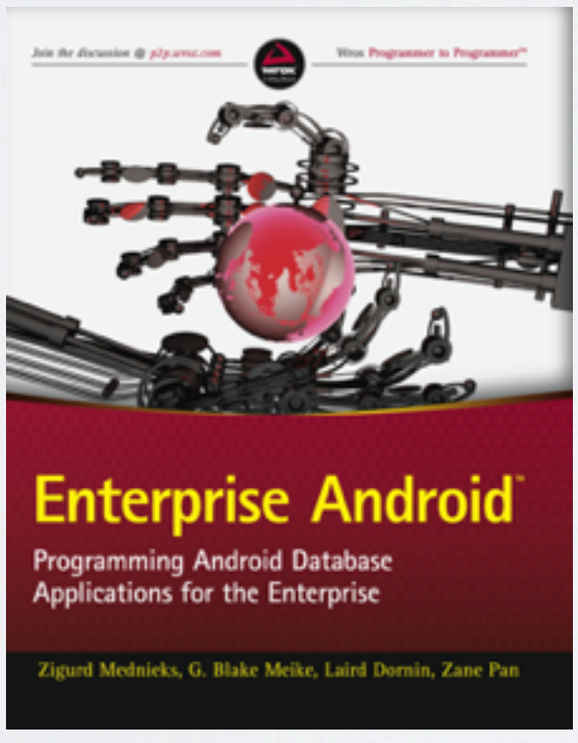
TAMING THE WILD
SYNC ADAPTER



Blake Meike

- blake.meike@gmail.com
- @callmeike

MVC

As are many design patterns, MVC is fractal-like.

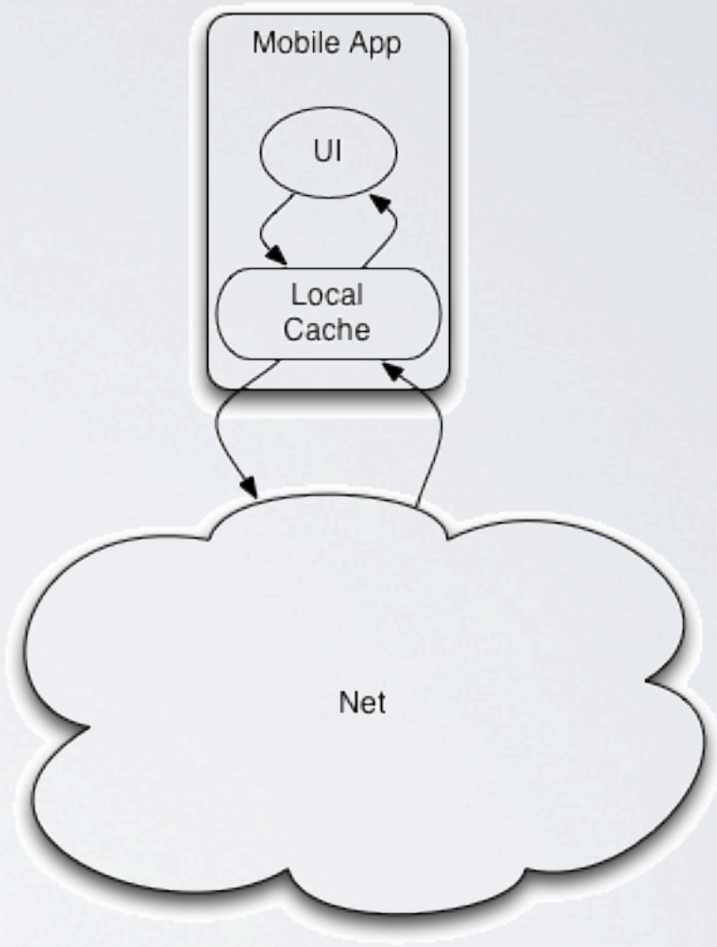
UI	Activity/Service
Application	Mobile App/Backend Server

This is not an invitation to merge layers!

Application layer MVC

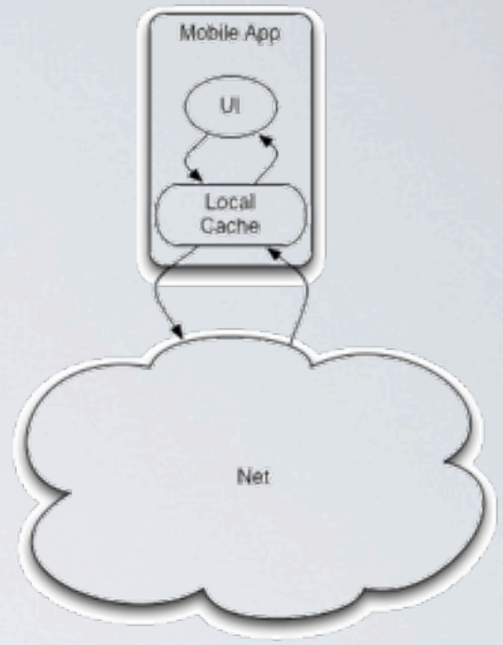
The UI sees only local cache

When necessary the cache sends updates to and receives them from the net



The UI is completely isolated from the network

Why?



