Google™ 09 I/O

GWT App Architecture Best Practices

Ray Ryan
28 May 2009

What are we talking about?

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

If you remember nothing else...

If you remember nothing else...

- Get browser history right, and get it right early

If you remember nothing else...

- Get browser history right, and get it right early

If you remember nothing else...

- Get browser history right, and get it right early
- Use an Event Bus to fight spaghetti

If you remember nothing else...

- Get browser history right, and get it right early
- Use an Event Bus to fight spaghetti

If you remember nothing else...

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

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 Injection plus
  Model / View / Presenter
  for the win!

Demo new AdWords UI

Three major themes

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

Embrace Asynchrony

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

- Everything might require an async call sometimes
- So assume it does all the time

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

- Everything might require an async call sometimes
- So assume it does all the time

```
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

- Everything might require an async call sometimes
- So assume it does all the time

```
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

- Everything might require an async call sometimes
- So assume it does all the time

```
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
  - GWT.runAsync()
  - Undo / Redo
  - Gears / HTML5 DB

Use Command pattern RPC

```
/** The name Command is taken */
interface Action<T extends Response> { }

interface Response { }

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> details) {
    this.details = details;
  }

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

Command style RPC example

…plus convenience callback

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

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

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

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

Command style RPC example

Make it go

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

Always be decoupling

Always be decoupling

With the combination of

- An event bus
- MVP pattern for your custom widgets
- Dependency injection of app-wide services

Always be decoupling

With the combination of

- An event bus
- MVP pattern for your custom widgets
- Dependency injection of app-wide services

You get...

Easy rejiggering of the app

Always be decoupling

With the combination of

- An event bus
- MVP pattern for your custom widgets
- Dependency injection of app-wide services

You get...

Easy rejiggering of the app

Easy to defer pokey DOM operations

Always be decoupling

With the combination of

- An event bus
- MVP pattern for your custom widgets
- Dependency injection of app-wide services

You get...

Easy rejiggering of the app

Easy to defer pokey DOM operations

Easy unit testing

Always be decoupling

With the combination of

- An event bus
- MVP pattern for your custom widgets
- Dependency injection of app-wide services

You get...

