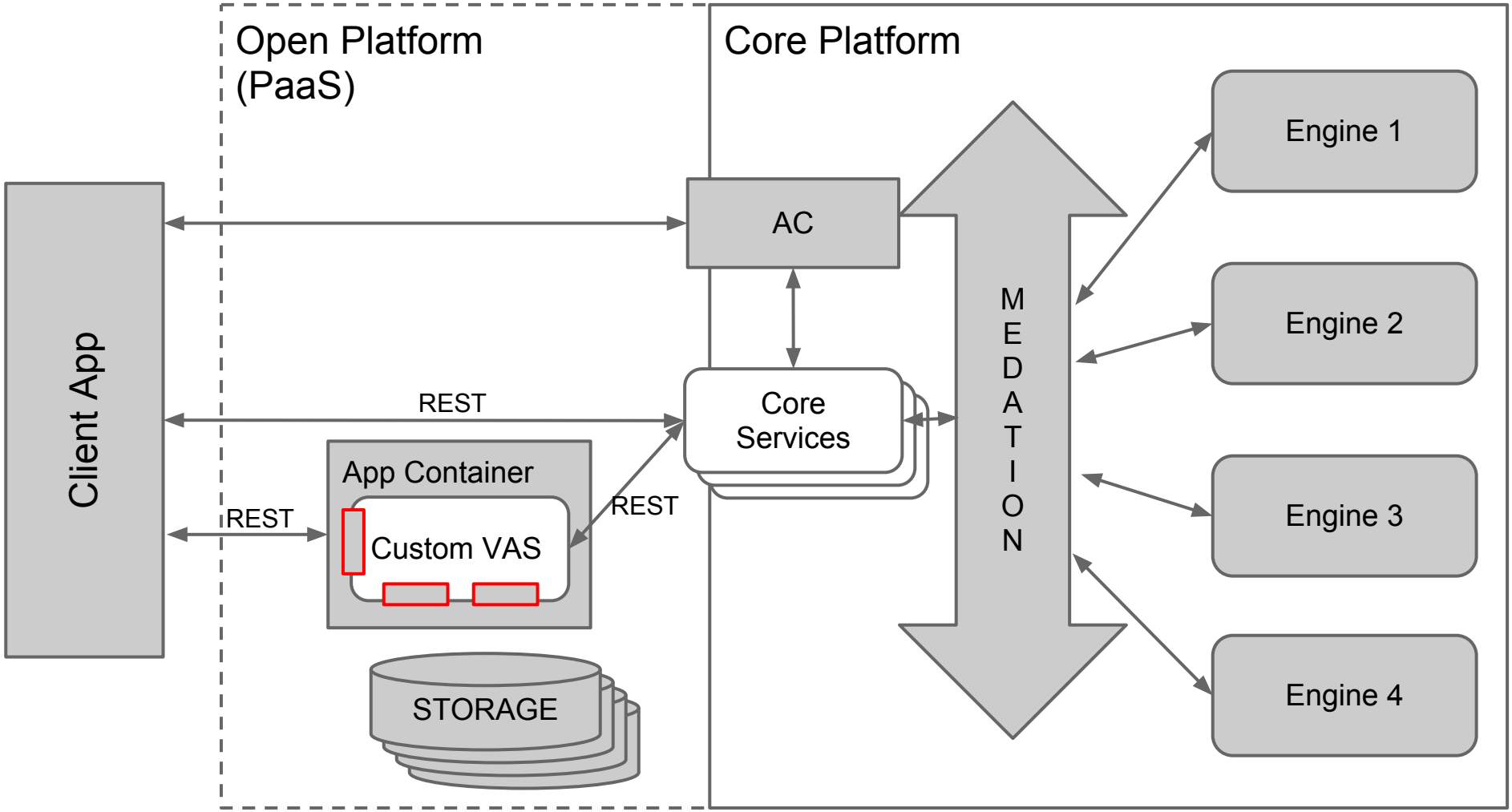
