
Google
Developer
Day2009

GWT App Architecture Best Practices

Fred Sauer

June 9, 2009

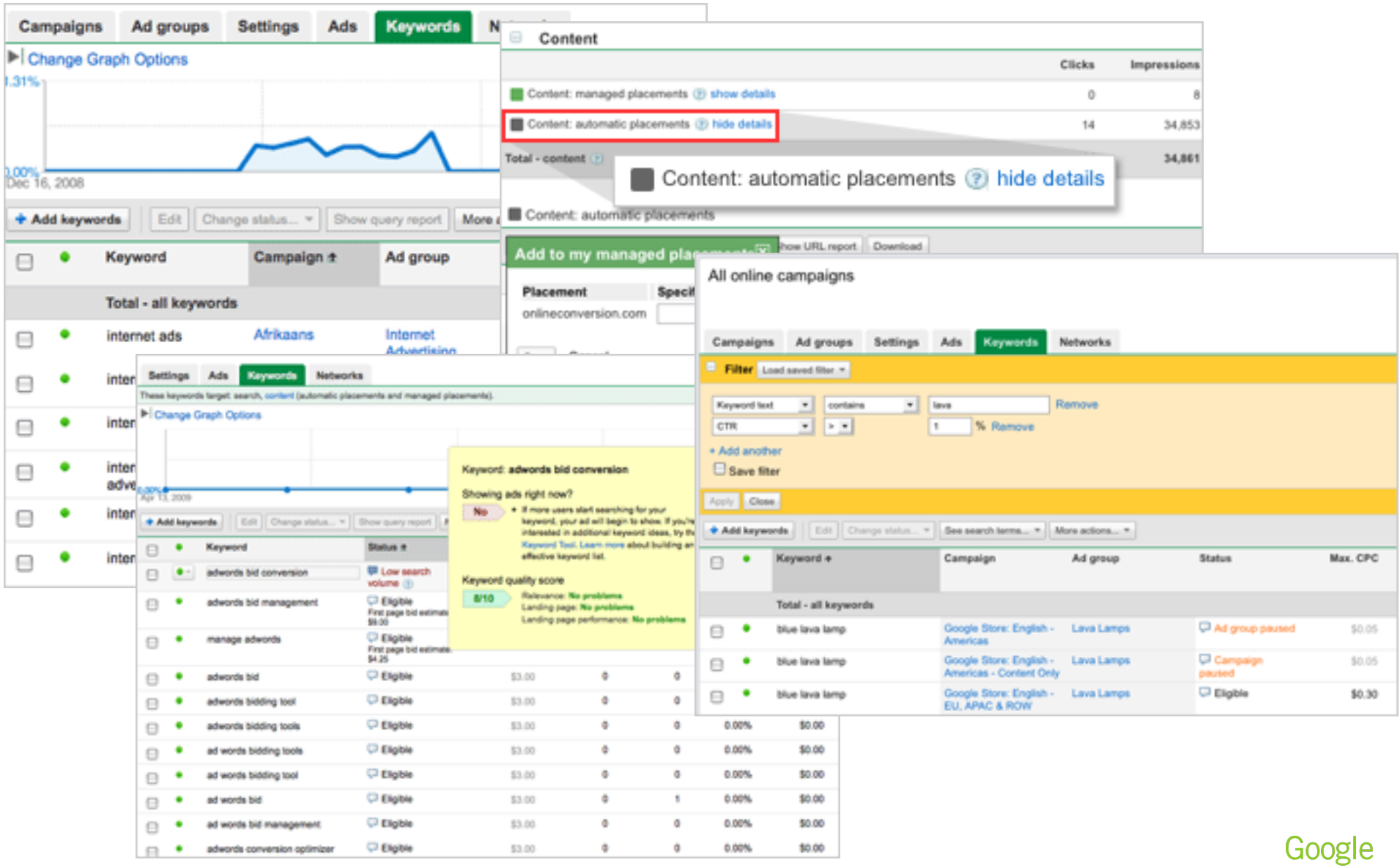
Based on original presentation by Ray Ryan

Google Developer Day 2009

What are we talking about?

- How to organize a nontrivial GWT application
- Particular focus on client side
- Lessons learned from new AdWords UI

New AdWords UI



http://www.google.com/adwords/newinterface/

Google Developer Day2009

If you remember nothing else...

- Get browser **History** right, and get it right early
- Use an **Event Bus** to fight spaghetti
- **DI** + **MVP** FTW

Dependency **I**njection
+
Model / **V**iew / **P**resenter

for the win!

Three major themes

1. Embrace asynchrony
2. Always be decoupling
3. Strive to achieve statelessness

Theme 1:
Embrace Asynchrony

Remember what that **A** in **A**JAX is for

```
class Contact {
  String name;
  List<ContactDetail> details;
  List<ContactDetail> getContactDetails() {
    return details;
  }

  String getName() {
    return name;
  }
}
```

Remember what that **A** in **A**JAX is for

```
class Contact {
  String name;
  ArrayList<ContactDetailId> detailIds;

  ArrayList<ContactDetailId> getDetailIds() {
    return detailIds;
  }

  String getName() {
    return name;
  }
}
```

Command pattern to make async tolerable

Leverage point for

- Caching
- Batching
- Centralize failure handling

Lays the groundwork for

- Code splitting via GWT.runAsync()
- Undo / Redo
- Gears / HTML5 DB

Use Command pattern RPC

```
// The name 'Command' is taken; Use 'Action' instead
interface Action<T extends Response> { }

interface Response { }

// Familiar GWT RPC sync/async method signatures
interface ContactsService extends RemoteService {
  <T extends Response> T execute(Action<T> action);
}

interface ContactsServiceAsync {
  <T extends Response> void execute(Action<T> action,
      AsyncCallback<T> callback);
}
```

Command style RPC example

Write an Action…

```
class GetDetails implements Action<GetDetailsResponse> {
  private final ArrayList<ContactDetailId> ids;

  public GetDetails(ArrayList<ContactDetailId> ids) {
    this.ids = ids;
  }

  public ArrayList<ContactDetailId> getIds() {
    return ids;
  }
}
```

Command style RPC example

…and its Response…

```
class GetDetailsResponse implements Response {
  private final ArrayList<ContactDetail> details;

  public GetDetailsResponse(ArrayList<ContactDetail> contacts) {
    this.details = contacts;
  }

  public ArrayList<ContactDetail> getDetails() {
    return new ArrayList<ContactDetail>(details);
  }
}
```

Command style RPC example

…plus convenience AsyncCallback...

```
abstract class GotDetails implements
    AsyncCallback<GetDetailsResponse> {

  public void onFailure(Throwable oops) {
    /* default app wide failure handling */
  }

  public void onSuccess(GetDetailsResponse result) {
    got(result.getDetails());
  }

  public abstract void got(ArrayList<ContactDetail> details);
}
```

Command style RPC example

Make it go

```
// Ask that a contact be shown
void showContact(final Contact contact) {
  service.execute(new GetDetails(contact.getDetailIds()),
    new GotDetails() {
      public void got(ArrayList<ContactDetail> details) {
        renderContact(contact);
        renderDetails(details);
      }
   });
}
```

Theme 2:
Always be decoupling

Decoupling advantages

With a combination of...

- An **Event Bus**
- **D**ependency **I**njection of app-wide services
- **MVP** pattern for your custom widgets

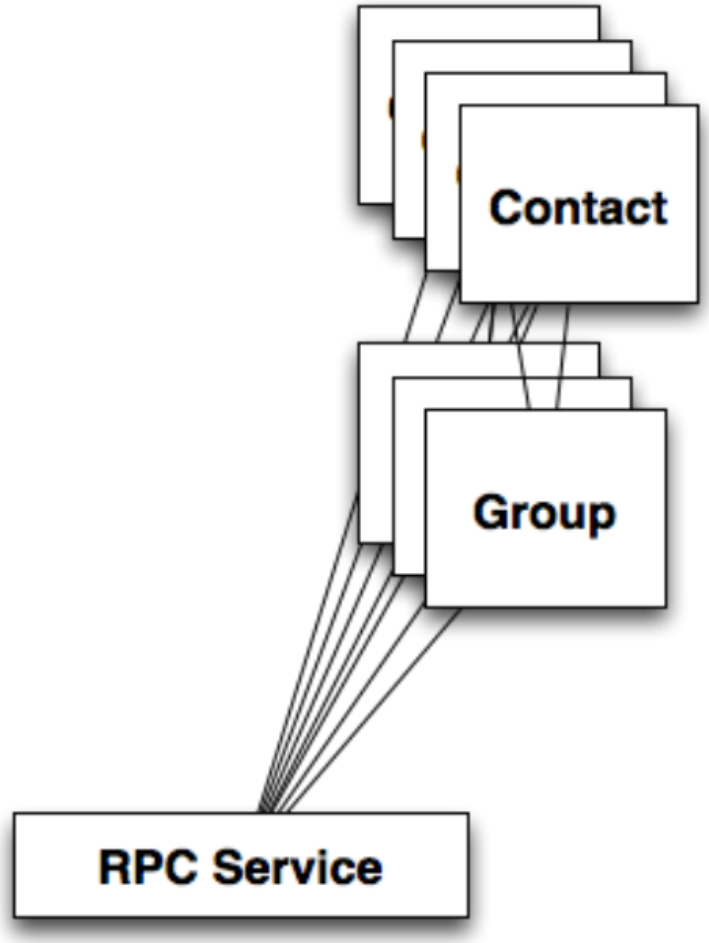
You get...

