A tag-based, event-driven Flex framework

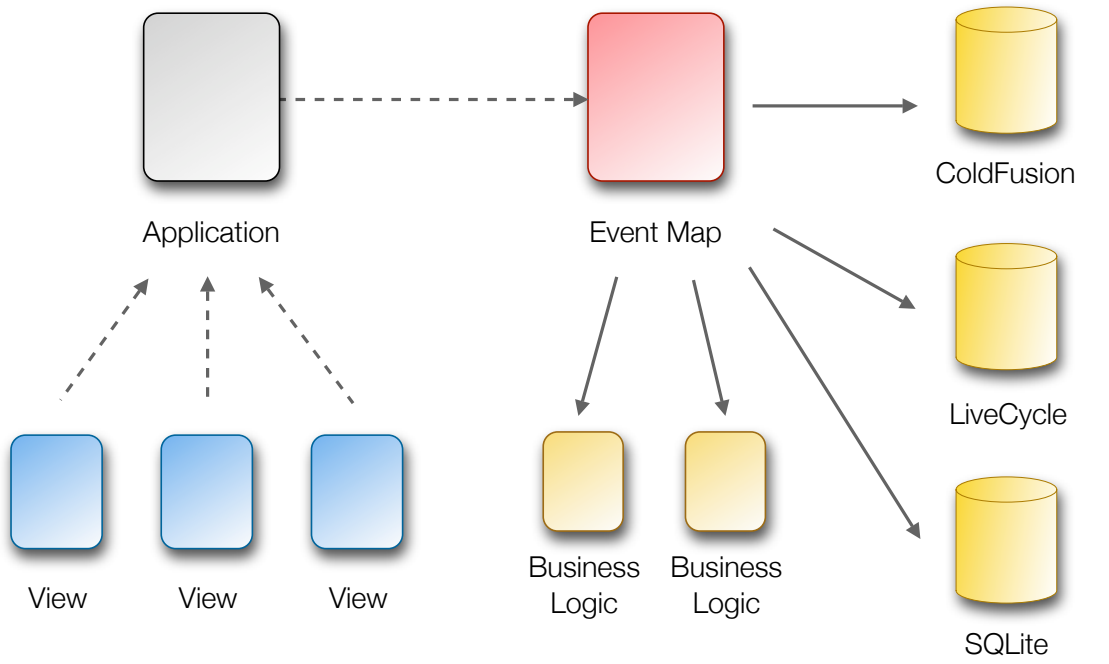
Laura Arguello

What is Mate?

- A Flex framework
- Not an ActionScript framework
- Currently in Alpha (but internally at iteration 4)

Why we created it

- To solve recurring problems
- Easy to understand
- Non-invasive
- Easy handling of data retrieval from server
- Easy to extend



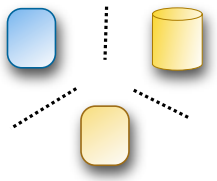
How does it do it?

1. You give it a list of actions for every event
2. Mate performs them when the event is dispatched

Using tags to accomplish it!

Common Problem

How to separate the service layer from the views and business logic



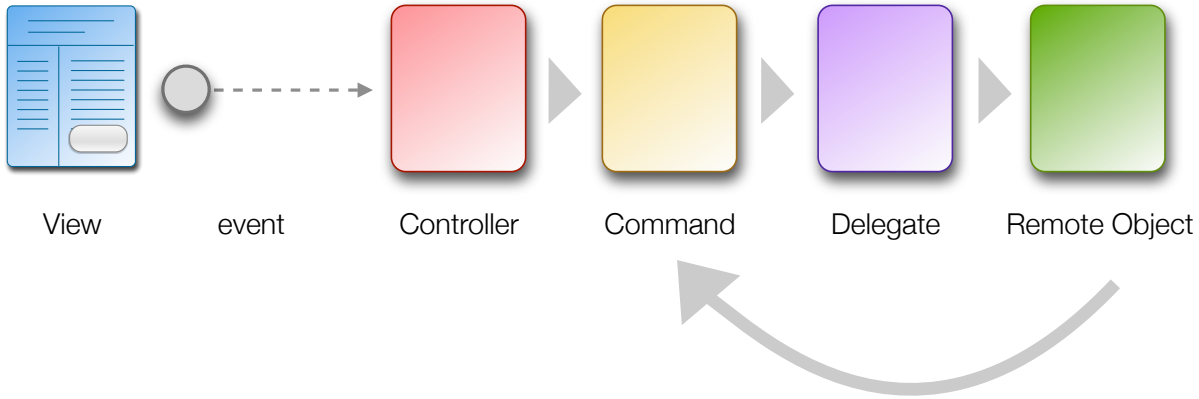
A view

MySearchView.mxml

All-in-one (everything in the view)

```
<mx:RemoteObject result="handleResult(event)"
                 fault="handleFault(event)">


private function handleResult(event:ResultEvent):void {
                  //parse results
                        }

private function handleFault(event:FaultEvent):void {}


<mx:Button label="Search" click="..."

/>
```

MySearchView.mxml

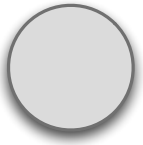


Cairngorm style

How does Mate work?