Easy rejiggering of the app

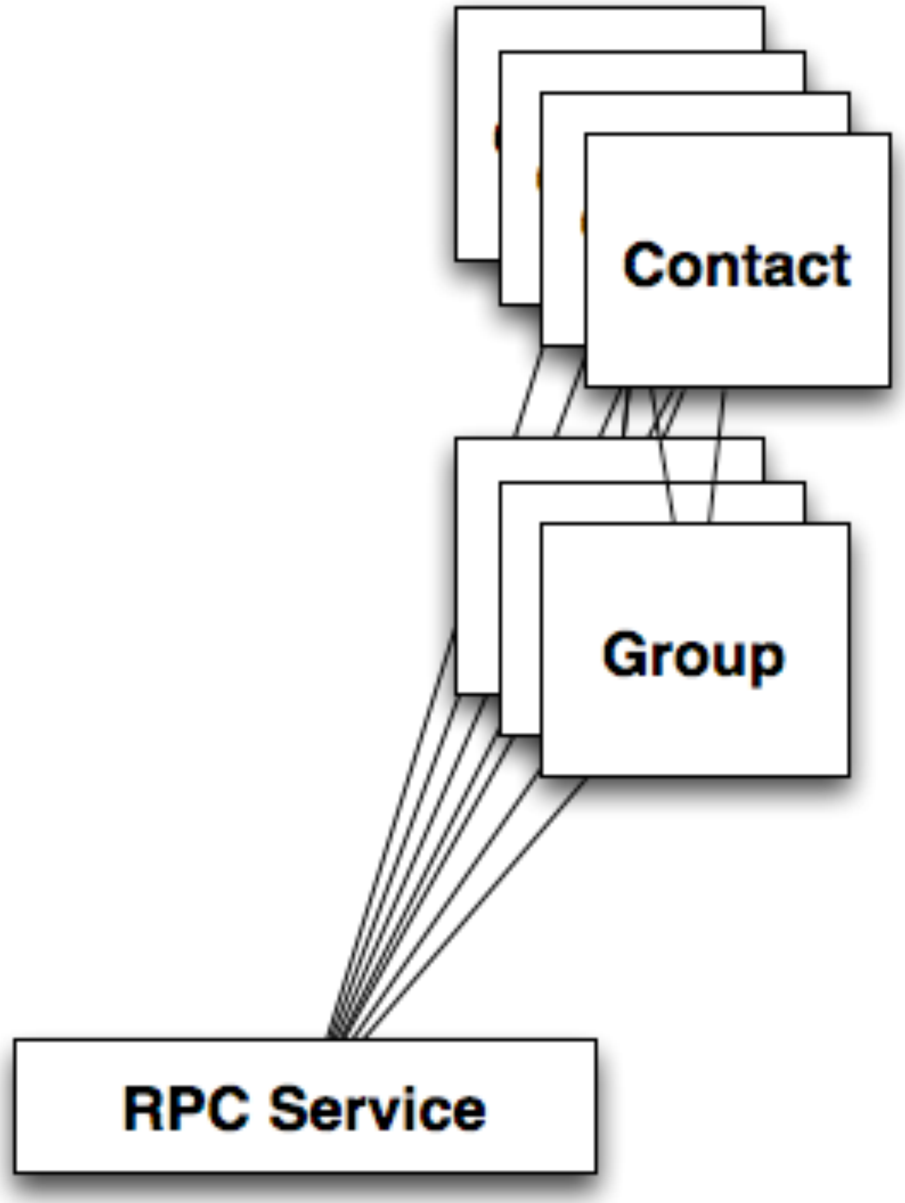
Easy to defer pokey DOM operations

Easy unit testing

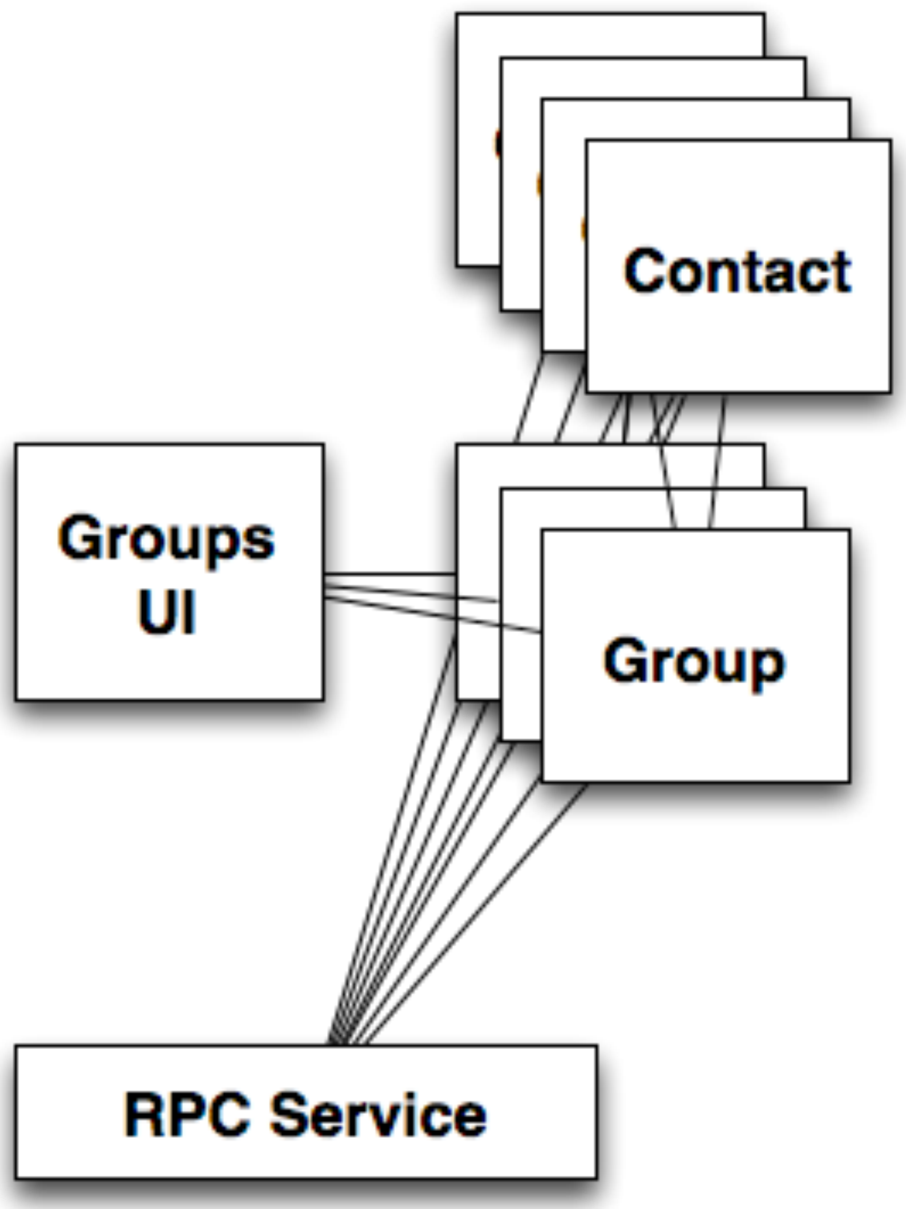
Fast test execution

Decoupling via event bus

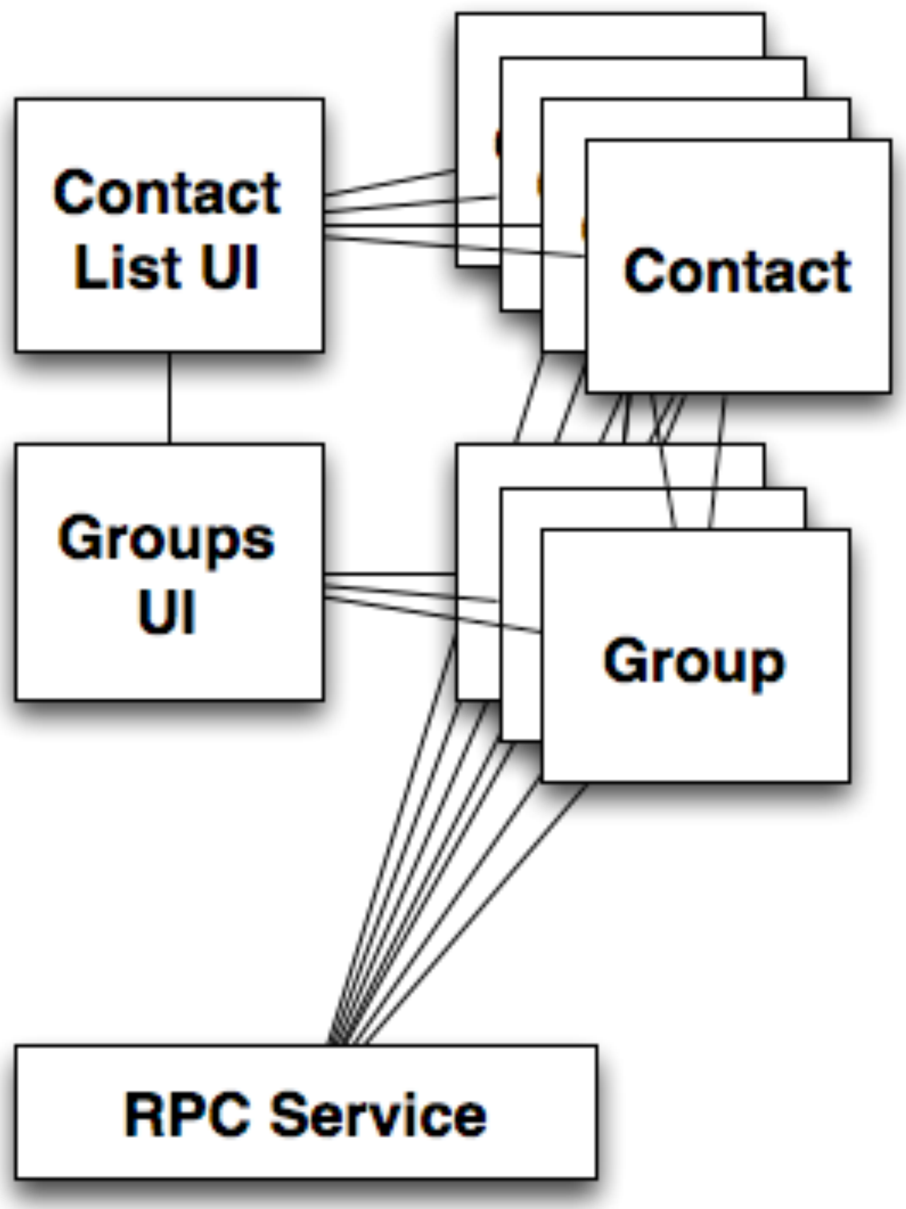
Coupling



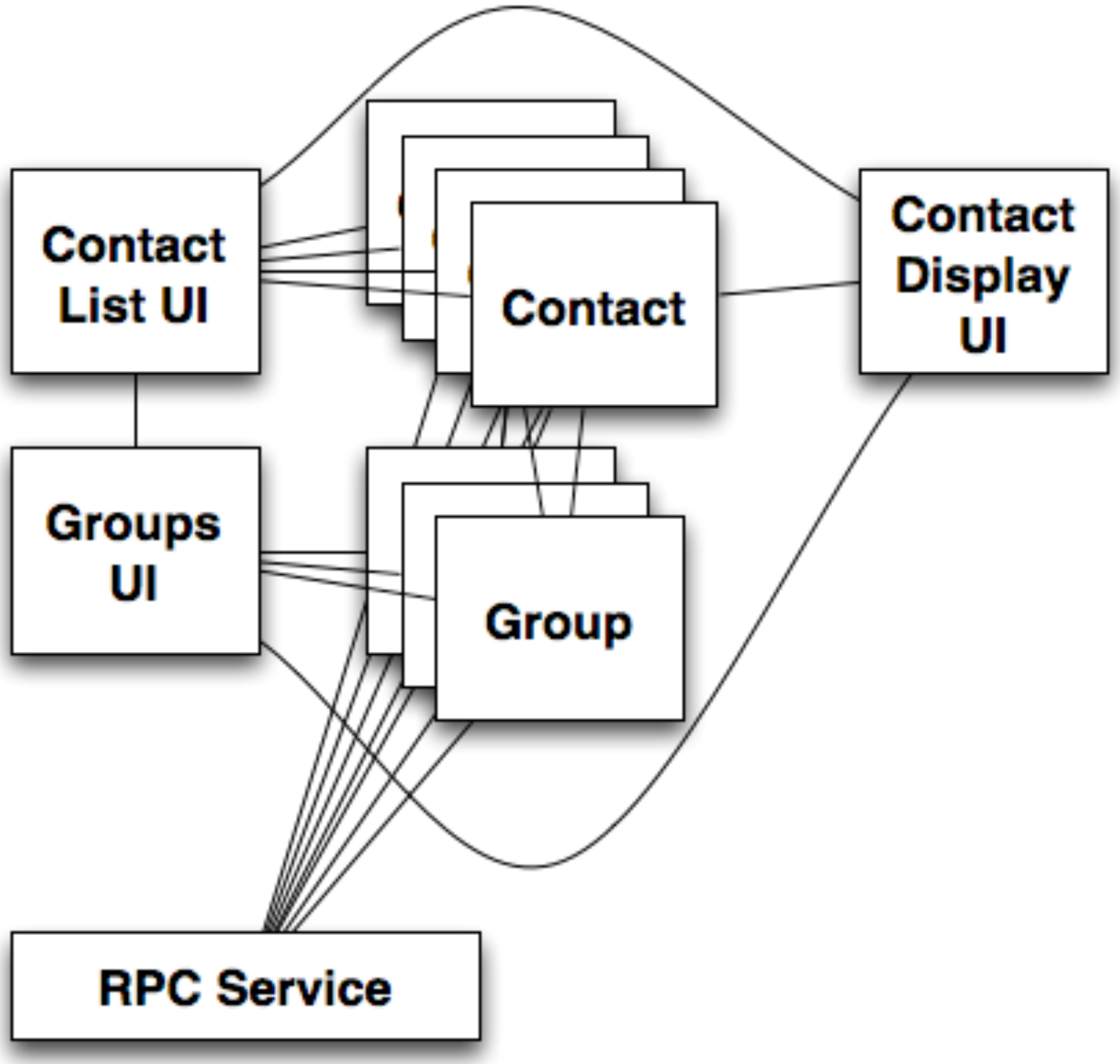
Coupling



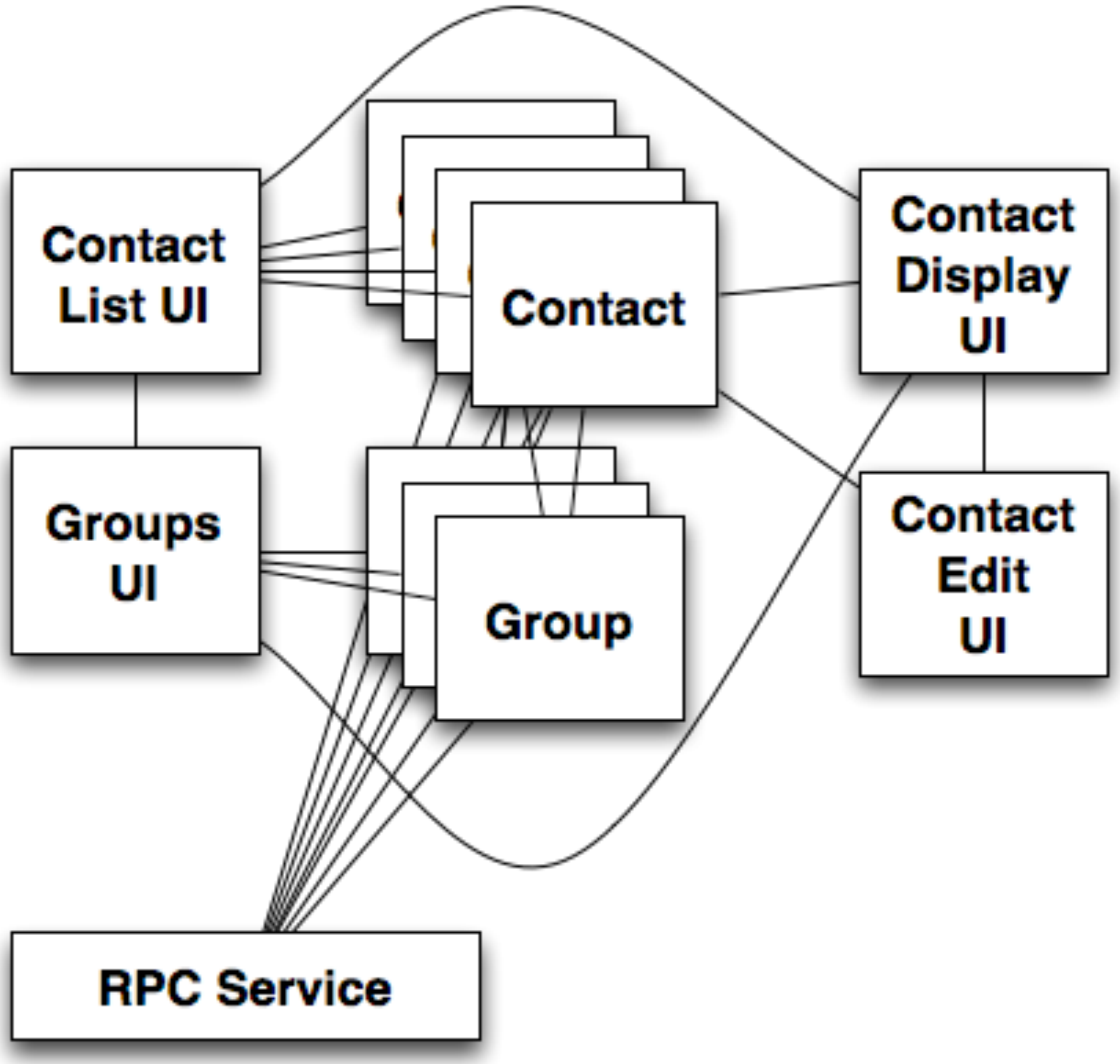
Coupling



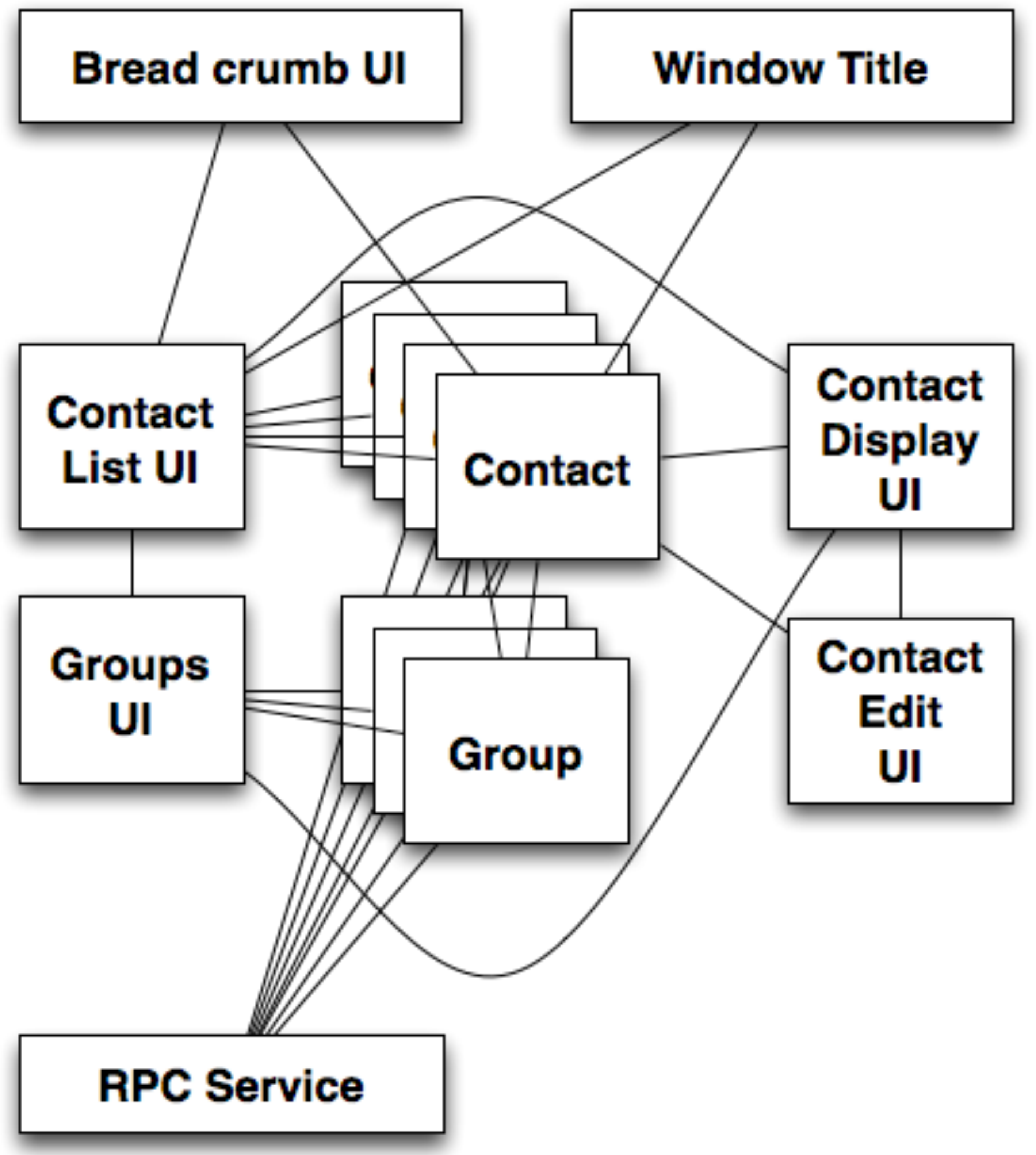