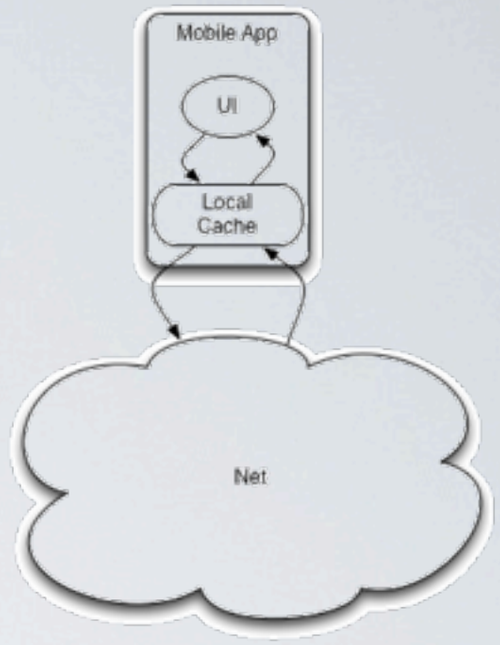
- UI code is plenty hard to understand already. No need to combine it with the complexity of the network

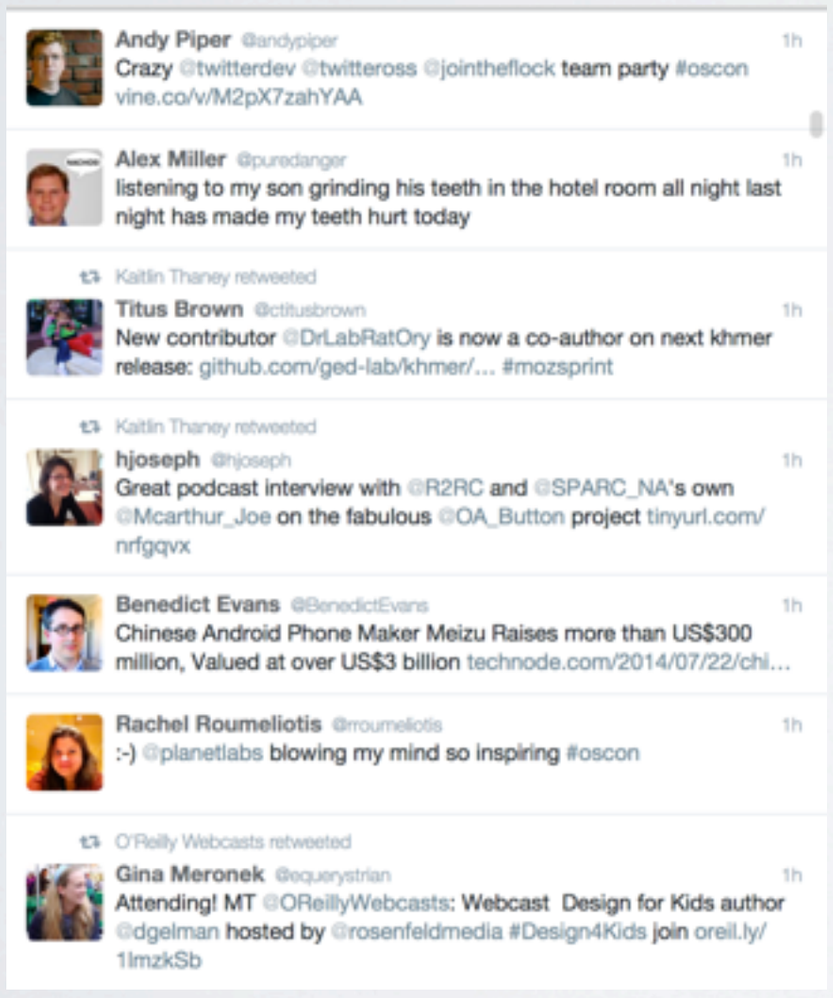
- Consistent user experience: If she saw something once, she can see it again.

- Separation of concerns: Nice clean code

When?

There are times that the pattern does not fit.

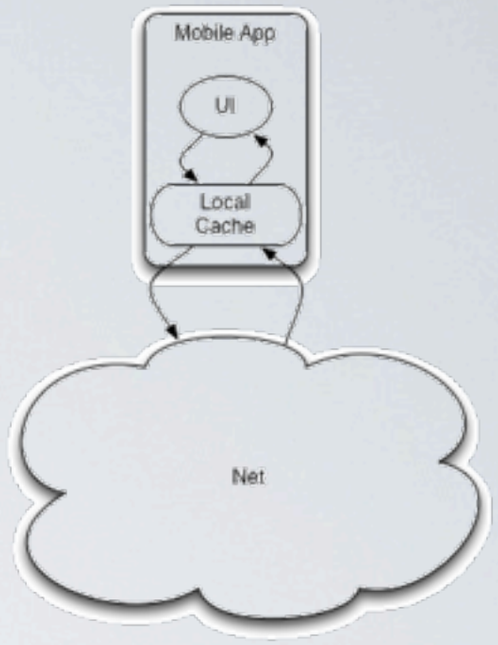




Consider, e.g., Volley when the remote data is not essential

Network CRUD

You could proxy REST calls to the remote...