- Easy rejiggering of the app
- Easy to defer pokey DOM operations
- Easy unit testing
- Fast test execution

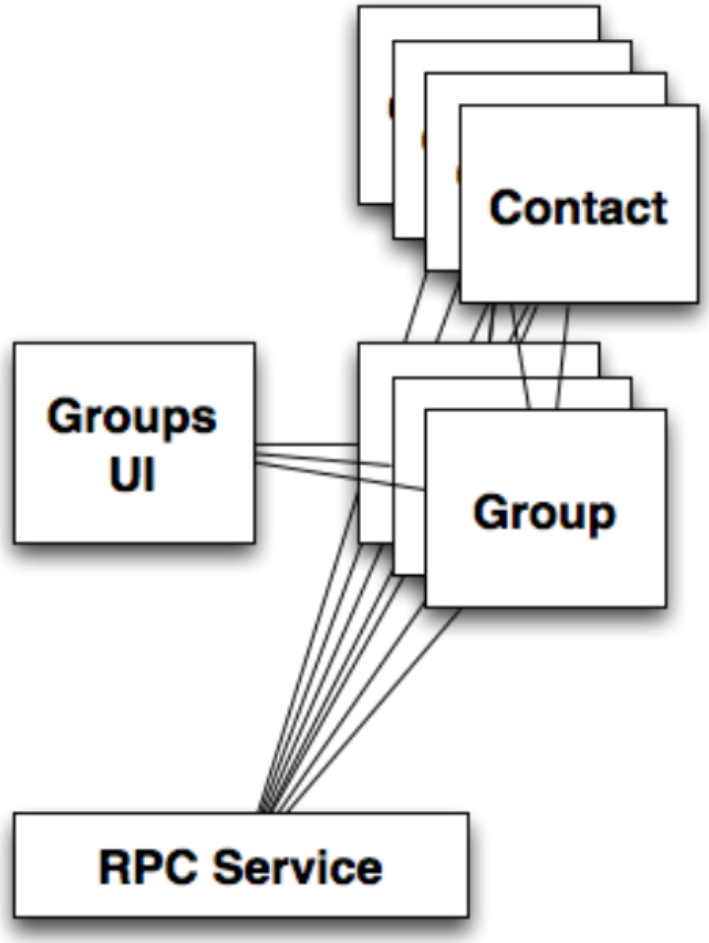
Always be decoupling: Decoupling via the Event Bus