Coupling


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

Coupling

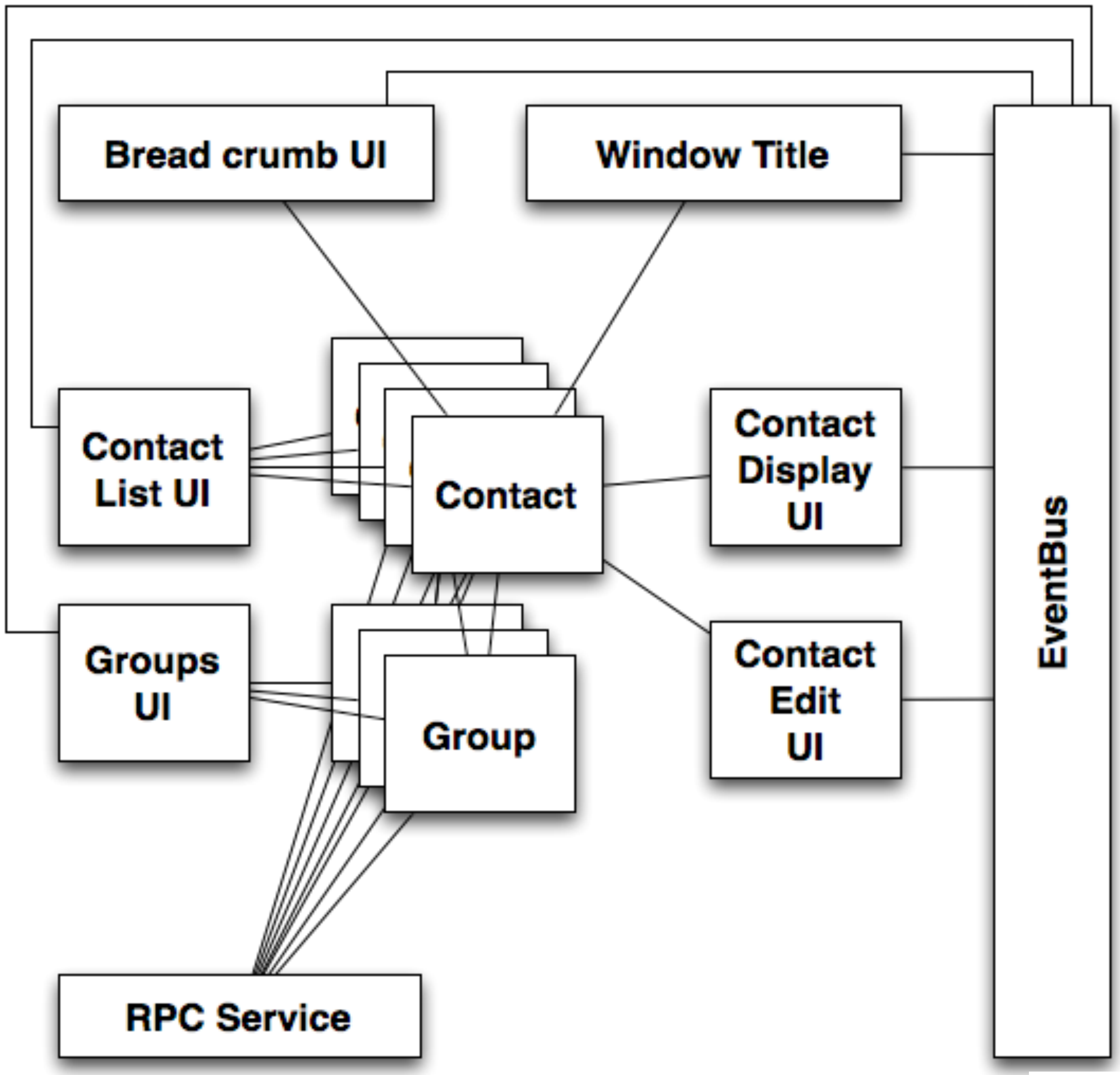

Contact List UI
Contact
Contact Display UI
Groups UI
Group
Contact Edit UI
RPC Service

Coupled

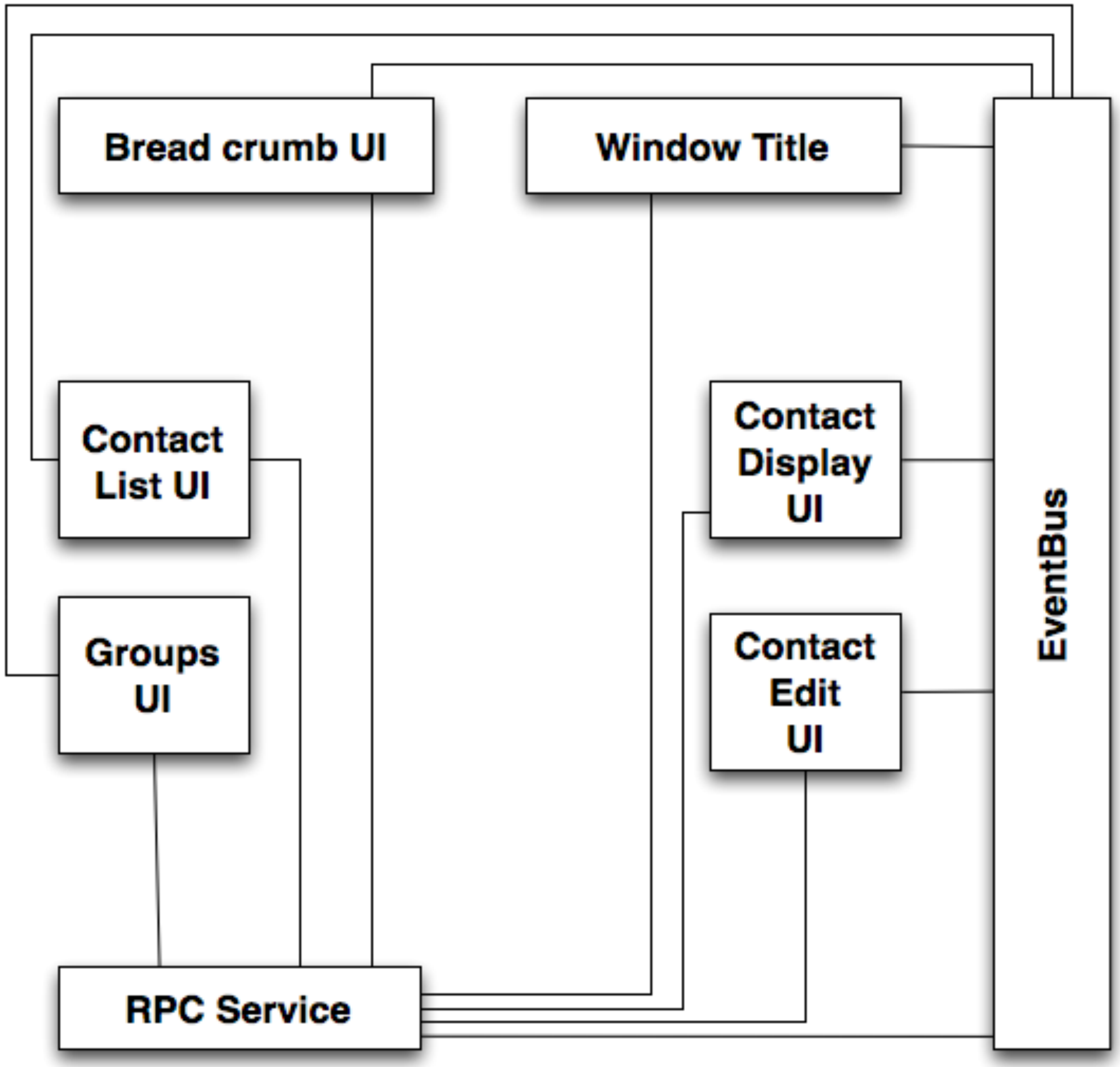


Bread crumb UI
Window Title
Contact List UI
Contact
Contact Display UI
Groups UI
Group
Contact Edit UI
RPC Service