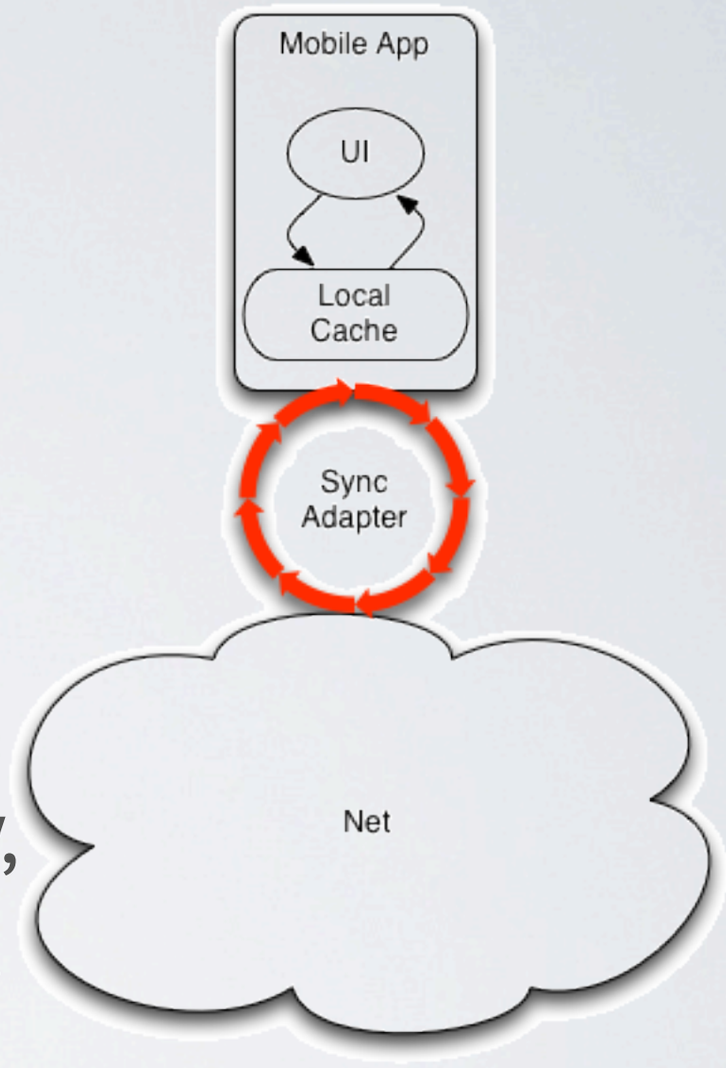


..just don't

The Sync Adapter

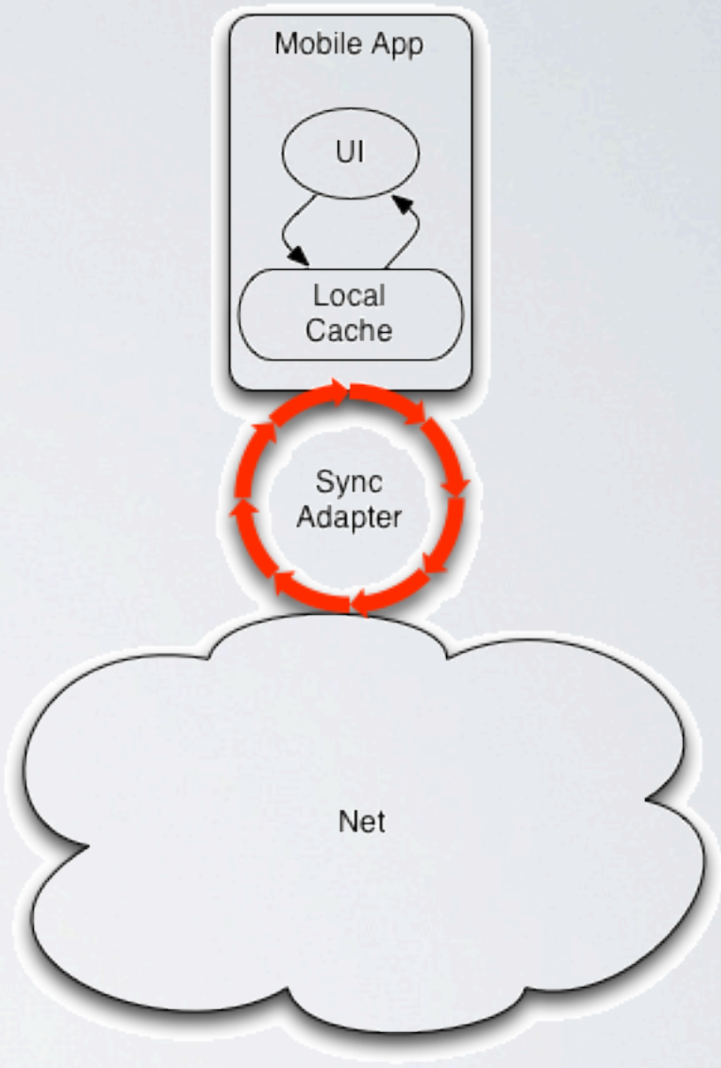
- Using the radio is one of the most battery intensive things the device can do. Once it is turned on, it stays on for several seconds, needed or not

- It’s tricky to synchronize asynchronously, if it requires securely stored credentials.



