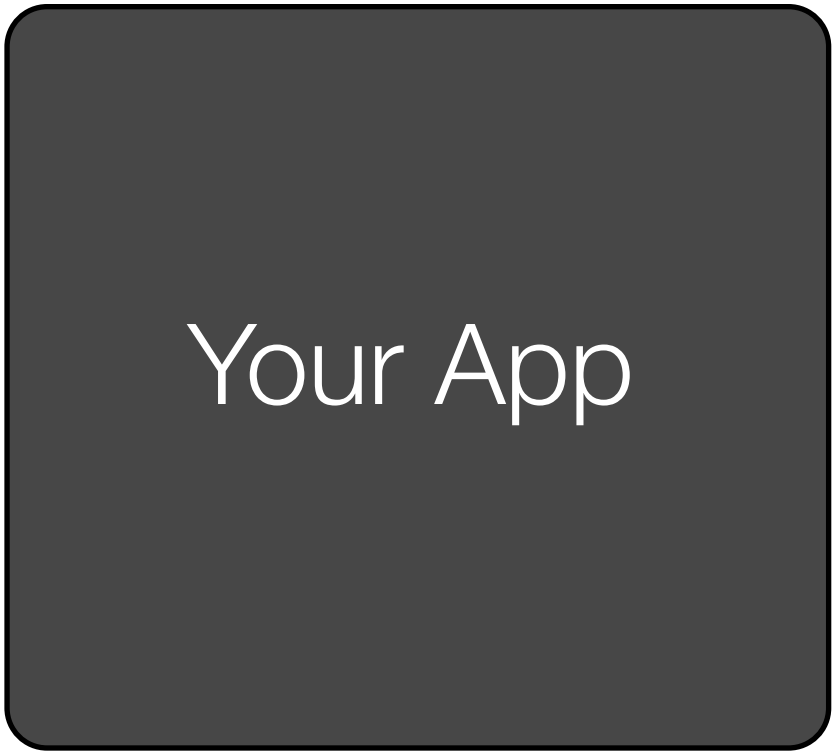
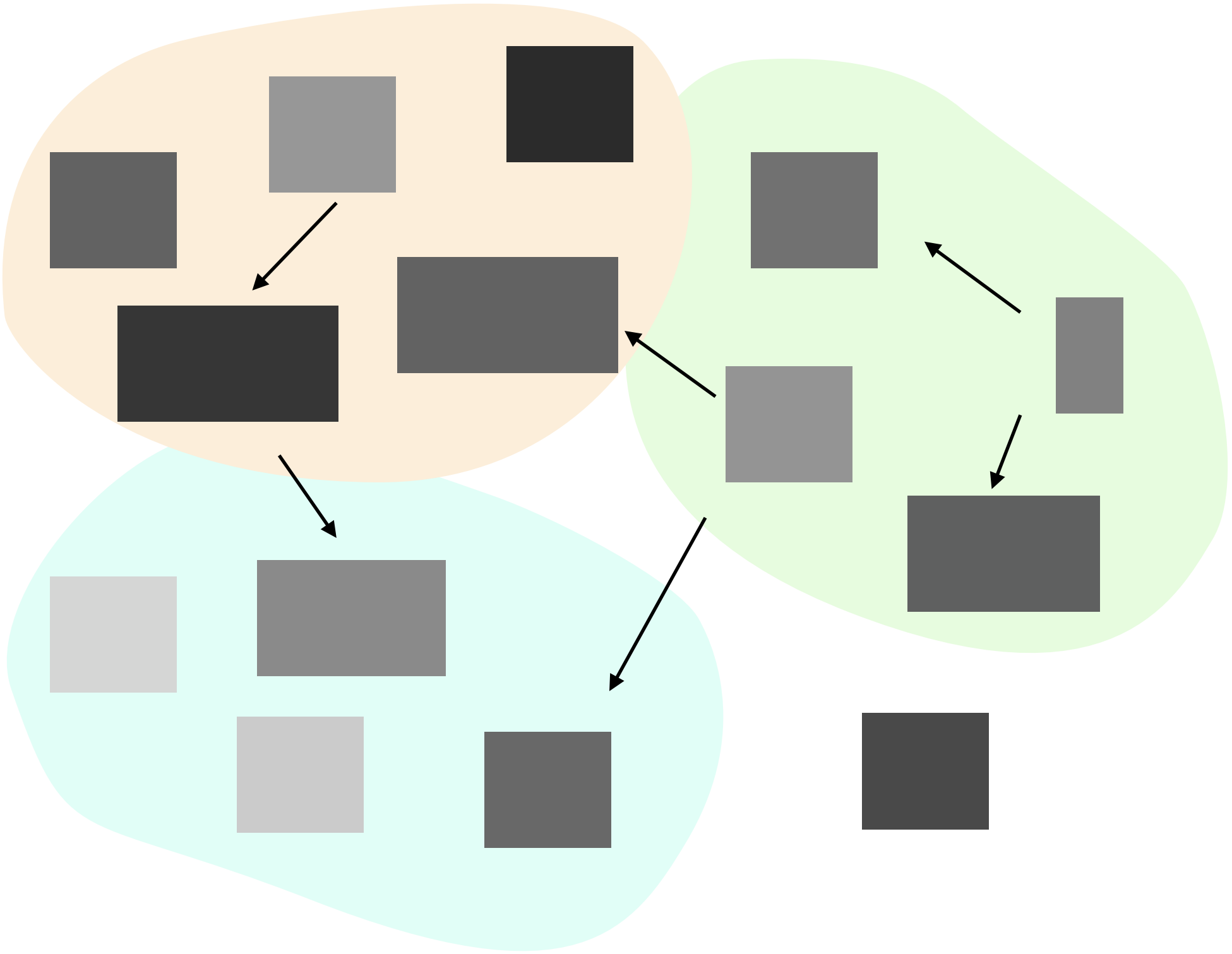
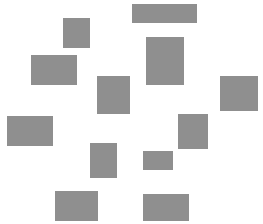
Breaking down your apps with