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

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

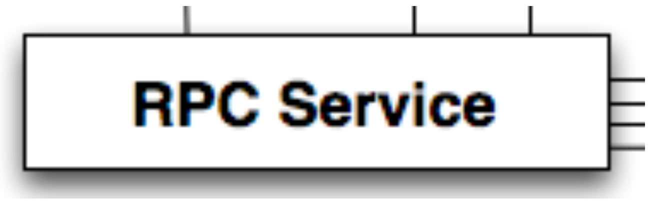
EventBus

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

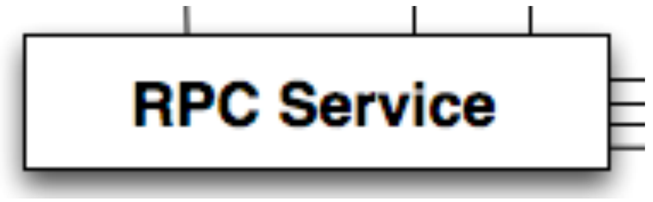
Tossing the event



```java
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

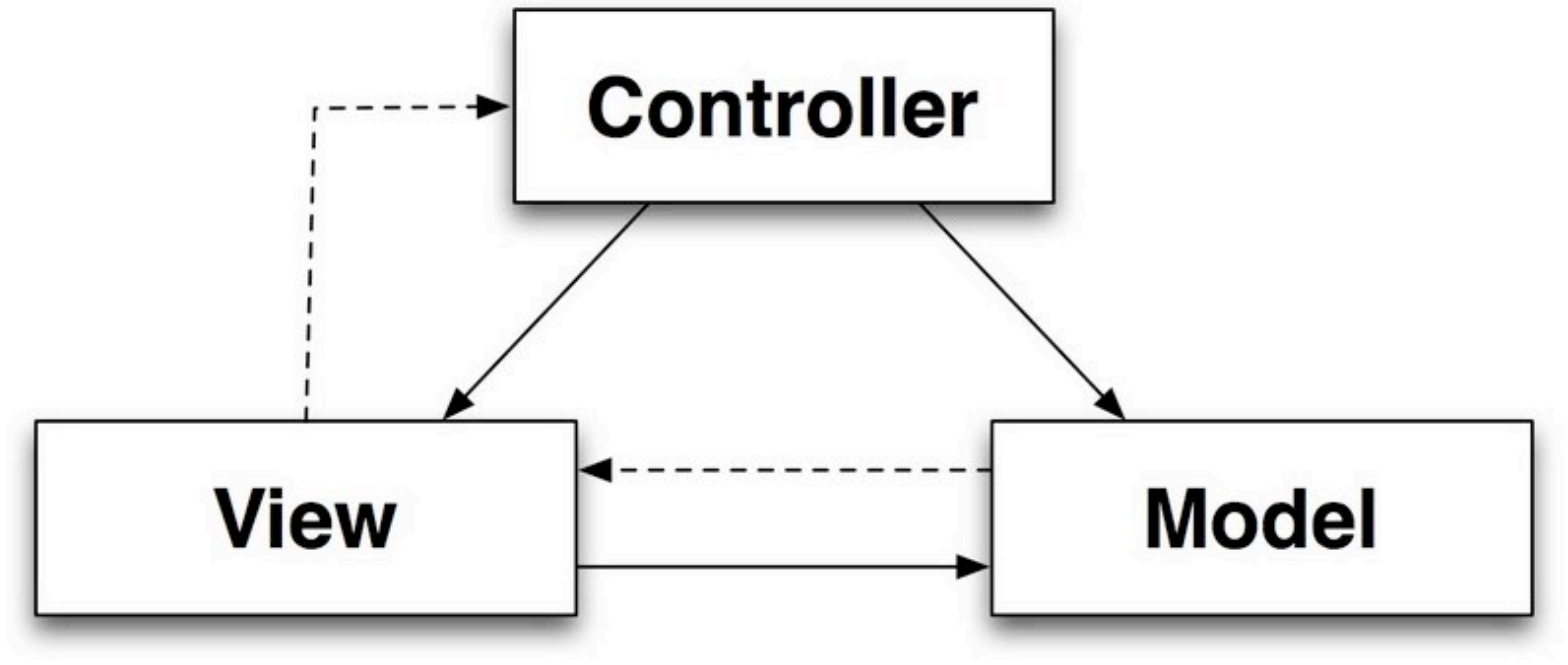
Tossing the event



