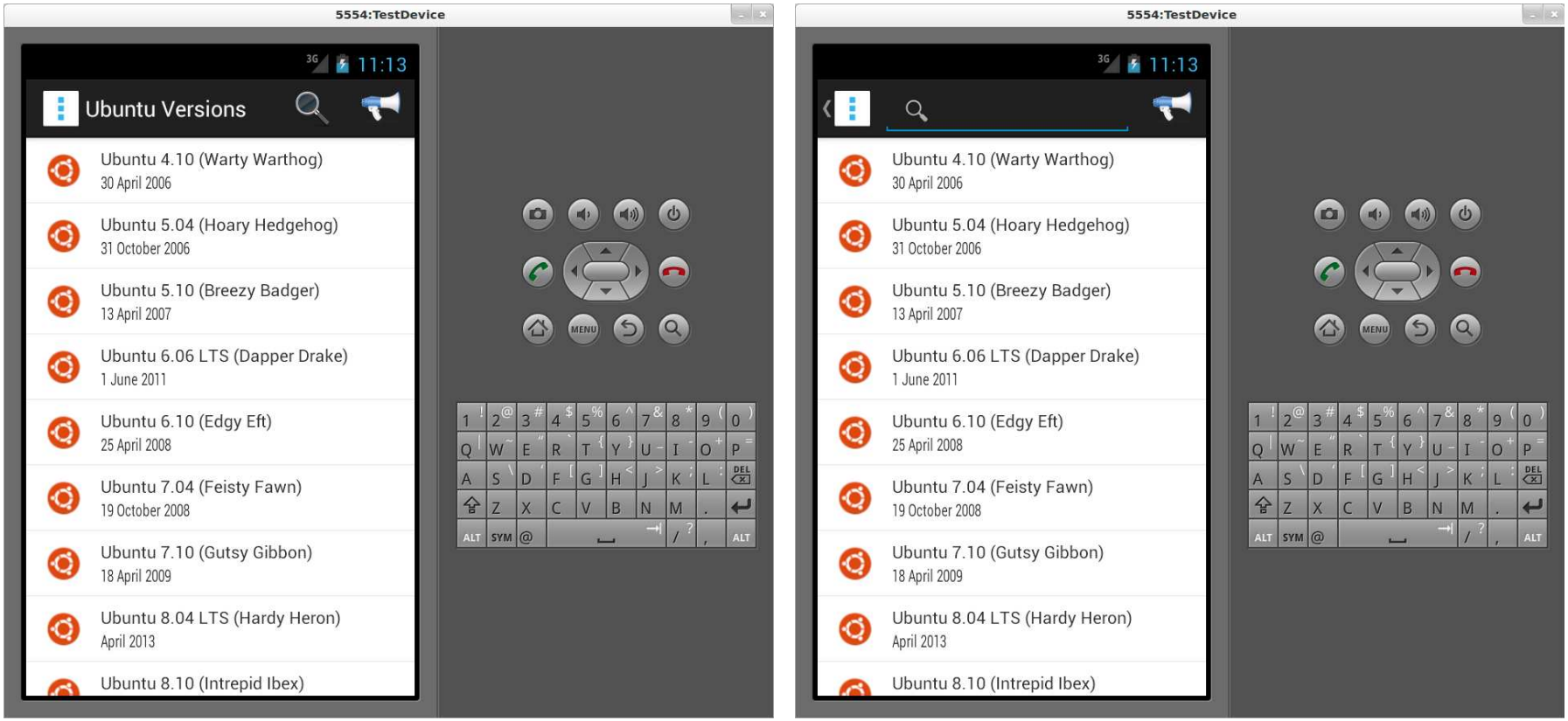
Action Views

- Fast beliebige Views in einer ActionBar
- Typische Anwendung Suchfelder & Auswahllisten
- Einbinden mit *android:actionViewClass* oder *android:actionLayout* in <*item*>-Element
- Zusammenklappen zu Action Item möglich (*collapseActionView*)