by


Your App



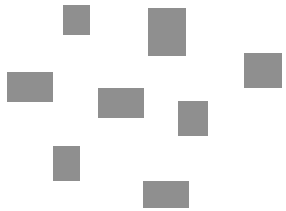


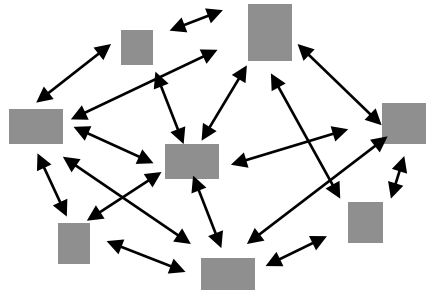
1

1 million

unmanageable

unmanageable





no relationships

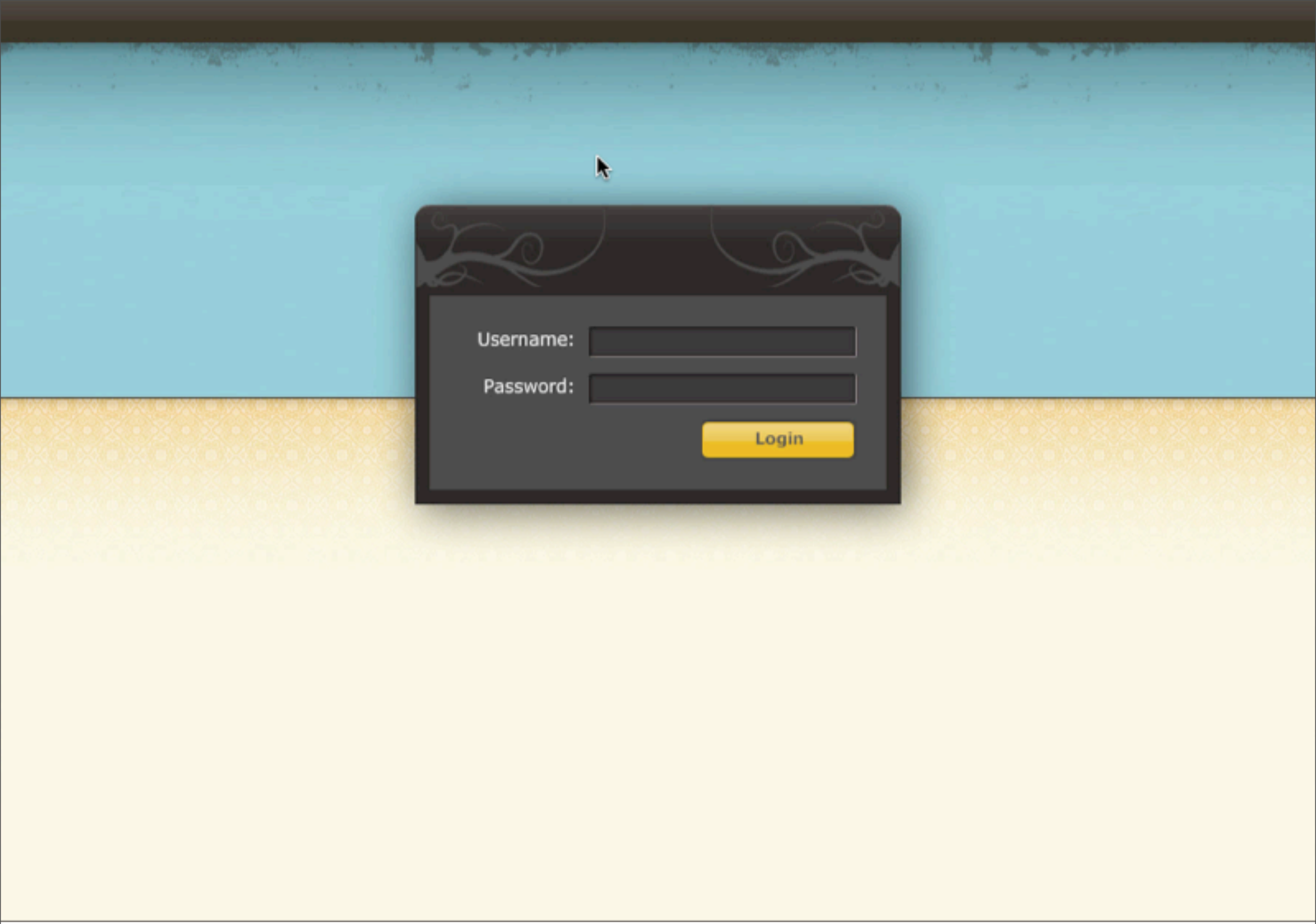
impossible
or monolithic

everything related to
everything

unmanageable

The sample app

Brownie-net


Username:
Password:
Login

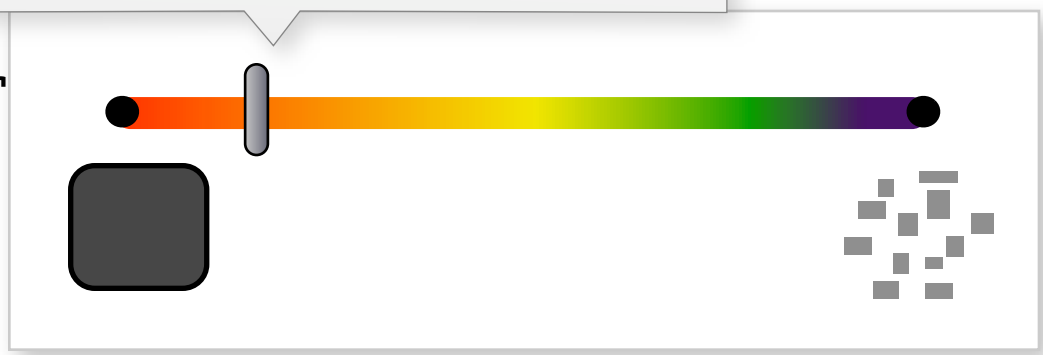
```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ViewStack>
                    <!-- Login view -->
                    <mx:Canvas>
                        <mx:Panel>
                            ......
                        </mx:Panel>
                    </mx:Canvas>

                    <!-- Contacts view -->
                    <mx:HBox>
                        <mx:Panel>
                            ....
                        </mx:Panel>
                    </mx:HBox>


                    <!-- Admin view -->
                    <mx:HBox>
                        <mx:Panel>
                            ....
                        </mx:Panel>
                    </mx:HBox>
    </mx:ViewStack>

</mx:Application>
```

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:ViewStack>
                    <!-- Login view -->

                     <views:Login />


                    <!-- Contacts view -->
                    <mx:HBox>
                        <mx:Panel>
                            ....
                        </mx:Panel>
                    </mx:HBox>


                    <!-- Admin view -->
                    <mx:HBox>
                        <mx:Panel>
                            ....
                        </mx:Panel>
                    </mx:HBox>
    </mx:ViewStack>

</mx:Application>
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
            [Bindable]
            private var contacts:ArrayCollection;

           private function handleResult(event:ResultEvent):void {
                contacts = new ArrayCollection(event.result);
           }
    </mx:Script>


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


     <mx:List id="list" dataProvider="{ contacts }" />


  </mx:Panel>
```

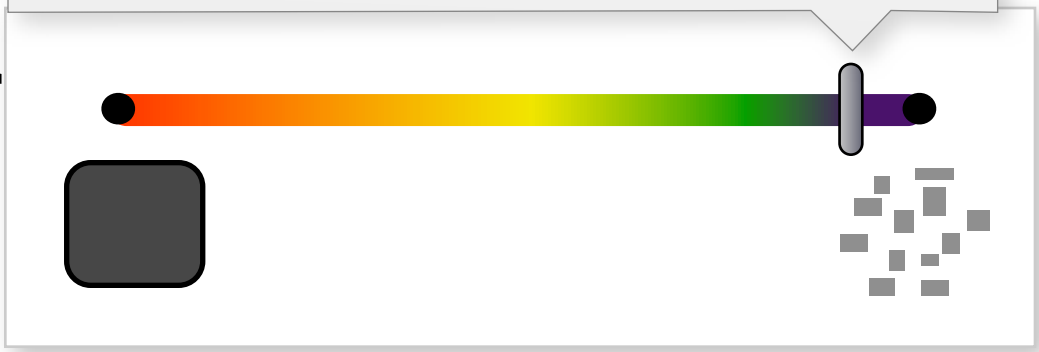
```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
            [Bindable]
            private var contacts:ArrayCollection;

           private function handleResult(event:ResultEvent):void {
               contacts = new ArrayCollection(event.result);
           }
    </mx:Script>


     <mx:RemoteObject id="service"
           result="handleResu
            fault="handleFaul