```
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);
      }
    });
}
```

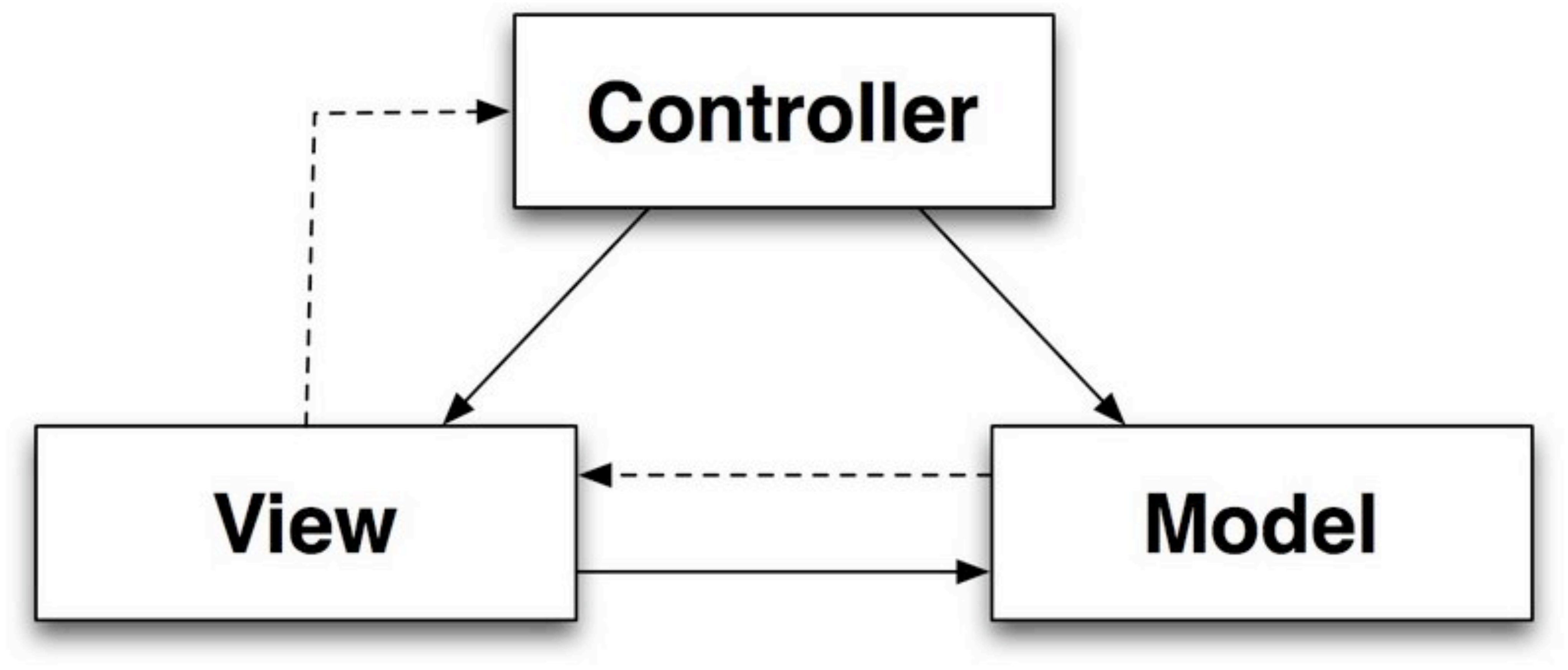
Decoupling via MVP

Classic Model View Controller pattern



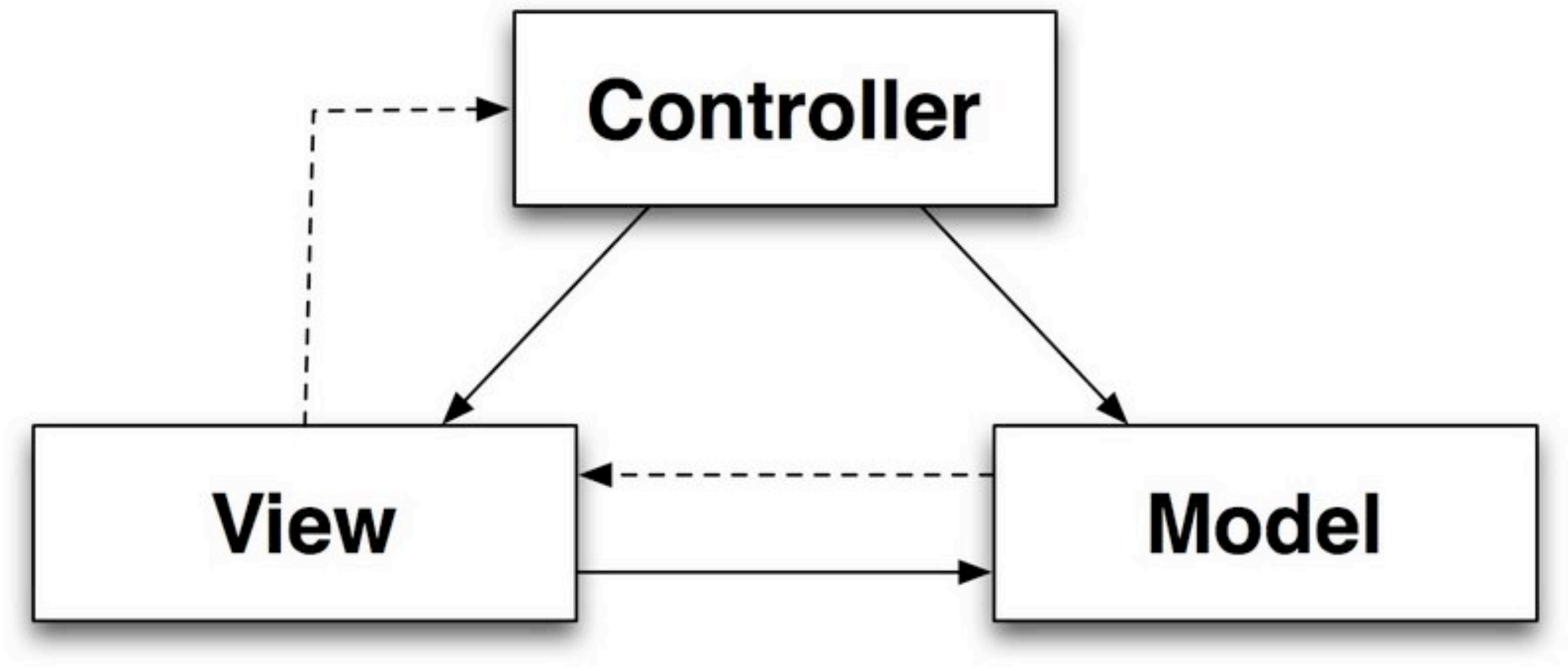
Classic Model View Controller pattern

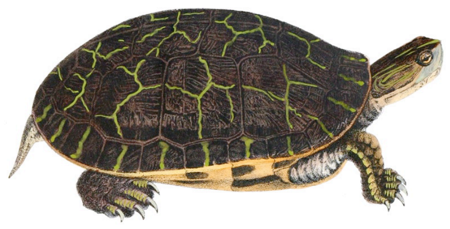
How 1980s



Classic Model View Controller pattern

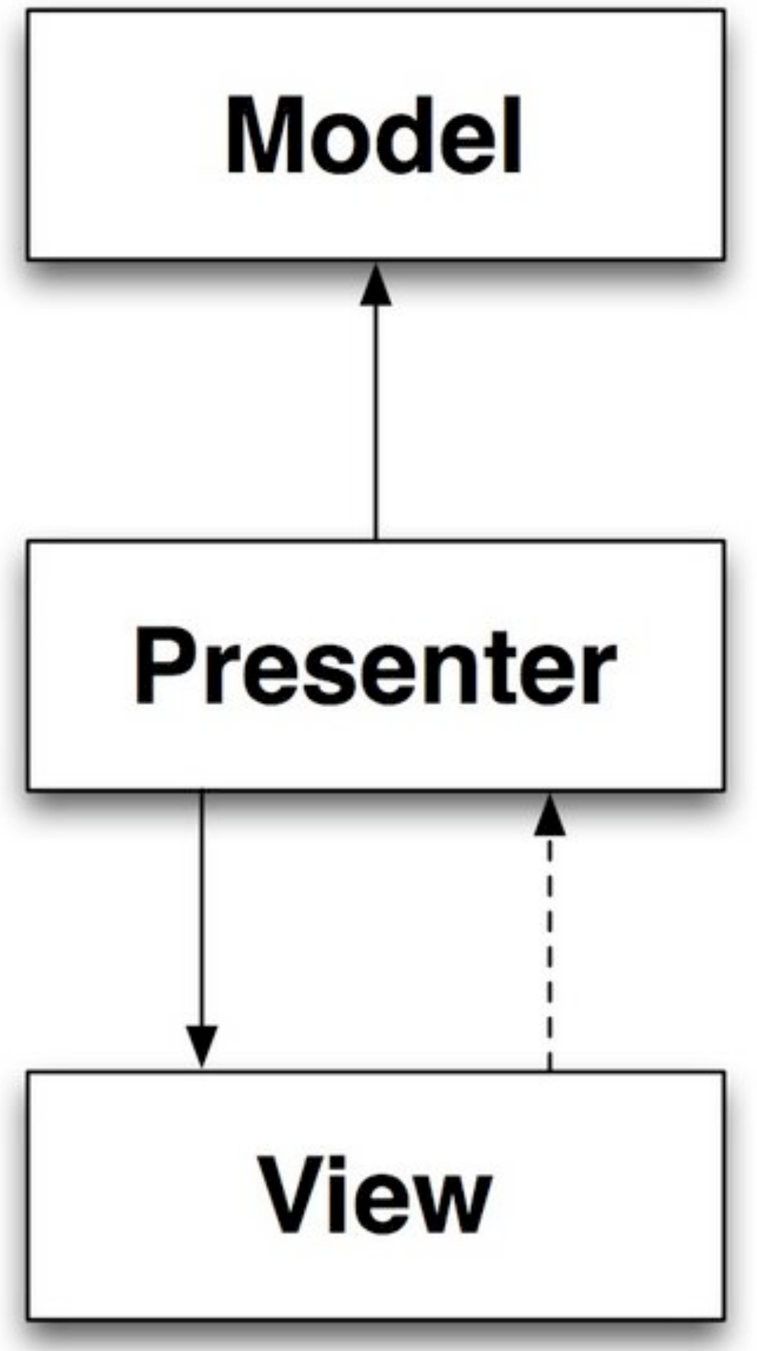
How 1980s

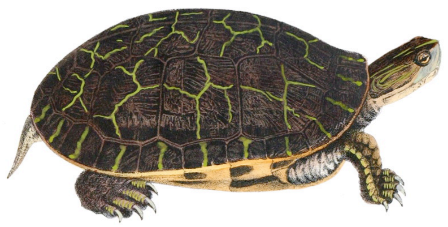




public domain image http://vintageprintable.com/wordpress/wp-content/uploads/2009/04/deirochelys_reticulariaholbrookv1p07a.jpg

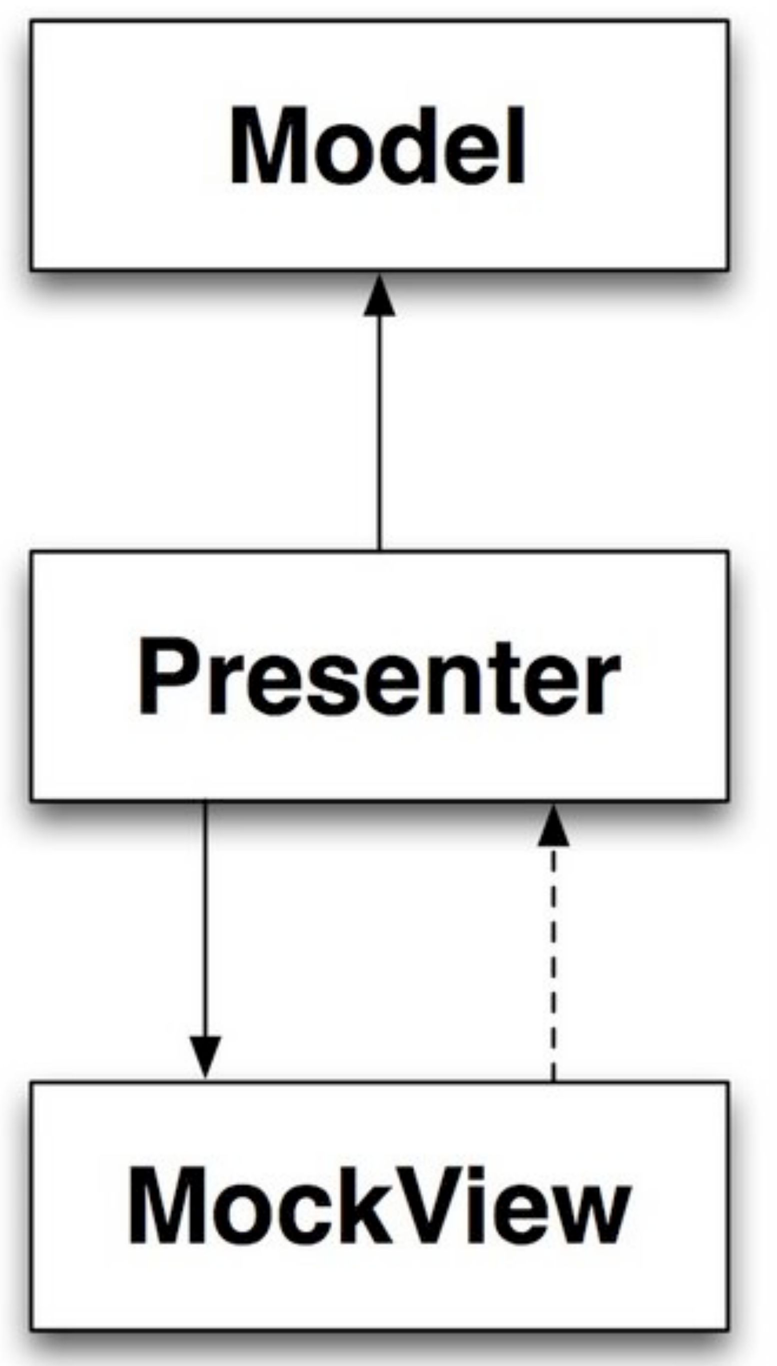
MVP



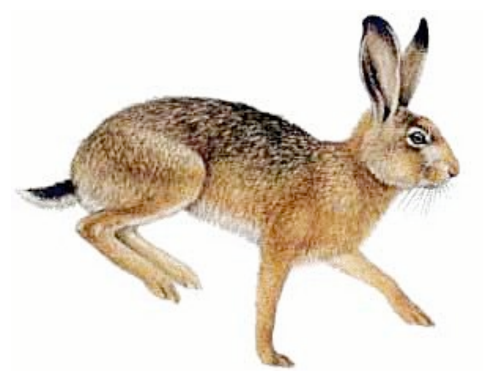


public domain image http://vintageprintable.com/wordpress/wp-content/uploads/2009/04/deirochelys_reticulariaholbrookv1p07a.jpg

MVP

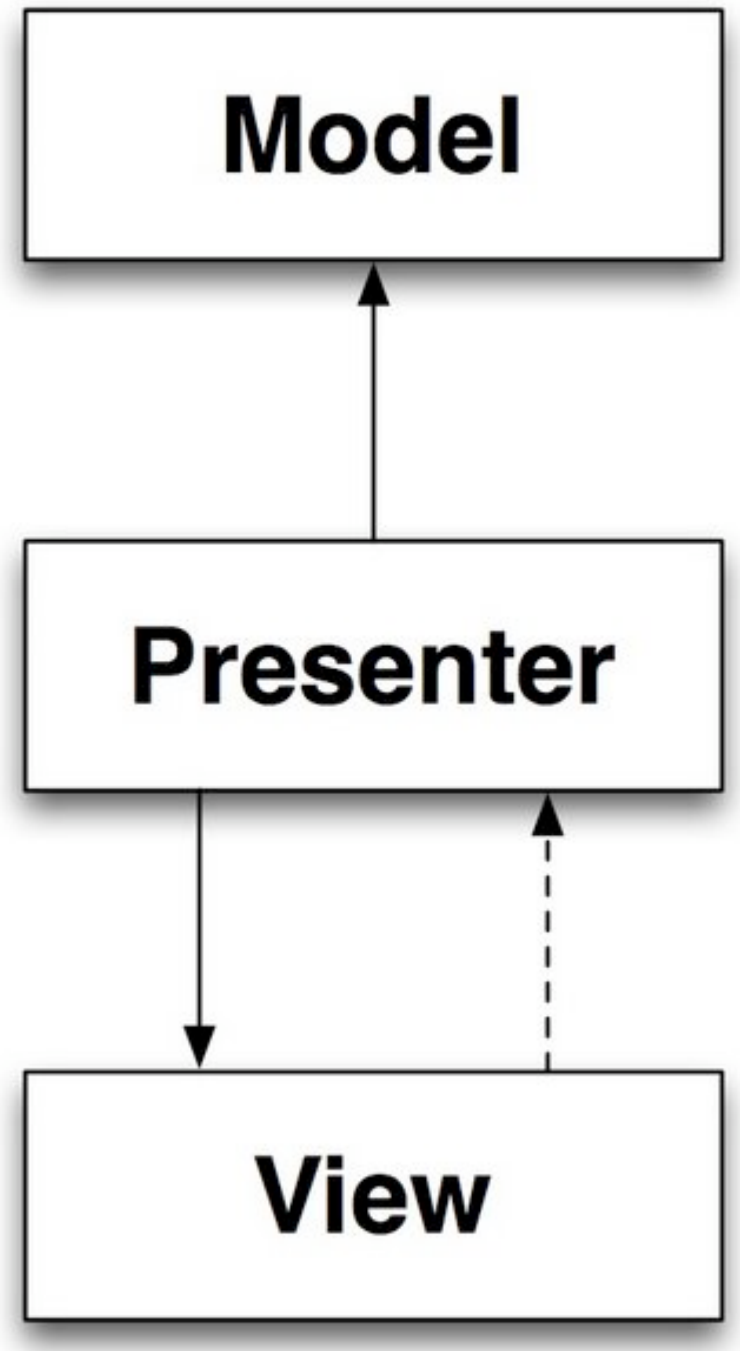


GWTMockUtilities



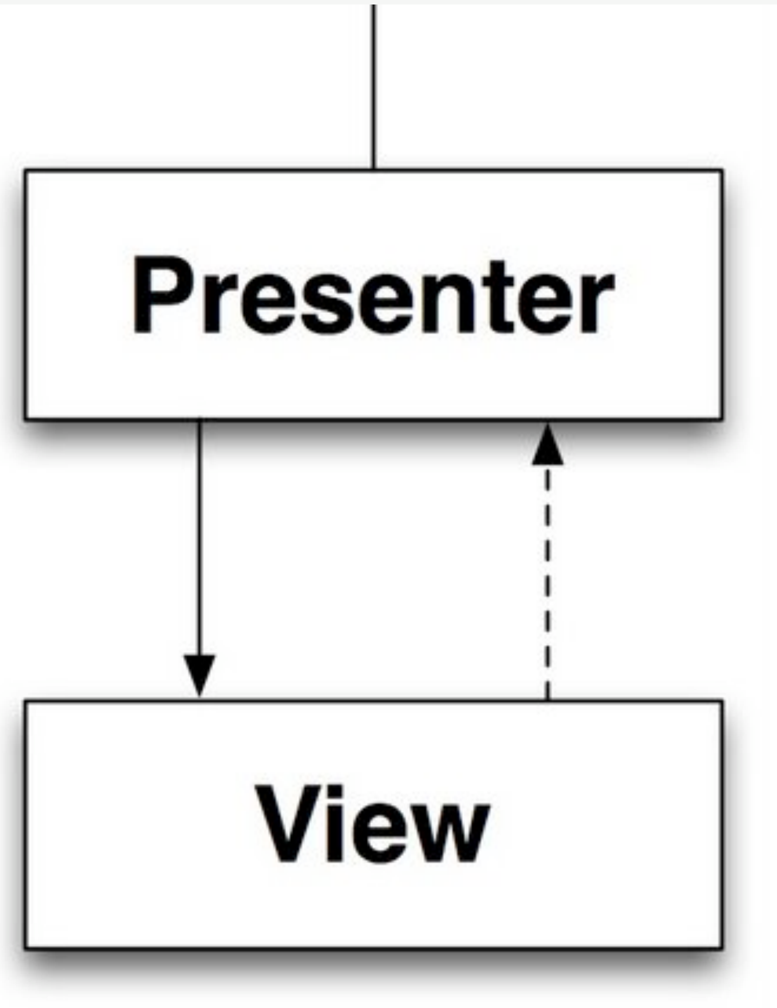