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_search"
        android:title="@string/menu_search"
        android:icon="@drawable/ic_menu_search"
        android:showAsAction="ifRoom|collapseActionView"
        android:actionViewClass="android.widget.SearchView" />

    ...
</menu>
```

Listing : Action View Deklaration

Screenshot



(a) Action View zusammengeklappt

(b) Action View ausgeklappt

Abbildung: Anwendung von Action Views

Verwendung von Listenern

- Zuweisung in *onCreateOptionsMenu()*
- Laden des Menüeintrags mit *findItem()*
- Suchen des dazugehörigen Views mit *getActionView()*

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu declaration ...
    getMenuInflater().inflate(R.menu.options, menu);

    // ... and load the search view.
    SearchView searchView = (SearchView) menu.findItem(R.id.menu_search).getActionView();

    return super.onCreateOptionsMenu(menu);
}
```

Listing : Laden des Action Views

Hinweise

Der Hardware-Such-button

Manche Android-Geräte haben einen speziellen Hardware-Button, der die Suche öffnet. Das Überschreiben der Methode *onKeyUp()* der betreffenden Aktivität ermöglicht das Behandeln des Events mit der ID *KEYCODE_SEARCH*.

Zusammenklappen von Action Views

Das Zusammenklappen eines Aktion Views kann, falls dies nötig ist, mit einem *OnActionExpandListener* abgefangen und behandelt wereden.

Action-Provider

- Bindet eigenes View mit spezieller Event-Behandlung ein
- Typische Anwendung anbieten verschiedener Applikationen zum ”*sharen*” von Inhalten
- Einbinden mit *android:actionProviderClass*
- Übergabe des dazugehörigen Intents in*onCreateOptionsMenu()*

```
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <item
        android:id="@+id/menu_share"
        android:title="@string/menu_share"
5       android:showAsAction="ifRoom"
        android:actionProviderClass="android.widget.ShareActionProvider" />

    ...
</menu>
```

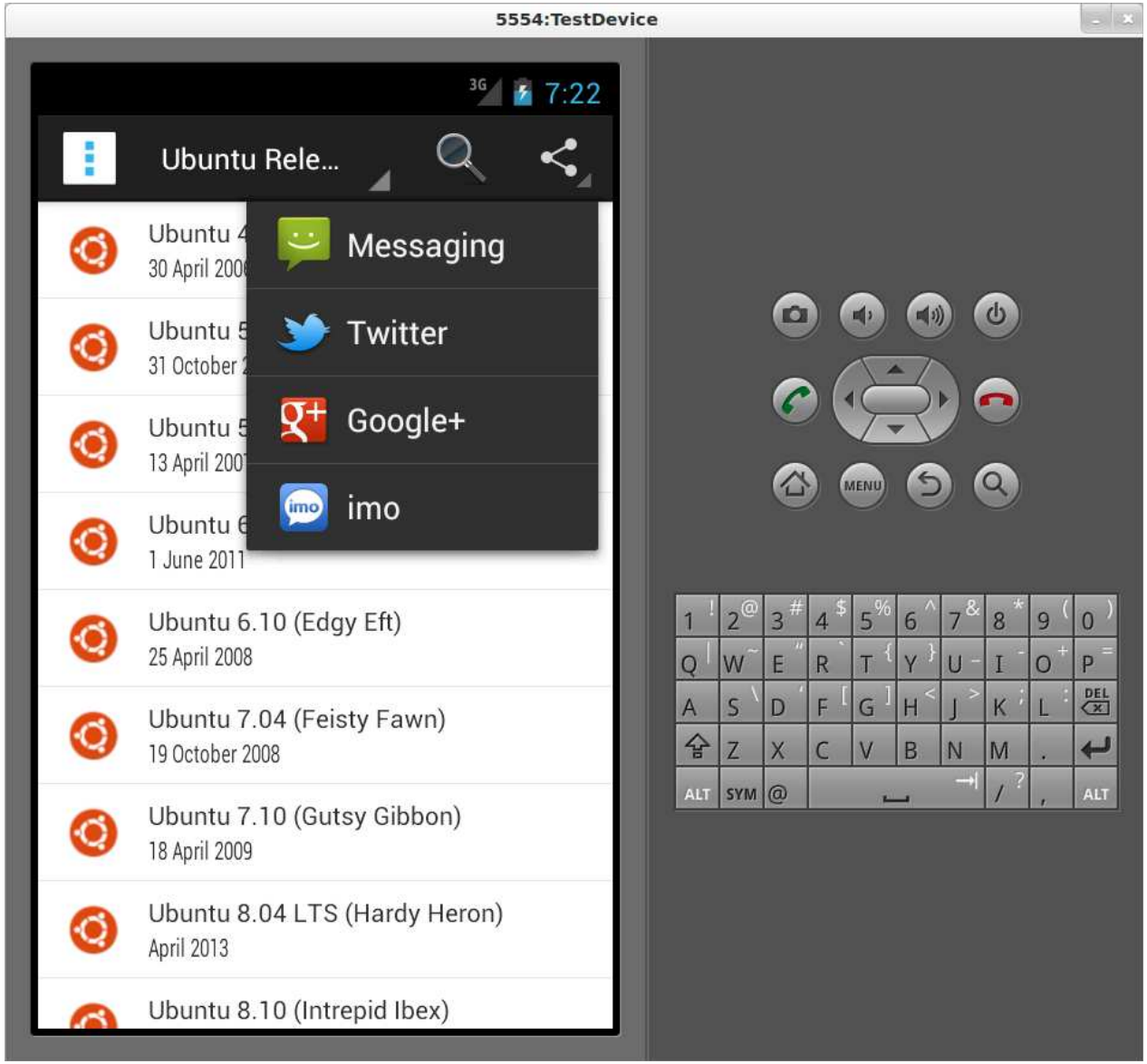
Listing : Deklaration des ShareActionProviders