Google Developer Day 2009

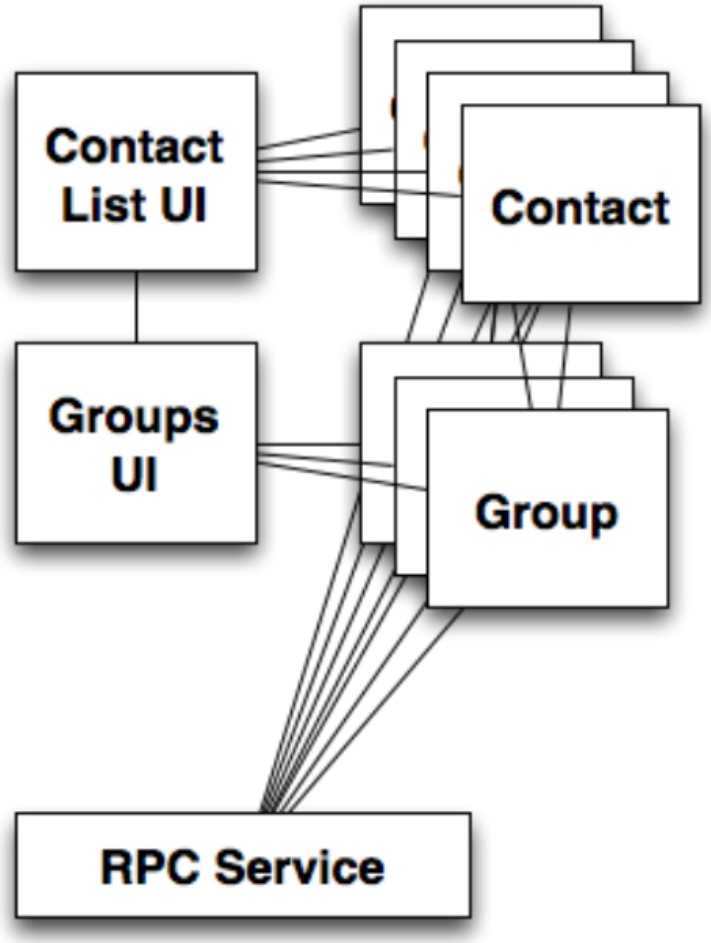
Coupling



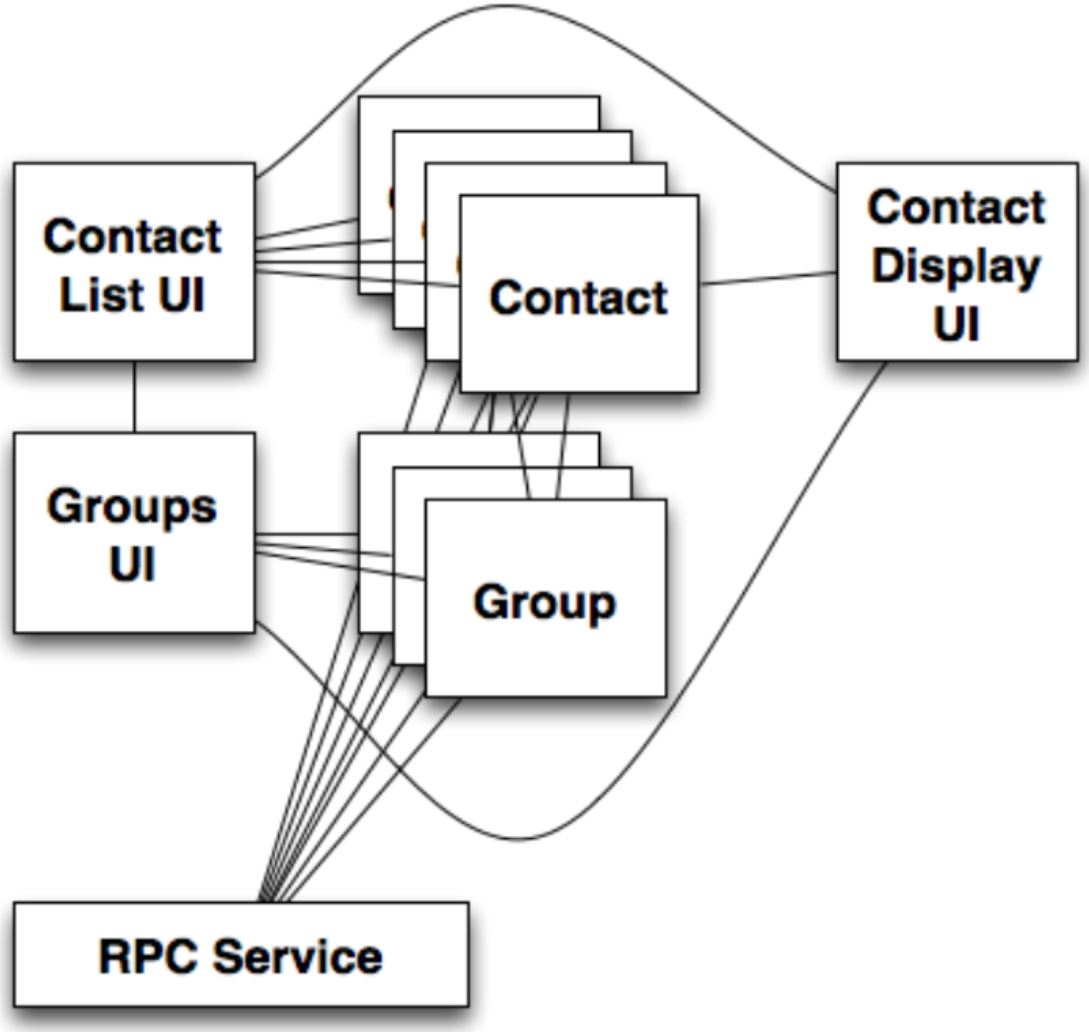
Coupling



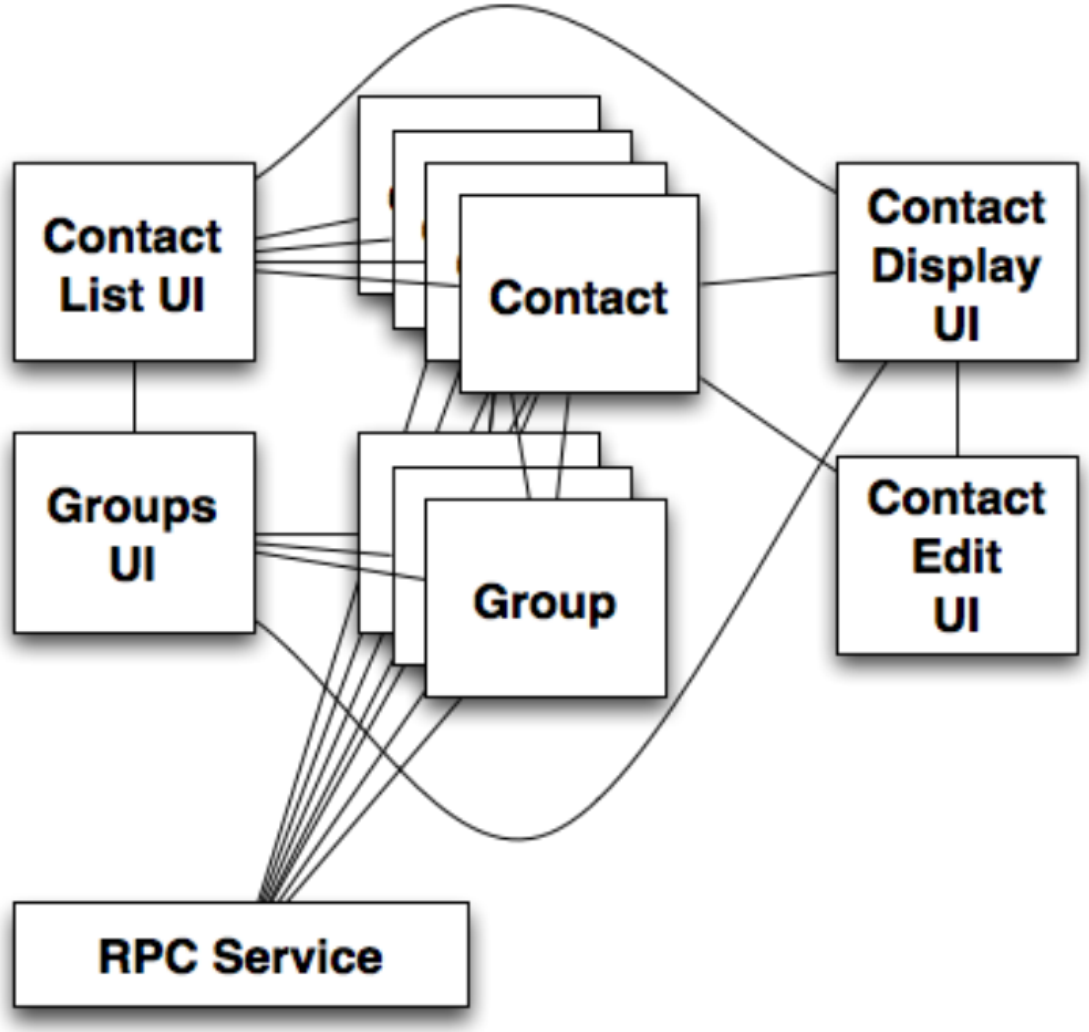
Coupling



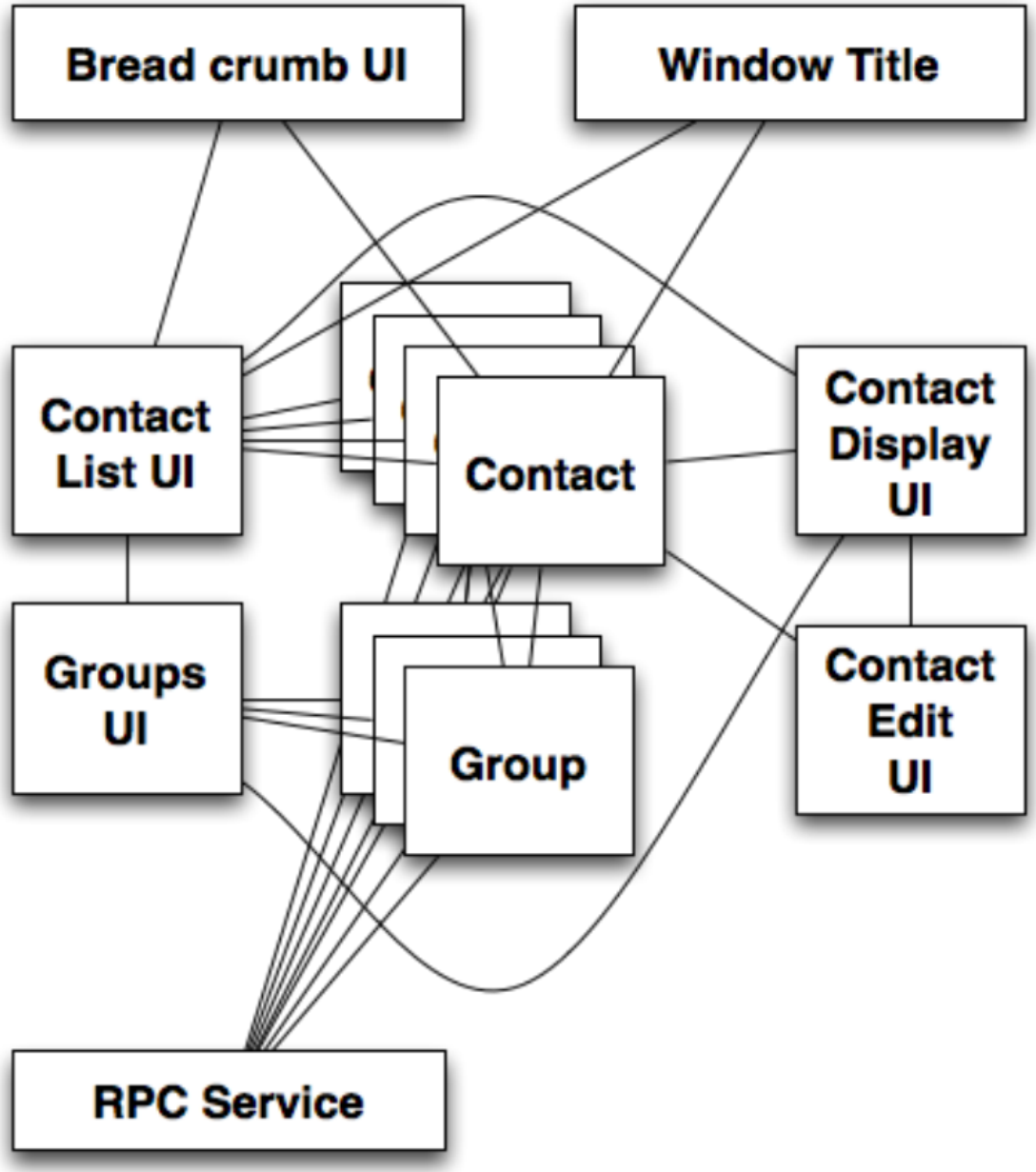
Coupling


Contact List UI
Contact
Contact Display UI
Groups UI
Group
RPC Service

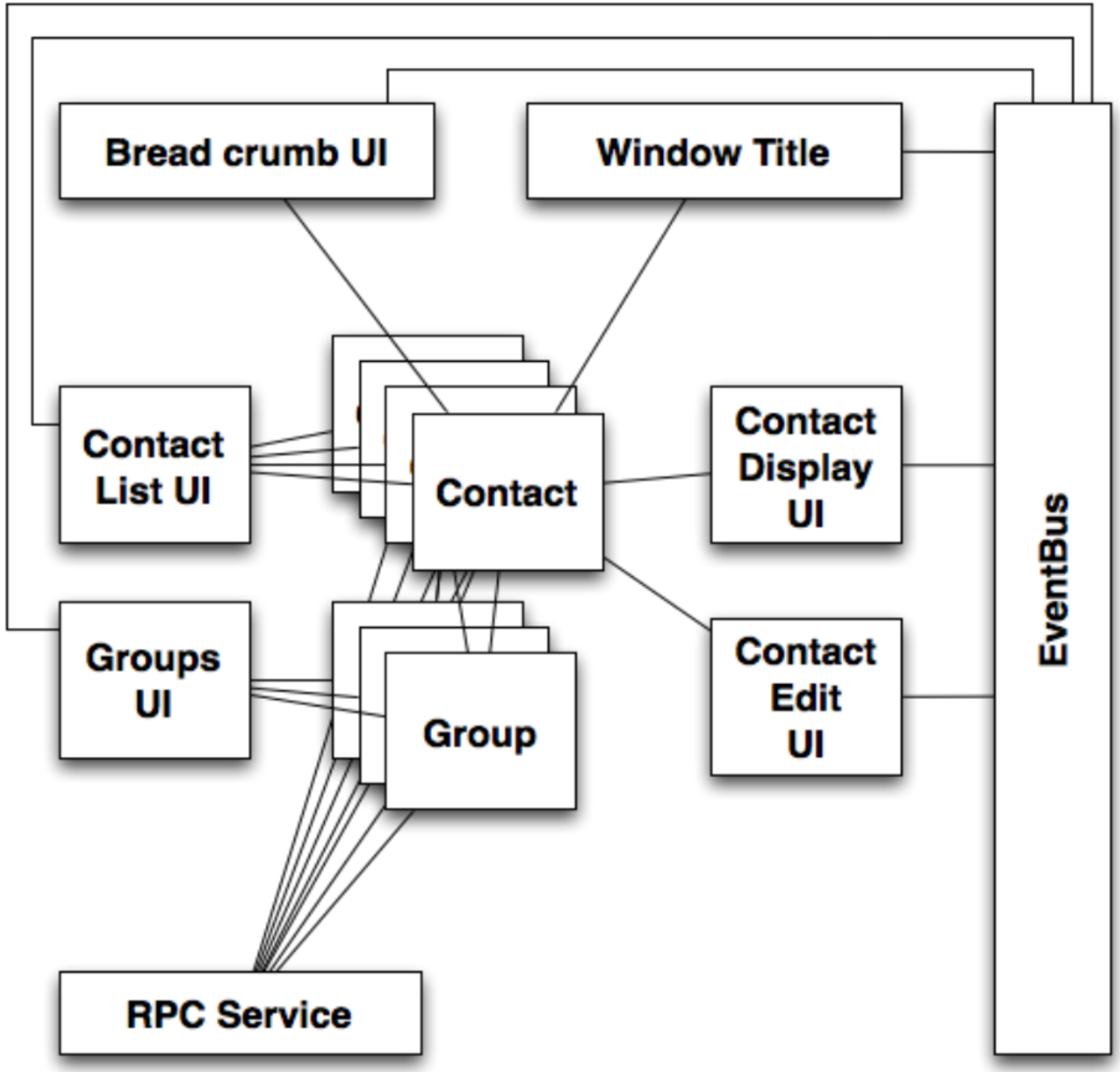
Coupling


Contact List UI
Contact
Contact Display UI
Groups UI
Group
Contact Edit UI
RPC Service