1. You give it a **list of actions** for every **event**

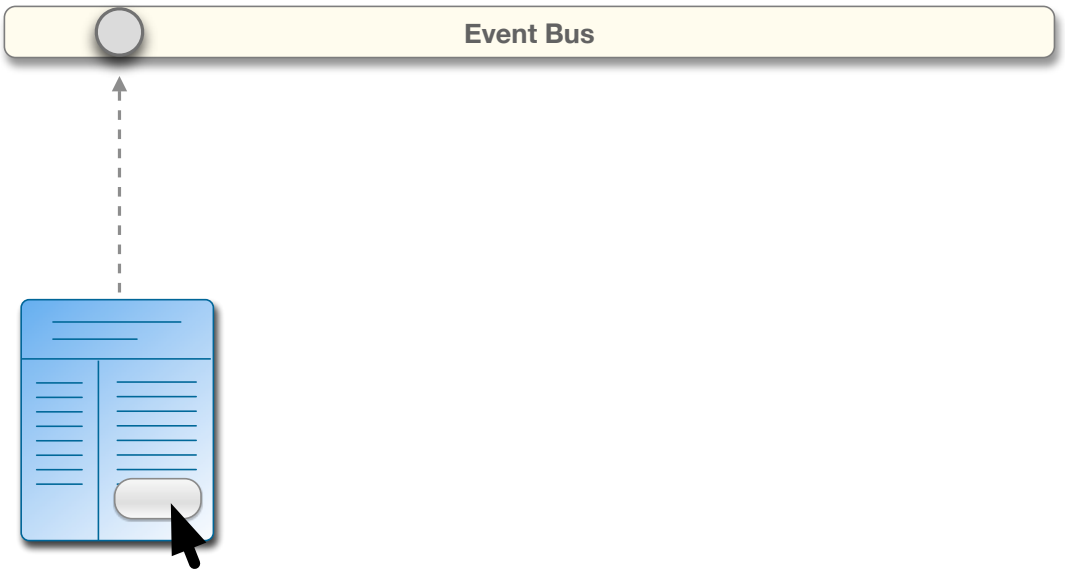
What you need



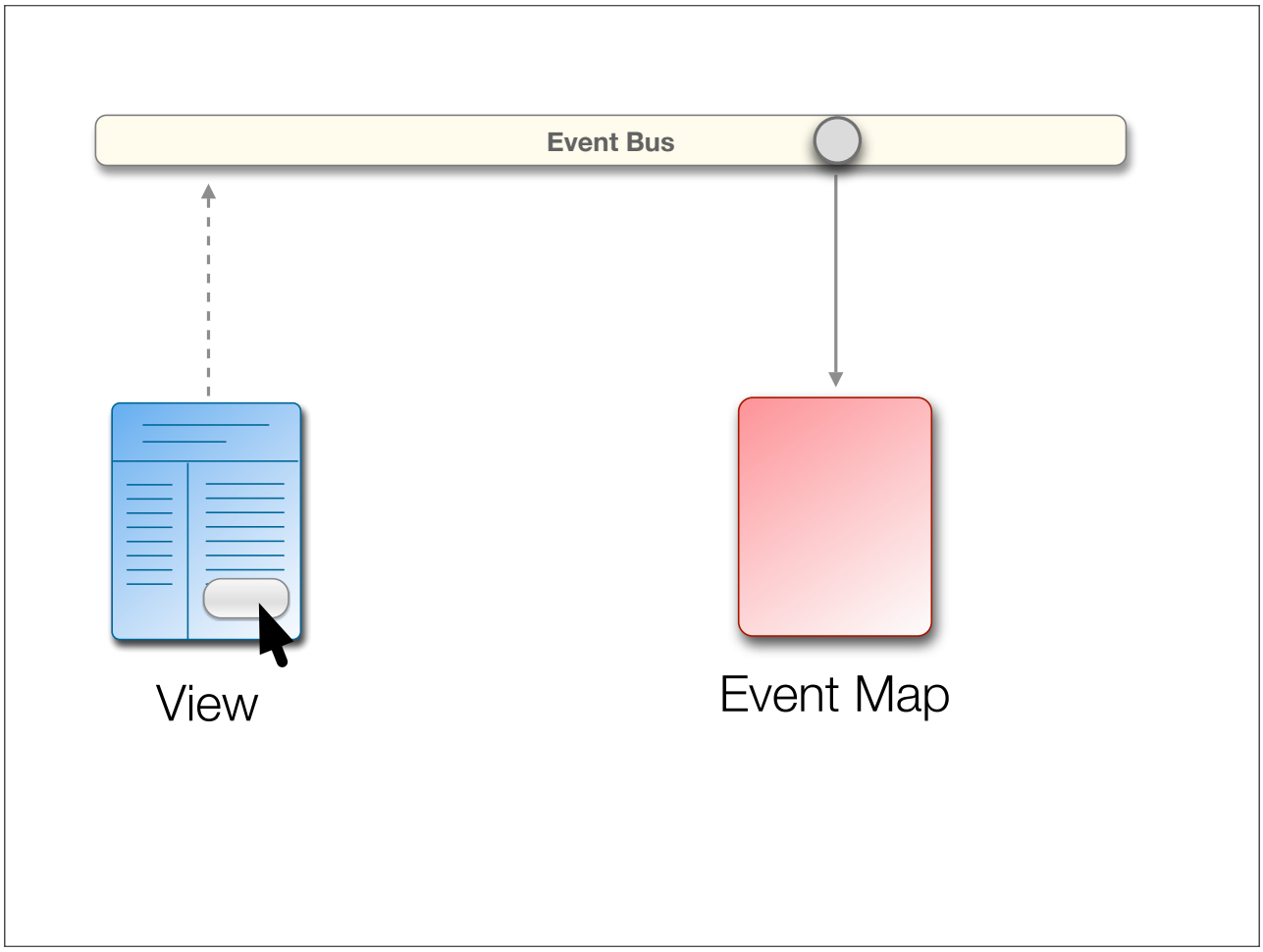
an Event

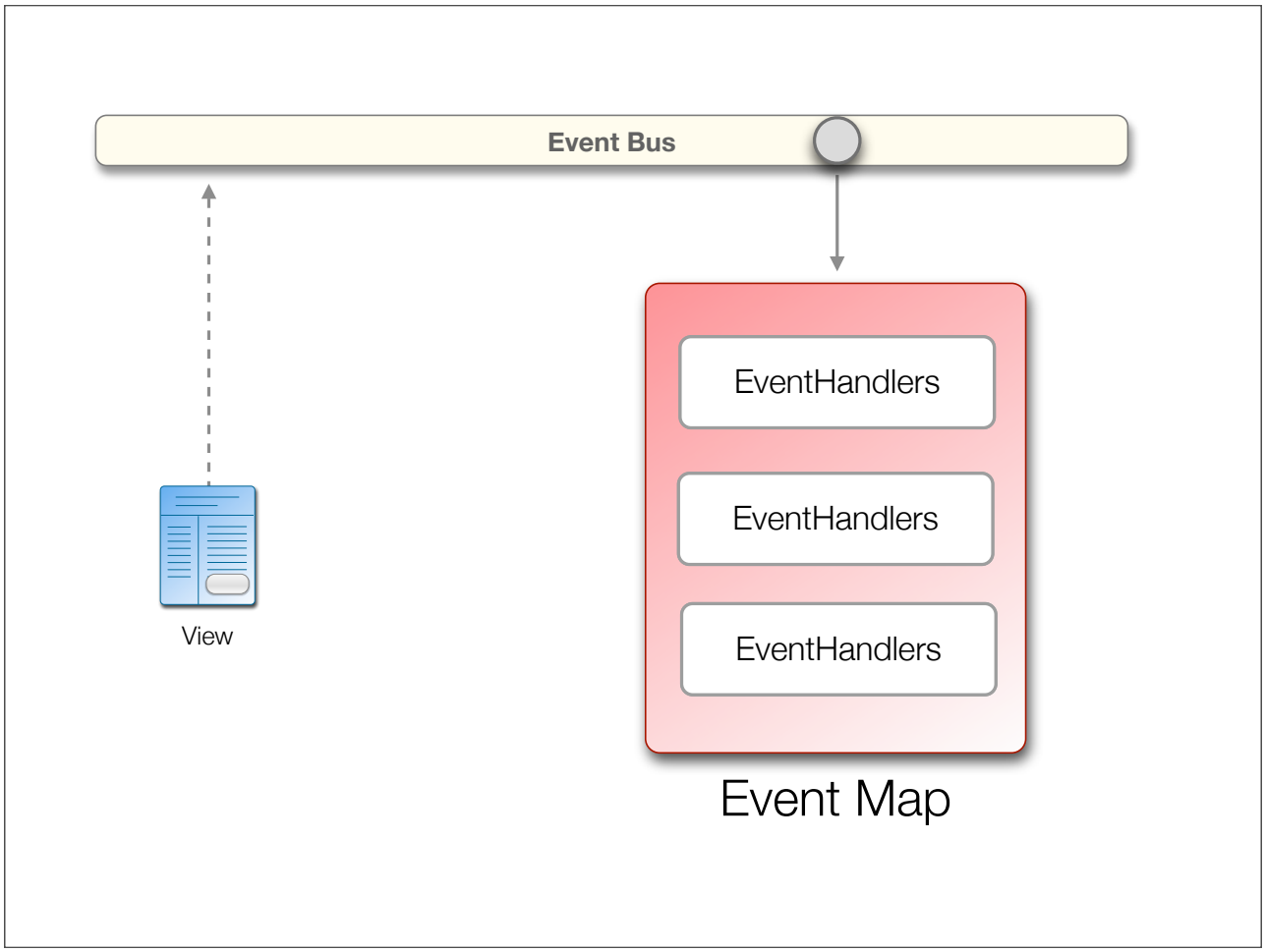
- ✔ Call the server
- ✔ Store the data
- ✔ ....

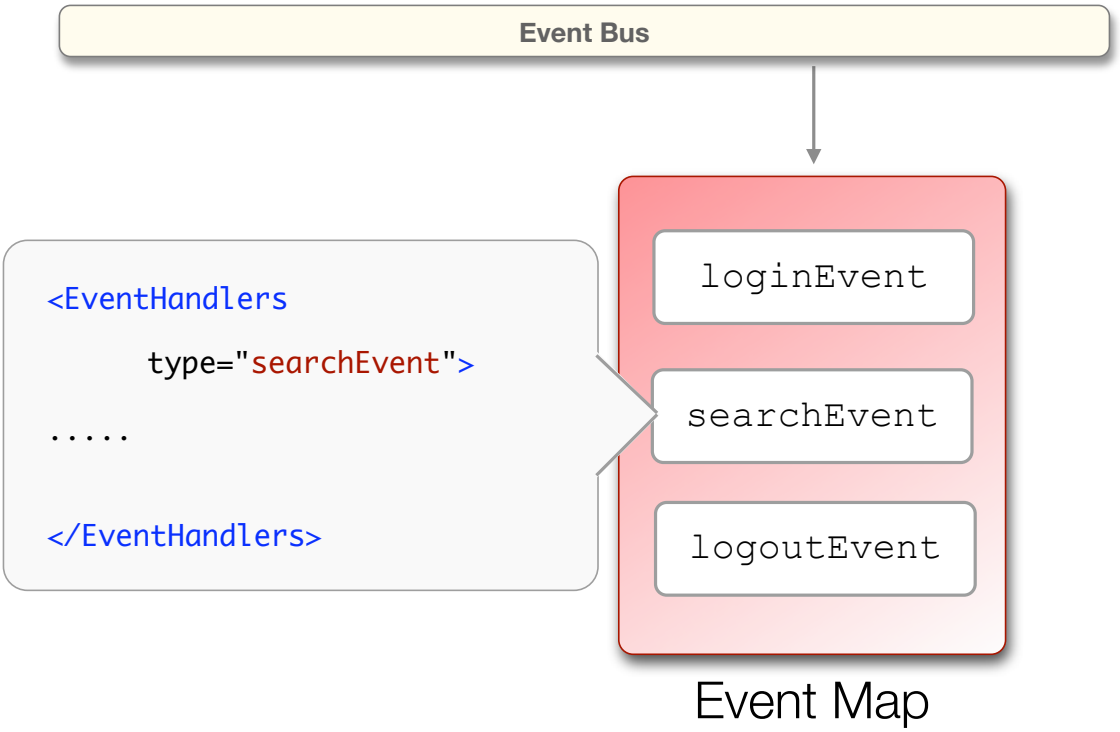
a list of actions
to perform



View


Event Bus
View
Event Map


Event Bus
View
EventHandlers
EventHandlers
EventHandlers
Event Map



Event Handlers

```
<mate:EventHandlers type="searchEvent">
```

- ✔ Call the server
- ✔ Store the data
- ✔ ....

```
</mate:EventHandlers>
```

Event Handlers

```
<mate:EventHandlers type="searchEvent">

    <mate:RemoteObjectInvoker ...>
    <mate:MethodInvoker ...>
    <mate: ...>

</mate:EventHandlers>
```

Event Handlers

```
<mate:EventHandlers type="searchEvent">

    <mate:RemoteObjectInvoker ...>
    <mate:MethodInvoker ...>
    <ns:YourTag ... >

</mate:EventHandlers>
```

EventHandlers

```
<mate:EventHandlers type="searchEvent">


      <mate:RemoteObjectInvoker ...>

      <mate:MethodInvoker
              generator="MyClass"
              method="saveResult"
              arguments="{[...]}>


</mate:EventHandlers>
```

```
var worker:MyClass = new MyClass();

worker.saveResult(...);
```

resultHandlers for service calls

```
<EventHandlers type="searchEvent">


     <RemoteObjectInvoker ...>
            <resultHandlers>

                 <MethodInvoker ... />

            </resultHandlers>
     </RemoteObjectInvoker>


</EventHandlers>
```

faultHandlers for service calls

```
<EventHandlers type="searchEvent">

    <RemoteObjectInvoker ...>

        <resultHandlers>
            <MethodInvoker ... />
        </resultHandlers>

        <faultHandlers>
            <MethodInvoker ... />
        </faultHandlers>

    </RemoteObjectInvoker>

</EventHandlers>
```

Your main application

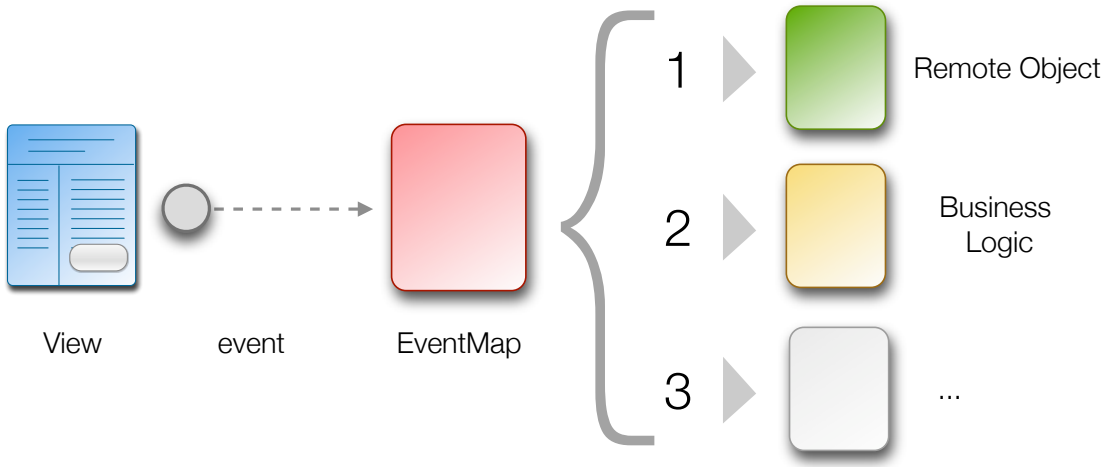
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application ...>

    <!-- Event Maps  -->
    <maps:MyEventMap />

    <!-- Views  -->
    <views:MainUI />

</mx:Application>
```

The Event Map

```
<?xml version="1.0" encoding="utf-8"?>
<EventMap ...>
    <EventHandlers type="searchEvent">

1       <RemoteObjectInvoker ...>
            <resultHandlers>
2               <MethodInvoker ... />
            </resultHandlers>
        </RemoteObjectInvoker>

