SCO Forum 2006

MOBILITY EVERYWHERE >

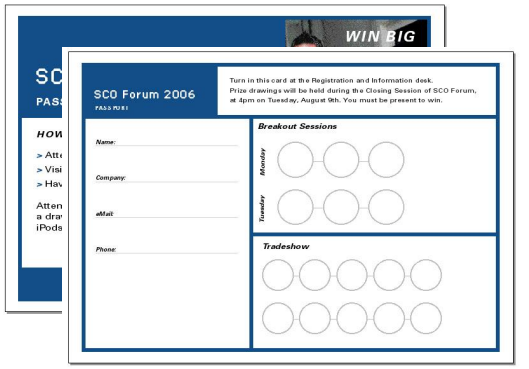
# Building EdgeClick clients for cell phones and smartphones with Java ME

**Cliff Jansen**

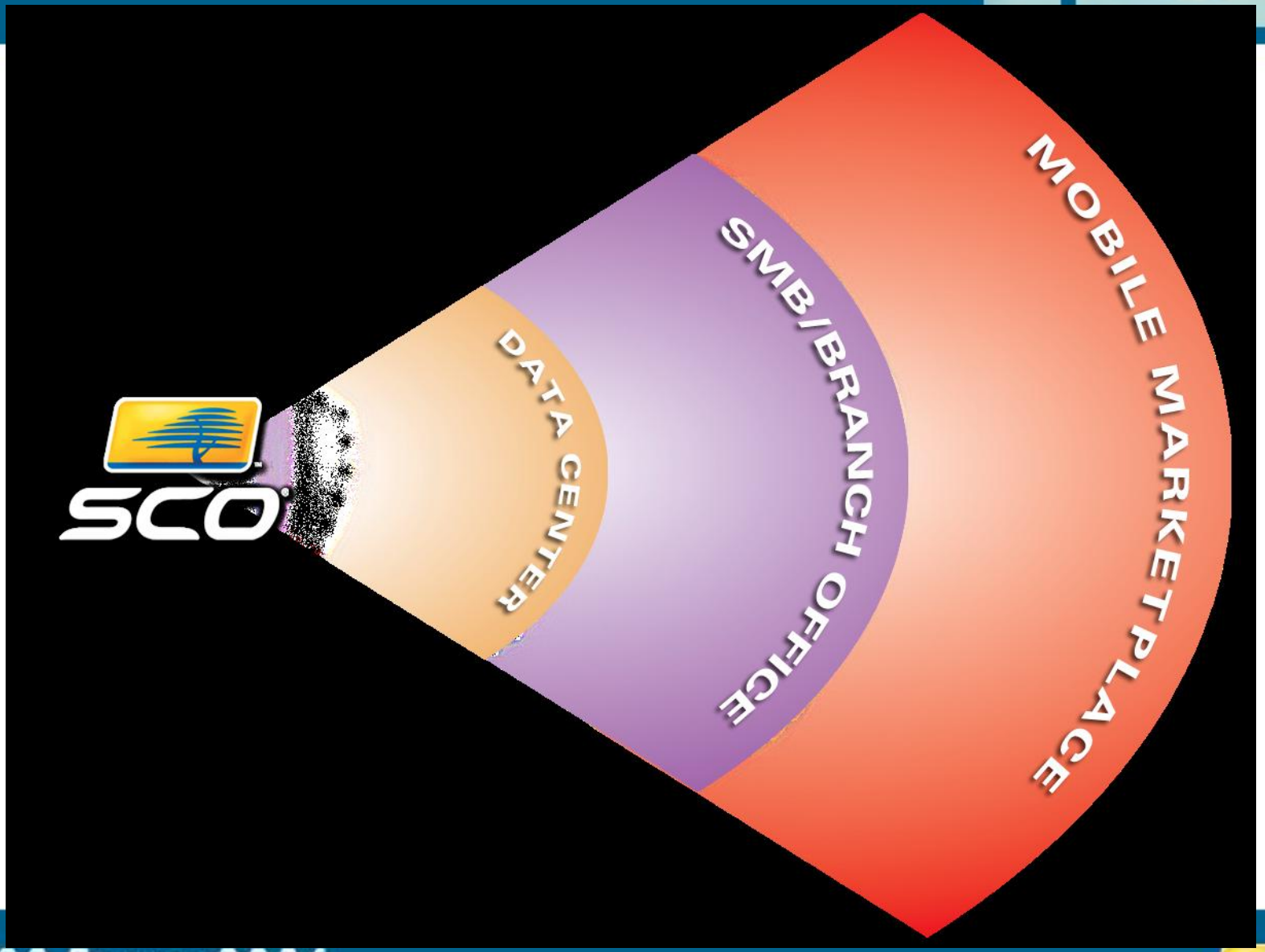
**Session ID: 110**

# Get Your Passport Stamped

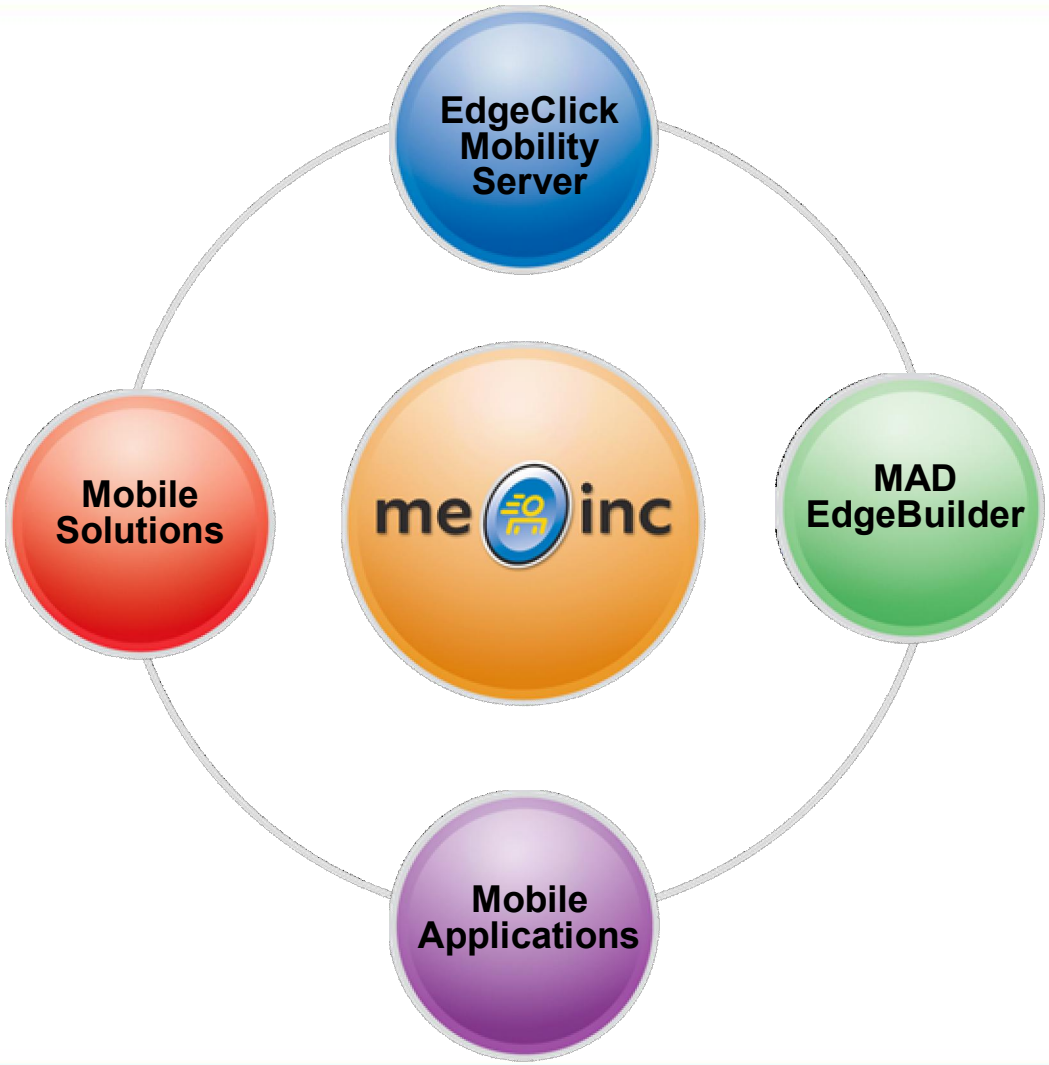
- Be sure to get your Passport stamped.
  - Get your passport stamped
    - By breakout session instructors
    - By exhibitors in the exhibit hall
  - Turn in your Passport
    - After the last breakout session on Wednesday
    - Drawing for great prizes for Wrap-up Session
- Remember to complete the breakout session evaluation form, too



# SCO Automates Transactions



# Mobility Everywhere



# Agenda

- Java ME and EdgeClick
- EdgeClick Architecture
- EdgeClick Java ME API
- Constructing a Java ME client application
- Finding your own Java ME Sweet Spot

# The Hype

- 600 Million Java ME enabled phones sold in the last year alone!
- Devices very inexpensive; often free!
- It's Java.  Write once run everywhere!