The Sync Adapter

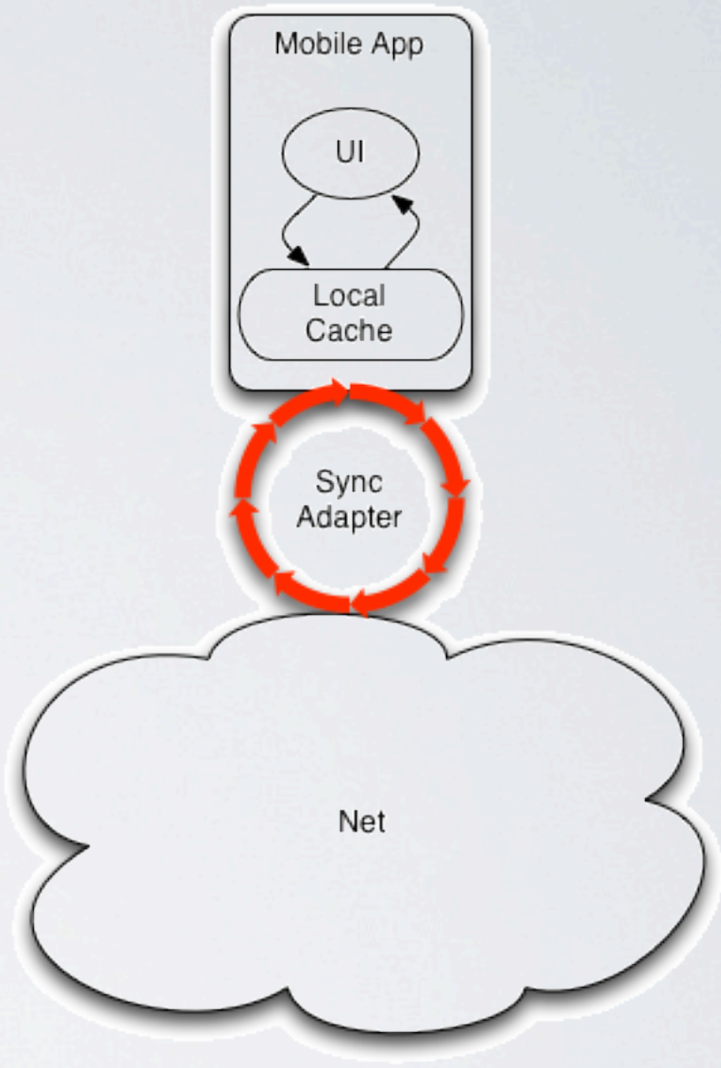
- Efficient: cross-device synchronization coordination
- Secure: supports a very wide range of authentication schemes
- Egalitarian: requires hundreds of lines of code for even the simplest synchronization strategy
- Magic....



The Sync Adapter

A bindable service:

- Part of your application
- Bound by the SyncManager
- Run as your app (uid)




```
public abstract void onPerformSync(
    Account account,
    Bundle extras,
    String authority,
    ContentProviderClient provider,
    SyncResult syncResult);
```

Account?

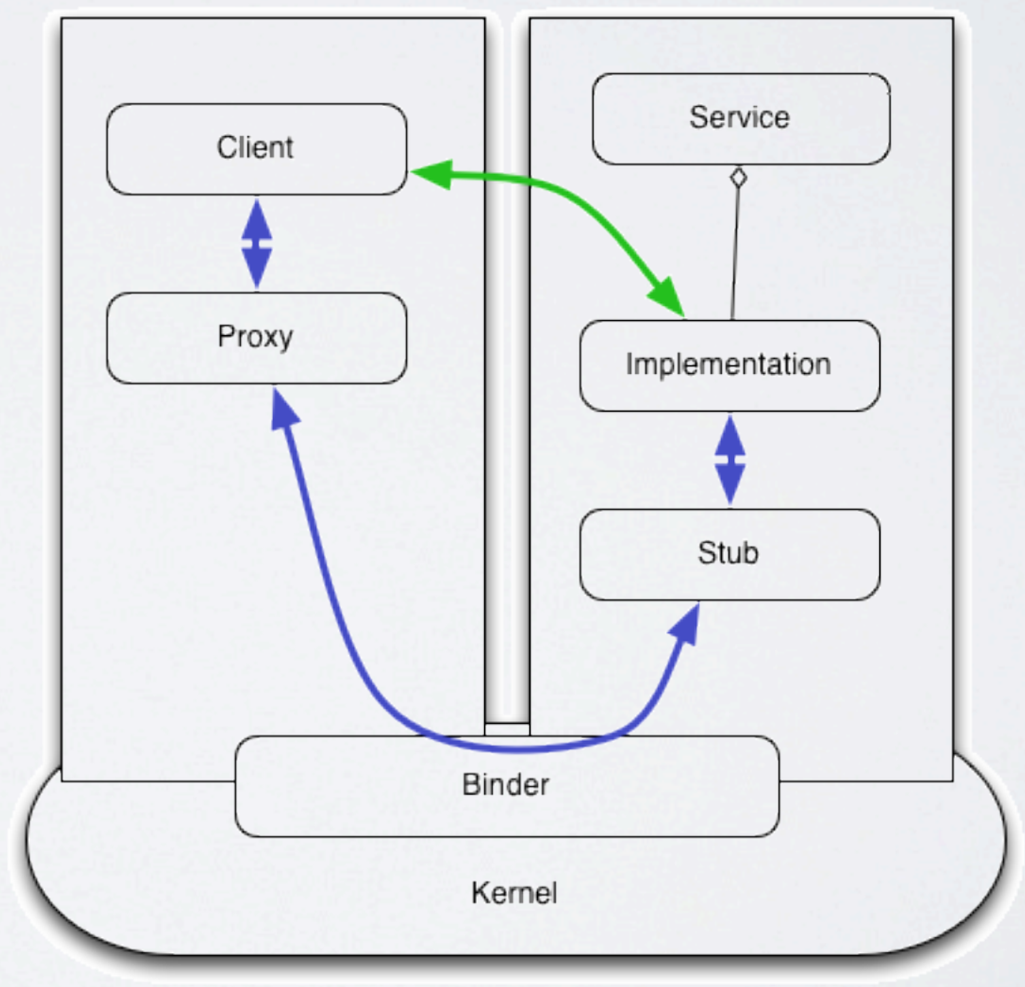
The unit of synchronization, for a SyncAdapter,
is an `Account`