public domain image by <u>http://www.freeclipartnow.com/animals/rabbits/Brown-Hare.jpg.html</u>

MVP



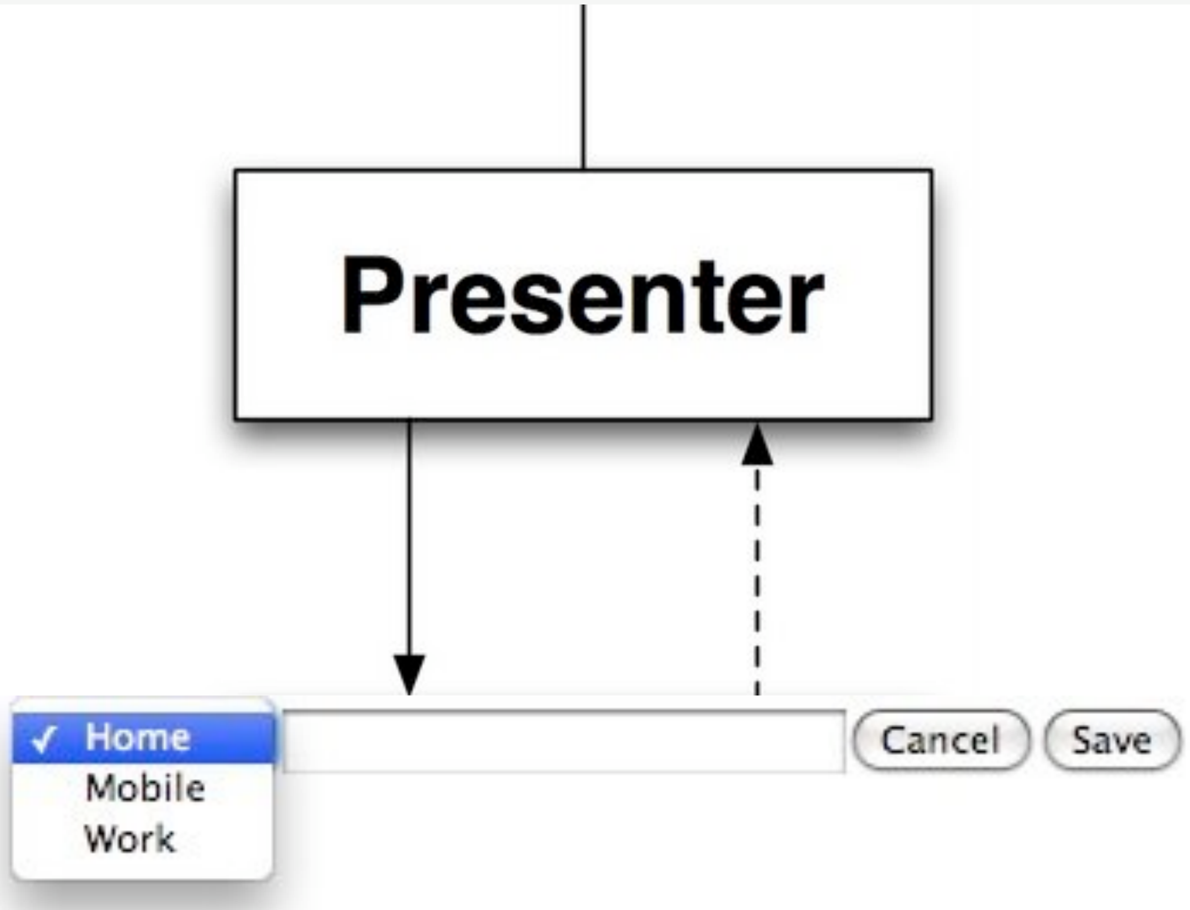
MVP

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



MVP

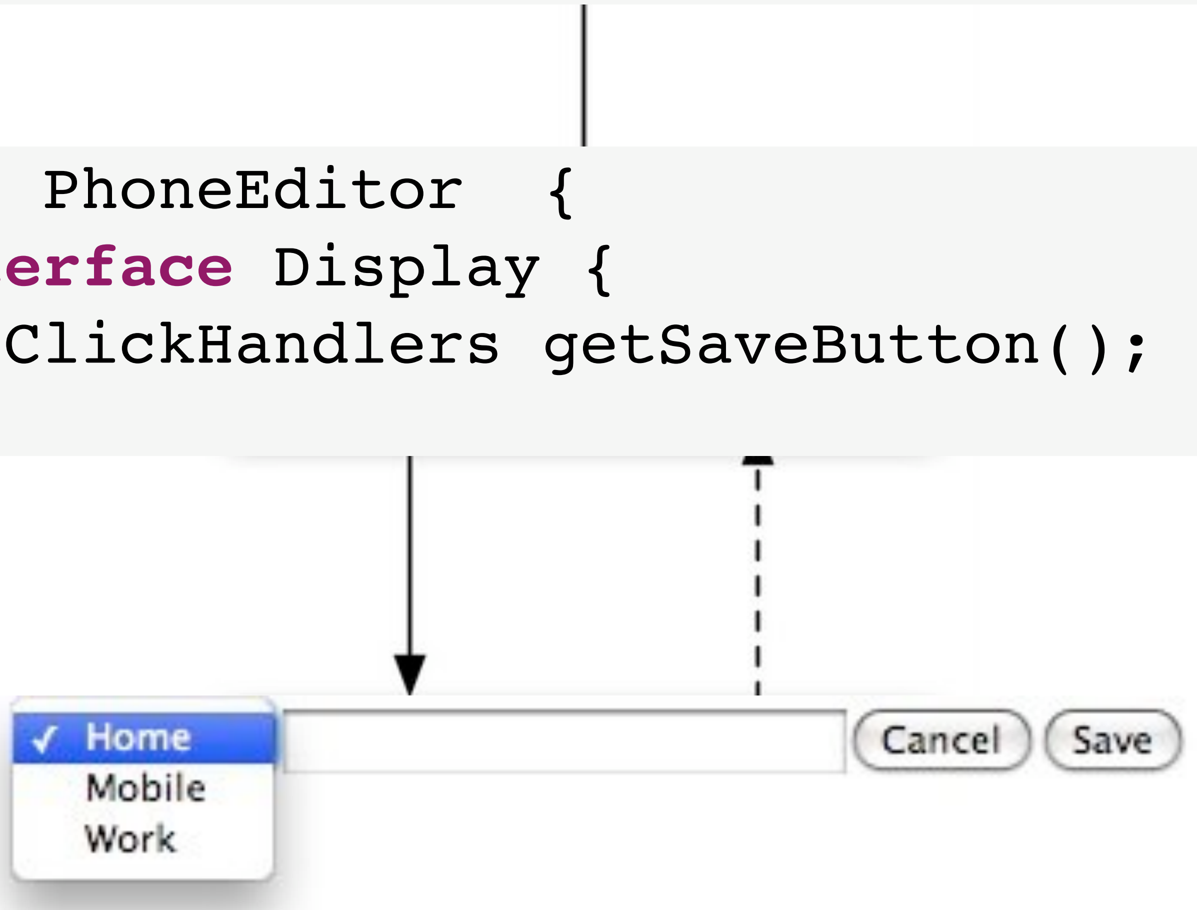
```
class Phone implements ContactDetail {
  final ContactDetailId id;
  String number;
  String label; // work, home...
}
```



MVP

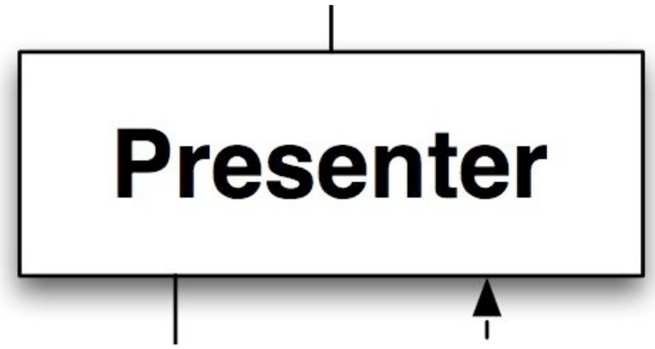
```
class Phone implements ContactDetail {
  final ContactDetailId id;
  String number;
  String label; // work, home...
}
```

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



Sample Presenter: PhoneEditor

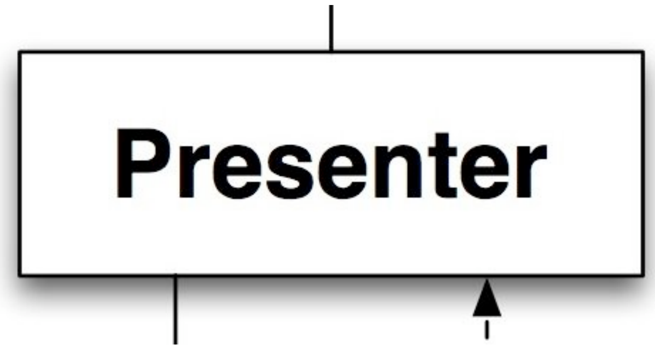
Display interface using characteristic interfaces



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

Sample Presenter: PhoneEditor

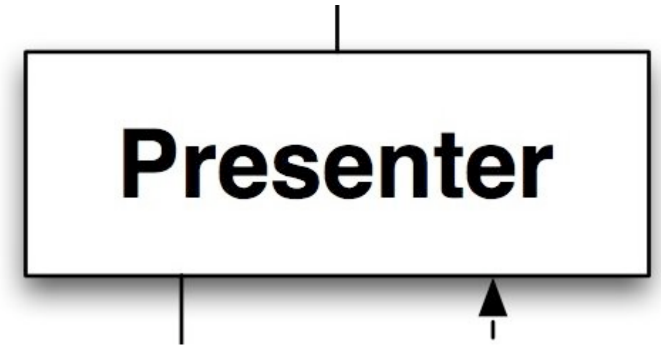
Binding the display