# The Critics Respond

- It's not really Java.
- Ifdef's?  See point #1.
- They forgot the "I" in I/O.
- They forgot a lot of the "O" too.
- Just try to get TCP/IP configured.
- Security and Permissions are a nightmare.
- Stop glossing over the obvious:
  "Fragmentation" really means "Total Chaos"

# The Market

- Slim handsets are selling like hot cakes
  - Surprise: small “I” and small “O”
- The “little Java that could” is right at home and continues to grow in popularity
- EVDO and HSDPA changing that “Wait and Pay” feeling
- The Opera Mini phenomenon: users (and carriers too) waking up to the possibilities of mobile data

# Which Java ME?

- Java ME is an umbrella term
- EdgeClick runs on MIDP 2.0 and CLDC 1.x
- EdgeClick falls in the MIDP 2.0 sweet spot:
  - You can get your application to the phone
  - Your application can talk to the EdgeClick Processor
- But… the application still has to wrestle with fragmentation issues outside the scope of the EdgeClick library.

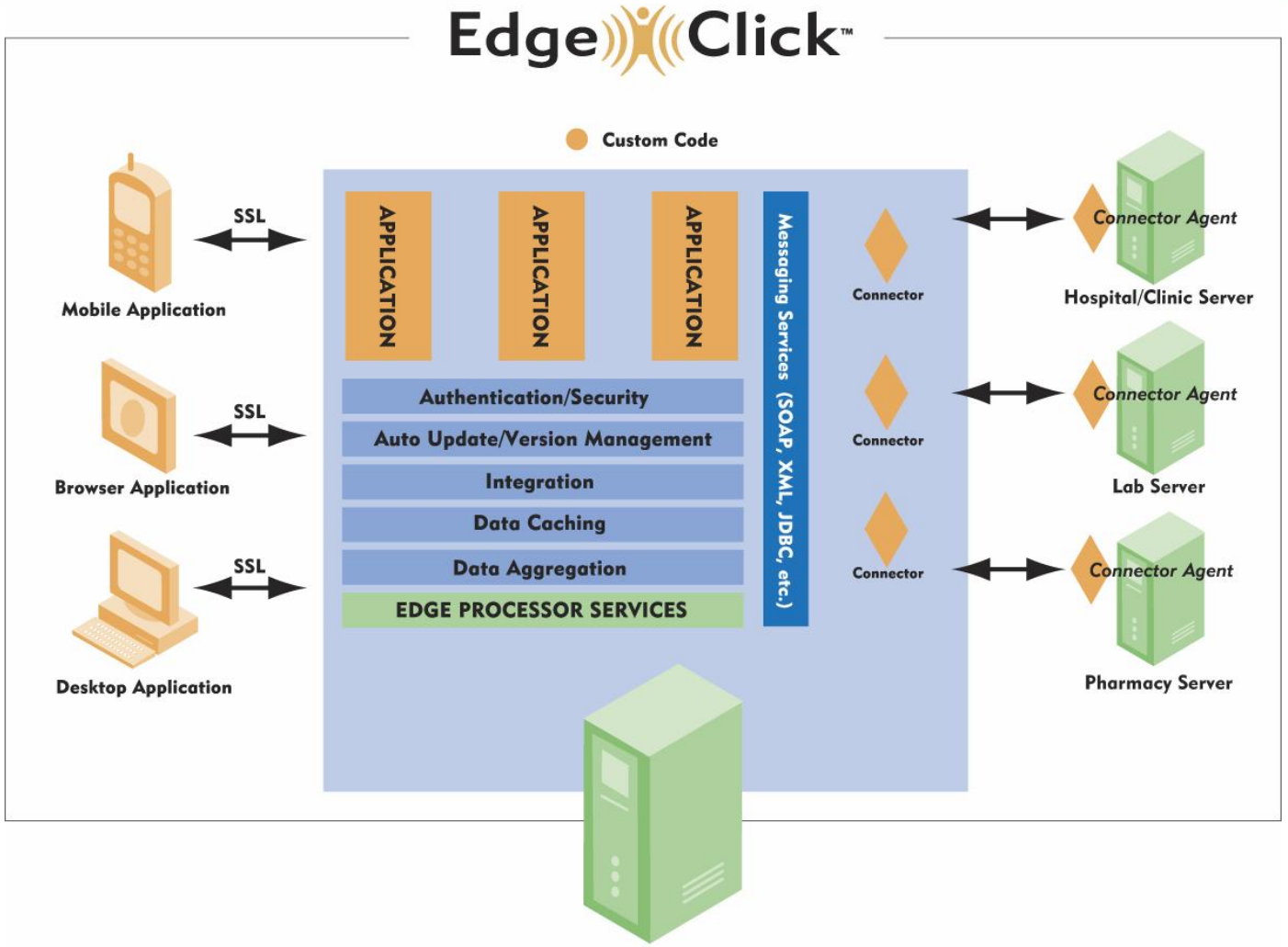
# MIDP 2.0 Devices

- Low end:
  - 8k persistent store
  - 64k jar size
  - Cramped display (90x90)
- High end
  - PDA Specs
- EdgeClick applications run on the full spectrum

# EdgeClick Java ME Client Libraries

- Very light weight
  - Compact code and runtime footprint
- No reliance on any extensions (i.e. XML, FC, PIM)
- No thrown exceptions
- Thread protection through MIDP event model
- GUI agnostic
  - Canvas
  - Form
- Control flow agnostic
  - Exception: pushView and popView
- Can be used with any Java ME IDE or toolkit

# EdgeClick Architecture



# EdgeClick Java ME Class Overview

- edgeclick.accounts
  - **Account** – class representing a single account
  - **AccountMgr** – class to manage multiple accounts
  - **AccountView, AccountListView** – optional UI classes
- edgeclick.groups
  - **Contact** – class representing a single contact
  - **GroupMgr** – keeps track of selected groups and contacts
  - **ContactsView, SyncGroupsView** – UI classes for selecting recipients and refreshing data from the EdgeClick Processor

# Class Overview (continued)

- edgeclick.net
  - **Http** – class for sending HTTP requests in separate thread
  - **HttpUtil** – helper functions for parsing EdgeClick server XML responses
- EdgeClick.App
  - class that provides application properties that are required to be defined by every EdgeClick application. Controls lifecycle of the application (subclass of MIDlet).

# Class Overview (continued)

- edgeclick.mid
  - **MidpControllable** – lifecycle notifications (see App.register())