     <mx:List id="list" dataProvider


</mx:Panel>
```

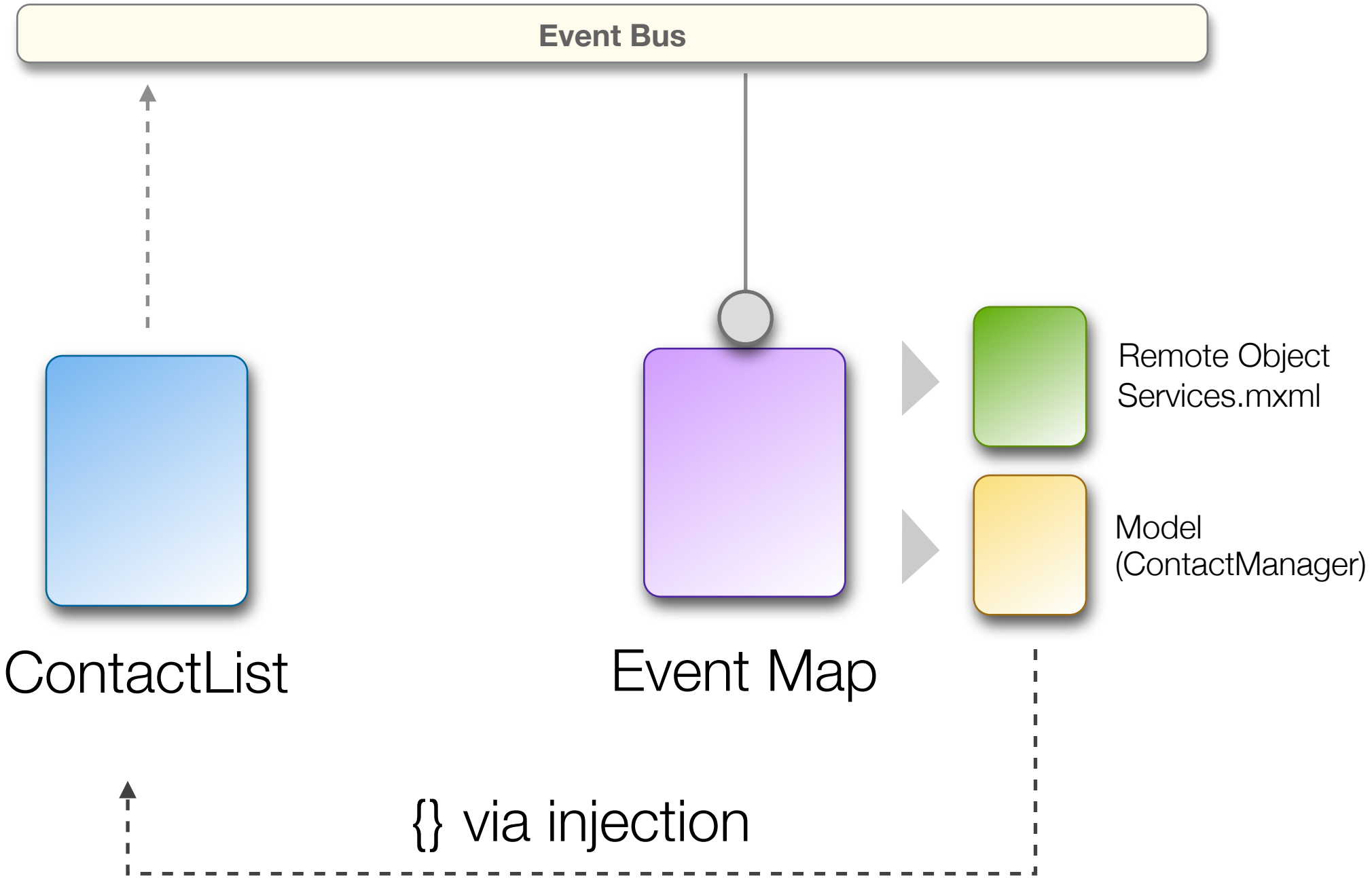
Too many responsibilities

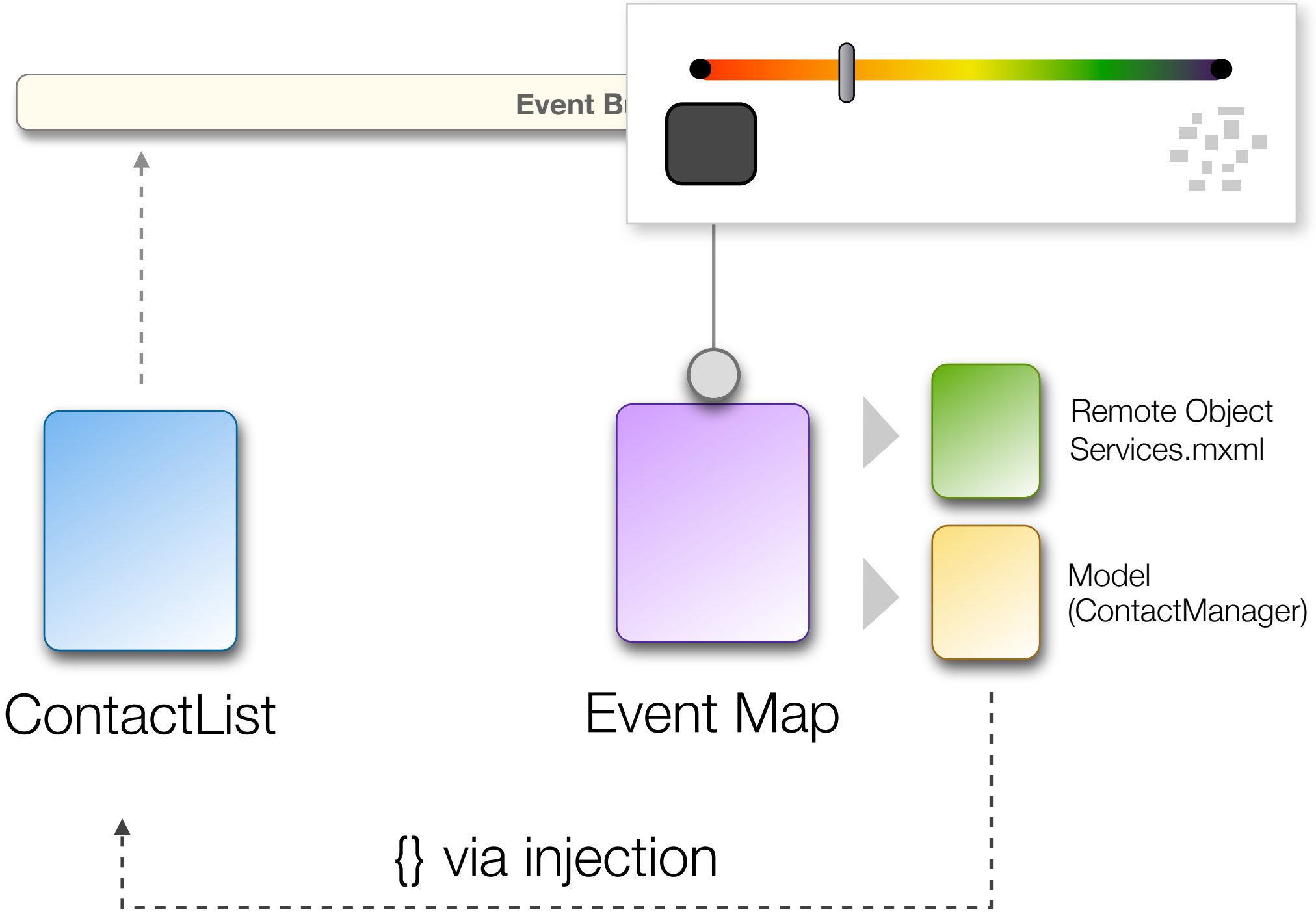


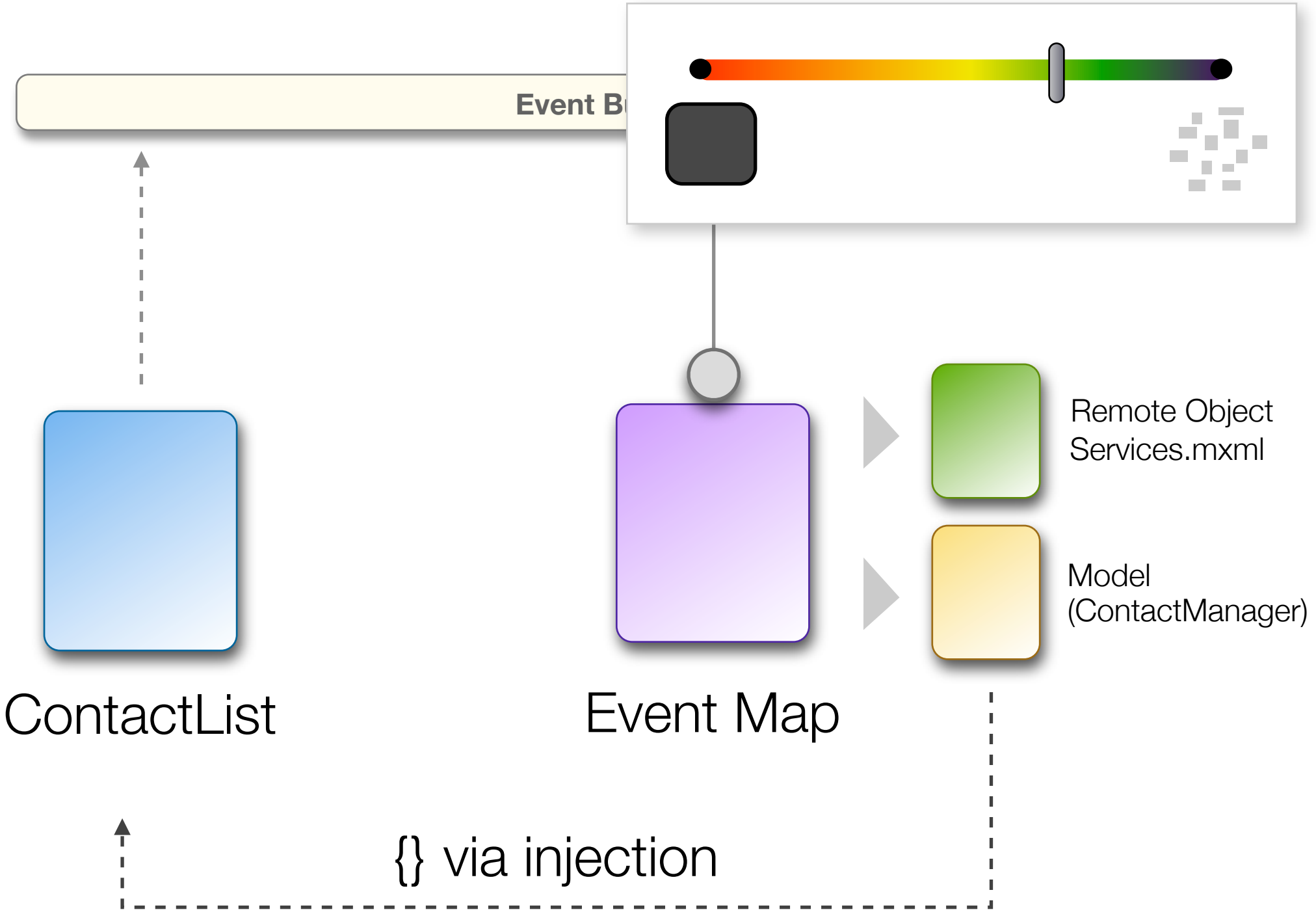
```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>
            [Bindable]
            private var contacts:ArrayCollection;

           private function handleResult(event:ResultEvent):void {
                contacts = new ArrayCollection(event.result);
           }

    </mx:Script>


      <mx:RemoteObject id="service"
            result="handleResult(eve
             fault="handleFault(even


     <mx:List id="list" dataProvider


  </mx:Panel>
```

Business logic, data and services are too distributed =
Can't be shared
Can't easily change




Event Bus
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
Event Map
{} via injection


Event B
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
Event Map
{} via injection


Event B
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
Event Map
{} via injection

View

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>

          [Bindable]
          public var contacts:ArrayCollection;









    </mx:Script>

    <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />

</mx:Panel>
```

View

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>

        [Bindable]
        public var contacts:ArrayCollection;

    </mx:Script>

    <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />

</mx:Panel>
```

Data gets injected

View

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>

        [Bindable]
        public var contacts:ArrayCollection;

        private function listChangeHandler():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.contact = list.selectedItem as Contact;
            dispatchEvent(event);
        }
    </mx:Script>

    <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />

</mx:Panel>
```

View

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>

        [Bindable]
        public var contacts:ArrayCollection;

        private function listChangeHandler():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.contact = list.selectedItem as Contact;
            dispatchEvent(event);
        }
    </mx:Script>

    <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />

</mx:Panel>
```

View

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>

        [Bindable]
        public var contacts:ArrayCollection;

        private function listChangeHandler():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.contact = list.selectedItem as Contact;
            dispatchEvent(event);
        }
    </mx:Script>

    <mx:List id="list" dataProvider="{co

</mx:Panel>
```