We need two detours here:

- Bound Services
- Accounts

Bound Service

Nearly unrelated to a Service, a BoundService is Android's richest form of Inter-process communication



Binding a Service

1) Bind the service:

```
Intent i = new Intent(this, LocalService.class);
bindService(i, this, Context.BIND_AUTO_CREATE);
```

Binding a Service

2) Await the asynchronous connection callback:

```
public void onServiceConnected(
  ComponentName className,
  IBinder service)
{
  this.service =  IMyService.Stub.asInterface(service);
}
```

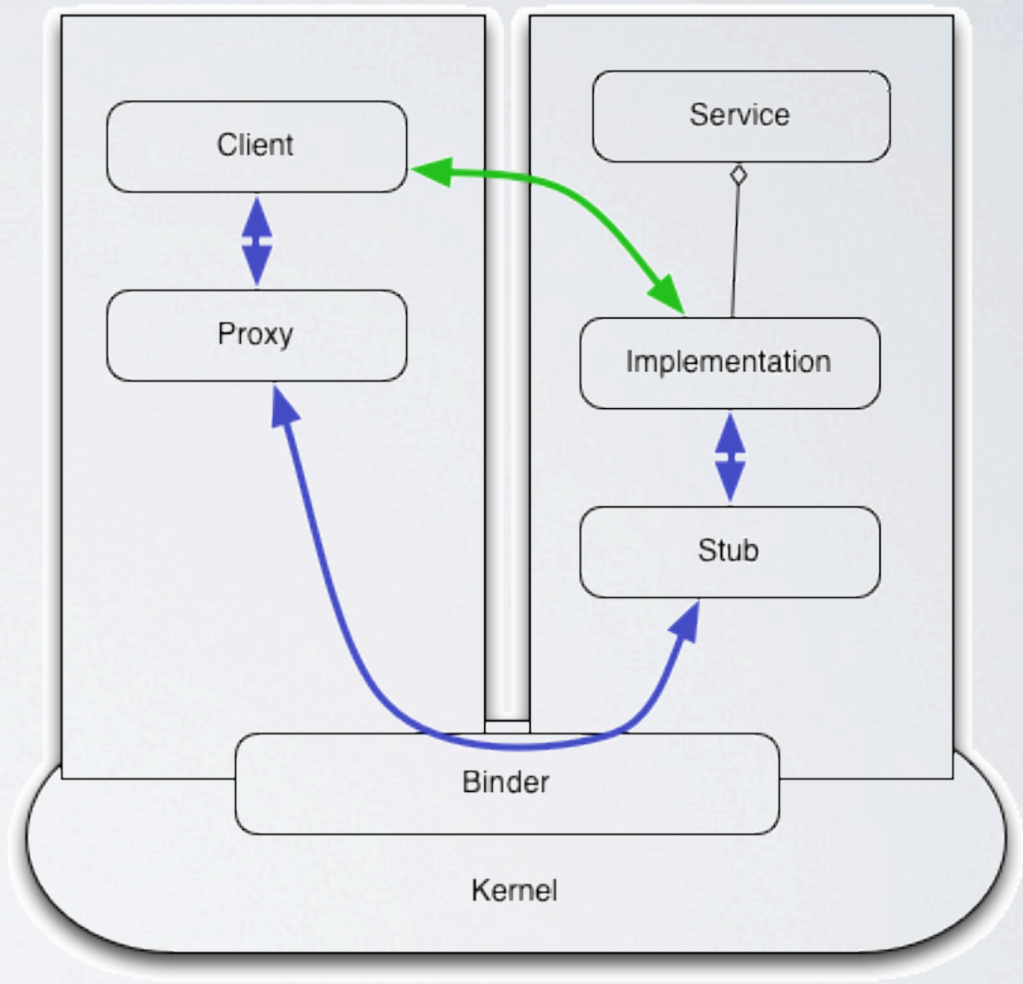
Binding a Service

3) Profit!

The service is a factory

The implementation is running in the process of the owning application

... but appears to your app to be a local class.



Back to Accounts

A SyncAdapter synchronizes an account.

So, what's an account?

Accounts

This is an account:

```
Account account = new Account(acctName, accountType);
acctExtras = new Bundle();  // add other stuff here
AccountManager.get(this).addAccountExplicitly(
    account,
    password,
    acctExtras);
```

1. the account type
2. the account name
3. an “extras” bundle
4. created with addAccountExplicitly

But wait...

- What is an AccountType?
- How will the SyncAdapter find an account?

```
public abstract void onPerformSync(
    Account account,
    Bundle extras,
    String authority,
    ContentProviderClient provider,
    SyncResult syncResult);
```

It’s complicated...