    </EventHandlers>

</EventMap>
```

Mate's solution

How to separate the service layer from the views and business logic



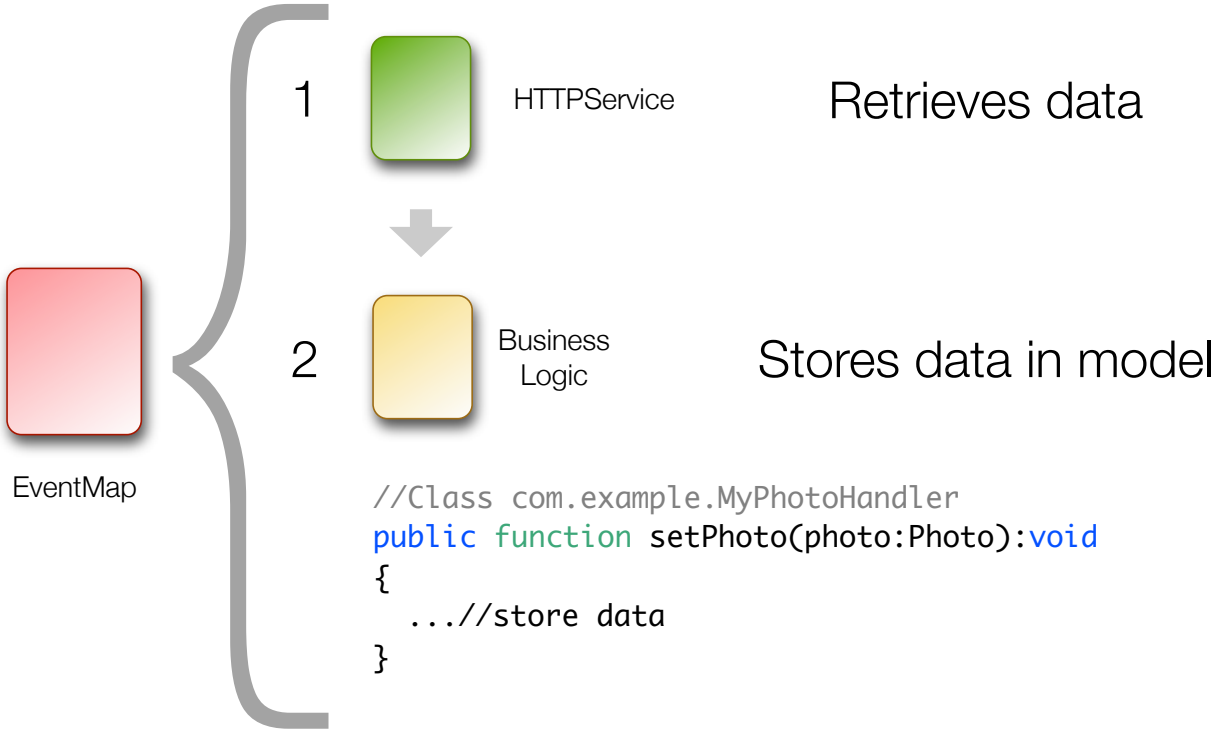
Common Problem

Parsing data coming from server 

Call Flickr to get a photo

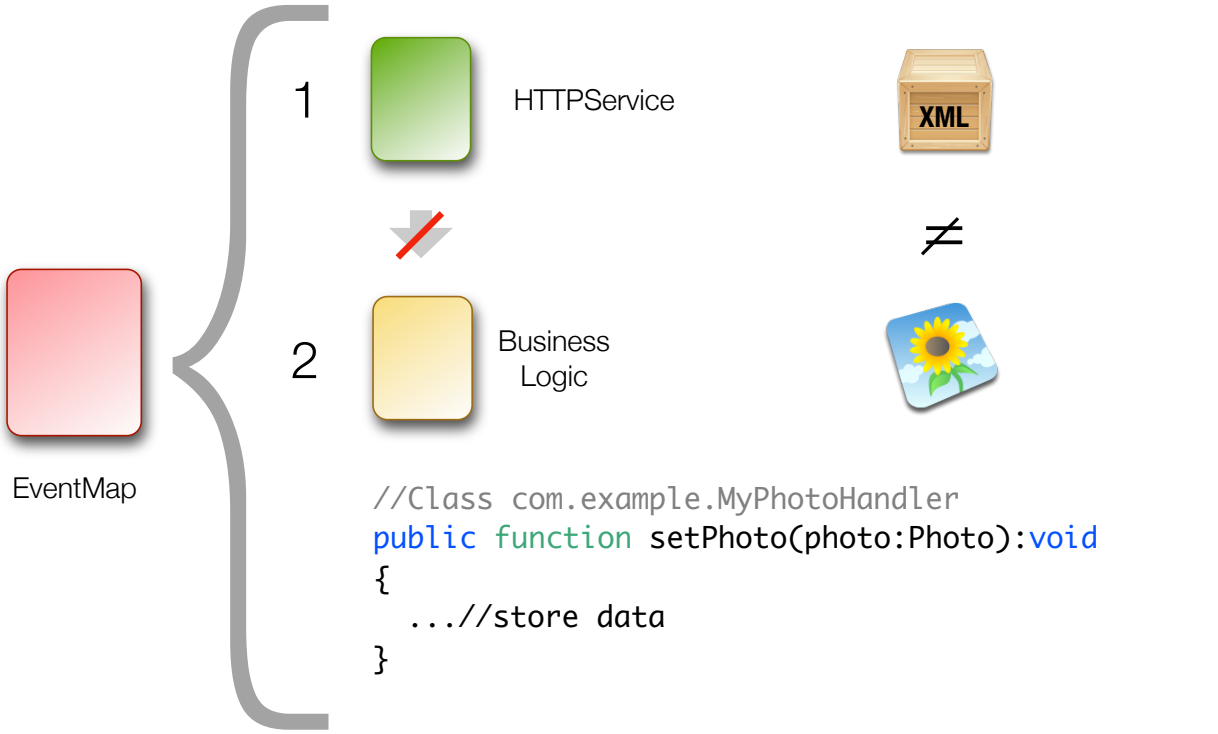
1. Call Flickr
2. Receive XML with photo information
3. Parse XML
4. Store the photo information as a Photo object

Parsing XML



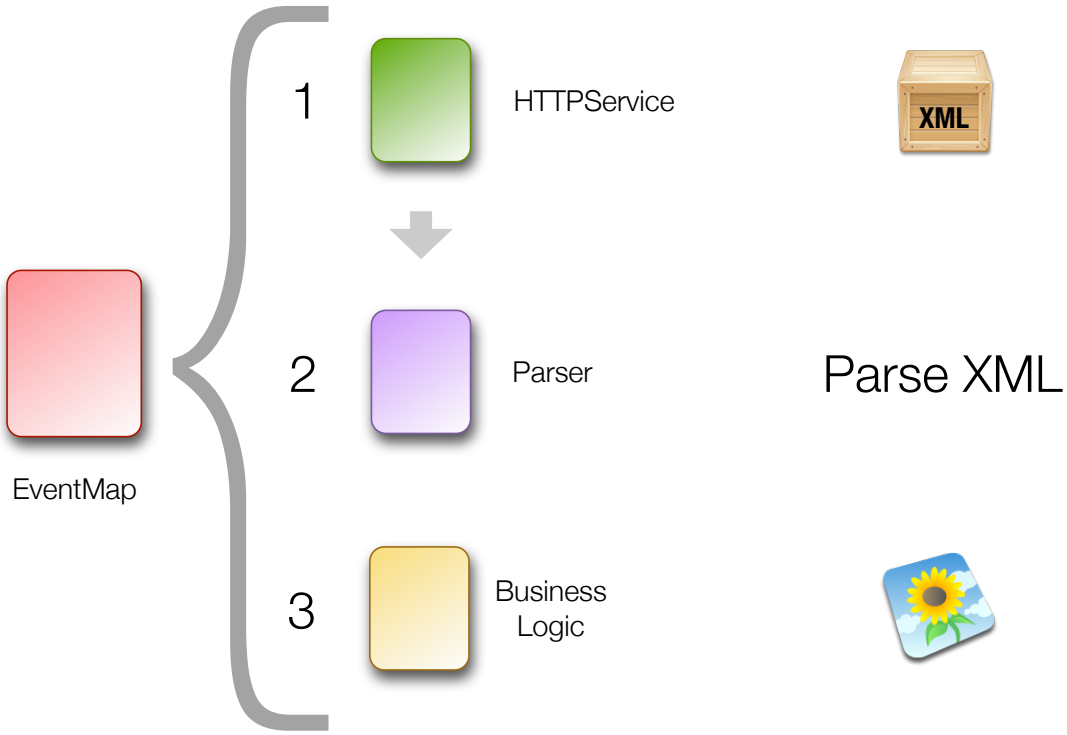

```
//Class com.example.MyPhotoHandler
public function setPhoto(photo:Photo):void
{
  ...//store data
}
```

Parsing XML

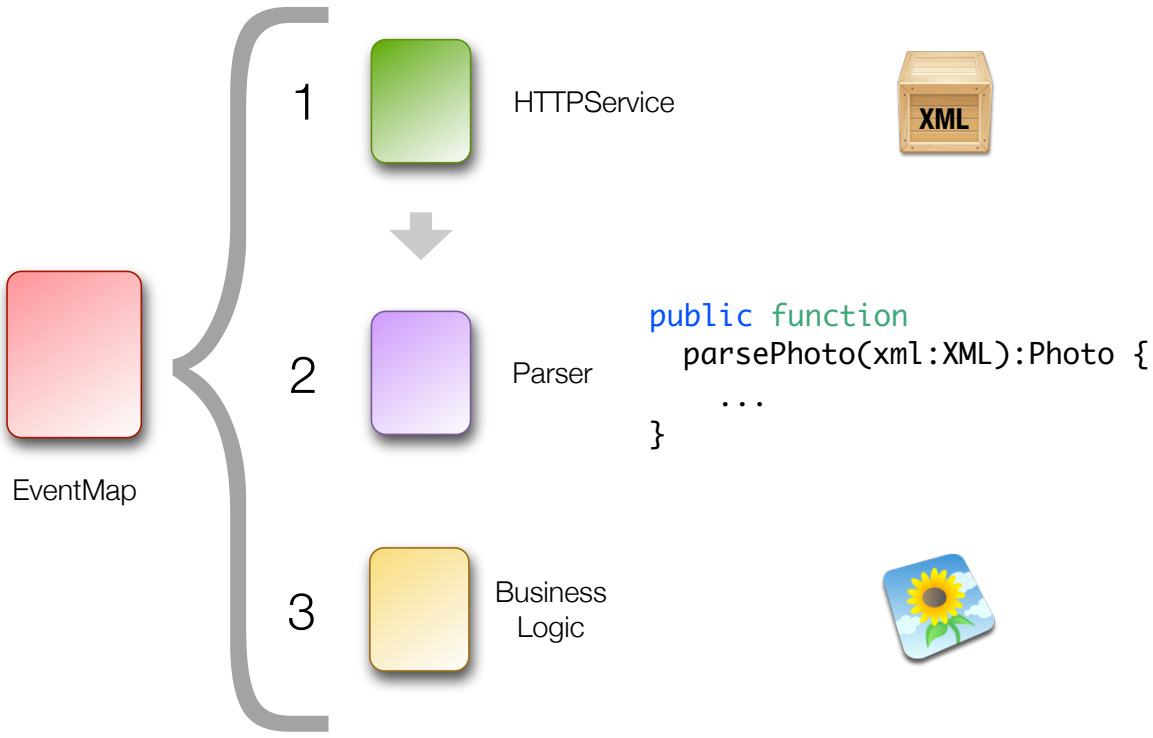