```
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

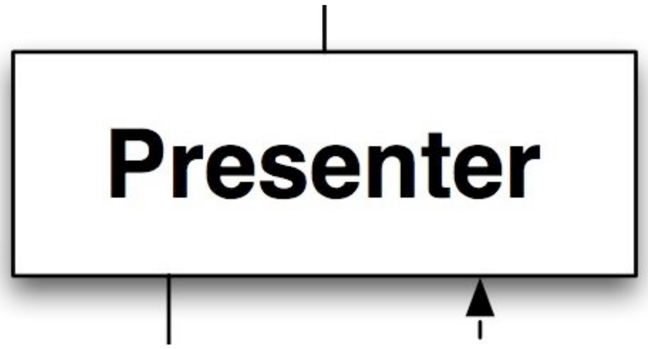
Start editing



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

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

Sample Presenter: PhoneEditor

Save it



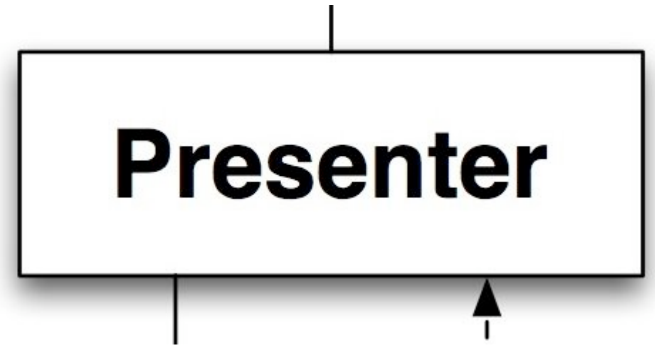
```
void doSave() {
  phone.setNumber(display.getNumberField().getValue());
  phone.setLabel(display.getLabelPicker().getValue());

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

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

Decoupling via DI

Dependency Injection

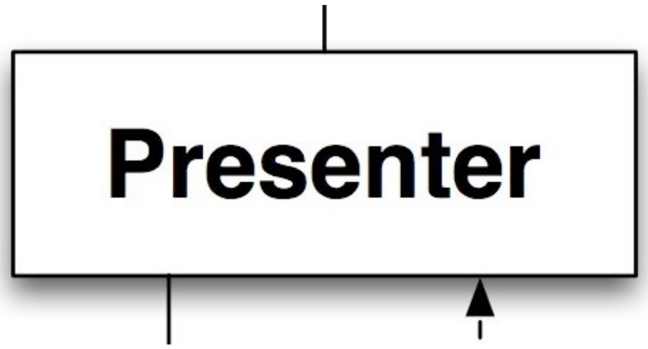
- Just a pattern:
    - No globals
    - No service locator
    - Dependencies pushed in, preferably via constructor
- Not hard to do manually
- GIN (client) and Guice (server) can automate it
    - http://code.google.com/p/google-guice/
    - http://code.google.com/p/google-gin/

DI slice for the PhoneEditor



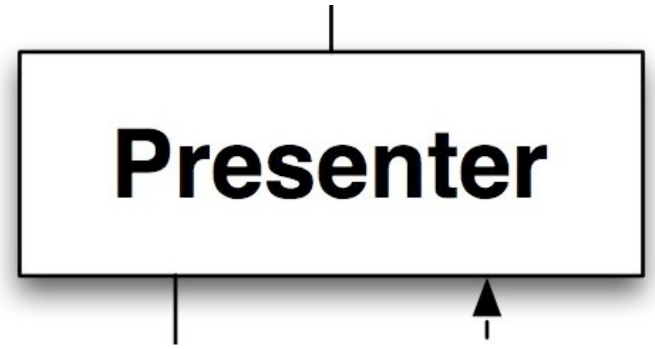
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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



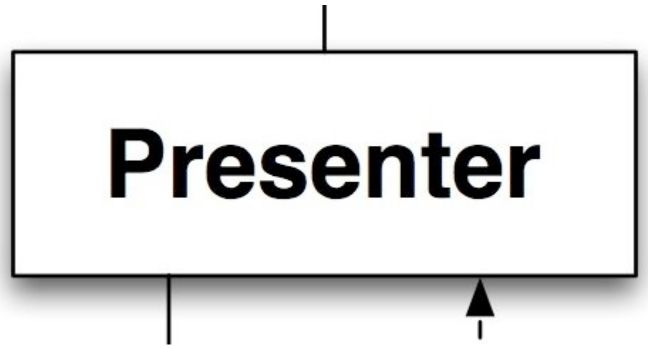
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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



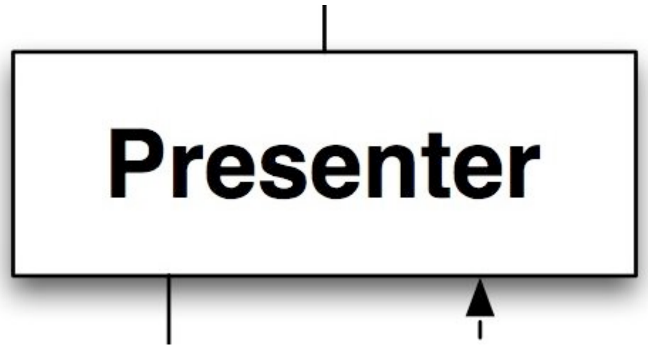
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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



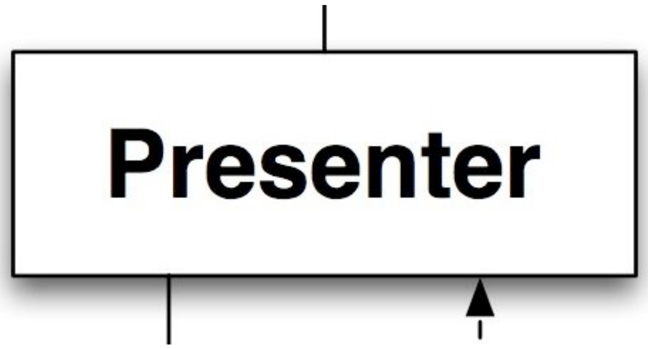
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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



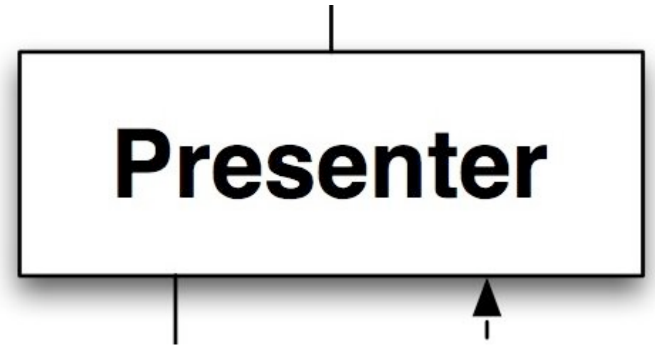
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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



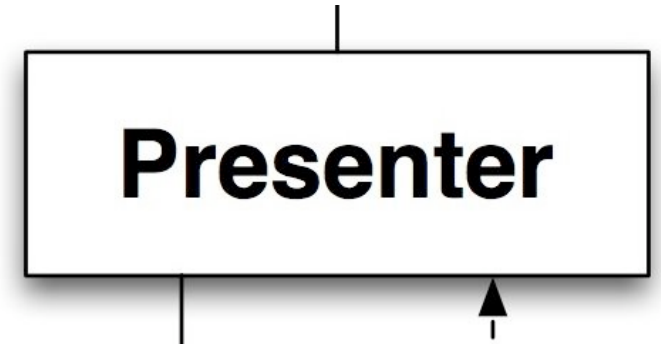
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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



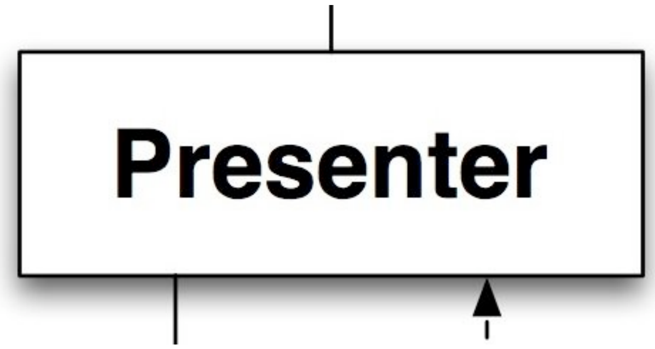
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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



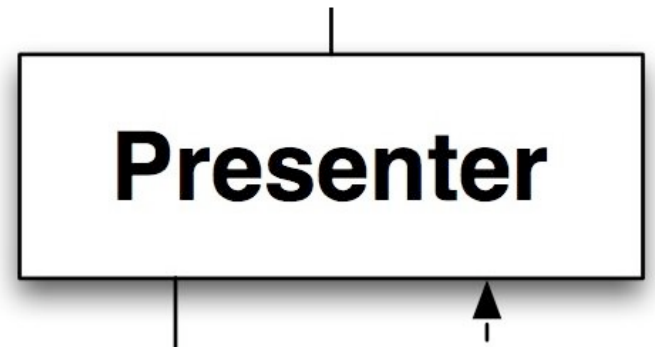
```
public void onModuleLoad() {
  ContactsServiceAsync realService =
    GWT.create(ContactsServiceAsync.class);
  CachedBatchingService rpcService =
    new CachedBatchingService(realService);

  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

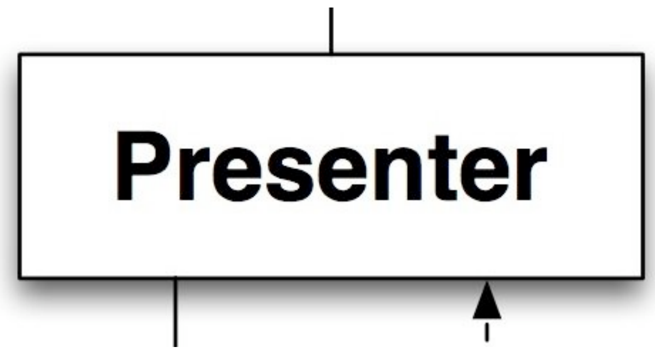


```
public void onModuleLoad() {
  MyGinjector factory = GWT.create(MyGinjector.class);

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

Decoupling payoff: fast tests

Test the PhoneEditor

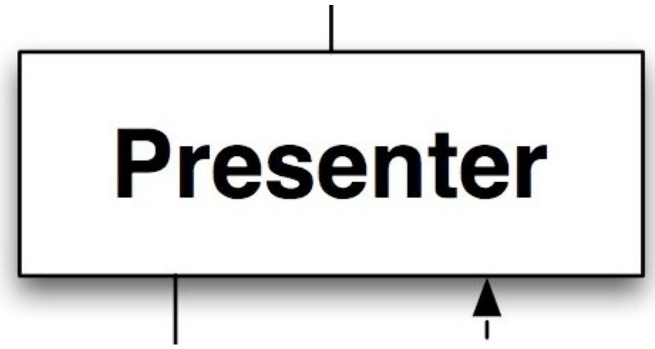
Define some mocks