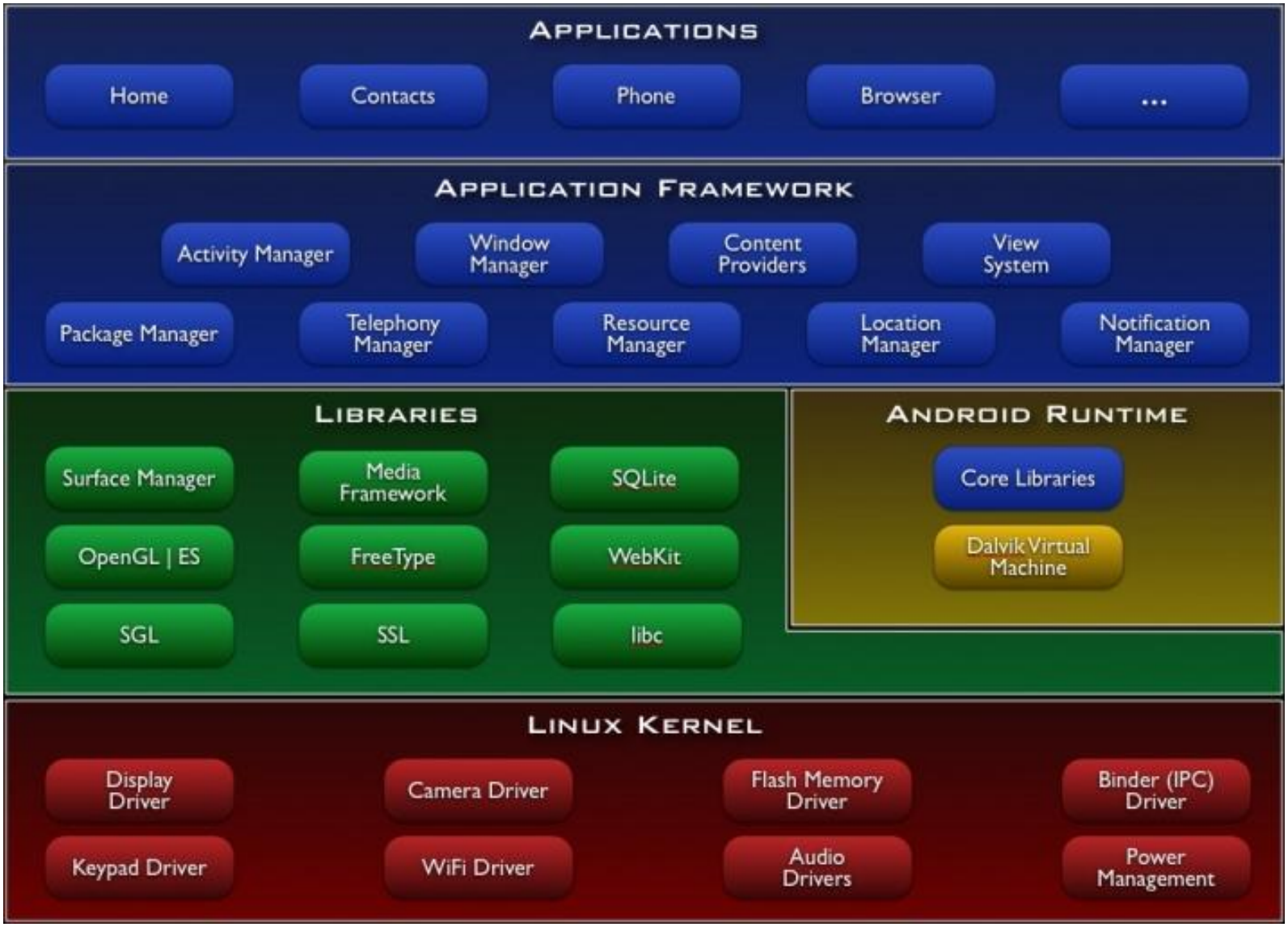