Initialisierung

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    ...

    shareProvider = (ShareActionProvider) menu.findItem(R.id.menu_share).getActionProvider();
    shareProvider.setShareHistoryFileName(ShareActionProvider.DEFAULT_SHARE_HISTORY_FILE_NAME);
    shareProvider.setShareIntent(createShareIntent());

    ...

    return super.onCreateOptionsMenu(menu);
}

private Intent createShareIntent() {
    Intent shareIntent = new Intent(android.content.Intent.ACTION_SEND);
    shareIntent.setType("text/plain");
    shareIntent.putExtra(android.content.Intent.EXTRA_TEXT, "The Ubuntu Release List");
    return shareIntent;
}
```

Listing : Initialisierung des ShareActionProviders

Screenshot


5554:TestDevice
7:22
Ubuntu Rele...
Messaging
Twitter
Google+
imo
Ubuntu 6.10 (Edgy Eft)
25 April 2008
Ubuntu 7.04 (Feisty Fawn)
19 October 2008
Ubuntu 7.10 (Gutsy Gibbon)
18 April 2009
Ubuntu 8.04 LTS (Hardy Heron)
April 2013
Ubuntu 8.10 (Intrepid Ibex)
MENU

Anmerkungen

- Anpassung des Verhaltens bei Klick in *onOptionsItemSelected()*
- Ansonsten Aufruf von *onPerformDefaultAction()* durch System
- *onOptionsItemSelected()* wird nur aufgerufen wenn Eintrag kein Untermenü einbindet

Aktivierung des Navigationsmodus

Ein Bug sorgt manchmal dafür, dass auch wenn eine einzelne Anwendung installiert ist, die dieses Intent verarbeiten könnte, der Button des dazugehörigen Action Providers im Emulator inaktiv ist. Dieses Problem lässt sich durch installieren weiterer Programme mit der *Android Debug Bridge* (adb) einfach beheben.

Implementierung

- Überschreiben von *onCreateActionView()* der Klasse *ActionProvider*
- Bei Platzmangel sollte *onPerformDefaultAction()* überschreiben werden
- *onPerformDefaultAction()* wird nur aufgerufen wenn Eintrag kein Untermenü einbindet