Creating an Account

1. Declare an account authentication service in the manifest
2. Request the permissions necessary to manage accounts
3. Create a meta-data declaration that refers to a resource describing the application's account type and add it to the service declaration.
4. Implement the service. It must return an instance of a subclass of AbstractAccountAuthenticator
5. Implement the Authenticator
6. Optionally, implement preference pages for the account

1. Declare an account authentication service

AndroidManifest.xml

```
<service
    android:name=".AccountService"
    android:exported="false">
    <intent-filter>
        <action
            android:name="android.accounts.AccountAuthenticator"/>
    </intent-filter>

    <!-- watch this space -->
</service>
```

2. Request permissions

AndroidManifest.xml

```
<uses-permission
    android:name="android.permission.AUTHENTICATE_ACCOUNTS"/>
<uses-permission
    android:name="android.permission.GET_ACCOUNTS"/>

<uses-permission
    android:name="android.permission.WRITE_SYNC_SETTINGS"/>

<uses-permission
    android:name="android.permission.USE_CREDENTIALS"/>

<uses-permission
    android:name="android.permission.MANAGE_ACCOUNTS"/>
```

3. Define the AccountType

account.xml

```
<?xml version="1.0" encoding="utf-8"?>
<account-authenticator
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="@string/account_type"
    android:icon="@drawable/ic_launcher"
    android:label="@string/app_name"
    android:smallIcon="@drawable/ic_launcher" />
```

AndroidManifest.xml

```
<service
    android:name=".AccountService"
    android:exported="false">
    <!-- intent filter... -->
    <meta-data
        android:name="android.accounts.AccountAuthenticator"
        android:resource="@xml/account"/>
</service>
```

4. Implement the Service

AccountService.java

```
public class AccountService extends Service {
    private volatile AccountMgr mgr;

    @Override
    public void onCreate() {
        super.onCreate();
        mgr = new AccountMgr(getApplication());
    }

    @Override
    public IBinder onBind(Intent intent) {
        return mgr.getIBinder();
    }
}
```

5. Implement the Authenticator

Not strictly necessary!

```
Account account = new Account(acctName, accountType);
acctExtras = new Bundle();  // add other stuff here
AccountManager.get(this).addAccountExplicitly(
    account,
    password,
    acctExtras);
```

Remember the definition of an account?

As long as an account of the syncable type exists, and the SyncManager can get a token for it, it can sync.

```
ContentResolver.setIsSyncable(account, provider, 1);
ContentResolver.setSyncAutomatically(account, provider, true);
ContentResolver.addPeriodicSync(account, provider, new Bundle(), 8 * 60 * 60);
```

Implementing AbstractAccountAuthenticator .addAccount

Three modes:

- Immediate
- Intent
- Delayed

addAccount in Immediate Mode

Simply return a bundle containing values for the caller

AccountMgr.java

```
public Bundle addAccount(
    AccountAuthenticatorResponse resp,
    String accountType,
    String authTokenType,
    String[] requiredFeatures,
    Bundle options)
    throws NetworkErrorException
{
    Bundle reply = new Bundle();
    reply.putString(AccountManager.KEY_ACCOUNT_TYPE, R.string.account_type);
    reply.putString(AccountManager.KEY_ACCOUNT_NAME, "MyAccount");
    reply.putString(AccountManager.KEY_AUTHTOKEN, "SEKRIT");
    reply.putString(AccountService.ENDPOINT, "http://my.service.com");
    return reply;
}
```

addAccount in Intent Mode

Return a bundle containing an Intent that will start an Account creation activity.

AccountMgr.java

```
public Bundle addAccount(
    AccountAuthenticatorResponse resp,
    String accountType,
    String authTokenType,
    String[] requiredFeatures,
    Bundle options)
    throws NetworkErrorException
{
        if (0 < AccountManager.get(app).getAccountsByType(at).length) {
            reply.putInt(AccountManager.KEY_ERROR_CODE, -1);
            reply.putString(AccountManager.KEY_ERROR_MESSAGE, "Account exists");
            return reply;
        }
        Intent intent = new Intent(app, NewAccountActivity.class);
        reply.putParcelable(AccountManager.KEY_INTENT, intent);

        return reply;
}
```

addAccount in Intent Mode

1. Subclass AccountAuthenticatorActivity
2. Do whatever you want: consult local data, the network, eyeball scanners or goat entrails
3. Call finish() !!