View

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

    <mx:Script>

        [Bindable]
        public var contacts:ArrayCollection;

        private function listChangeHandler():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);

            event.contact = 
            dispatchEvent(eve
        }
    </mx:Script>

    <mx:List id="list" dataProvider="{co

</mx:Panel>
```

The map bridges the model and the view

Model

```
public class ContactManager extends EventDispatcher
{
    private var _contacts:Array;
    [Bindable(Event="contactsChange")]
    public function get contacts():Array
    {
        return _contacts;
    }

    public function storeContacts( contacts:Array ):void
    {
        _contacts = contacts;
        dispatchEvent( new Event( "contactsChange" ) );
    }
}
```

Model

```
public class ContactManager extends EventDispatcher
{
    private var _contacts:Array;
    [Bindable(Event="contactsChange")]
    public function get contacts():Array
    {
        return _contacts;
    }

    public function storeContacts( contacts:Array ):void
    {
        _contacts = contacts;
        dispatchEvent( new Event( "contactsChange" ) );
    }
}
```

Model

```
public class ContactManager extends EventDispatcher
{
    private var _contacts:Array;
    [Bindable(Event="contactsChange")]
    public function get contacts():Array
    {
        return _contacts;
    }
    public function storeContacts( contacts:Array ):void
    {
        _contacts = contacts;
        dispatchEvent( new Event( "contactsChange" ) );
    }
}
```

Data

```
private var _contacts:Array;
[Bindable(Event="contactsChange")]
public function get contacts():Array
{
    return _contacts;
}
```

Business Logic

```
public function storeContacts( contacts:Array ):void
{
    _contacts = contacts;
    dispatchEvent( new Event( "contactsChange" ) );
}
```

Data

```
private var _contacts:Array;
[Bindable(Event="contactsCha
public function get contacts():Array
{
    return _contacts;
}
```

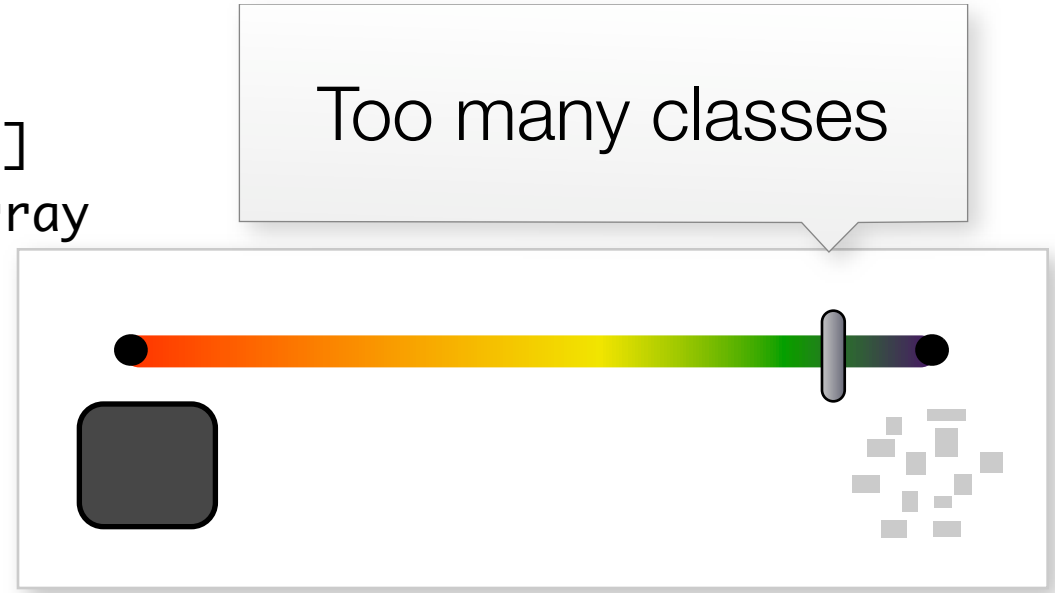
Monolithic model

Business Logic

```
public function storeContacts( contacts:Array ):void
{
    _contacts = contacts;
    dispatchEvent( new Event( "contactsChange" ) );
}
```

Data

```
private var _contacts:Array;
[Bindable(Event="contactsChange")]
public function get contacts():Array
{
    return _contacts;
}
```

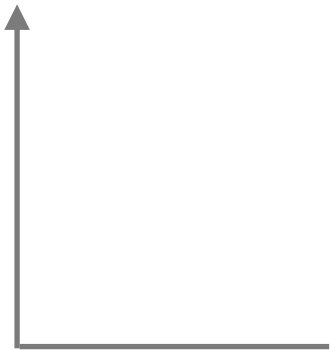


Business Logic

```
public function storeContacts( contacts:Array ):void
{
    _contacts = contacts;
    dispatchEvent( new Event( "contactsChange" ) );
}
```

Data

```
private var _contacts:Array;
[Bindable(Event="contactsChange")]
public function get contacts():Array
{
    return _contacts;
}
```



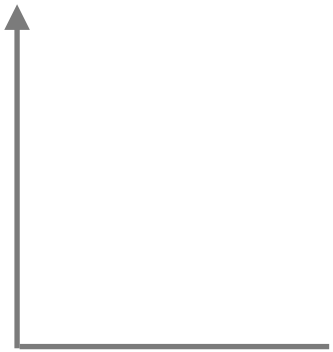
Needs reference to model

Business Logic

```
public function storeContacts( contacts:Array ):void
{
    _contacts = contacts;
    dispatchEvent( new Event( "contactsChange" ) );
}
```

Data

```
private var _contacts:Array;
[Bindable(Event="contactsChange")]
public function get contacts():Array
{
    return _contacts;
}
```



Needs reference to model

Business Logic

```
public function storeContacts( contacts:Array ):void
{
    Singleton.getInstance().contacts = contacts;
}
```

Data

```
private var _contacts:Array;
[Bindable(Event="contactsChange")]
public function get contacts():Array
{
    return _contacts;
}
```

Too many references to the model

Needs reference to model

Business Logic

```
public function storeContacts( contacts:Array ):void
{
    Singleton.getInstance().contacts = contacts;
}
```

Events

```
public class ContactEvent extends flash.events.Event {

        public static const GET_ALL:String = "getAllContactEvent";


        public function ContactEvent(type:String, bubbles:Boolean=true,
cancelable:Boolean=false)
        {
            super(type, bubbles, cancelable);
        }

}
```

Events

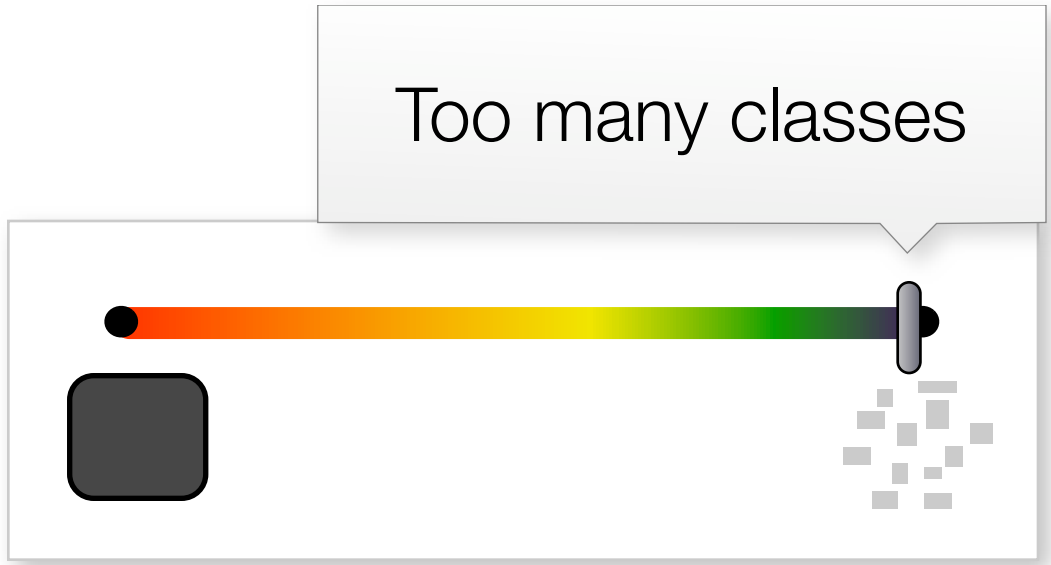
```
public class ContactEvent extends flash.events.Event {

        public static const GET_ALL:String = "getAllContactEvent";


        public function ContactEvent(type:String, bubbles:Boolean=true,
cancelable:Boolean=false)
        {
            super(type, bubbles, cancelable);
        }

}
```

Events

```
public class ContactEvent extends flash.events.Event {

        public static const GET_ALL:String = "getAllContactEvent";


        public function ContactEvent(type:String, bubbles:Boolean=true,
cancelable:Boolean=false)
        {
            super(type, bubbles, cancelable);
        }

}
```

Events

```
public class ContactEvent extends flash.events.Event {

        public static const GET_ALL:String = "getAllContactEvent";


        public function ContactEvent(type:String, bubbles:Boolean=true,
cancelable:Boolean=false)
        {
            super(type, bubbles, cancelable);
        }