```
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

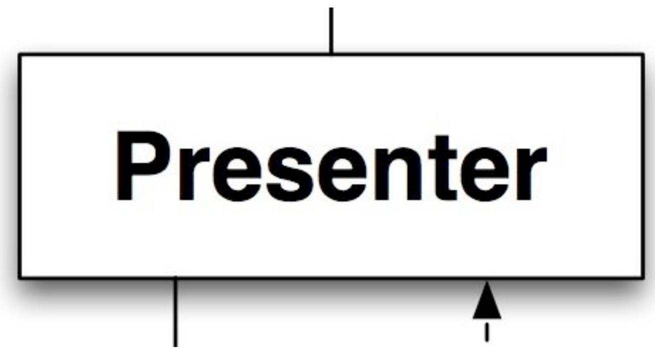
Define some mocks



```
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

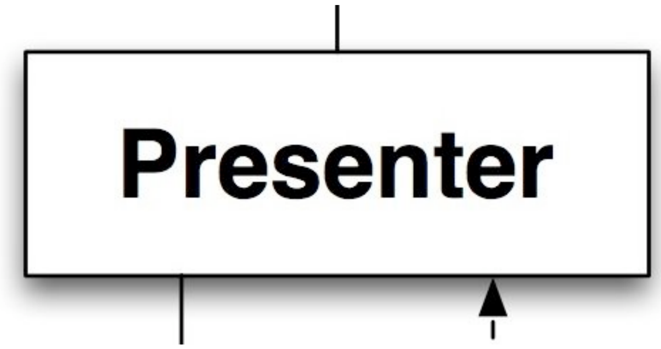
Define some mocks



```
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

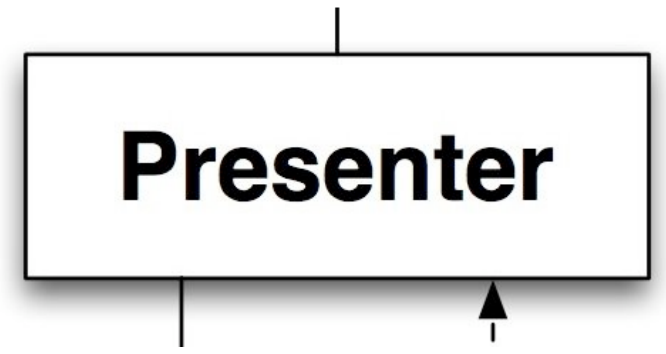
Define some mocks



```
class MockPhoneEditorDisplay implements PhoneEditor.Display {
  MockHasClickHandlers 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

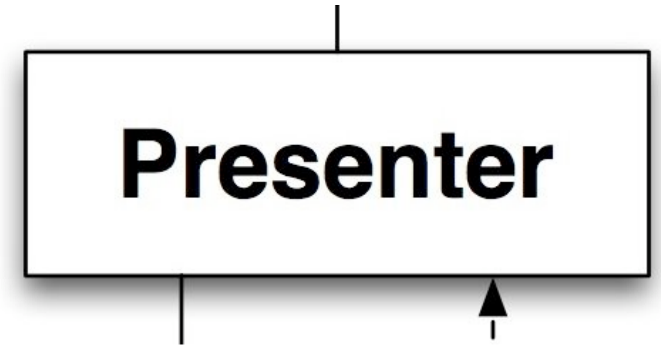
Set up the test...



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

  // Build before and after values

  ContactDetailId id = new ContactDetailId();

  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());
}
```

Strive to achieve statelessness

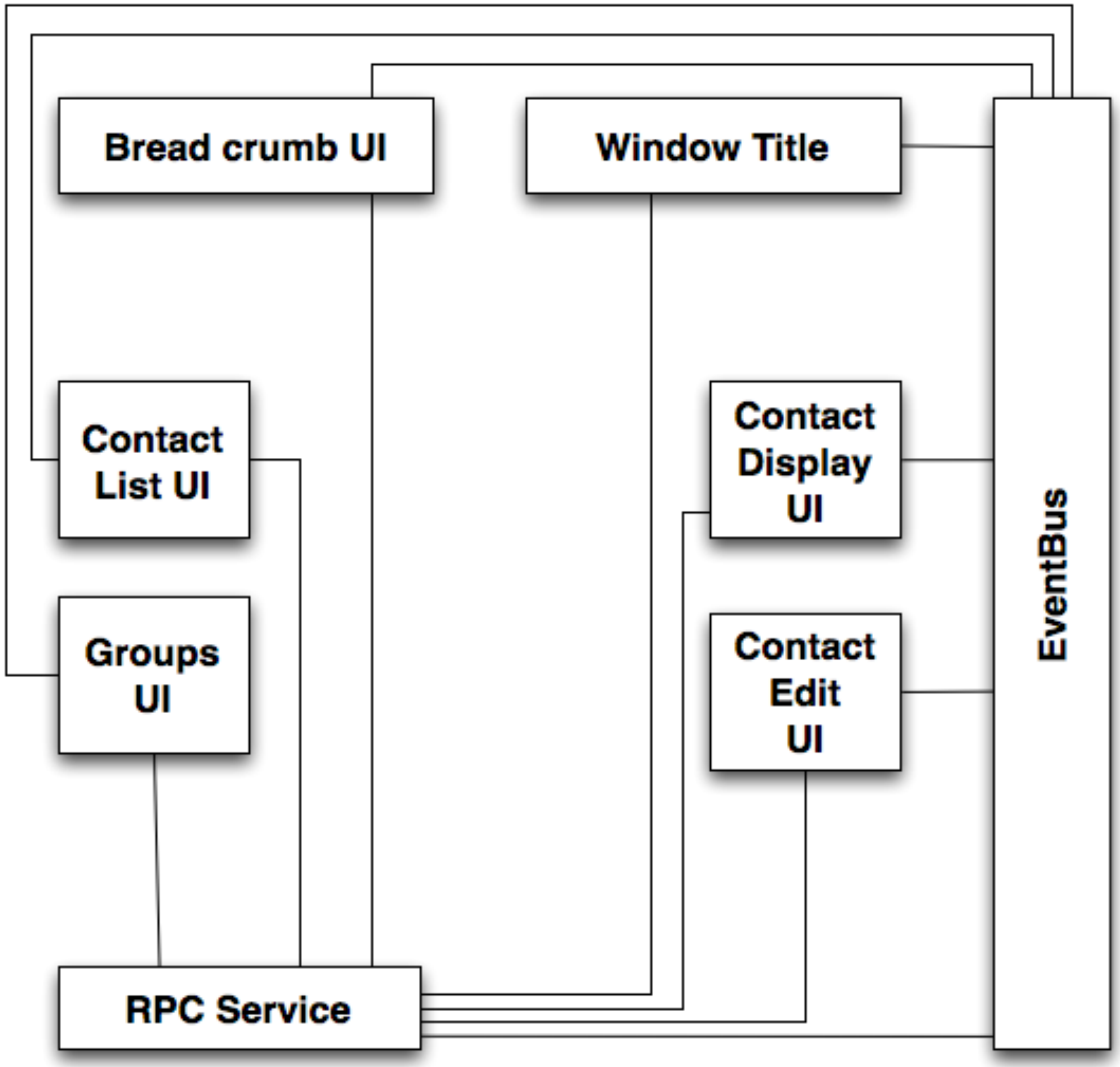
Disposable servers

- The browser embodies the session
- Server effectively stateless (except for caching)
- User should not notice server restart
- On AppEngine, MemCache works well with this attitude
    - "Values can expire from the memcache at any time"

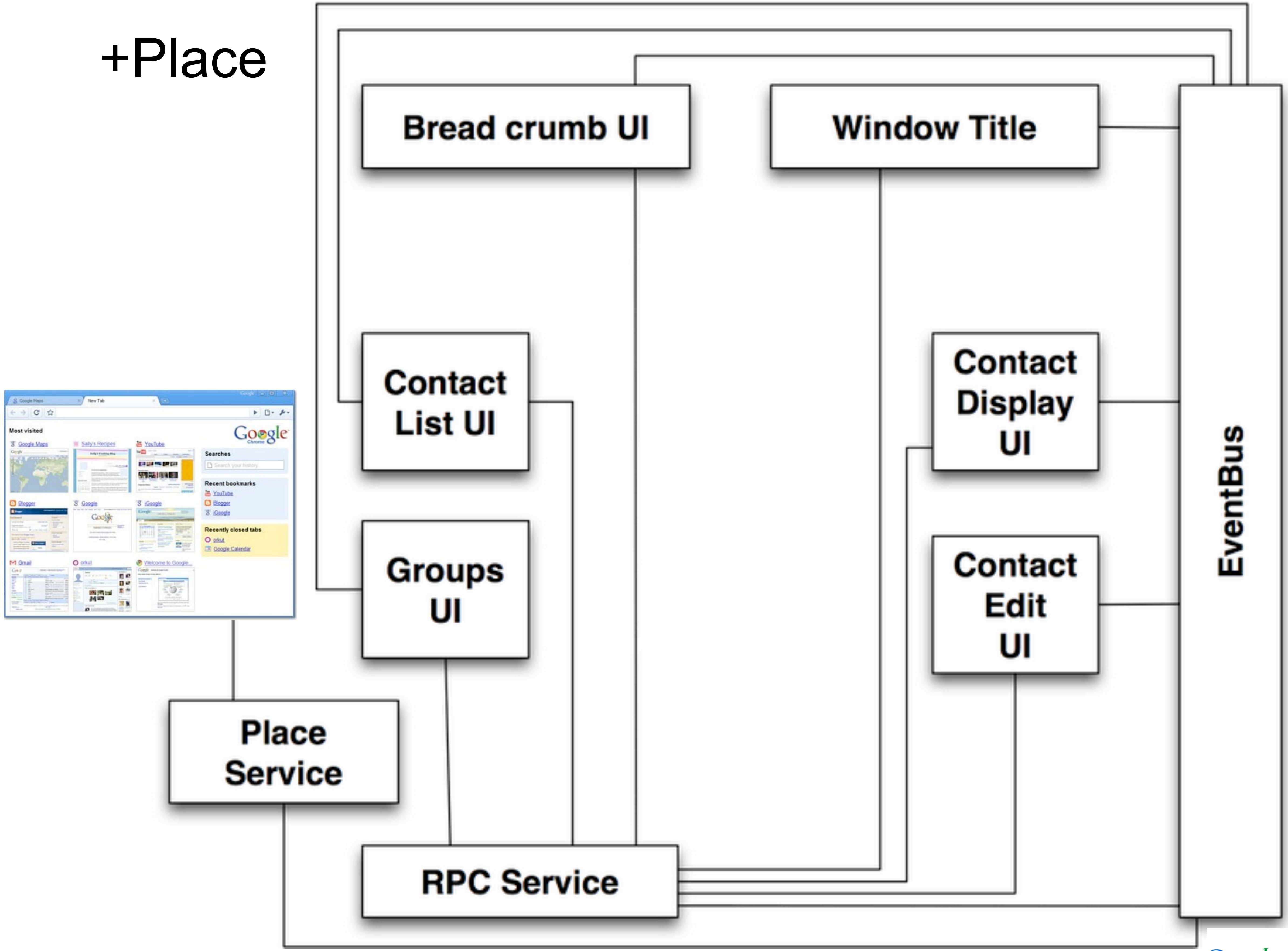
Disposable clients

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

Recall


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

+Place


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

Q & A

Google™ 09 I O