addAccount in Delayed Mode

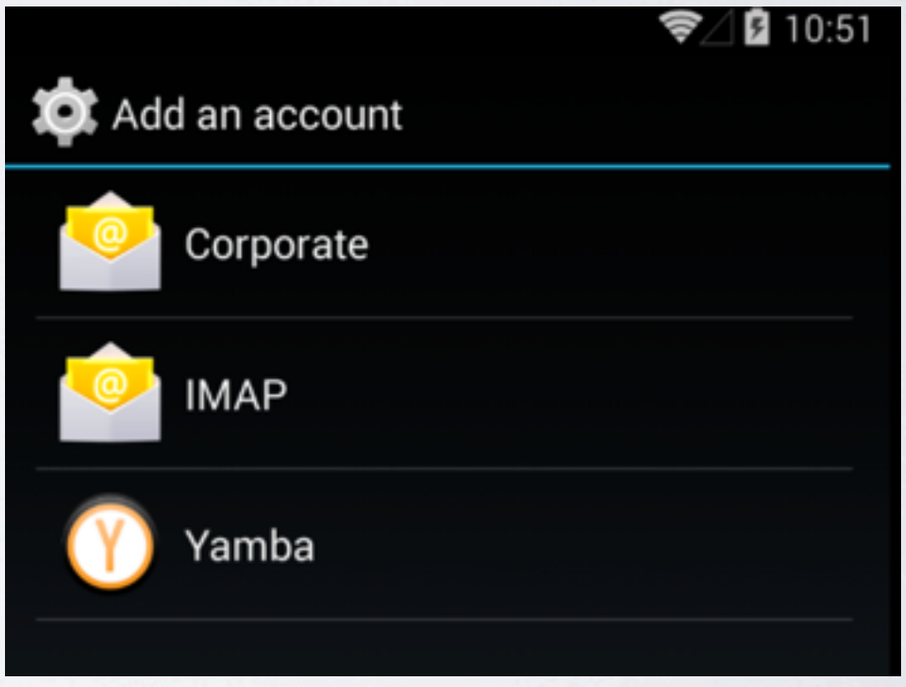
Return null.

This is dangerous...

Using the Account Manager

The canonical app Settings is a great example.

It first iterates over all AccountTypes and displays the associated icon.

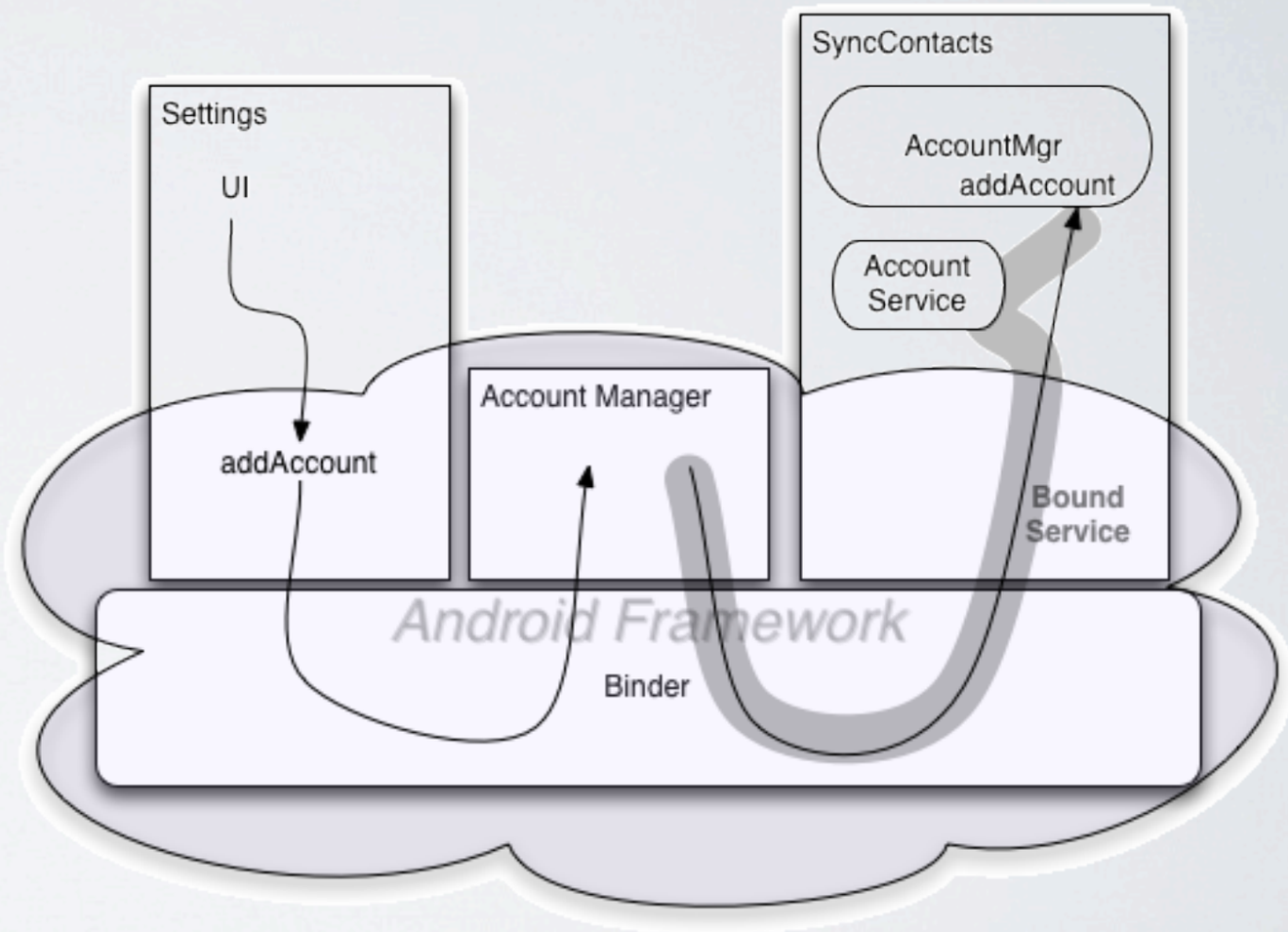


Settings

Setting uses the AccountManager to create an account

- Local library call binds AccountManager
- AccountManager binds your AccountMgr

Secure!