```
//Class com.example.MyPhotoHandler
public function setPhoto(photo:Photo):void
{
  ...//store data
}
```

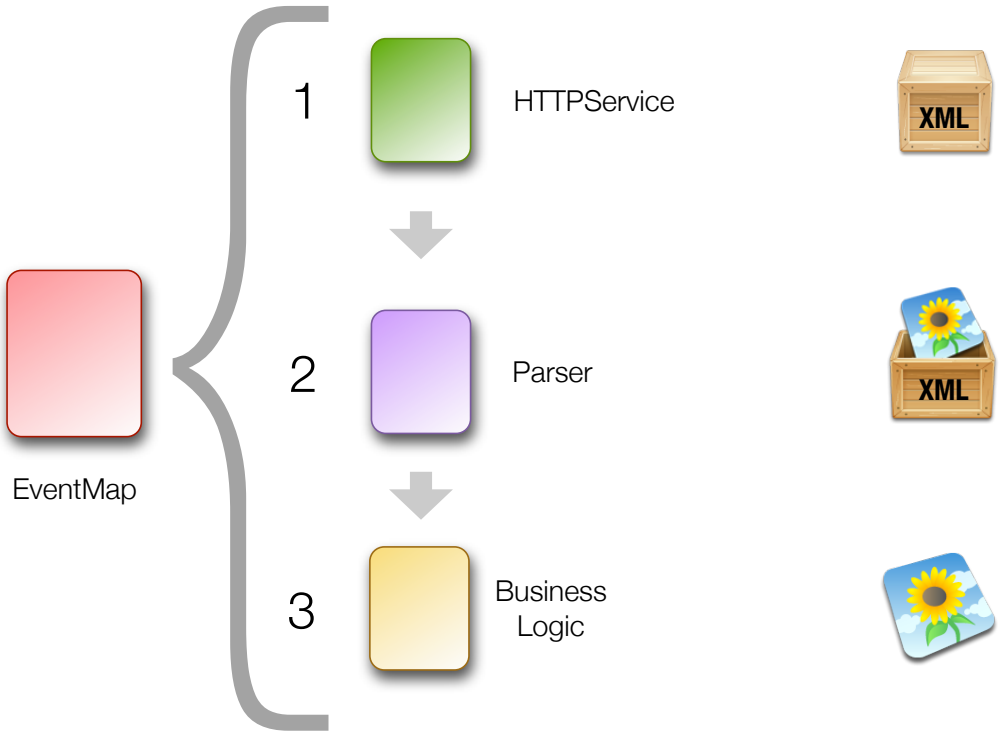
Parsing XML



Parsing XML



Parsing XML



In the EventMap (MyEventMap.mxml)

```
<EventHandlers type="{PhotoEvent.GET}">
    <HTTPServiceInvoker .....>
        <resultHandlers>

        </resultHandlers>
    </HTTPServiceInvoker>
</EventHandlers>
```

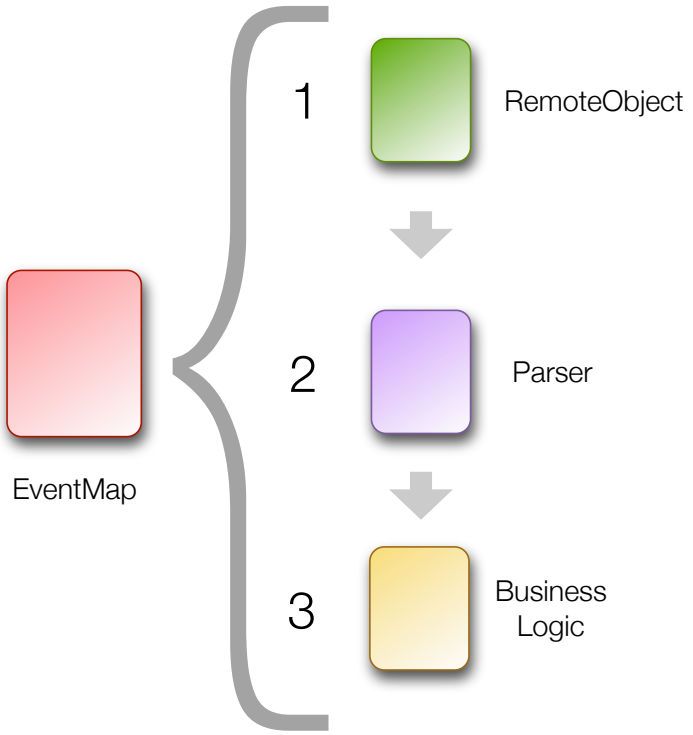
In the EventMap (MyEventMap.mxml)

```
<EventHandlers type="{PhotoEvent.GET}">
    <HTTPServiceInvoker .....>
        <resultHandlers>
            <MethodInvoker
                generator="com.example.MyXMLParser"
                method="parsePhoto"
                arguments="{resultObject}" />
            <MethodInvoker
                generator="com.example.MyPhotoHandler"
                method="setPhoto"
                arguments="{lastReturn}" />
        </resultHandlers>
    </HTTPServiceInvoker>
</EventHandlers>
```

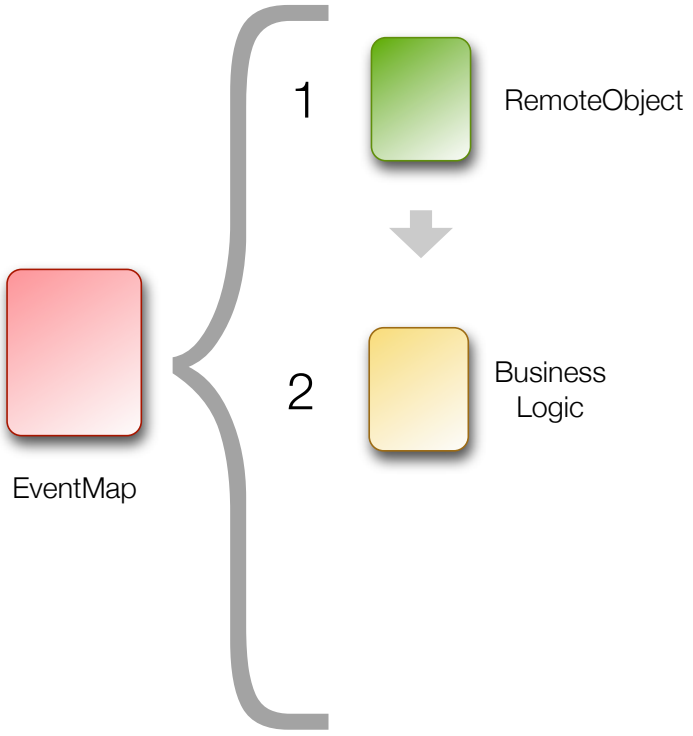
Problem

How to switch between different types of services 

XML to Remote Object



XML to Remote Object



XML to Remote Object

```
<EventHandlers type="{PhotoEvent.GET}">
    <RemoteObjectInvoker .....>
        <resultHandlers>

            <MethodInvoker
                generator="com.example.MyPhotoHandler"
                method="setPhoto"
                arguments="{resultObject}" />

        </resultHandlers>
    </RemoteObjectInvoker>
</EventHandlers>
```

Other handy features

Little things that'll make you happy 

Listen for initialization events

```
<mx:Application ...
   creationComplete="init()" >


      <mx:Script>
          private function init():void {

          var event:MyInitEvent = new MyInitEvent();
          ...
          dispatchEvent(event);

      </mx:Script>

</mx:Application>
```

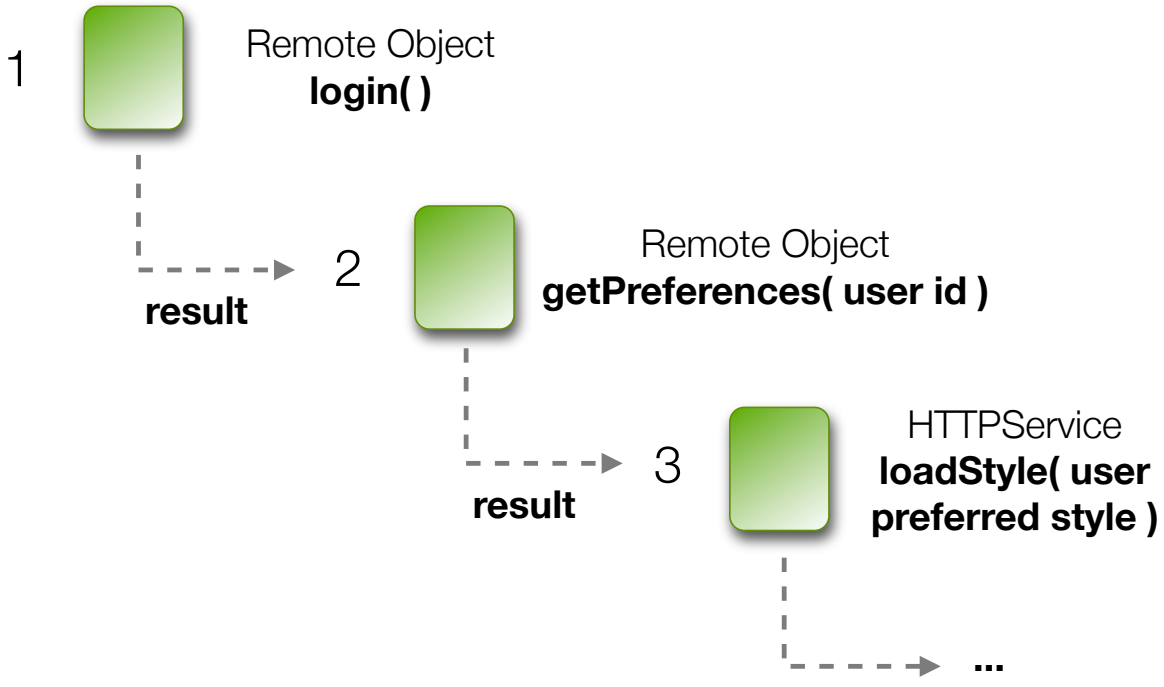
Listen for initialization events

```
<mx:Application ... >




      <maps:MyEventMap />




</mx:Application>
```

Listen for Flex events

```
<EventMap ...>
    <EventHandlers type="{FlexEvent.INITIALIZE}">
        ... actions to perform ...
    </EventHandlers>
</EventMap>
```

Chaining service calls



Chaining service calls

```
<EventHandlers type="{LoginEvent.LOGIN}">

     <RemoteObjectInvoker method="login" ...>

       <resultHandlers>

            ... actions to perform on result ...