```
public View onCreateActionView() {
    LayoutInflater layoutInflater = LayoutInflater.from(this.context);
    View view = layoutInflater.inflate(R.layout.edittext_provider, null);
    EditText txt = (EditText) view.findViewById(R.id.my_edittext);
    txt.addTextChangedListener(
        new TextWatcher() {
            public void afterTextChanged(Editable s) {
                // Do something...
            }
        }
    );

    return view;
}
```

Listing : Implementierung des ActionProviders

Tab basierte Navigation

- Platzsparende Alternative zu TabWidget
- Anzeige direkt in ActionBar oder in seperater Bar
- Manchmal wird Tab-Navigation auch in DropDown angezeigt um Platz zu sparen
- Inhalte von Tabs werden als Fragment bereitgestellt
- Wechsel zwischen Tabs nutzt *FragmentTransaction*
- Platzierung der Fragments in ViewGroup oder Standard-Layout
- Fragment muss gesamte Aktivität (ausgenommen ActionBar) füllen

Implementierung

- Aktivierung des Tab-Modus mit *setNavigationMode()* und dem Wert *NAVIGATION_MODE_TABS*
- Instanzierung eines *ActionBar.Tab* für jedes Fragment
- Zuweisung von Icon und Titel mit *seticon()* bzw. *setText()*
- Implementierung eines *ActionBar.TabListener*
- TabListener muss sich selbst um die Verwaltung des Fragments kümmern
- Eigene Instanz des TabListeners für jedes Tab
- Hinzufügen des Tabs mit *addTab()*

Der TabListener

```
public static class TabListener<T extends Fragment> implements ActionBar.TabListener {
   private Fragment fragment;
   private final Activity activity;
   private final String tag;
   private final Class<T> clazz;

   public TabListener(Activity activity, String tag, Class<T> clz) {
      this.activity = activity;
      this.tag = tag;
      this.clazz = clz;
      this.fragment = null;
   }

   public void onTabSelected(Tab tab, FragmentTransaction fragTrans) {
      //Instantiate and add the fragment if it doesN't exist.
      if (this.fragment == null) {
         this.fragment = Fragment.instantiate(this.activity, this.clazz.getName());
         fragTrans.add(android.R.id.content, this.fragment, this.tag);
      } else {
         // If it exists, show it.
         fragTrans.attach(this.fragment);
      }
   }

   public void onTabUnselected(Tab tab, FragmentTransaction fragTrans) {
      if (this.fragment != null) {
         // Hide the fragment.
         fragTrans.detach(this.fragment);
      }
   }