  - **MidpUtil** – simple persistent store and debugging
- EdgeClick.ui
  - AppView – GUI navigation interface
  - MsgScreen – shared message display form
  - LogoLine – class to provide consistent look for forms
  - PleaseWait – interface for the networking progress screen
  - Button

# EdgeClick Java ME API – AccountMgr

- Only one instance of the **edgeclick.accounts.AccountMgr** class can exist, and is referenced via **App.accountMgr**
- Responsible for maintaining the account persistent store.
- Although multiple accounts can be created and authenticated, only one can be active (or current) at one time.
- **App.accountMgr.currentAccount** will point to the current, authenticated account.

# Constructing a Java ME client application

1. Inherit the App class for the top level of your application.
2. Define some required EdgeClick properties for your application (name, version number, etc)
3. Initialize and install your first screen
4. Obtain authentication credentials from the EdgeClick processor
5. Use the edgeclick.net API to send commands to the EdgeClick processor
6. Parse the results
7. Error handling

# Step 2 – Define application properties

- In *YourApp.java*, define a number of EdgeClick application properties needed by the EdgeClick library:

```
// Application's name
appName = "Sample";

// Application's version
appVersion = "1.0.0";

// PleaseWait override (optional)
pleaseWait = new MyPleaseWait();
```

# Step 3 – Initialize

- For greatest flexibility, override App.midletInit():

```
public void midletInit (MIDlet m) {

    // Things that must be first go here.

    super.midletInit (m);   // registered Objects next

    // Remainder of initialization

    if (myStartupFailed) {
        pushView (new FatalError ("Initialization failed"));
        return;
    }

    pushView (new MyFirstView());
}
```

# Step 4 – Authenticate

- The AccountMgr remembers the current account from the last time the application ran. The current account must be established on the first invocation. The ecp.properties resource file defines default account settings.

```
if (!AccountMgr.currentAccountSet()) {

   // Optionally display "Please Login" message

   // Call the GUI interface

   App.pushView(AccountListView.getView());
}
```

# Step 5 – Send commands to the EP

- Communication between the phone and the EdgeClick Processor is done via HTTP POST request.
- The results are returned as XML.
- Commands are identified via a unique integer code.
- Any additional data specific to that command can be specified via attribute/value pairs, constructed using **Http.addTuple()**.