       </resultHandlers>

     </RemoteObjectInvoker>

</EventHandlers>
```

Result has its own set of actions to perform

Chaining service calls

```
<EventHandlers type="{LoginEvent.LOGIN}">

     <RemoteObjectInvoker method="login" ...>

       <resultHandlers>
            <RemoteObjectInvoker method="getPreferences" >
                 <resultHandlers>
                   ... actions to perform on result ...
                 </resultHandlers>
            </RemoteObjectInvoker>

       </resultHandlers>

     </RemoteObjectInvoker>

</EventHandlers>
```

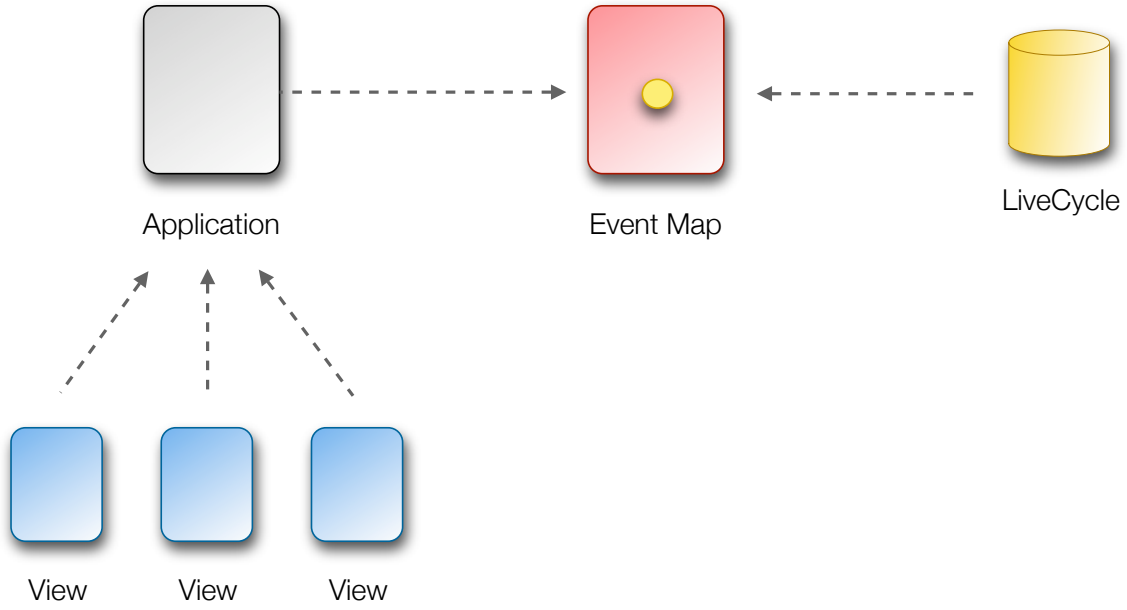
Stopping the handlers

```
<EventHandlers type="{SearchEvent.FIND_ALL}">

    1  <MethodInvoker ... />

    2  <MethodInvoker ... />

       <StopHandlers ... />

    3  <MethodInvoker ... />   ?

</EventHandlers>
```

Flex messaging integration

Handling Message events

Flex messaging integration



Flex messaging integration

```
<MessageHandlers destination="ColdFusionGateway">

    ... actions to perform on message received ...

</MessageHandlers>
```

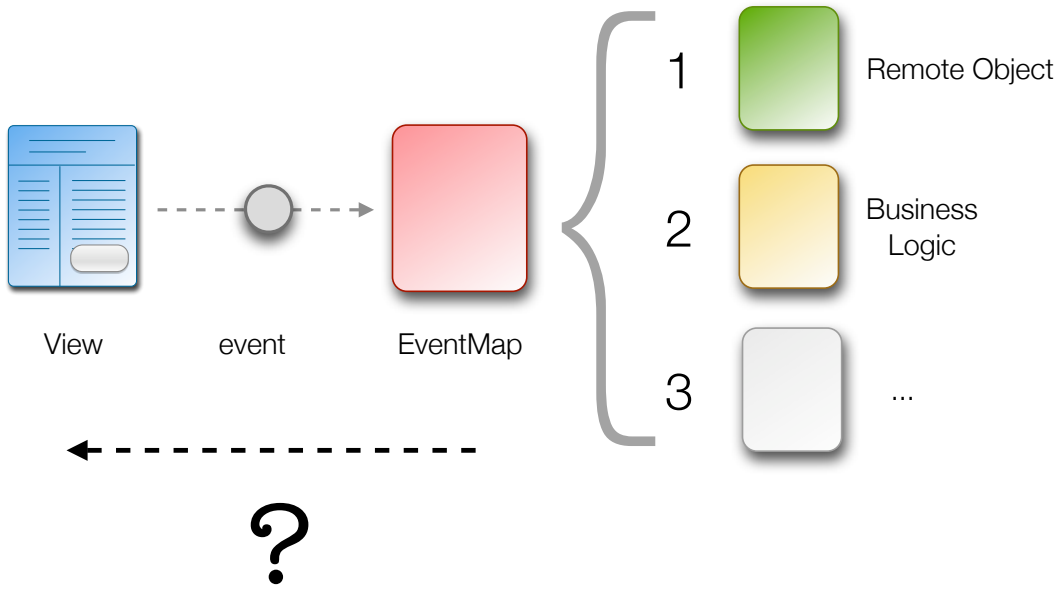
Multi-way communication

Views-views, event map-views

A view

MySearchView.mxml

How do we get back to the view?



Receiving a response in the view

```
var event:SearchEvent =
        new SearchEvent("searchEvent", true);

event.keyword = textinput.text;

myDispatcher.dispatchEvent(event);
```

Receiving a response in the view

```
var event:SearchEvent =
       new SearchEvent("searchEvent", true);

event.keyword = textinput.text;

myDispatcher.dispatchEvent(event);


<mate:Dispatcher id="myDispatcher">


</mate:Dispatcher>
```

Receiving a response in the view

```
var event:SearchEvent =
       new SearchEvent("search

event.keyword = textinput.text

myDispatcher.dispatchEvent(eve
```

```
<EventMap ...>

  <EventHandlers type="searchEvent">

    <ResponseAnnouncer .../>

  </EventHandlers>

</EventMap>
```

```
<mate:Dispatcher id="myDispatcher">
    <mate:ResponseHandler type="searchResultResponse"
                response="handleResult()" .../>
</mate:Dispatcher>
```

Receiving a response in the view

```
var event:SearchEvent =
       new SearchEvent("search

event.keyword = textinput.text

myDispatcher.dispatchEvent(eve
```

```
<EventMap ...>

  <EventHandlers type="searchEvent">

    <ResponseAnnouncer .../>

  </EventHandlers>

</EventMap>
```

```
<mate:Dispatcher id="myDispatcher">
   <mate:ResponseHandler type="searchResultResponse"
               response="handleResult()" .../>
</mate:Dispatcher>
```

Receiving a response in the view

```
var event:SearchEvent =
       new SearchEvent("searchEvent", true);

event.keyword = textinput.text;

myDispatcher.dispatchEvent(event);
```

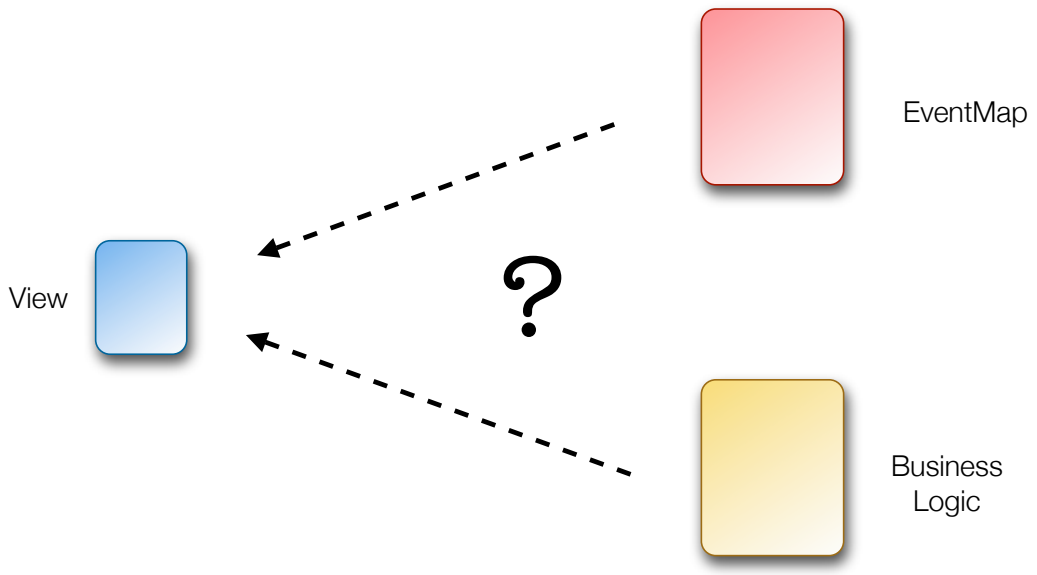
```
<mate:Dispatcher id="myDispatcher">
   <mate:ResponseHandler type="searchResultResponse"
               response="handleResult()" .../>
</mate:Dispatcher>
```

Receiving a response in the view

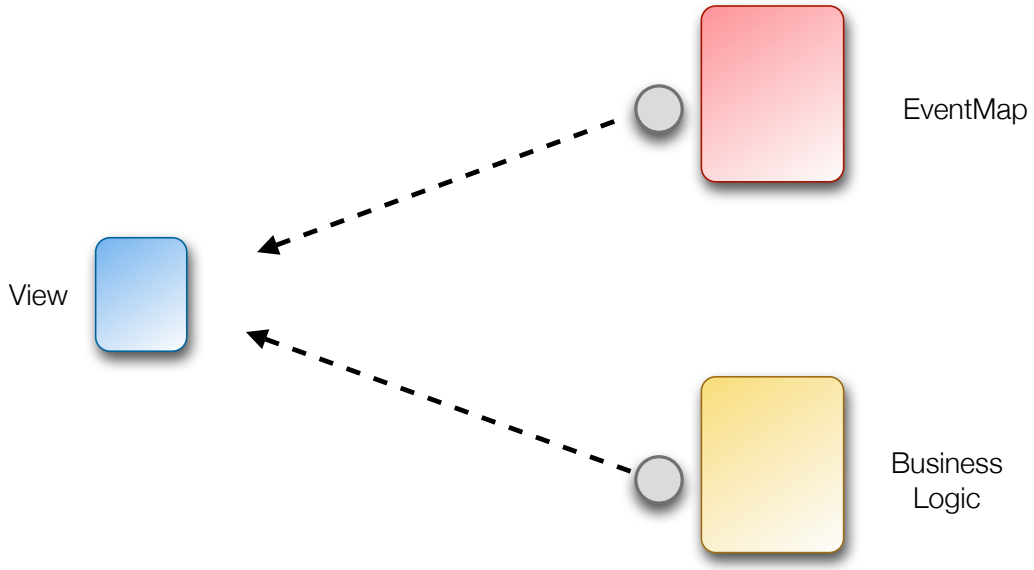
```
private function handleResult():void {
      //remove loading animation
}
```