   public void onTabReselected(Tab tab, FragmentTransaction fragTrans) {
      // Do nothing here or refresh fregment data.
   }
}
```

Listing: Implementierung des TabListeners

Spinner Navigation

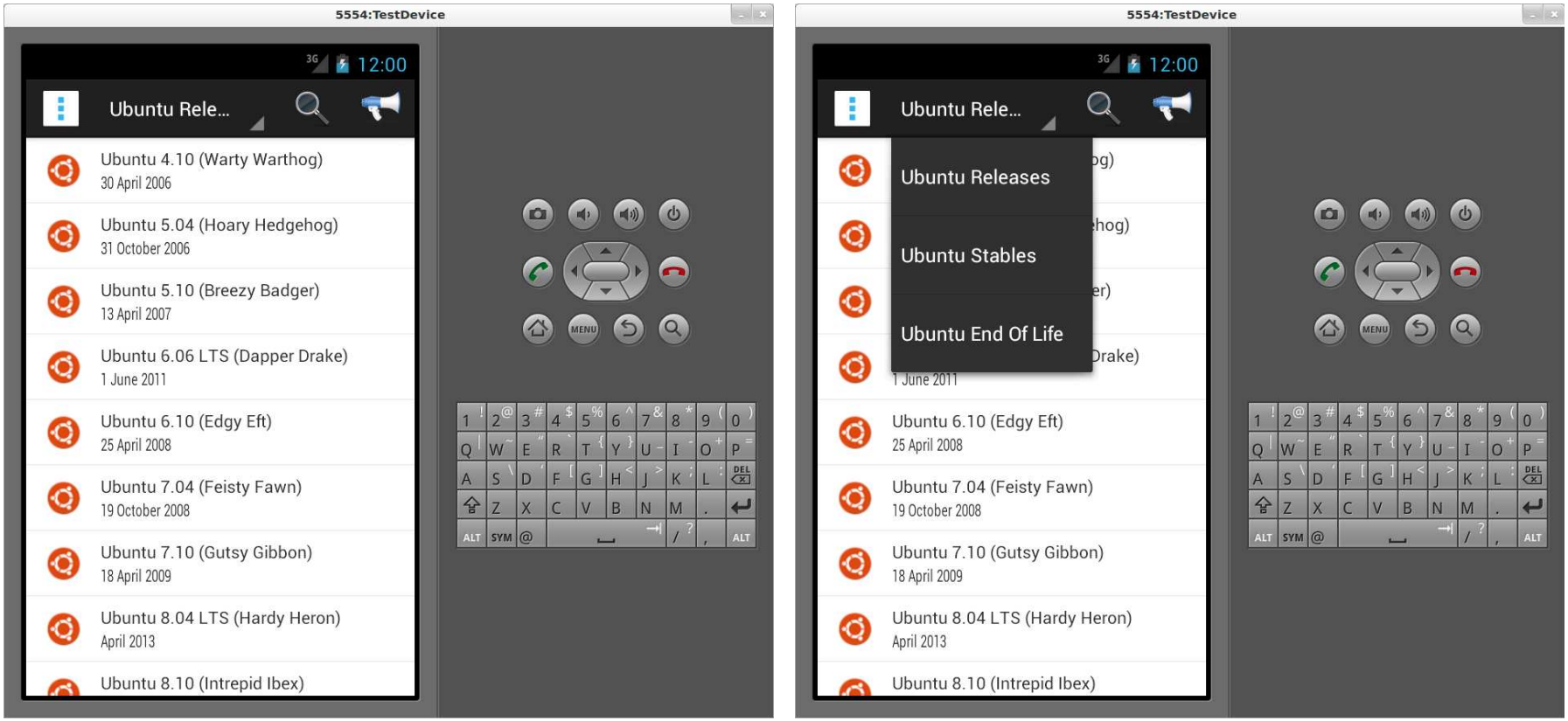
- Bietet Möglichkeit der Filterung oder Navigation
- Typische Anwendung Sortierung bzw. Filterung von Listen
- Versorgung mit Daten über Adapter
- Zuweisung des Adapters mit *setListNavigationCallbacks()*
- *OnNavigationListener* kümmert sich um Ausführung von Operationen
- Aktivierung des Spinners mit *setNavigationMode()*

Implementierung

```
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    MenuInflater inflater = getMenuInflater();
    inflater.inflate(R.menu.ubuntu_menu, menu);

    ActionBar actionBar = getActionBar();
    actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_LIST);
    actionBar.setListNavigationCallbacks(
        ArrayAdapter.createFromResource(
            this, R.array.ubuntu_selections, R.layout.spinner_dropdown_item),
        new ActionBar.OnNavigationListener() {
            public boolean onNavigationItemSelected(int itemPosition, long itemId) {
                // Do something crazy here!
                return false;
            }
        }
    );

    return true;
}
```

Listing : ActionBar Spinner

Screenshot



(a) Action Spinner (b) Action Spinner ausgeklappt

Abbildung: Anwendung von Action Spinner

Anmerkung

Schrift des Spinners

Da die Schriftfarbe des durch Android zur Verfügung gestellten Standard-Layouts *android.R.layout.simple_spinner_dropdown_item* schwarz ist, lohnt es sich ein eigenes Layout zu deklarieren. Man sollte in diesem Fall unbedingt auf die View-Klasse *CheckedTextView* zurückgreifen, um die Auswahl der einzelnen Einträge mit einem grafischen Feedback zu verschönern.