}
```



Events

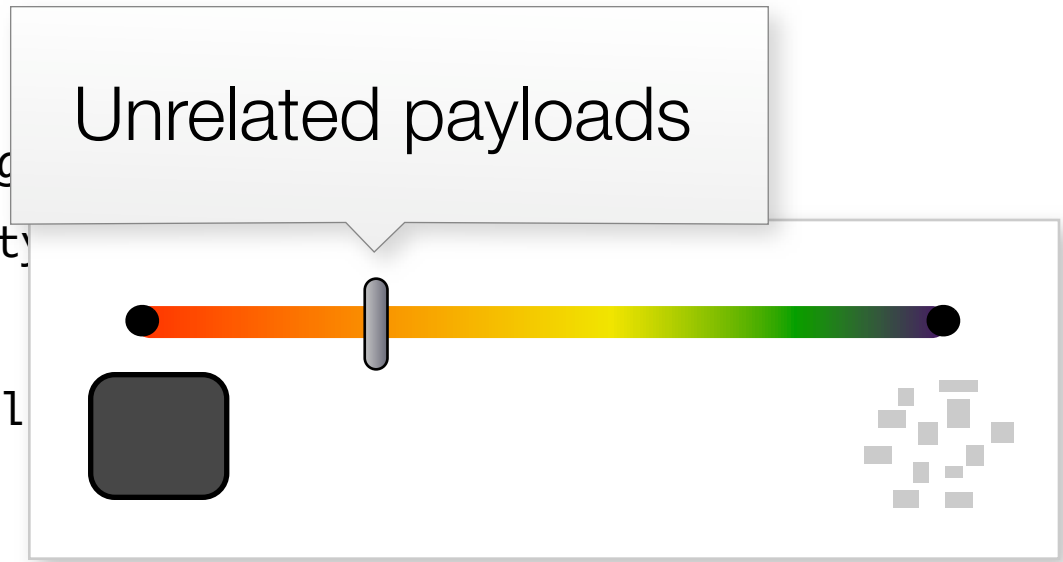
```
public class ContactEvent extends flash.events.Event {

        public static const ADD:String = "addContactEvent";
        public static const REMOVE:String = "removeContactEvent";
        public static const UPDATE:String = "updateContactEvent";

        public var contact:Contact;


        public function ContactEvent(type:String, bubbles:Boolean=true,
cancelable:Boolean=false)
        {
            super(type, bubbles, cancel
        }

}
```

Events

```
public class ContactEvent extends flash.events.Event {

        public static const ADD:String = "addContactEvent";
        public static const REMOVE:String = "removeContactEvent";
        public static const UPDATE:String = "updateContactEvent";

        public var contact:Contact;


        public function ContactEvent(type:String, bubbles:Boolean=true,
cancelable:Boolean=false)
        {
            super(type, bubbles, cancel
        }

}
```

Events

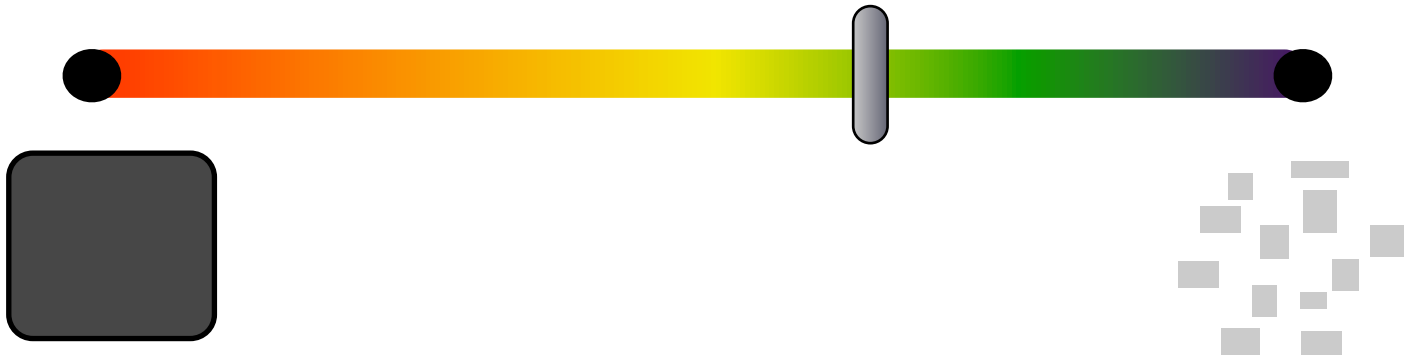
```
public class ContactEvent extends flash.events.Event {

        public static const ADD:String = "addContactEvent";
        public static const REMOVE:String = "removeContactEvent";
        public static const UPDATE:String = "updateContactEvent";
        public static const SEARCH:String = "searchContactEvent";
        public static const SORT_BY:String = "sortByContactEvent";

        public var contact:Contact;
        public var keyword:String;
        public var sortCriteria:String
        public function ContactEvent(ty
cancelable:Boolean=false)
        {
            super(type, bubbles, cancel
        }

}
```



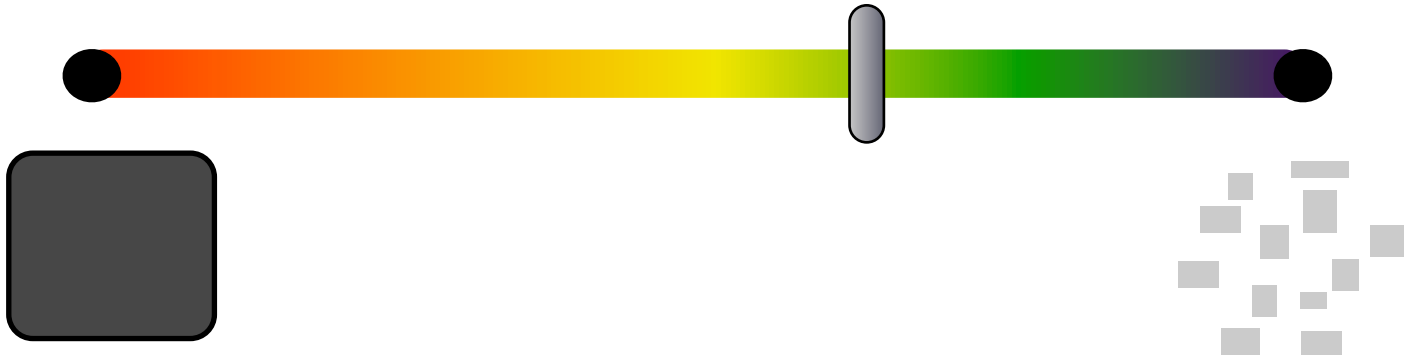
Event Map

```
<EventMap>

    <EventHandlers type="{ ContactEvent.GET_ALL }">
        <RemoteObjectInvoker instance="{ services.contacts }"
            method="getAll">
            <resultHandlers>
                <MethodInvoker generator="{ ContactManager }"
                    method="storeContacts" arguments="{ resultObject }"/>
            </resultHandlers>
        </RemoteObjectInvoker>
    </EventHandlers>

    <Injectors target="{ ContactList }">

        <PropertyInjector targetKey="contacts"
        source="{ ContactManager }"  sourceKey="contacts"/>