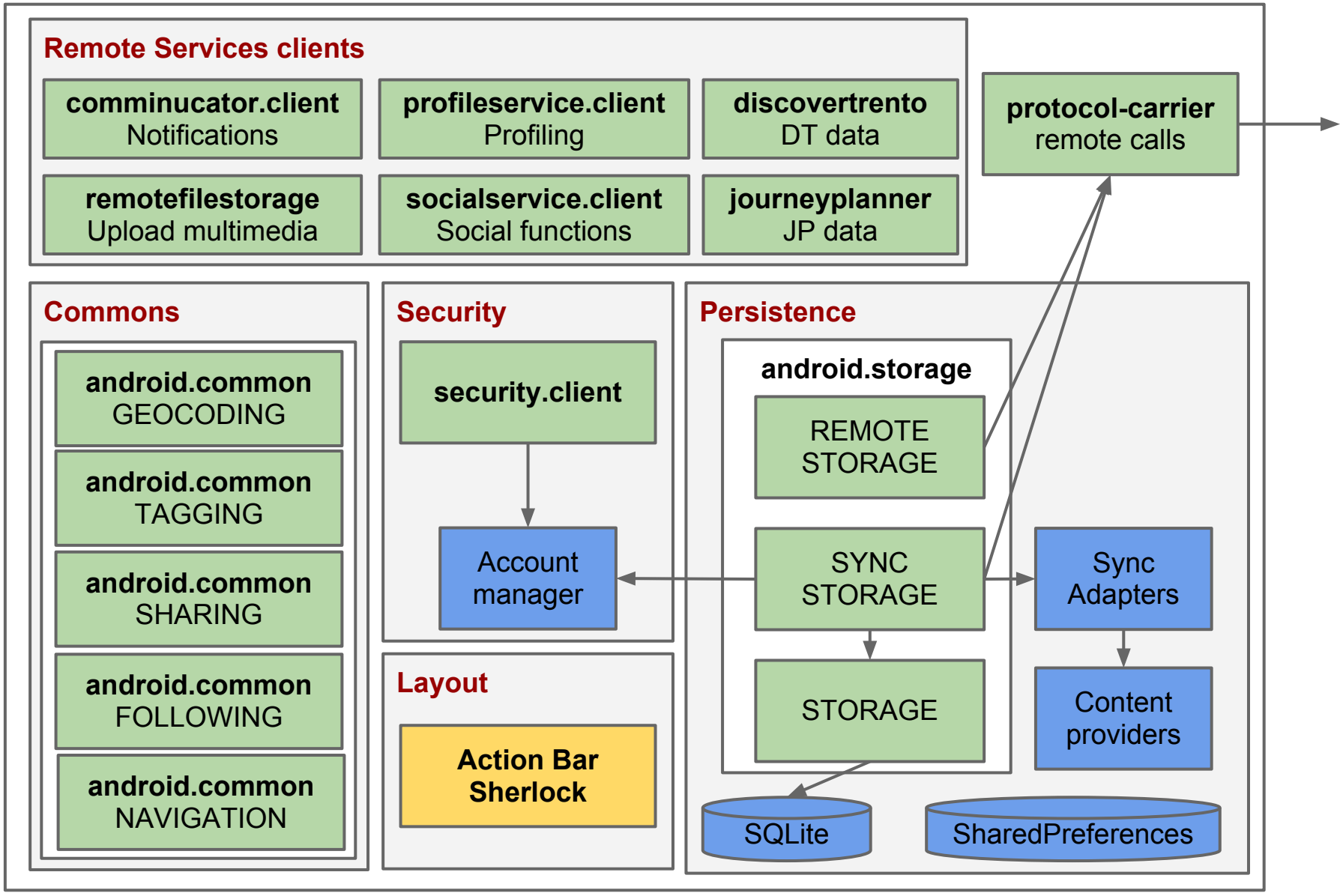
Open platform