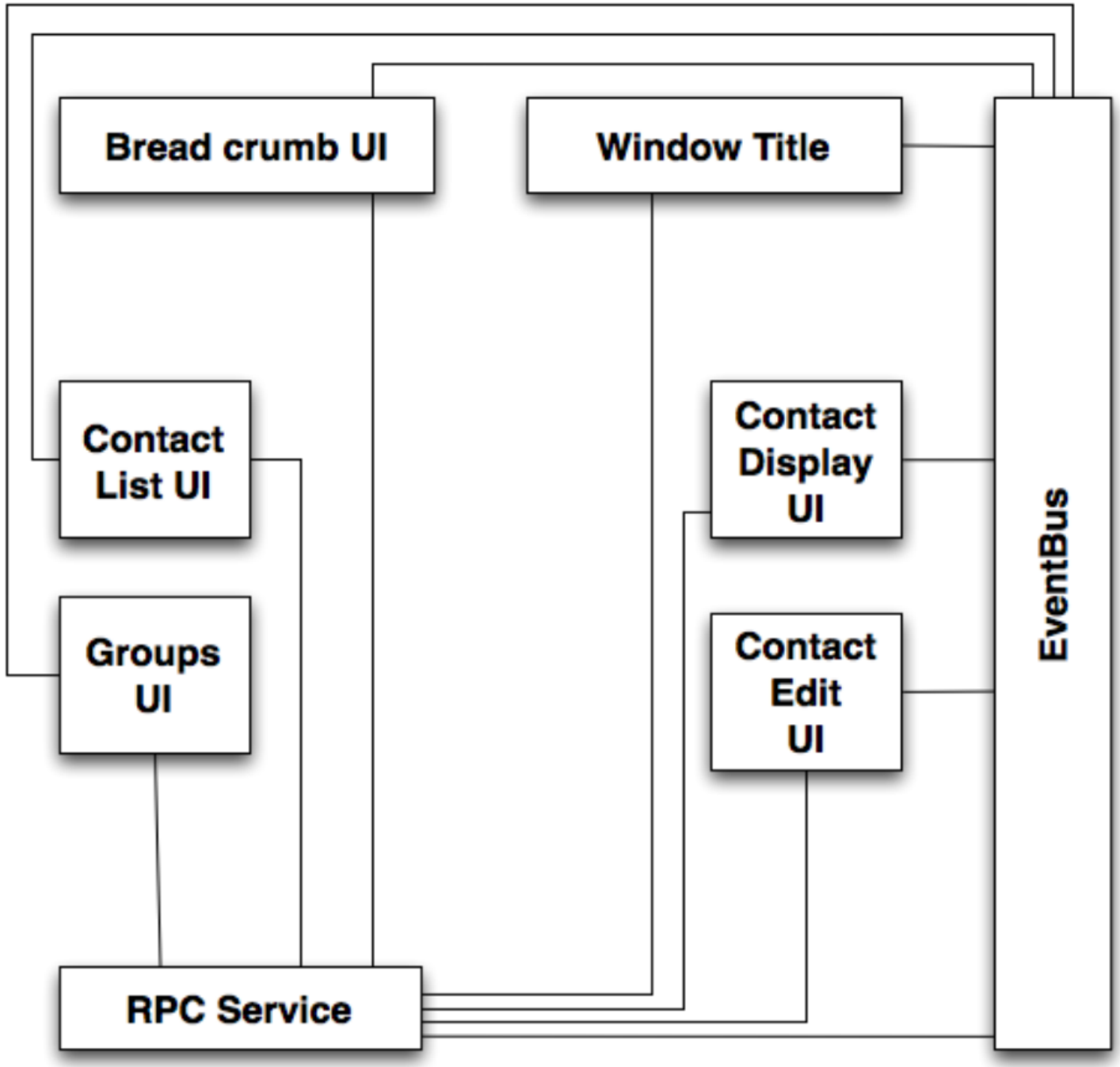
Coupled


Bread crumb UI
Window Title
Contact List UI
Contact
Contact Display UI
Groups UI
Group
Contact Edit UI
RPC Service

Looser


Bread crumb UI
Window Title
Contact List UI
Contact
Contact Display UI
Groups UI
Group
Contact Edit UI
EventBus
RPC Service

Loose


Bread crumb UI
Window Title
Contact List UI
Contact Display UI
Groups UI
Contact Edit UI
EventBus
RPC Service

EventBus

Implement with GWT HandlerManager


```
// Recall this method from an earlier slide

void showContact(final Contact contact) {
  service.execute(new GetDetails(contact.getDetailIds()),
    new GotDetails() {
      public void got(ArrayList<ContactDetail> details) {
        renderContact(contact);
        renderDetails(details);
      }
    }
  });
}
```

EventBus

Implement with GWT HandlerManager


```
ContactId currentContact;

void showContact(final Contact contact) {
  if (!currentContactId.equals(contact.getId())) {
    currentContactId = contact.getId();
    service.execute(new GetDetails(contact.getDetailIds()),
      new GotDetails() {
        public void got(ArrayList<ContactDetail> details) {
          renderContact(contact);
          renderDetails(details);
        }
      });
  }
}
```

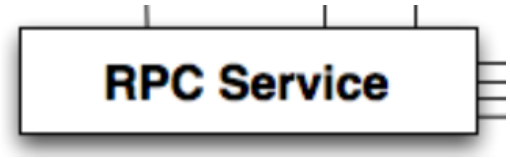
EventBus

Implement with GWT HandlerManager


```
HandlerManager eventBus;

void listenForContactUpdates() {
  eventBus.addHandler(ContactChangeEvent.TYPE,
    new ContactChangeEventHandler() {
      public void onContactChange(ContactChangeEvent event) {
        Contact contact = event.getContact();
        if (currentContactId.equals(contact.getId())) {
          renderContact(contact);
        }
      }
  });
}
```

EventBus

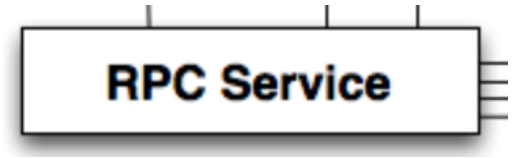
Tossing the event




```
public void execute(final UpdateContact update,
    final AsyncCallback<GetContactsResponse> cb) {

  realService.execute(update,
      new AsyncCallback<UpdateContactResponse>() {
        public void onFailure(Throwable caught) {
          cb.onFailure(caught);
        }

        public void onSuccess(UpdateContactResponse result) {
          recache(update.getContact());
    cb.onSuccess(result);
          ContactChangeEvent e =
              new ContactChangeEvent(update.getContact());
          eventBus.fireEvent(e);
        }
  });
}
```

EventBus

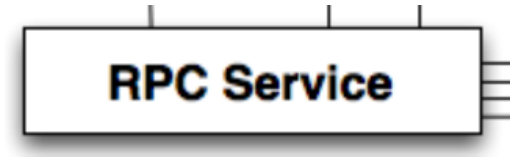
Tossing the event




```
public void execute(final UpdateContact update,
    final AsyncCallback<GetContactsResponse> cb) {

  realService.execute(update,
      new AsyncCallback<UpdateContactResponse>() {
        public void onFailure(Throwable caught) {
          cb.onFailure(caught);
        }

        public void onSuccess(UpdateContactResponse result) {
          recache(update.getContact());
    cb.onSuccess(result);
    ContactChangeEvent e =
                new ContactChangeEvent(update.getContact());
          eventBus.fireEvent(e);
        }
  });
}
```

EventBus

Tossing the event



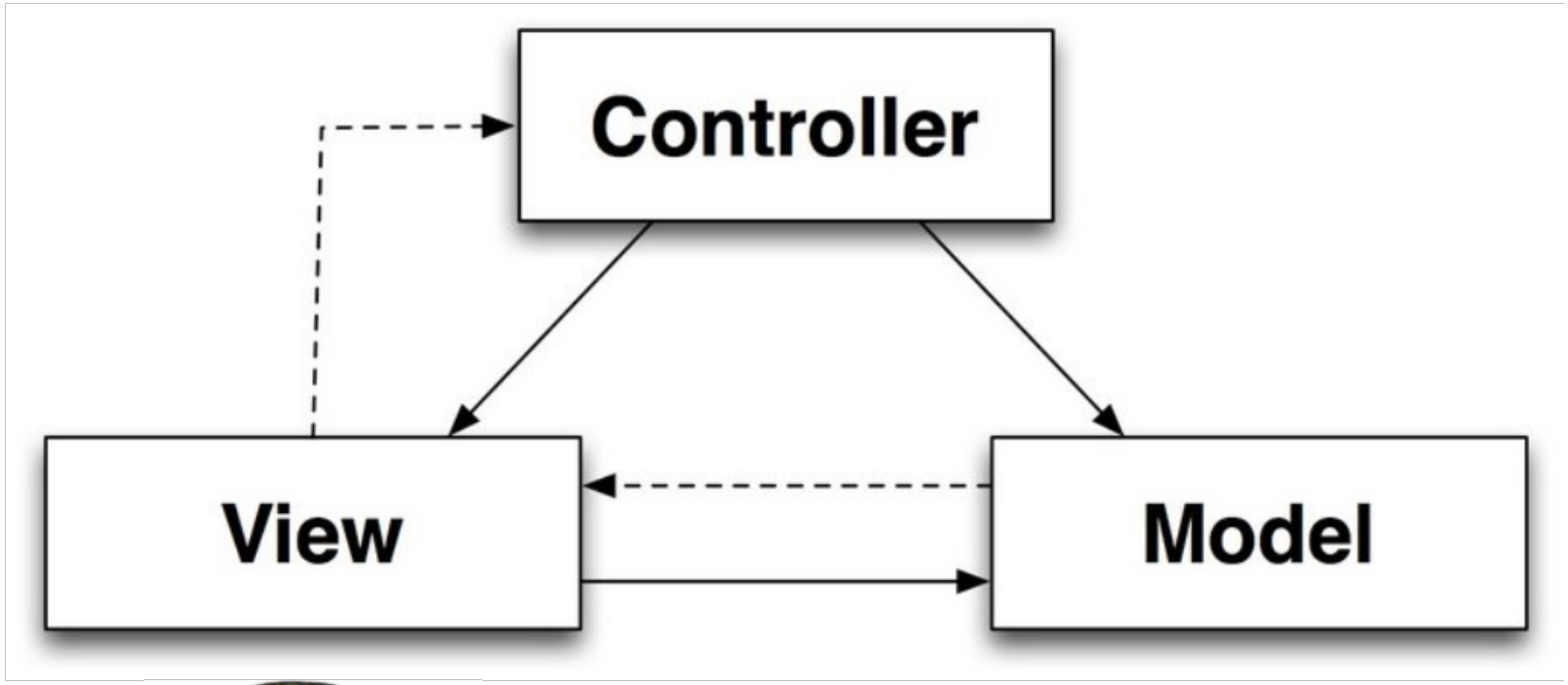

```
public void execute(final UpdateContact update,
    final AsyncCallback<GetContactsResponse> cb) {

  realService.execute(update,
      new AsyncCallback<UpdateContactResponse>() {
        public void onFailure(Throwable caught) {
          cb.onFailure(caught);
        }

        public void onSuccess(UpdateContactResponse result) {
          recache(update.getContact());
    cb.onSuccess(result);
          ContactChangeEvent e =
              new ContactChangeEvent(update.getContact());
          eventBus.fireEvent(e);
        }
  });
}
```