```
<mate:Dispatcher id="myDispatcher">
   <mate:ResponseHandler type="searchResultResponse"
                response="handleResult()" .../>
</mate:Dispatcher>
```

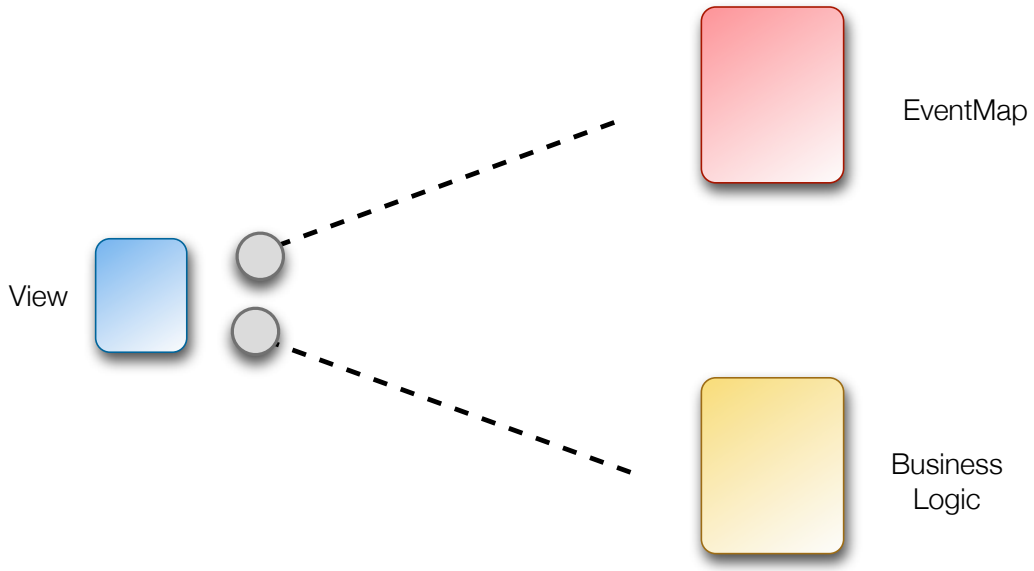
Can we communicate with views?



Can we communicate with views?

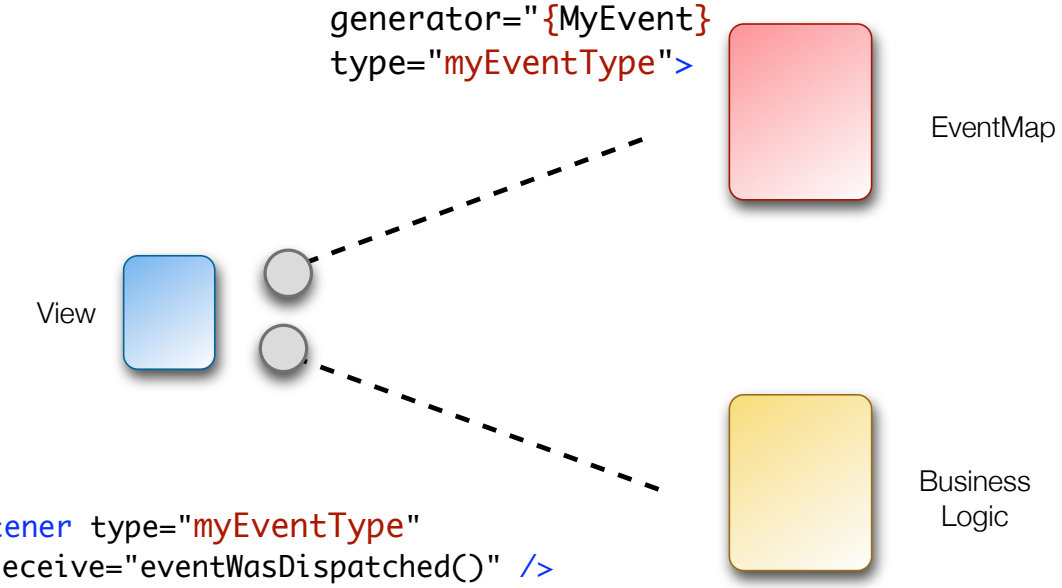


Can we communicate with views?



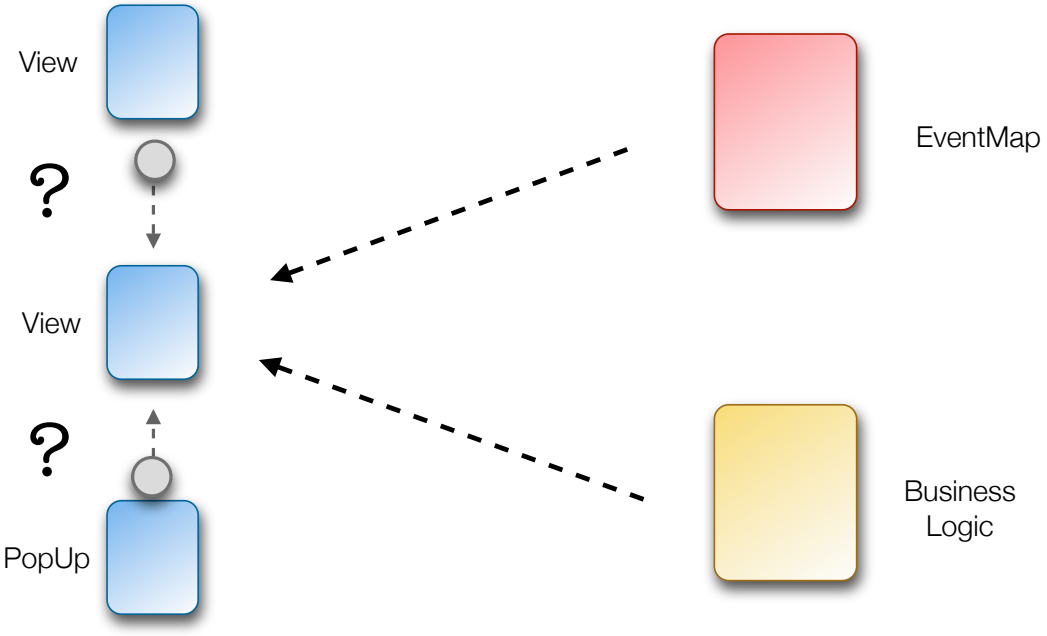
Can we communicate with views?

```
<EventAnnouncer
  generator="{MyEvent}
  type="myEventType">
```

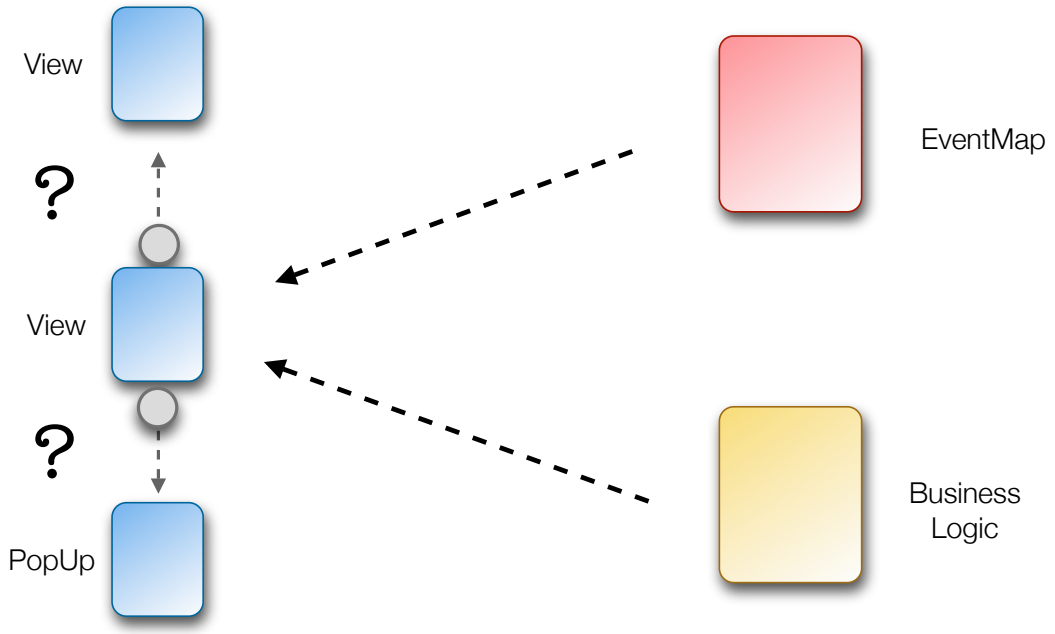



```
<Listener type="myEventType"
    receive="eventWasDispatched()" />
```

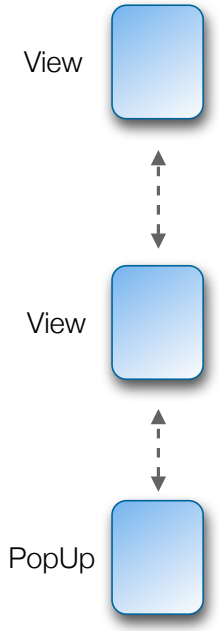
Can we communicate between views?



Can we communicate between views?



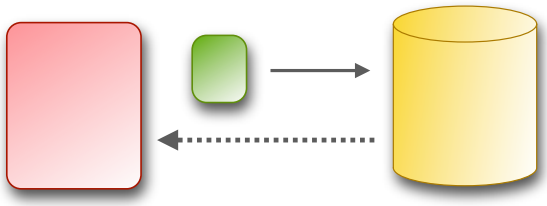
Can we communicate between views?




```
<Listener type="myEventType"
    receive="eventWasDispatched()" />
```

Dependency Injection

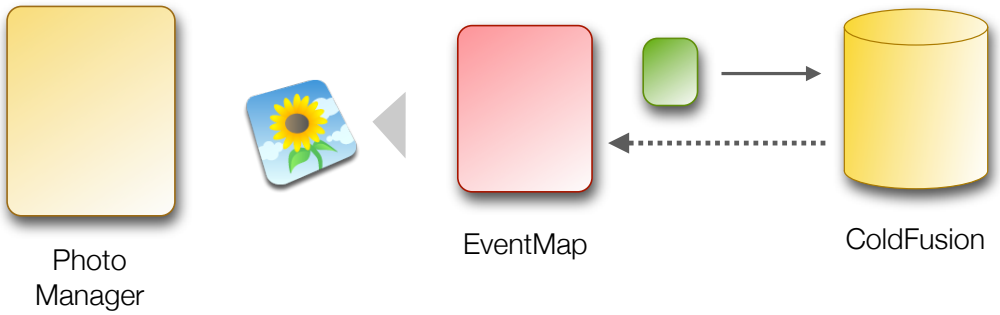
Photo
Manager



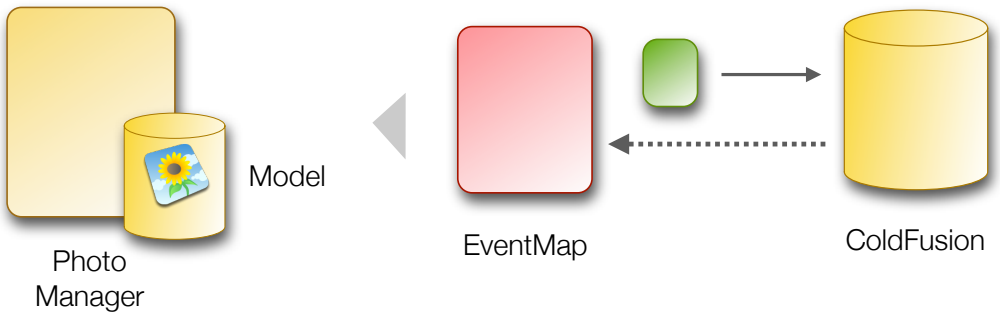
EventMap

ColdFusion