Open platform: Android



Open platform: Android in SC



ActionBarSherlock

See http://actionbarsherlock.com/usage.html

Support for Android Action Bar design patterns on old devices.

Usage:

MyActivity extends **Sherlock**FragmentActivity

get**Support**ActionBar(), get**Support**FragmentManager(), etc.

Typical usage for tabs:

```
ActionBar actionBar = getSupportActionBar();
actionBar.setDisplayShowTitleEnabled(true); // system title
actionBar.setDisplayShowHomeEnabled(true); // home icon bar
actionBar.setNavigationMode(ActionBar.NAVIGATION_MODE_TABS); // tabs bar
// Tab 1
ActionBar.Tab tab = actionBar.newTab();
tab.setText(R.string.tab1);
tab.setTabListener(new TabListener<MyTab1Fragment>(this, "tag1", MyTab1Fragment.class));
actionBar.addTab(tab);
```

On implementation of the tab listener etc see http://developer.android.com/guide/topics/ui/actionbar.html#Tabs

Security Client

Provides authentication functionality for SC platform.

Usage (standalone):

1. Declare account metadata in AndroidManifest.xml

```
<meta-data android:name="SHARED_PACKAGE" android:value="smartcampus.android.template.standalone"/>
<meta-data android:name="ACCOUNT_TYPE" android:value="smartcampus.android.template.standalone"/>
<meta-data android:name="ACCOUNT_NAME" android:value="SmartCampus Android Template"/>
```

2. Declare account service and activity

```
<service android:name="eu.trentorise.smartcampus.ac.authenticator.AuthenticationService"
    android:exported="true" android:permission="eu.trentorise.smartcampus.ac.AUTHENTICATE">
    <intent-filter><action android:name="android.accounts.AccountAuthenticator" /></intent-filter>
    <meta-data android:name="android.accounts.AccountAuthenticator" android:resource="@xml/authenticator"
/>
</service>
<activity android:name="eu.trentorise.smartcampus.ac.authenticator.AuthenticatorActivity"/>
```

3. Declare authenticator descriptor in res/xml/authenticator.xml

```
<account-authenticator xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="smartcampus.android.template.standalone" android:icon="@drawable/ic_launcher"
    android:smallIcon="@drawable/ic_launcher" android:label="@string/app_name"/>
```

Security Client

4. In your activity use the AccessProvider class and ask for a token.

```
SCAccessProvider mAccessProvider = new AMSCAccessProvider();
mToken = mAccessProvider.getAuthToken(this, null);
if (mToken != null) {
        // do somthing...
}
```

5. Handle authentication result:

```
protected void onActivityResult(int requestCode, int resultCode, Intent data) {
        if (requestCode == SCAccessProvider.SC_AUTH_ACTIVITY_REQUEST_CODE) {
                if (resultCode == RESULT_OK) {
                        mToken = data.getExtras().getString(AccountManager.KEY_AUTHTOKEN);
                } else if (resultCode == RESULT_CANCELED) {
                } else {
                        Log.i(TAG, "Authentication failed: "+error);
                }
        }
        super.onActivityResult(requestCode, resultCode, data);
}
```

6. Access token and user data directly (null if not present)

```
String token = mAccessProvider.readToken(this, null);
UserData data = mAccessProvider.readUserData(this, null);
```

Storage

See http://developer.android.com/guide/topics/data/data-storage.html

Store application data on the device.

- SharedPreferences: simple key-value storage in a file.
- SQLite: simple database with SQL support and conventional methods
- ContentProvider: abstraction on top of any other that allows for sharing data across apps.

  Example: contacts.

SC Library: **android.storage**

- Common model: JavaBeans storage with CREATE-DELETE-UPDATE-READ operations
- Local storage extension:
  - SQLite database
  - SQL queries on model
- Remote storage extension:
  - RESTful JSON service backend
  - generic support for object search
- Synchronized storage extension: local storage with remote sync
  - Only 'delta' is sent/received
  - Works well with Android SyncAdapter

Storage: remote

Usage:

1. Create storage instance

```
RemoteStorage remoteStorage = new RemoteStorage(mContext, "myapp");
remoteStorage.setConfig(authToken, remoteAppHost, remoteAppService);
```

2. Perform operations

```
MyBean bean = remoteStorage.getObjectById(id, MyBean.class);
remoteStorage.searchObjects(filter,MyBean.class);
remoteStroage.create(new MyBean());
```

Service conventions:

- for object creation: POST <host>/<service>/<canonical-name-of-object-java-class>.
- for object update: PUT <host>/<service>/<canonical-name-of-object-java-class>/<id>.
- for object delete: DELETE <host>/<service>/<canonical-name-of-object-java-class>/<id>.
- for individual object access: GET <host>/<service>/<canonical-name-of-object-java-class>/<id>.
- for object access by type: GET <host>/<service>/<canonical-name-of-object-java-class>.
- for object search: GET <host>/<service>/objects. Query object is passed as "filter" query parameters in JSON representation. Should return the JSON map with full java class names as the keys and the corresponding array of the objects of those classes matching the criteria.

Protocol carrier

Support for synchronous/asynchronous and immediate/deferred remote calls.

Synchronous immediate call usage:

- Create a request object

```
MessageRequest request = new MessageRequest(host, service);
request.setMethod(Method.GET);
request.setQuery("queryParameter=" + parameter);
```

- Perform a call

```
MessageResponse response = new ProtocolCarrier(ctx, "myApp").invokeSync(request, "myApp", authToken);
```

- process the result (using Utils: the Jackson library wrapper in android.common)

```
String body = response.getBody();
MyBean bean = Utils.convertJSONToObjects(body, MyBean.class);
```