Always be decoupling: Decoupling via MVP

Google Developer Day 2009

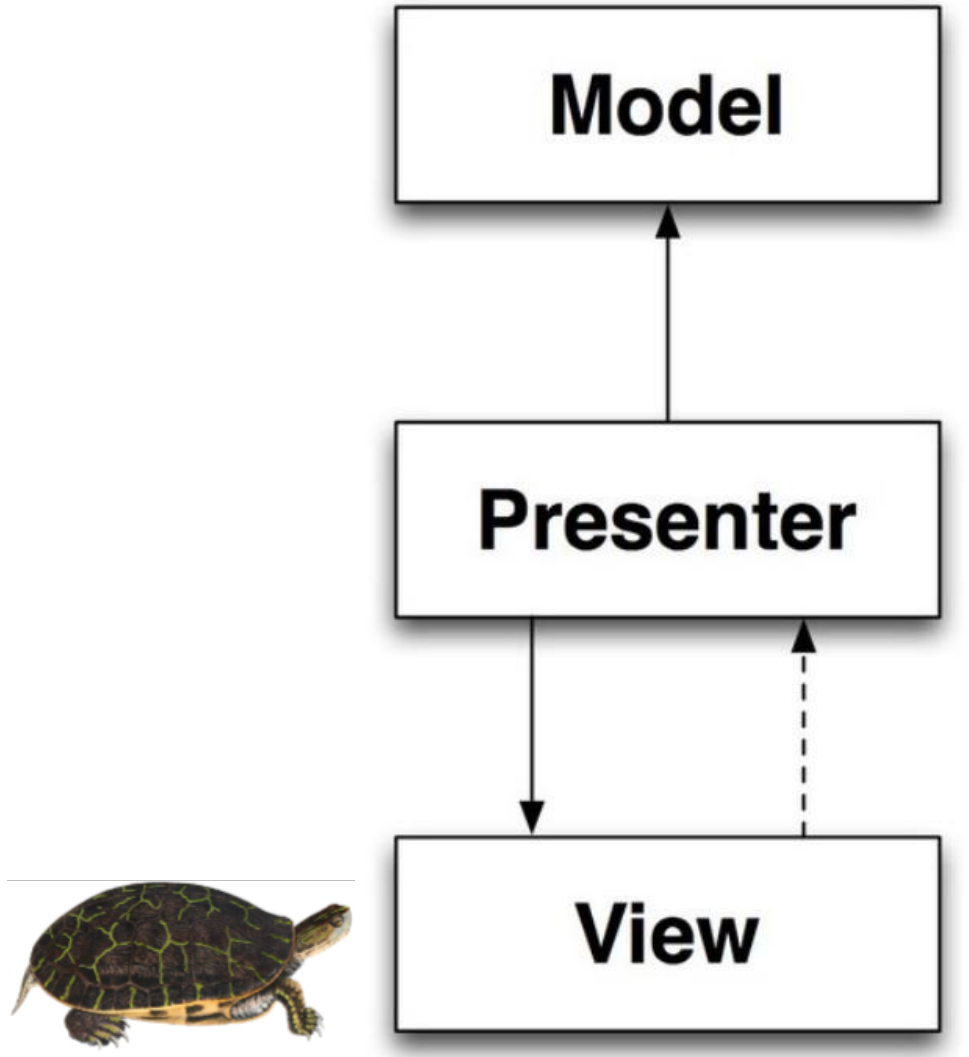
Classic Model View Controller pattern

How 1980s



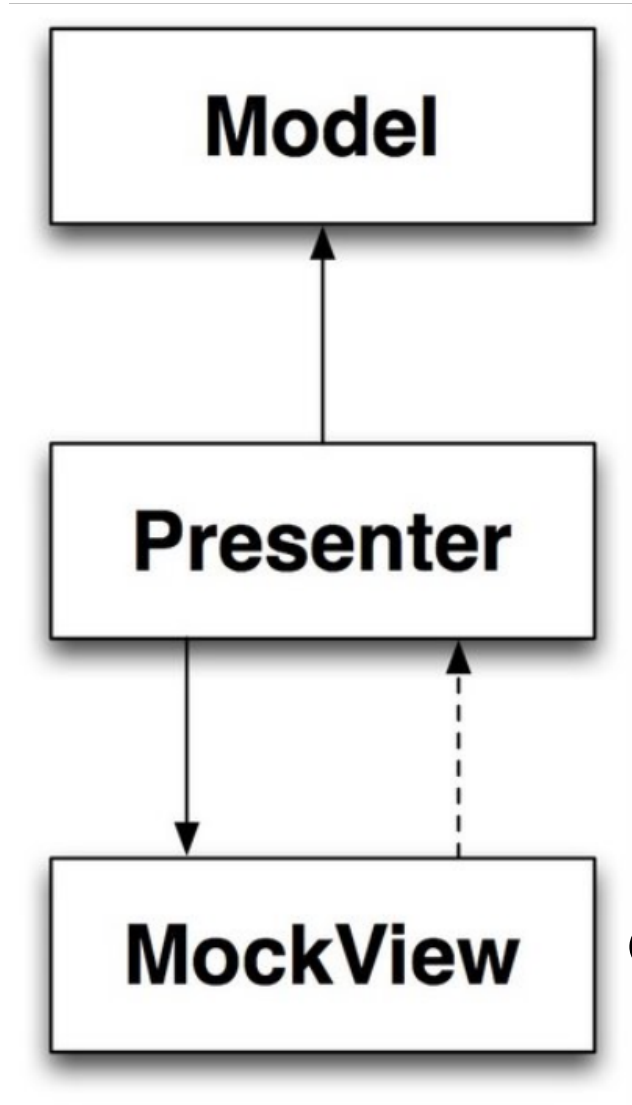
public domain image
http://vintageprintable.com/wordpress/wp-content/uploads/2009/04/deirochelys_reticulariaholbrookv1p07a.jpg

MVP



public domain image
http://vintageprintable.com/wordpress/wp-content/uploads/2009/04/deirochelys_reticulariaholbrookv1p07a.jp

MVP



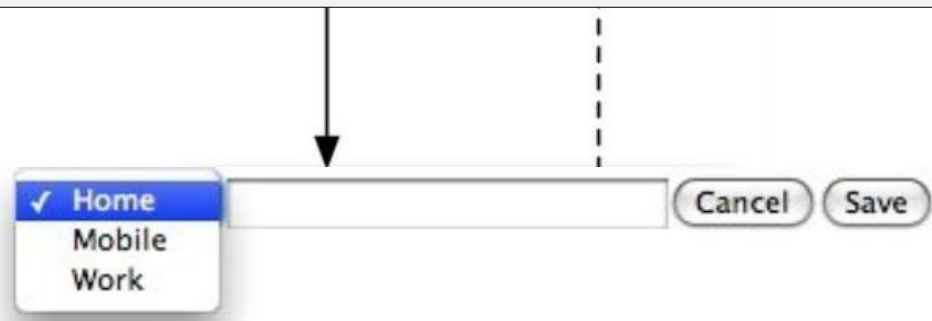
GWTMockUtilities

public domain image
http://www.freeclipartnow.com/animals/rabbits/Brown-Hare.jpg.html

MVP

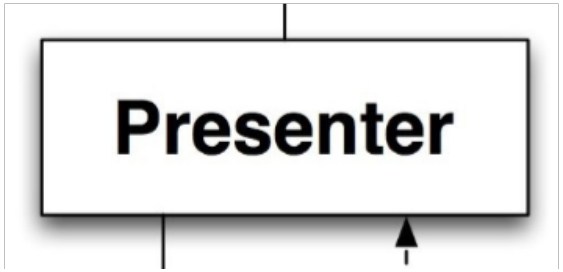
```
class Phone implements ContactDetail {
  final ContactDetailId id;
  String number;
  String label; // work, home...
}
```

```
class PhoneEditor {
  interface Display {
    HasClickHandlers getSaveButton();
    ...
```