```
//Class com.example.PhotoManager
public function setPhoto(photo:Photo):void
{
  ...//store data
}
```



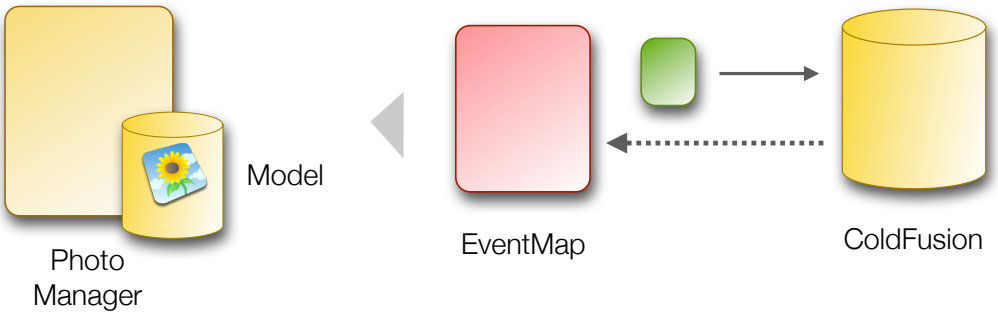

```
<MethodInvoker generator="{PhotoManager}"
  method="setPhoto" arguments="{resultObject}"
  cache='true' />
```



How can the view get this photo?

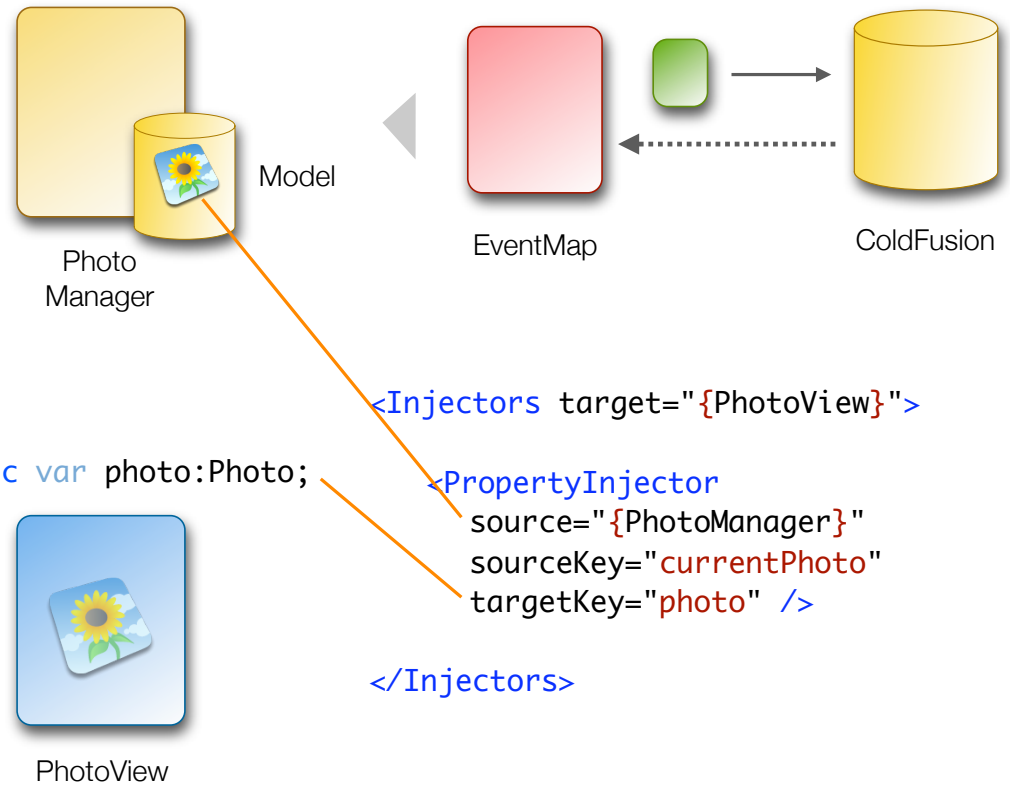
without:

- Direct reference to model
  (ie: ModelLocator.getInstance())
- Supplying it from parent view



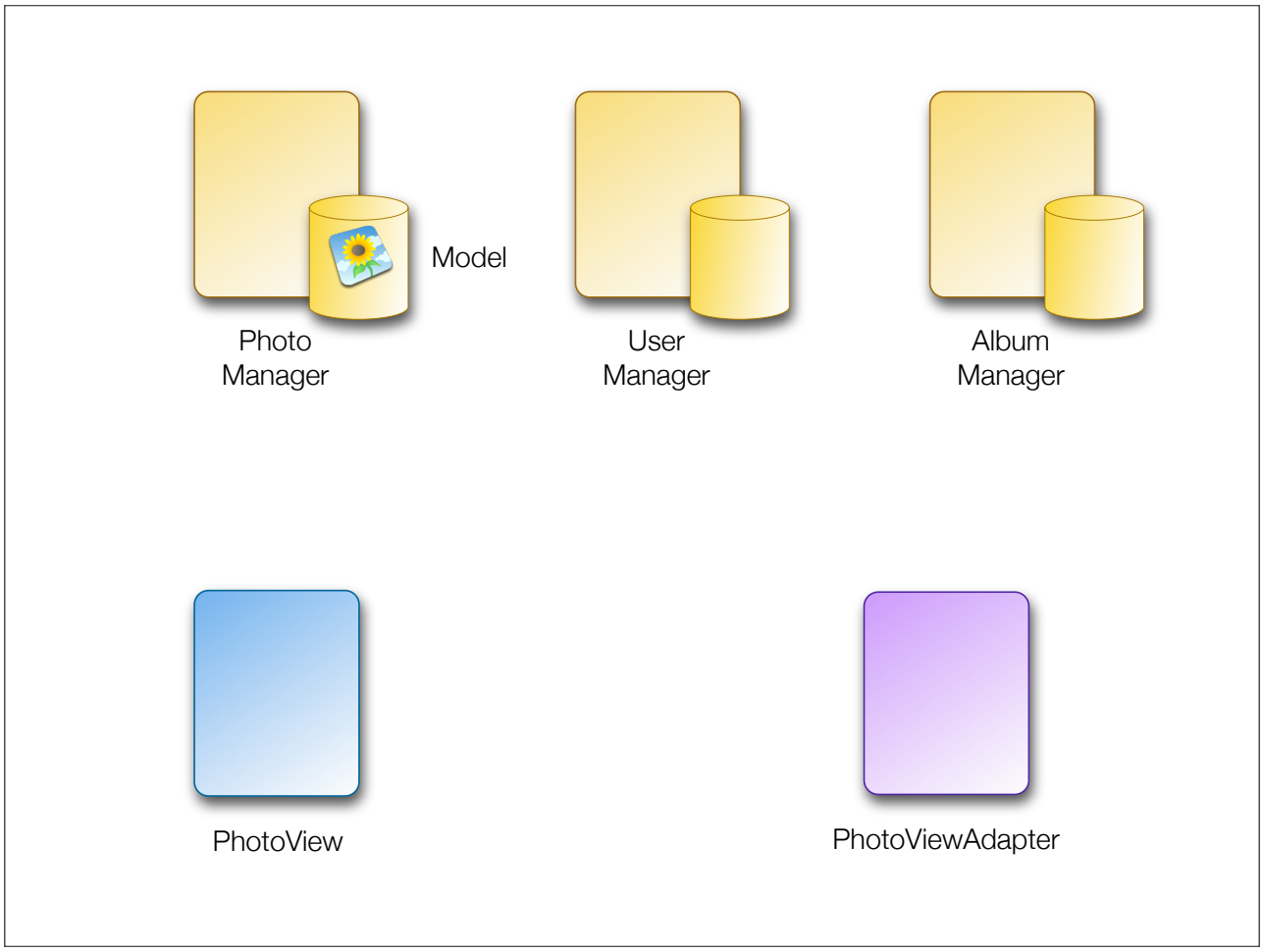