# Step 5 – Sample code

```
private void myOrderEcpRequest() {
    pleaseWait = App.getApp().getPleaseWait();
    plsWaitMsg = "Retrieving customer orders”;
    pleaseWait.show (plsWaitMsg, http_, myNextScreen);

    http_.addTuple ("CMD", CUSTOMER_ORDERS);
    http_.addTuple (“order_type", “recent”);
    http_.post (this, App.currentAccount.ecpUrl());
    // New Http worker thread now launched
}
```

# Step 5 – Sample code

- HttpListener callback is called when the request completes. Executes in separate worker thread.

```
public void httpDone() {

   pleaseWait.hide();
   if (/* recoverable error */) {
       pleaseWait.show ("try #2", http_, );
       http_.post (this, App.currentAccount.ecpUrl());
       return;
   }

   // Process response here as worker thread or
   // transfer control back to UI thread.
}
```

# Step 6 – Parse the Results

- The XML returned has the following format:

```
<return>
  <code>return_code</code>
  <result>XML_result</result>
</return>
```

- The **Http** class defines all the possible return codes.
- In practice the code will always be:
  - 0 – **CODE_SUCCESS** or,
  - 10 – **CODE_NEW_VERSION_AVAILABLE**.
- All other return codes will indicate an unexpected error condition.

# Step 6 – XML strategies

- On success, the response is either in your designated OutputStream or available from:

```
Http.getResponse()
```

- JSR 172 – the "Web Services APIs" extension – is frequently available on newer devices. Standards based. Program won't run on devices without the extension.
- Third party XML parsers are available that run on MIDP2.0. Will run on any device. Must be packaged inside you application jar file.
- Parse XML data directly using String and StringBuffer. Examples can be found in the EdgeBuilder documentation.

# Step 7 – Error handling

- **Http.succeeded()** -> No networking errors encountered.  Http class can also be used to talk to arbitrary http servers.

- **Http.interrupted()** -> Http.interrupt() was called sometime after Http.post().  Usually because the user requested "Cancel" in the PleaseWait progress screen.

# Step 7 – Error handling

- XML helper functions:

```
String response = http_.getResponse();

int ecpCode = HttpUtil.getEcpCode (response);

String ecpMsg = HttpUtil.getEcpMsg (response);

boolean isExpired = HttpUtil.authExpired
                                    (response);
```

# Step 7 – Error handling

- Application error codes:

```
Http.CODE_NULL_COMMAND
Http.CODE_UNKNOWN_COMMAND
Http.CODE_UNSUPPORTED_VERSION
Http.CODE_INVALID_ARGUMENTS
Http.CODE_EXCEPTION
Http.CODE_APP_EXCEPTION
```

- the command code, application index, version number or arguments were not valid, or the EdgeClick service itself threw an exception

# Step 7 – Error handling and Authentication

- **HttpUtil.authExpired**(resp) – if true, the session ID for the current account has expired, the user must enter her password to re-authenticate:

  Use: AccountView.reAuthenticate ()

- **`ecpCode == Http.CODE_NOT_AUTHENTICATED`**

  Use: AccountView.edit(accountId)

# Step 7 – Error handling and Version Control

- **Http.CODE_NEW_VERSION_AVAILABLE.** The version number of the client is older than the current version, but not old enough to be unsupported
- It is up to you how to handle this code, but it is recommended that it is checked for and a message displayed when sending the first command to theEdgeClick Processor, and ignored afterwards.

- **Http.CODE_UNSUPPORTED_VERSION.** The version number of the client is too old.
- There is no current support for upgrading an application. But see:

  MIDlet.platformRequest()

# Finding your own Java ME Sweet Spot

- Move functionality to the EdgeClick processor
  - Client is smaller
  - Contain the client upgrade cycle
- Vanilla should be your favorite flavor
- Narrow the range of target devices
- Use extensions for functional need, but confirm permissions for the device AND the carrier.
- Evaluate how the user will relate to the Small "I"

# Q & A

- Questions?