Sample Presenter: PhoneEditor

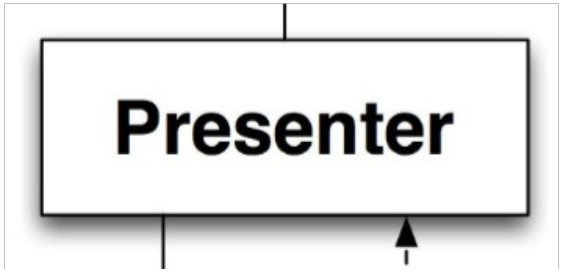
Display interface using characteristic interfaces




```
class PhoneEditor {
  interface Display {
    HasClickHandlers getSaveButton();
    HasClickHandlers getCancelButton();
    HasValue<String> getNumberField();
    HasValue<String> getLabelPicker();
    ...
}
```

Sample Presenter: PhoneEditor

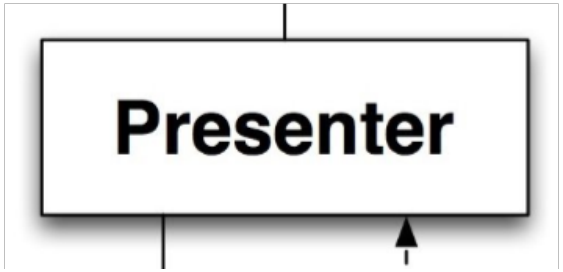
Binding the display




```
void bindDisplay(Display display) {
  this.display = display;

  display.getSaveButton().addClickHandler(new ClickHandler() {
    public void onClick(ClickEvent event) {
      doSave();
    }
  });

  display.getCancelButton().addClickHandler(new ClickHandler() {
    public void onClick(ClickEvent event) {
      doCancel();
    }
  });
}
```

Sample Presenter: PhoneEditor

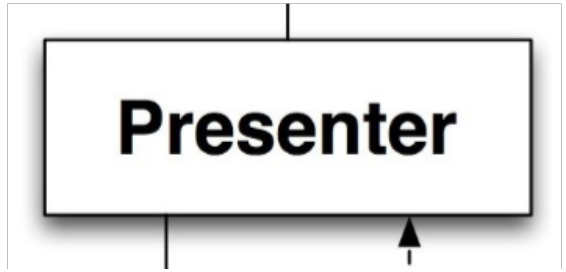
Start editing




```
void editPhone(Phone phone) {
  this.phone = Phone.from(phone);

  display.getNumberField().setValue(phone.getNumber());
  display.getLabelPicker().setValue(phone.getLabel());
}
```

Sample Presenter: PhoneEditor

Save it




```
void doSave() {
  phone.setNumber(display.getNumberField().getValue());
  phone.setLabel(display.getLabelPicker().getValue());

  service.execute(new UpdatePhone(phone), new UpdatedPhone() {
    public void updated() {
      tearDown();
    }

    public void hadErrors(HashSet<PhoneError> errors) {
      renderErrors(errors);
    }
  });
}
```

Always be decoupling: Decoupling via DI

Google Developer Day 2009

Dependency Injection

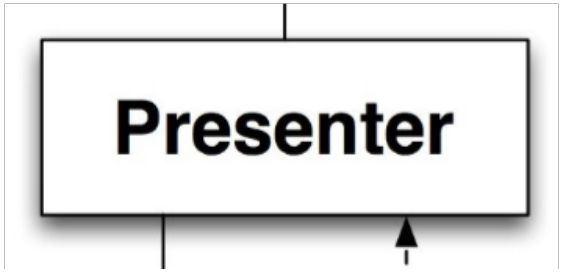
Just a pattern:

- No globals
- No service locator
- Dependencies pushed in, preferably via constructor

Not hard to do manually

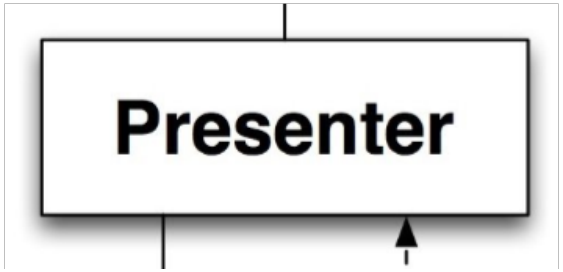
GIN (client) and Guice (server) can automate it
http://code.google.com/p/google-guice/
http://code.google.com/p/google-gin/

DI slice for the PhoneEditor



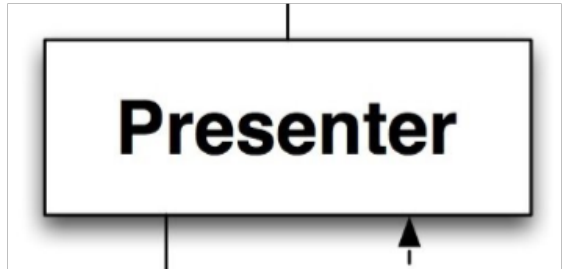

```
public void onModuleLoad() {
  ContactsServiceAsync realService =
      GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
      new CachedBatchingService(realService);
  HandlerManager eventBus = new HandlerManager(null);

  PhoneEditWidget phoneEditWidget = new PhoneEditWidget();
  PhoneEditor phoneEditor = new PhoneEditor(phoneEditWidget,
      rpcService);

  ContactWidget contactWidget = new ContactWidget();
  ContactViewer contactViewer =
      new ContactViewer(contactWidget, phoneEditor,
          rpcService, eventBus);
  contactViewer.go(RootPanel.get());
```

DI slice for the PhoneEditor

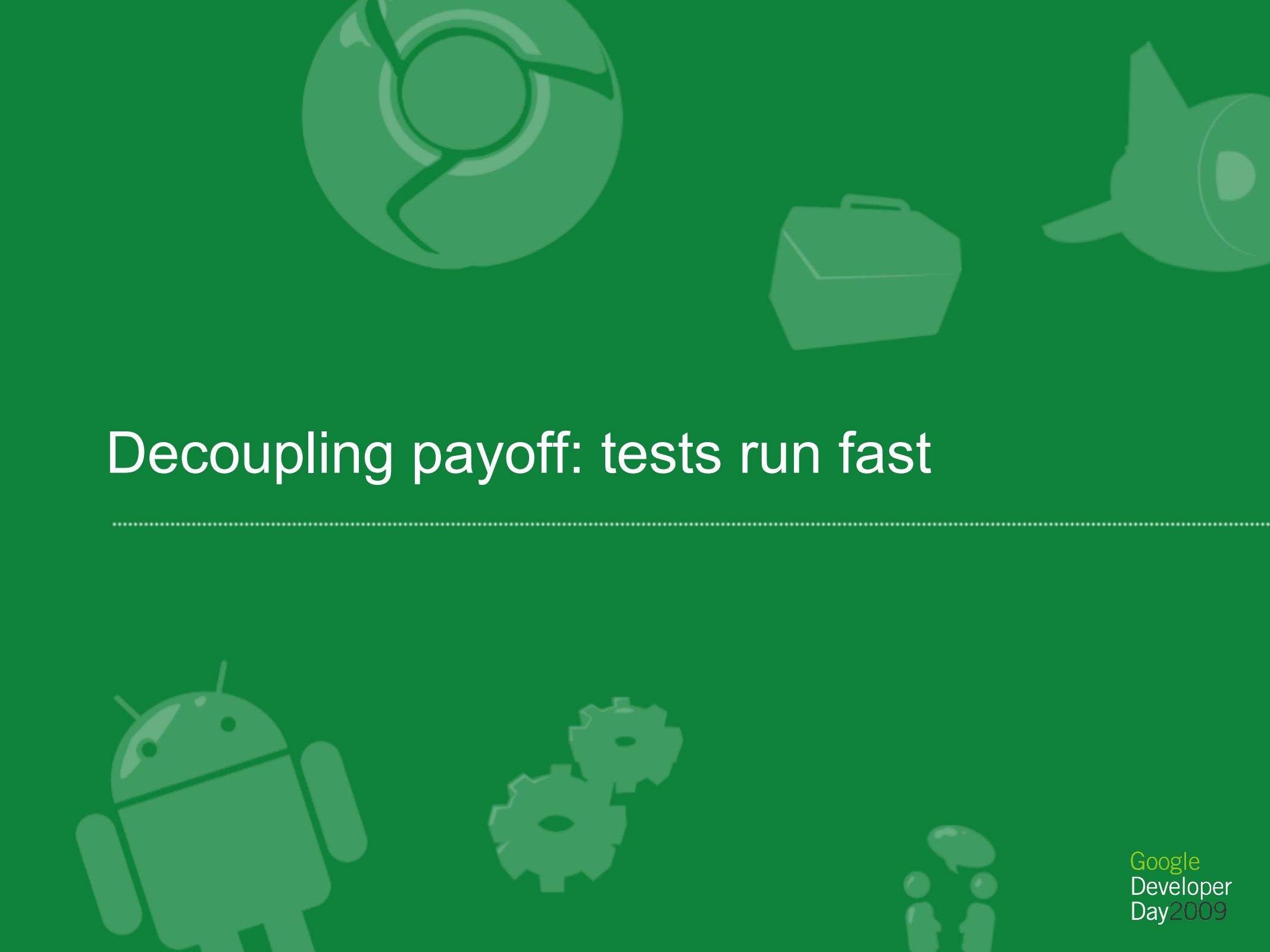



```
public void onModuleLoad() {
  ContactsServiceAsync realService =
      GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
      new CachedBatchingService(realService);
  GVoiceService voiceService = GWT.create(GVoiceService.class);
  HandlerManager eventBus = new HandlerManager(null);

  PhoneEditWidget phoneEditWidget = new PhoneEditWidget();
  PhoneEditor phoneEditor = new PhoneEditor(phoneEditWidget,
      rpcService, voiceService);

  ContactWidget contactWidget = new ContactWidget();
  ContactViewer contactViewer =
      new ContactViewer(contactWidget, phoneEditor,
          rpcService, eventBus);
  contactViewer.go(RootPanel.get());
```