    </Injectors>

<EventMap>
```

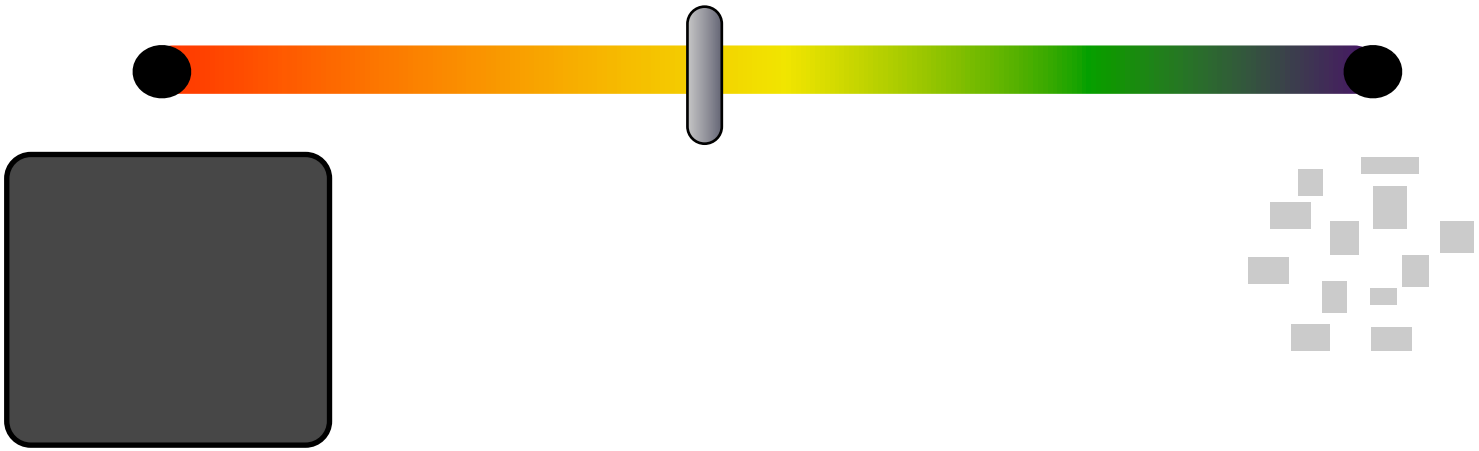
Mate’s solution



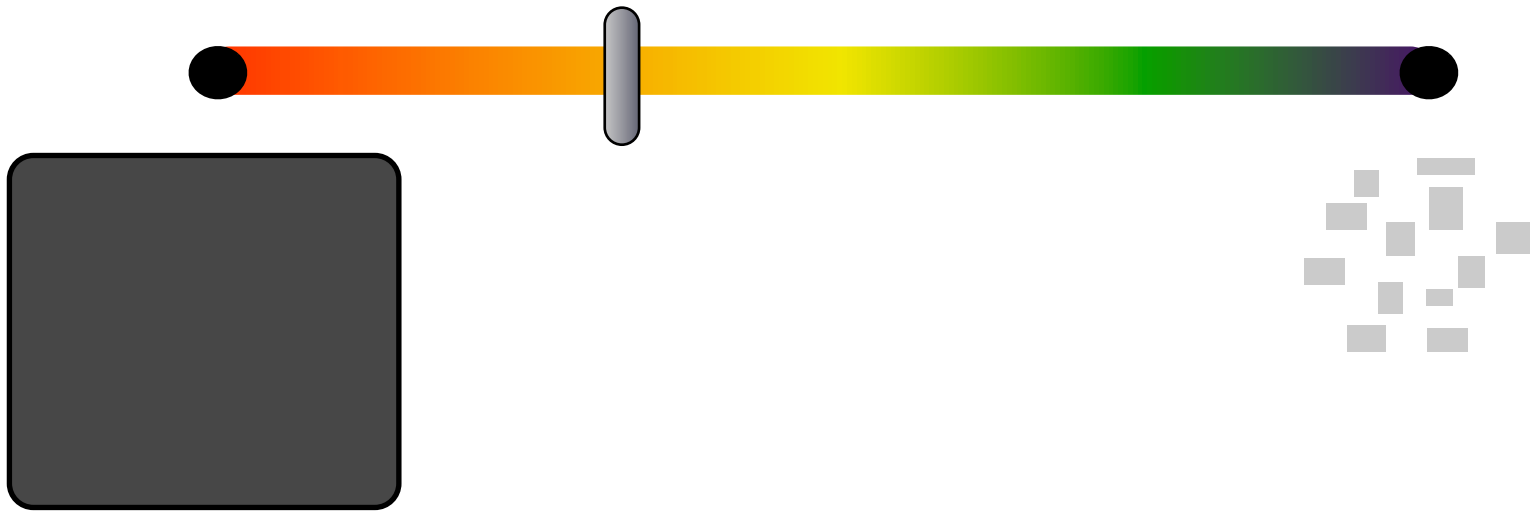
The problem



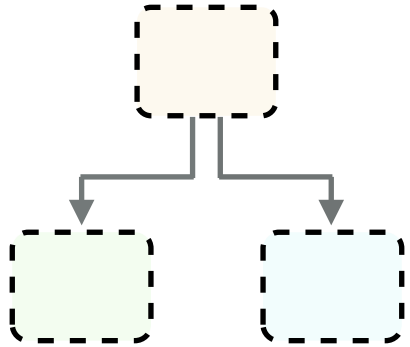
The problem



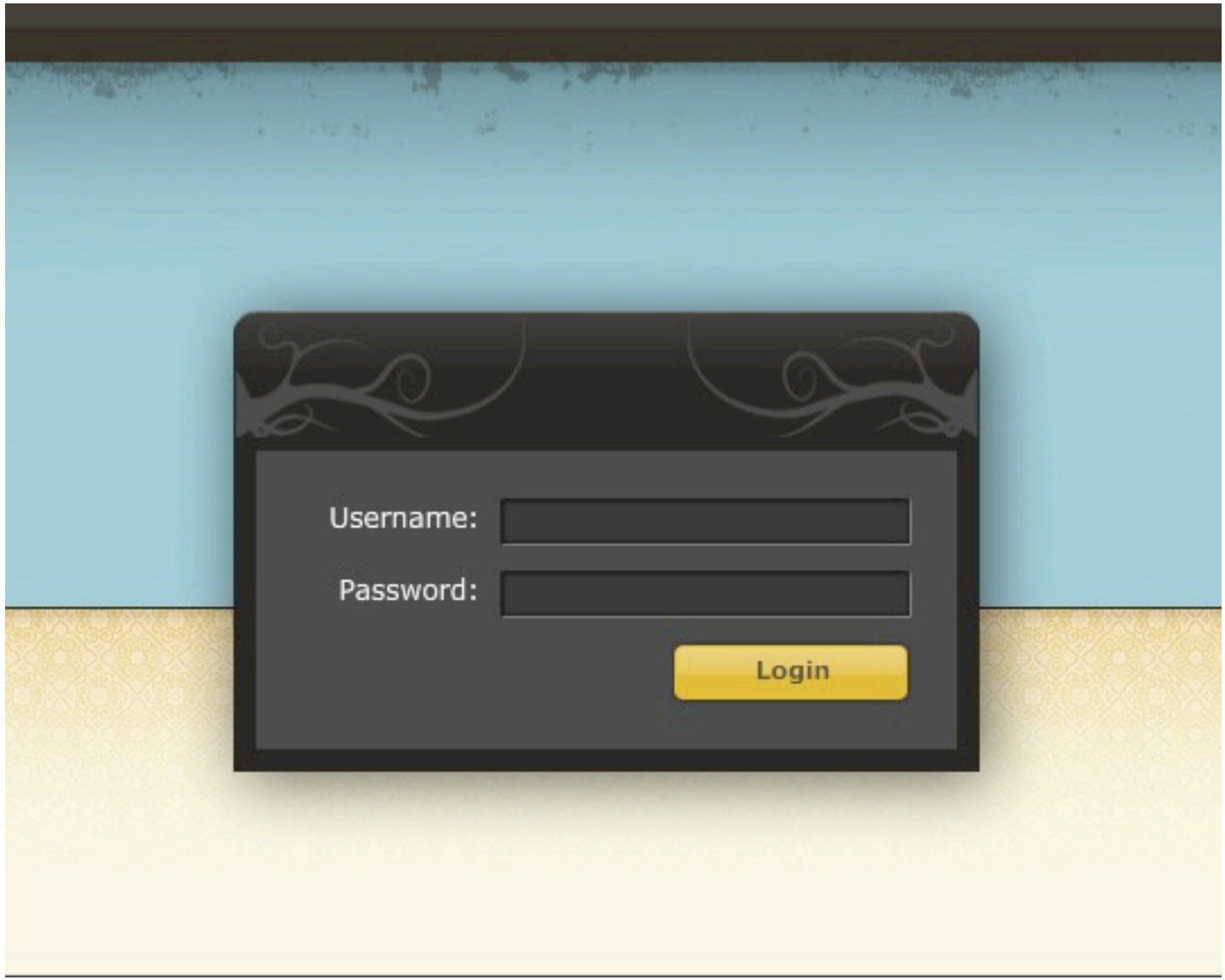
The problem

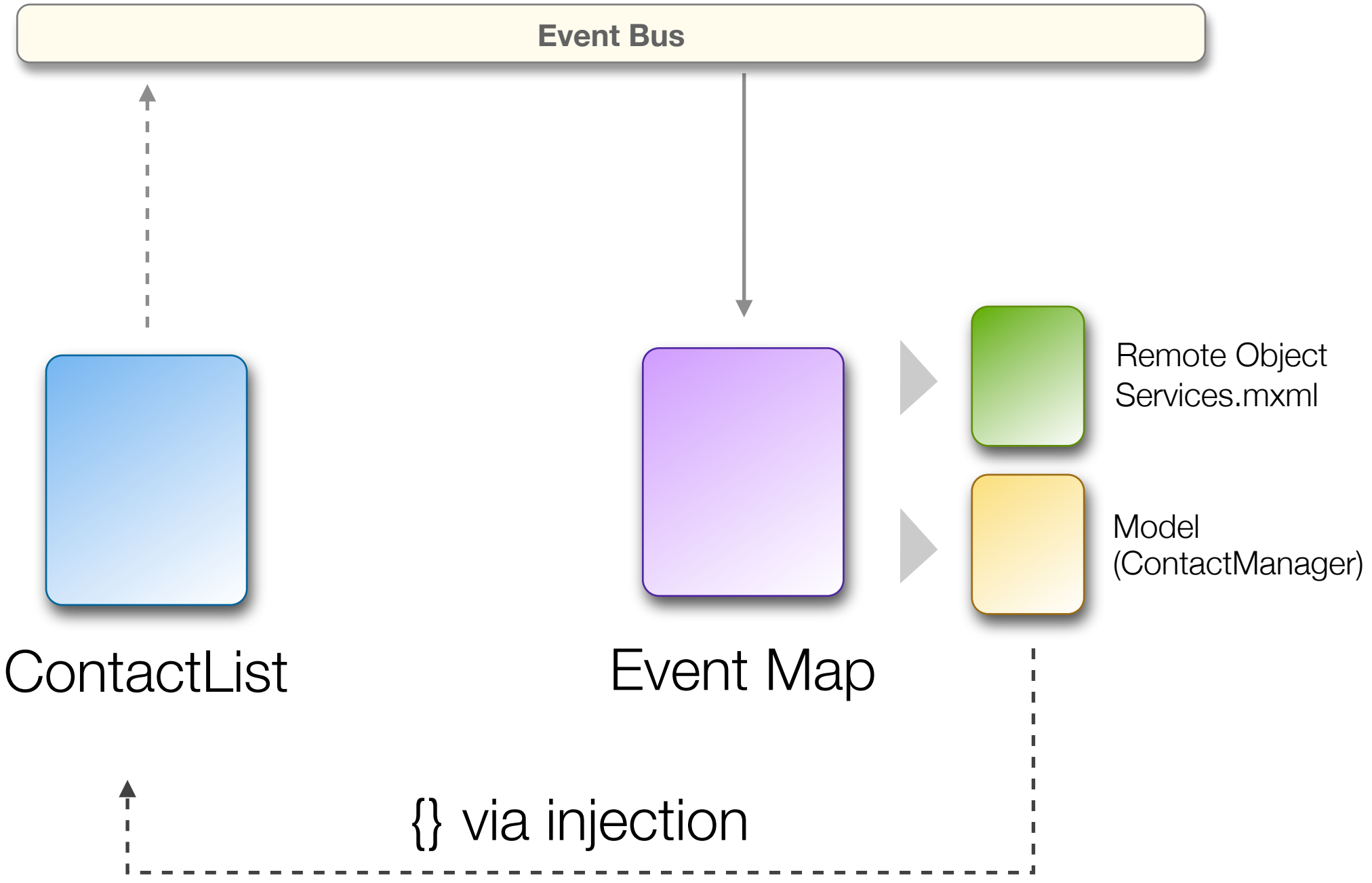


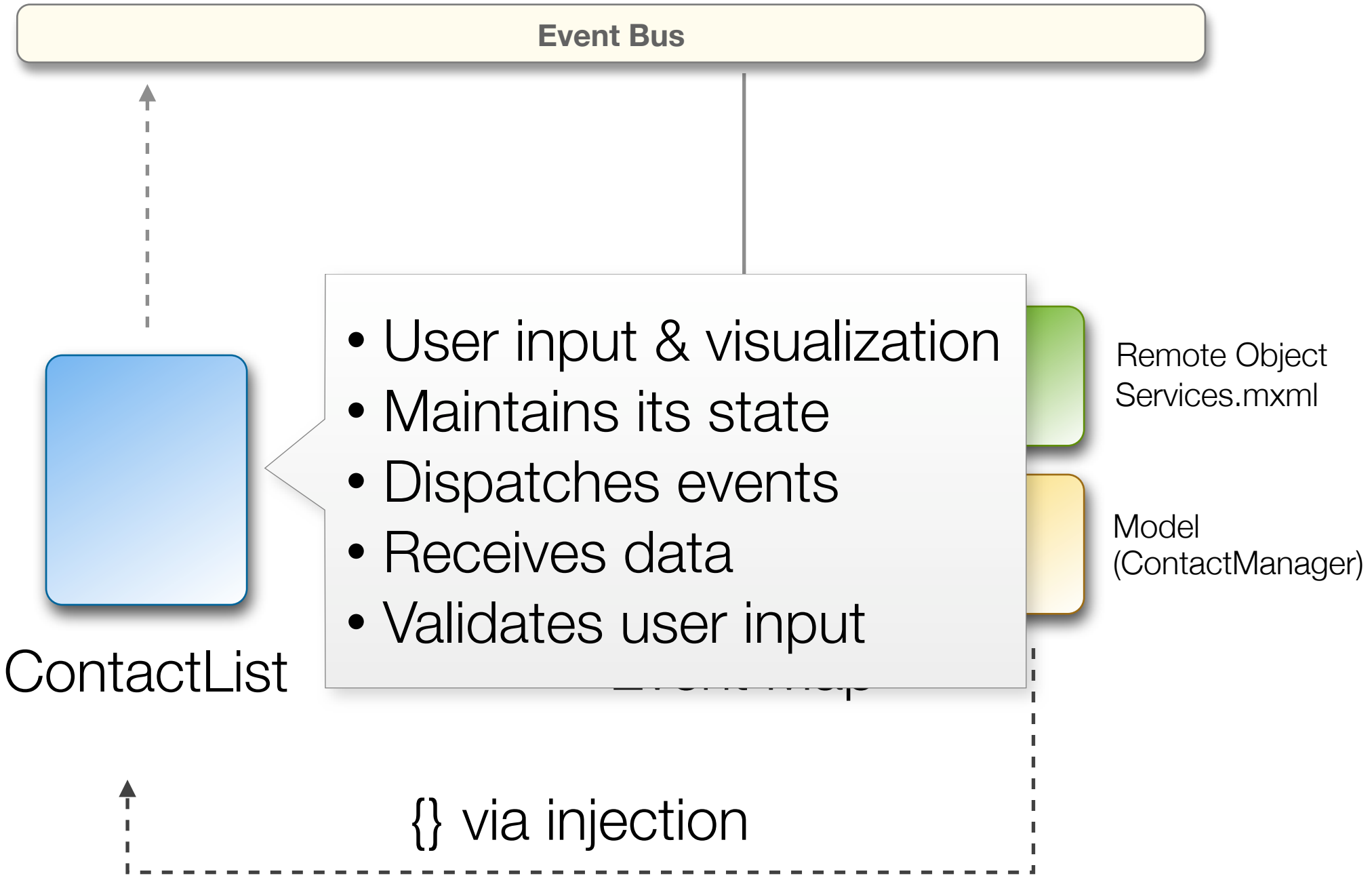
Helpful Patterns

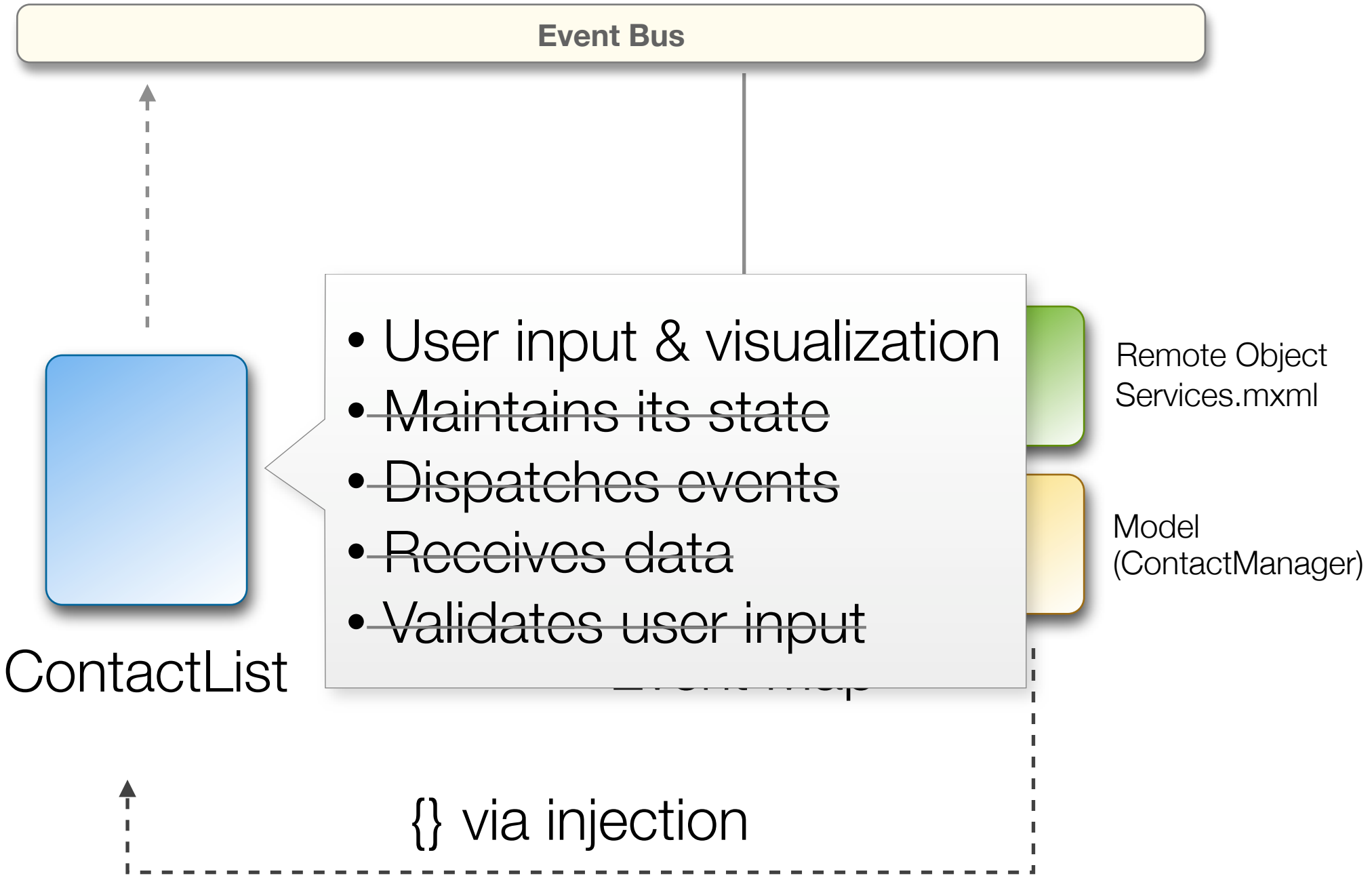


Presentation Model


Username:
Password:
Login


Event Bus
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
Event Map
{} via injection


Event Bus
• User input & visualization
• Maintains its state
• Dispatches events
• Receives data
• Validates user input
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
{} via injection


Event Bus
• User input & visualization
• Maintains its state
• Dispatches events
• Receives data
• Validates user input
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
{} via injection

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>

        [Bindable]
        public var contacts:Array;

        private function itemSelected():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.movie = list.selectedItem as Contact;
            dispatchEvent(event);
        }

        //logic to determine which state we should show

  </mx:Script>

   <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />
   <views:LoadingIndicator id="loader" />

    <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
    </mx:states>
</mx:Panel>
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>

        [Bindable]
        public var contacts:Array;

        private function itemSelected():void {
            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.movie = list.selectedItem as Contact;
            dispatchEvent(event);
        }

        //logic to determine which state we should show

    </mx:Script>

    <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />
    <views:LoadingIndicator id="loader" />

    <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
    </mx:states>
</mx:Panel>
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>

        [Bindable]
        public var contacts:Array;

        private function itemSelected():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.movie = list.selectedItem as Contact;
            dispatchEvent(event);
        }

        //logic to determine which state we should show

  </mx:Script>

   <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />
   <views:LoadingIndicator id="loader" />

    <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
    </mx:states>
</mx:Panel>
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>

        [Bindable]
        public var contacts:Array;

        private function itemSelected():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.movie = list.selectedItem as Contact;
            dispatchEvent(event);
        }

        //logic to determine which state we should show

  </mx:Script>

   <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />
   <views:LoadingIndicator id="loader" />

    <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
    </mx:states>
</mx:Panel>
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">
    <mx:Script>

  </mx:Script>

   <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />
   <views:LoadingIndicator id="loader" />

    <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
    </mx:states>
</mx:Panel>
```

```
[Bindable]
public var contacts:Array;

private function itemSelec

    var event:ContactEvent

    event.movie = list.sel
    dispatchEvent(event);
}

//logic to determine which
```

```
public class ContactListPresentationModel extends EventDispatcher {




    [Bindable]
    public var contacts:Array;

    private function itemSelected():void {

        var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                  true);
        event.movie = list.selectedItem as Contact;
        dispatchEvent(event);
    }

    //logic to determine which state we should show




}
```

```
public class ContactListPresentationModel extends EventDispatcher {

    public static const LOADING_STATE:String = "loadingState";
    public static const LOADED_STATE:String  = "loadedState";



    [Bindable]
    public var contacts:Array;

    private function itemSelected():void {