Using the Account Manager

```
public AccountManagerFuture<Bundle> addAccount(
    String accountType,
    String authTokenType,
    String[] requiredFeatures,
    Bundle addAccountOptions,
    Activity activity,
    AccountManagerCallback<Bundle> callback,
    Handler handler)
```

Notice that the return type is a Future<Bundle>

The Bundle is the one returned by your addAccount!

addAccount in Delayed Mode

In *delayed* mode, the framework is not responsible for fulfilling the Future.

```
public Bundle addAccount(
    AccountAuthenticatorResponse resp,
    String accountType,
    String authTokenType,
    String[] requiredFeatures,
    Bundle options)
    throws NetworkErrorException
```

The second argument to addAccount is an

`AccountAuthenticatorResponse`

Calling it's onResponse method unblocks the caller's Future.

Somebody better call it!

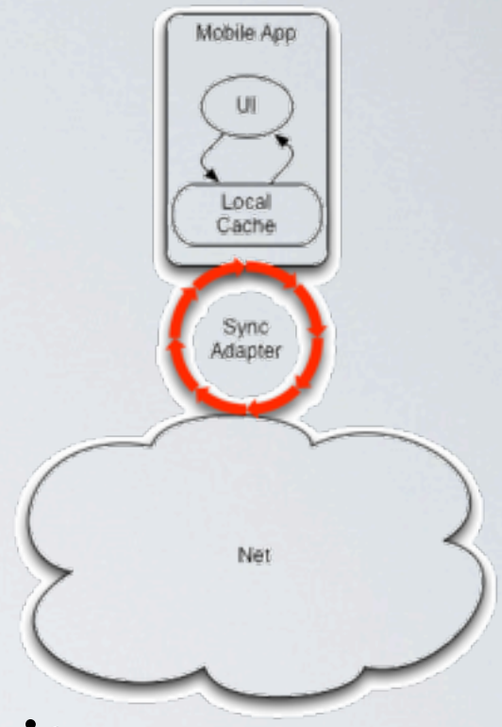
The AccountAuthenticatorResponse

- Parcelable
- In *Intent* mode, available to the Activity as

```
AccountManager.KEY_ACCOUNT_AUTHENTICATOR_RESPONSE
```

- Completely ignored by the Settings app

Remember the Sync Adapter?



Looks easy, compared to the Account Manager:

1. Declare an sync service in the manifest
2. Create a meta-data declaration that refers to a resource that connects an Account to an Authority. Add it to the service declaration
3. Implement the service. It must return an instance of a subclass of AbstractThreadedSyncAdapter
4. Implement AbstractThreadedSyncAdapter.onPerformSync

ContentProviders: Brief Reprise

Recall that, in Android, a URL can stand for a dataset.

An *Authority* is associated with a ContentProvider. The ContentResolver forwards CRUD calls to URIs of the form `content://authority/path` to the associated provider

Clients call:

```
getContentResolver().registerContentObserver(uri, true, obs)
```

... to be notified when there is a call to:

```
getContentResolver().notifyChange(uri, null)
```

... indicating that the dataset has changed.

1. Declare a sync service

AndroidManifest.xml

```
<service
    android:name=".SyncService"
    android:exported="false">
    <intent-filter>
        <action
            android:name="android.content.SyncAdapter"/>
    </intent-filter>

    <!-- watch this space -->
</service>
```

2. Relate an Account and an Authority

sync.xml

```
<?xml version="1.0" encoding="utf-8"?>
<sync-adapter
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:accountType="@string/account_type"
    android:contentAuthority="com.twitter.android.yamba"
    android:isAlwaysSyncable="true" />
```

AndroidManifest.xml

```
<service
    android:name=".SyncService"
    android:exported="false">
    <!-- intent filter... -->
    <meta-data
        android:name="android.contents.SyncAdapter"
        android:resource="@xml/sync"/>
</service>
```

3. Implement the Service

SyncService.java

```
public class SyncService extends Service {
    private static final String TAG = "SYNC_SVC";

    private volatile SyncAdapter synchronizer;

    @Override
    public void onCreate() {
        super.onCreate();
        synchronizer
            = new SyncAdapter(getApplication(), true);
    }

    @Override
    public IBinder onBind(Intent intent) {
        return synchronizer.getSyncAdapterBinder();
    }
}
```

4. Implement onPerformSync

Use:

`AccountManager.getAuthToken`

...to ensure that the account is authenticated.

It, also, returns a `Future<Bundle>` under the same tri-modal contract as `addAccount`.

Use:

`AccountManager.blockingGetAuthToken`

...to force wait for authentication: Probably what you want.

If this smells like OAuth, so be it.

Forcing a Sync

There are several ways to force a Sync:

On request:

```
ContentResolver.requestSync(ACCOUNT, AUTHORITY, null);
```

On network tickle:

```
getContentResolver()
    .setSyncAutomatically(ACCOUNT, AUTHORITY, true);
```

On periodically:

```
ContentResolver
    .addPeriodicSync(ACCOUNT, AUTHORITY, null, INTERVAL);
```

Closing the Loop!

... and Magic!

Remember that call to:

```
getContentResolver().notifyChange(uri, null)
```

... indicating that the dataset has changed?

There is an second method:

```
getContentResolver().notifyChange(uri, null, true)
```

... that causes the associated sync adapter be run for all accounts of the type associated with the URI!

Thank you!

Code and slides available:

bmeike@twitter.com