DI slice for the PhoneEditor



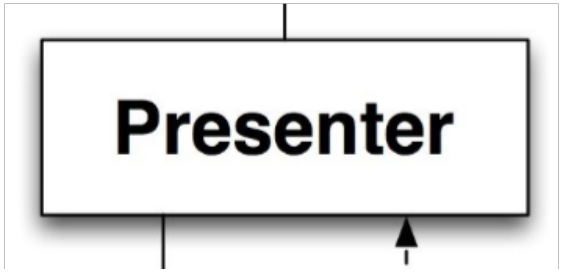

```
public void onModuleLoad() {
  MyGinjector factory = GWT.create(MyGinjector.class);

 factory.createContactViewer().go(RootPanel.get());

}
```



Decoupling payoff: tests run fast

Test the PhoneEditor

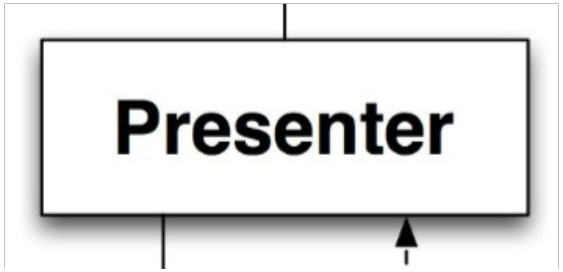
Define some mocks




```
class MockContactsService implements ContactsServiceAsync {
  Action<?> lastAction;
  AsyncCallback<?> lastCallback;

  public <T extends Response> void execute(Action<T> action,
      AsyncCallback<T> callback) {
    lastAction = action;
    lastCallback = callback;
  }
}
```

Test the PhoneEditor

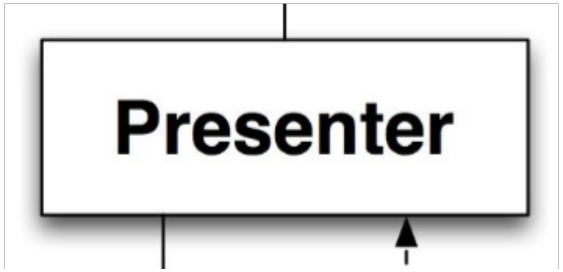
Define some mocks




```
class MockClickEvent extends ClickEvent { }

class MockHasClickHandlers implements HasClickHandlers {
  ClickHandler lastClickHandler;

  public HandlerRegistration addClickHandler(
      ClickHandler handler) {
    lastClickHandler = handler;

    return new HandlerRegistration() {
      public void removeHandler() { }
    };
  }

  public void fireEvent(GwtEvent<?> event) { }

}
```

Test the PhoneEditor

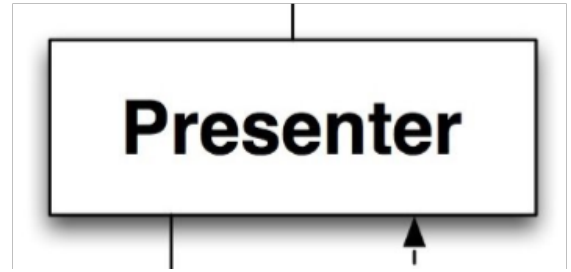
Define some mocks




```
class MockHasValue<T> implements HasValue<T> {
  T lastValue;

  public T getValue() {
    return lastValue;
  }

  public void setValue(T value) {
    this.lastValue = value;
  }

  public void setValue(T value, boolean fireEvents) {
    setValue(value);
  }

}
```

Test the PhoneEditor

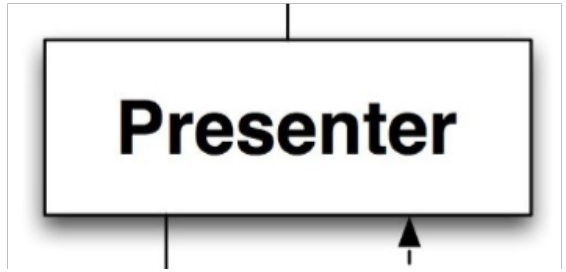
Define some mocks




```
class MockPhoneEditorDisplay implements PhoneEditor.Display {
  HasClickHandlers save = new MockHasClickHandlers();
  HasClickHandlers cancel = new MockHasClickHandlers();
  HasValue<String> labelPicker = new MockHasValue<String>();
  HasValue<String> numberField = new MockHasValue<String>();

  public HasClickHandlers getCancelButton() {
    return cancel;
  }
  public HasClickHandlers getSaveButton() { return save; }
  public HasValue<String> getLabelPicker() {
    return labelPicker;
  }
  public HasValue<String> getNumberField() {
    return numberField;
  }
}
```

Test the PhoneEditor

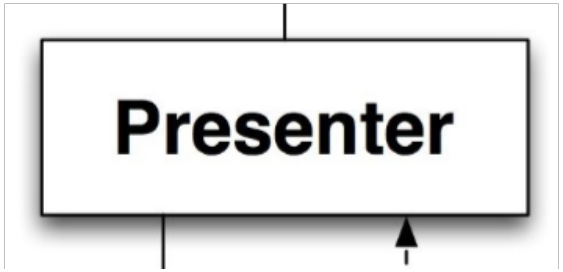
Set up the test...



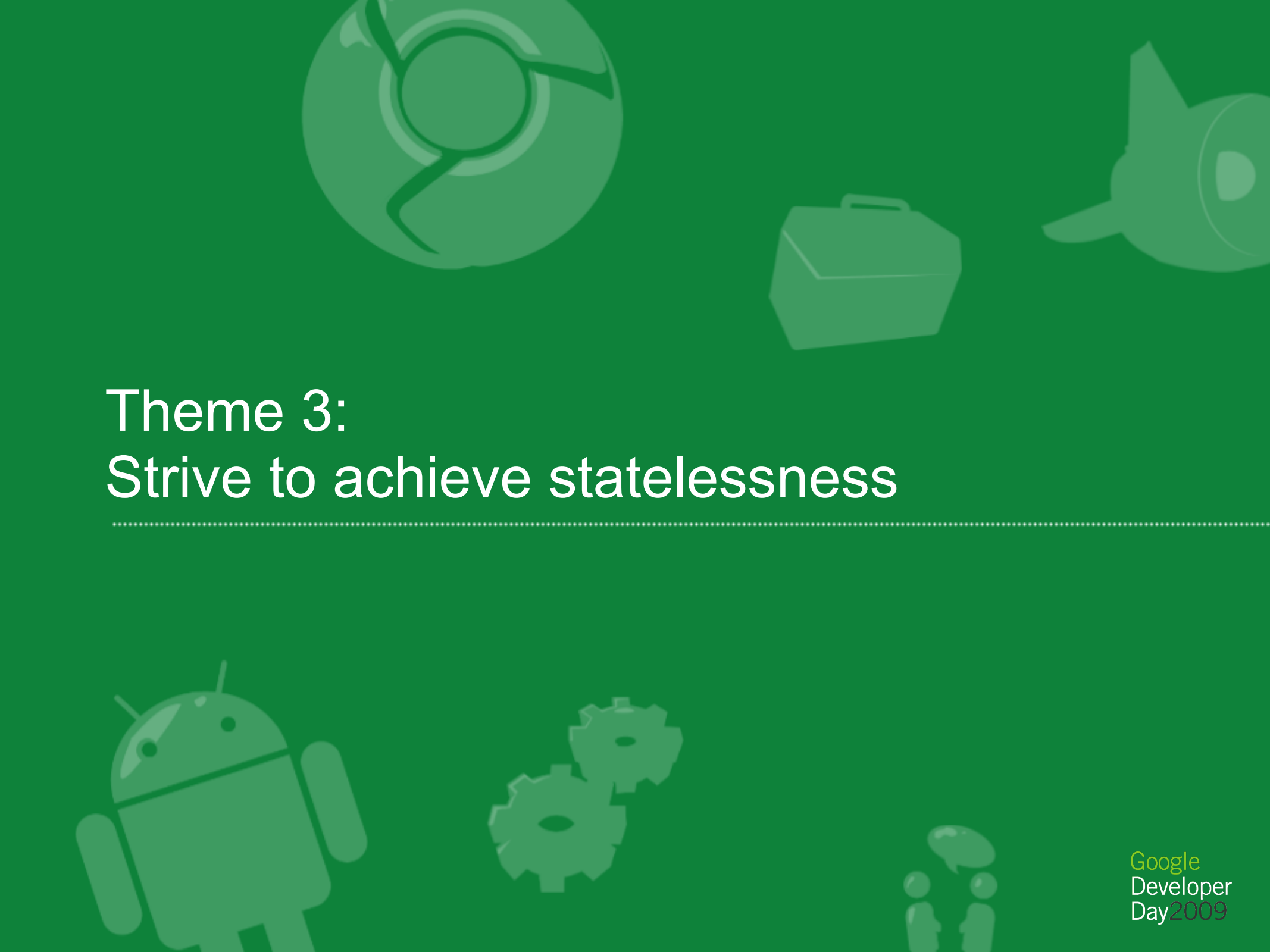

```
public void testSave() {
  MockContactsService service = new MockContactsService();
  MockPhoneEditorDisplay display =
      new MockPhoneEditorDisplay();
  ContactDetailId id = new ContactDetailId();

  // Build before and after values
  Phone before = new Phone(id);
  before.setLabel("Home");
  before.setNumber("123 456 7890");

  Phone expected = Phone.from(before);
  expected.setLabel("Work");
  expected.setNumber("098 765 4321");

 ...
```