        var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                  true);
        event.movie = list.selectedItem as Contact;
        dispatchEvent(event);
    }

    //logic to determine which state we should show




}
```

```
public class ContactListPresentationModel extends EventDispatcher {

    public static const LOADING_STATE:String = "loadingState";
    public static const LOADED_STATE:String  = "loadedState";


    private var _state:String = LOADING_STATE;
    [Bindable(Event="stateChange")]
    public function get state():String {
        return _state;
    }



     [Bindable]
     public var contacts:Array;

     private function itemSelected():void {

         var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                                    true);
         event.movie = list.selectedItem as Contact;
         dispatchEvent(event);
     }

     //logic to determine which state we should show

}
```

```
public class ContactListPresentationModel extends EventDispatcher {

    public static const LOADING_STATE:String = "loadingState";
    public static const LOADED_STATE:String  = "loadedState";


    private var _state:String = LOADING_STATE;
    [Bindable(Event="stateChange")]
    public function get state():String {
        return _state;
    }

    //logic to determine which state we should show

    [Bindable]
    public var contacts:Array;

    private function itemSelected():void {

        var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                  true);
        event.movie = list.selectedItem as Contact;
        dispatchEvent(event);
    }



}
```

```
public class ContactListPresentationModel extends EventDispatcher {

    public static const LOADING_STATE:String = "loadingState";
    public static const LOADED_STATE:String  = "loadedState";


    private var _state:String = LOADING_STATE;
    [Bindable(Event="stateChange")]
    public function get state():String {
        return _state;
    }

    //logic to determine which state we should show

    private var _contacts:Array;
    [Bindable(Event="contactsChange")]
    public function get contacts():Array {
        return _contacts;
    }


    private function itemSelected():void {

        var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                  true);
        event.movie = list.selectedItem as Contact;
        dispatchEvent(event);
    }
}
```

```
public class ContactListPresentationModel extends EventDispatcher {

    public static const LOADING_STATE:String = "loadingState";
    public static const LOADED_STATE:String  = "loadedState";


    private var _state:String = LOADING_STATE;
    [Bindable(Event="stateChange")]
    public function get state():String {
        return _state;
    }



     private var _contacts:Array;
     [Bindable(Event="contactsChange")]
     public function get contacts():Array {
         return _contacts;
     }
     public function set contacts( value:Array ):void {
         if( value != null ) {
             _state = LOADED_STATE;
             dispatchEvent( new Event( "stateChange" ) );
         }

         _contacts = value;
         dispatchEvent( new Event( "contactsChange" ) );
     }
```

```
        public function get state():String {
            return _state;
        }



        private var _contacts:Array;
        [Bindable(Event="contactsChange")]
        public function get contacts():Array {
            return _contacts;
        }
        public function set contacts( value:Array ):void {
            if( value != null ) {
                _state = LOADED_STATE;
                dispatchEvent( new Event( "stateChange" ) );
            }

            _contacts = value;
            dispatchEvent( new Event( "contactsChange" ) );
        }
        private function itemSelected():void {

            var event:ContactEvent = new ContactEvent(ContactEvent.SELECTED,
                                                      true);
            event.movie = list.selectedItem as Contact;
            dispatchEvent(event);
        }
    }
```

```
        public function get state():String {
            return _state;
        }



        private var _contacts:Array;
        [Bindable(Event="contactsChange")]
        public function get contacts():Array {
            return _contacts;
        }
        public function set contacts( value:Array ):void {