```
public var photo:Photo;
```

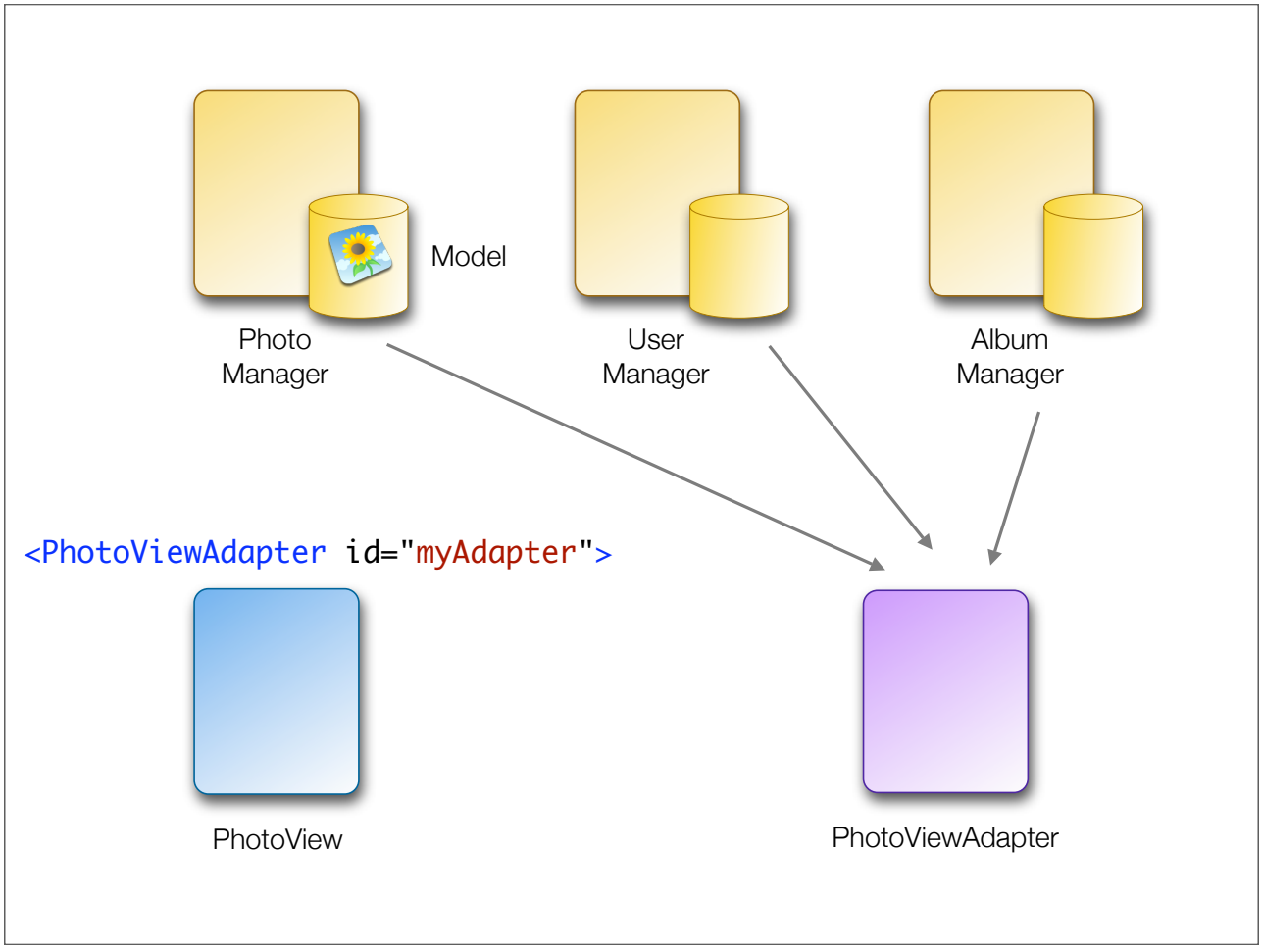
PhotoView

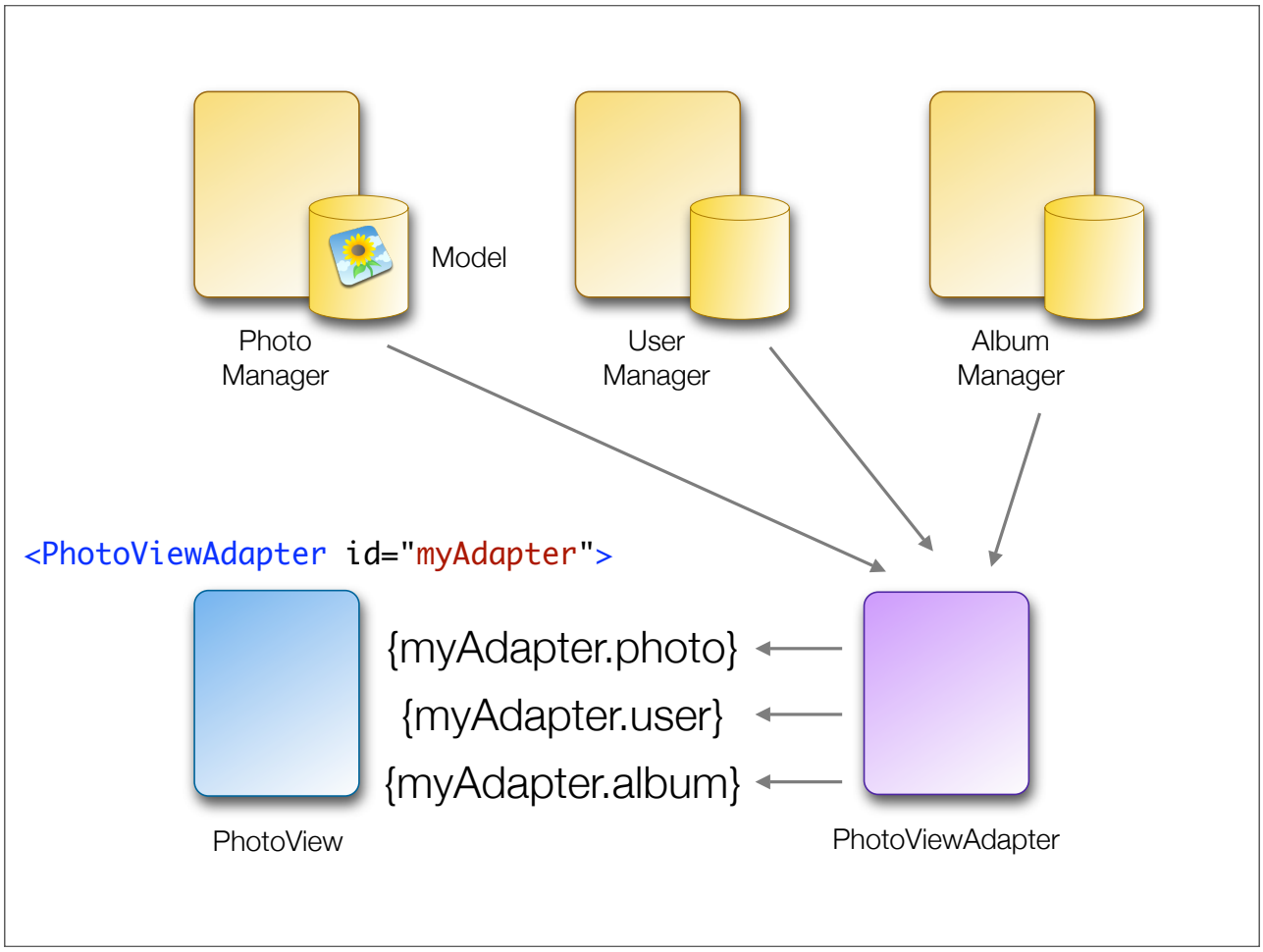


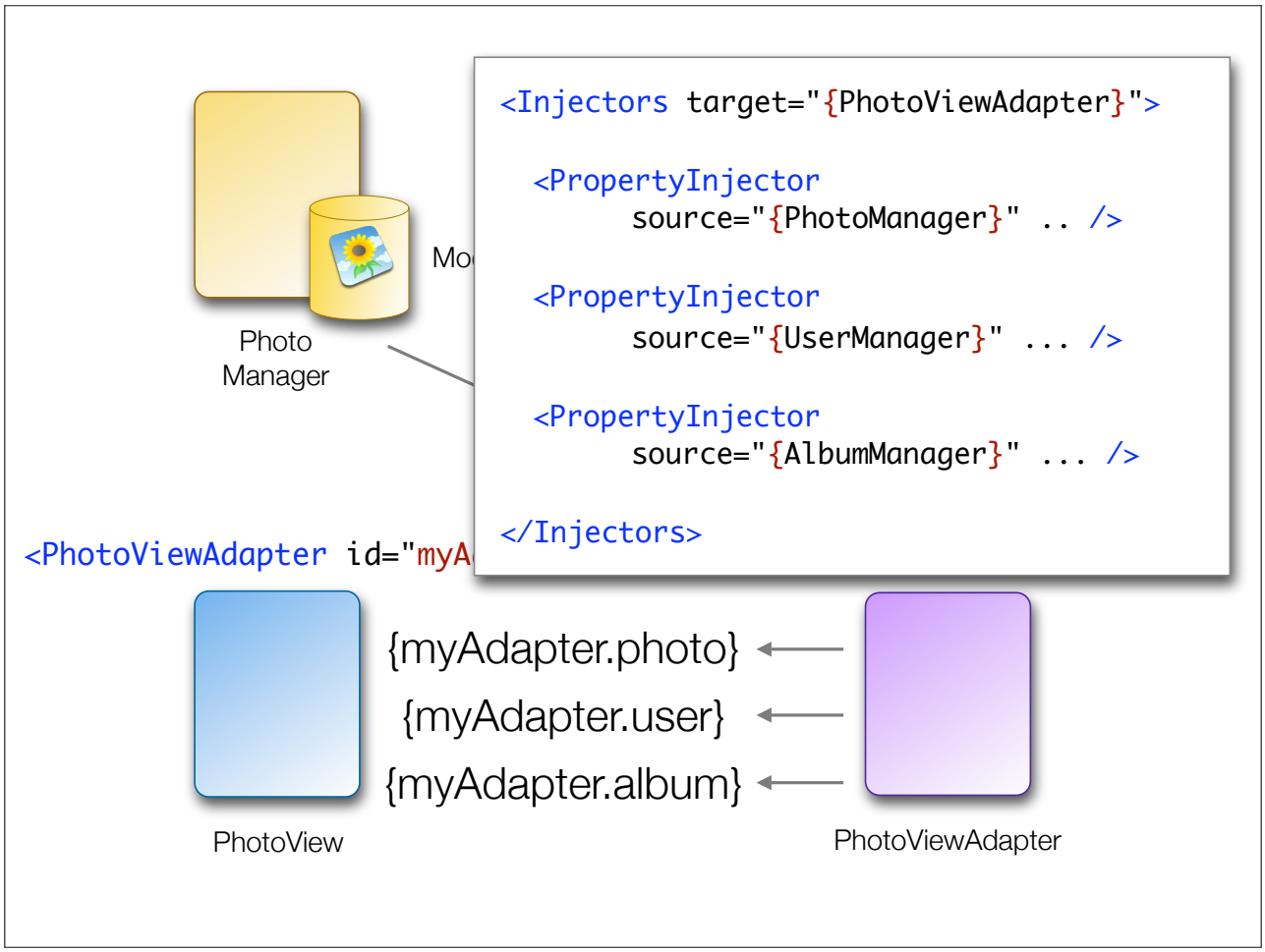

```
<Injectors target="{PhotoView}">

public var photo:Photo;
    <PropertyInjector
    source="{PhotoManager}"
    sourceKey="currentPhoto"
    targetKey="photo" />

</Injectors>
```


PhotoView


Model
Photo
Manager
User
Manager
Album
Manager
PhotoView
PhotoViewAdapter


Model
Photo
Manager
User
Manager
Album
Manager
<PhotoViewAdapter id="myAdapter">
PhotoView
PhotoViewAdapter


Model
Photo Manager
User Manager
Album Manager
<PhotoViewAdapter id="myAdapter">
{myAdapter.photo}
{myAdapter.user}
{myAdapter.album}
PhotoView
PhotoViewAdapter


<Injectors target="{PhotoViewAdapter}">
<PropertyInjector source="{PhotoManager}" .. />
<PropertyInjector source="{UserManager}" ... />
<PropertyInjector source="{AlbumManager}" ... />
</Injectors>
Photo Manager
<PhotoViewAdapter id="myA
{myAdapter.photo}
{myAdapter.user}
{myAdapter.album}
PhotoView
PhotoViewAdapter

Debugging

How to debug the event map 

Debugging a sequence

```
<EventMap ...>

    <Debugger level="{LogEventLevel.ALL}" />

    <EventHandlers debug="true"
            type="{PhotoEvent.SEARCH}">

        ... actions to perform ...
        <MethodInvoker generator="{FlickrHelper}" ...>

        <HTTPServiceInvoker .....>


    </EventHandlers>

</EventMap>
```

Debugging a sequence

```
<EventMap ...>

    <Debugger level="{LogEventLevel.ALL}" />

    <EventHandlers debug="true"
                   type="{PhotoEvent.SEARCH}">

        ... actions to perform ...
        <MethodInvoker generator="{FlickrHelper}" ...>

        <HTTPServiceInvoker .....>


    </EventHandlers>

</EventMap>
```

Debugging a sequence

```
<EventMap ...>

    <Debugger level="{LogEventLevel.ALL}" />

    <EventHandlers debug="true"
                   type="{PhotoEvent.SEARCH}">

        ... actions to perform ...
        <MethodInvoker generator="{FlickrHelper}" ...>

        <HTTPServiceInvoker .....>


    </EventHandlers>

</EventMap>
```

Debugging output

```
<EventHandlers (started)
type="PhotoEvent.SEARCH" (searchPhotoEvent)   priority="0">

    <MethodInvoker cache="true"
        arguments="[ arg1, null, 1 ]"
        method="getSearchUrl"
        generator="[class FlickrHelper]"/>

    <HTTPServiceInvoker   resultFormat="e4x"
        url="http://api.flickr.com/services/rest/?
        method=flickr.photos.search&api_key=..." method="GET" .../>

</EventHandlers (end)  type="PhotoEvent.SEARCH" (searchPhotoEvent)>
```

Benefits

Why you'll love it 

Highly decoupled

- Your classes do not extend from Mate
- Business logic is independent of framework
  - Can be tested & reused
- Business logic decoupled from events
- Business logic decoupled from services
- Events are normal Flash events
  - Can be reused

Get more

http://mate.asfusion.com