Test the PhoneEditor

...and run it




```
…

  PhoneEditor editor = new PhoneEditor(display, service);
  editor.editPhone(before);
  display.labelPicker.setValue("Work");
  display.numberField.setValue("098 765 4321");
  display.save.lastClickHandler.onClick(new MockClickEvent());
  // Verify
  UpdatePhone action = (UpdatePhone) service.lastAction;
  assertEquals(new UpdatePhone(expected), action);

  Phone actual = action.getPhone();

  assertEquals(expected.getLabel(), actual.getLabel());
  assertEquals(expected.getNumber(), actual.getNumber());
}
```



Theme 3:
Strive to achieve statelessness

Disposable servers

- The browser embodies the session
- Server effectively stateless (except for caching)
- User should not notice server restart
- On App Engine, memcache works like this

"Values can expire
from the memcache at any time"

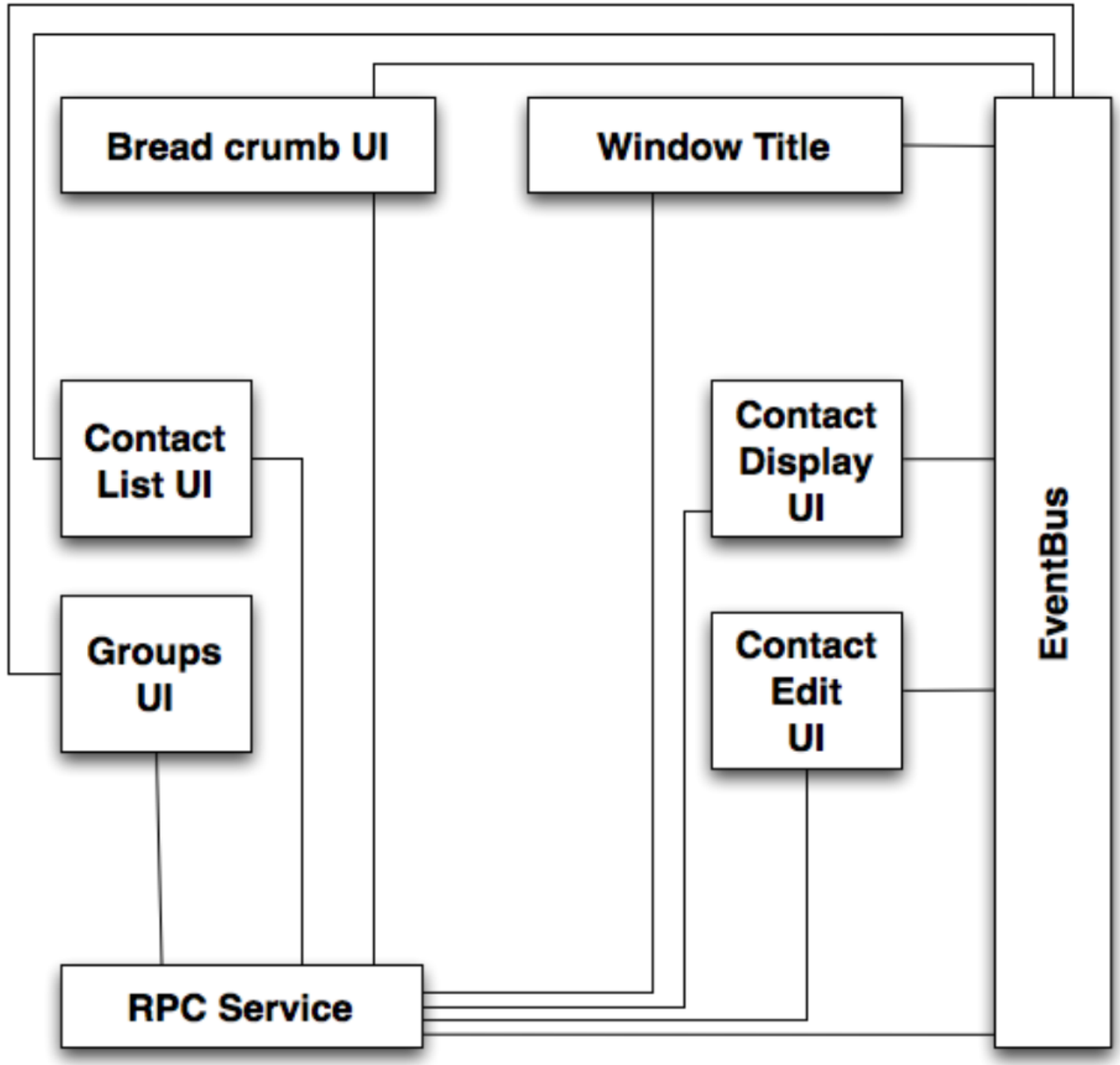
Disposable clients

- Use GWT History right, and get it right early
- Back button, refresh as a feature, not a catastrophe
- Bookmark your application's state
- Use Place abstraction
  - Layer above History
  - Announce place change via event bus

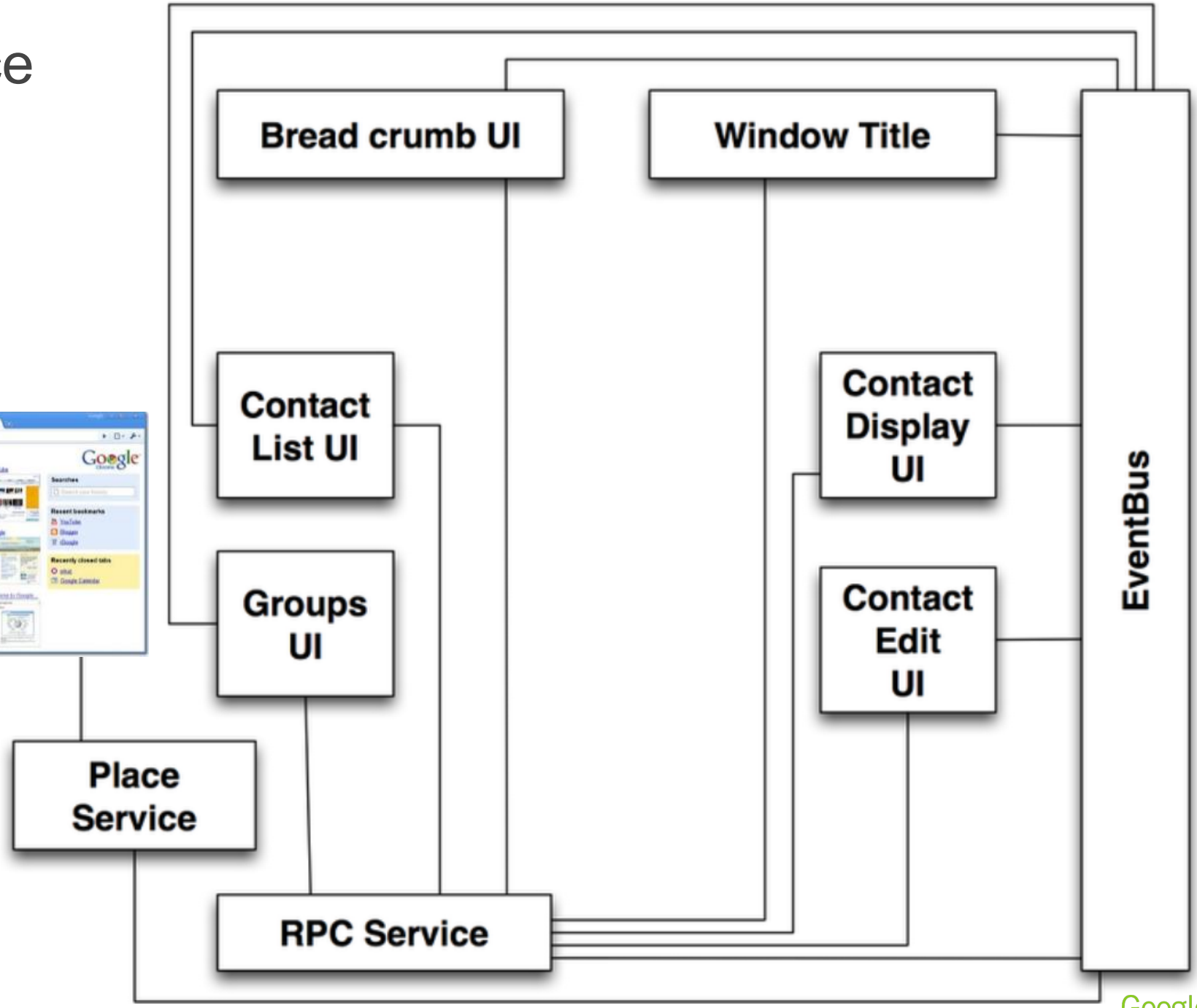
Disposable model objects

- Don't fire property change events
- Make defensive copies
- Okay to throw extra copies away
- Allow multiple instances
- Expect multiple instances

Recall


Bread crumb UI
Window Title
Contact List UI
Contact Display UI
Groups UI
Contact Edit UI
EventBus
RPC Service

+Place


Bread crumb UI
Window Title
Contact List UI
Contact Display UI
Groups UI
Contact Edit UI
EventBus
Place Service
RPC Service

Recap

If you remembered nothing else...

- Get browser **History** right, and get it right early
- Use an **Event Bus** to fight spaghetti
- **DI** + **MVP** FTW

Three major themes

1. Embrace asynchrony
2. Always be decoupling
3. Strive to achieve statelessness

Thank You

Read more

http://code.google.com/webtoolkit/

Contact info

Fred Sauer (twitter: @fredsa)Developer Advocate
fredsa@google.com

Questions

?


Google
Developer
Day2009