            if( value != null ) {
                _state = LOADED_STATE;
                dispatchEvent( new Event( "stateChange" ) );
            }

            _contacts = value;
            dispatchEvent( new Event( "contactsChange" ) );
        }
        public function itemSelected( contact:Contact ):void {

            var event:ContactEvent = new ContactEvent( ContactEvent.CONTACT_SELECT );
                event.contact = contact;

            dispatcher.dispatchEvent(event);


    }
```

```
    public function set contacts( value:Array ):void {
        if( value != null ) {
            _state = LOADED_STATE;
            dispatchEvent( new Event( "stateChange" ) );
        }

        _contacts = value;
        dispatchEvent( new Event( "contactsChange" ) );
    }
    public function itemSelected( contact:Contact ):void {

        var event:ContactEvent = new ContactEvent( ContactEvent.CONTACT_SELECT );
            event.contact = contact;

        dispatcher.dispatchEvent(event);
    }
```

```
private var dispatcher:IEventDispatcher;
public function
ContactListPresentationModel( dispatcher:IEventDispatcher ):void {

        this.dispatcher = dispatcher;

}
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">

  <mx:Script>
  [Bindable]
  public var model:ContactListPresentationModel;

   </mx:Script>

   <mx:List id="list" dataProvider="{contacts}" change="listChangeHandler()" />
   <views:LoadingIndicator id="loader" />

    <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
    </mx:states>
</mx:Panel>
```

```
class

ContactListPresentationModel


[Bindable]
public var contacts:Array;

[Bindable]
public var state:String;

public function
itemSelected(contact:Contact):
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">


  <mx:Script>
  [Bindable]
  public var model:ContactListPresentationModel;

   </mx:Script>




   <mx:List id="list" dataProvider="{model.contacts}"
                                          change="listChangeHandler()" />

   <views:LoadingIndicator id="loader" />

    <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
    </mx:states>
</mx:Panel>
```

```
class

ContactListPresentationModel


[Bindable]
public var contacts:ArrayCol

[Bindable]
public var state:String;

public function
itemSelected(contact:Contact):
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml">


   <mx:Script>
   [Bindable]
   public var model:ContactListPresentationModel;

    </mx:Script>




    <mx:List id="list" dataProvider="{model.contacts}"


    <views:LoadingIndicator id="loader" />

     <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
     </mx:states>
 </mx:Panel>
```

```
class

ContactListPresentationModel


[Bindable]
public var contacts:ArrayCol

[Bindable]
public var state:String;

public function
itemSelected(contact:Contact):
```

```
change="model.itemSelected(
    list.selectedItem as
    Contact)" />
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"


   <mx:Script>
   [Bindable]
   public var model:ContactListPresentationModel;

    </mx:Script>


    <mx:List id="list" dataProvider="{model.contacts}"
                                                  change="model.itemSelected(
                                                      list.selectedItem as
                                                      Contact)" />

    <views:LoadingIndicator id="loader" />

     <mx:states>
            <mx:State name="loading"/>
            <mx:State name="loaded">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
     </mx:states>
</mx:Panel>
```

```
class

ContactListPresentationModel


[Bindable]
public var contacts:ArrayCol

[Bindable]
public var state:String;

public function
itemSelected(contact:Contact):
```

```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    currentState="{ model.state }" >

  <mx:Script>
  [Bindable]
  public var model:ContactListPresentationModel;

  </mx:Script>

  <mx:List id="list" dataProvider="{model.contacts}"
                                    change="model.itemSelected(
                                        list.selectedItem as
                                        Contact)" />

  <views:LoadingIndicator id="loader" />

   <mx:states>
          <mx:State name="loading"/>
          <mx:State name="loaded">
              <mx:RemoveChild target="{ loader }"/>
          </mx:State>
   </mx:states>

</mx:Panel>
```

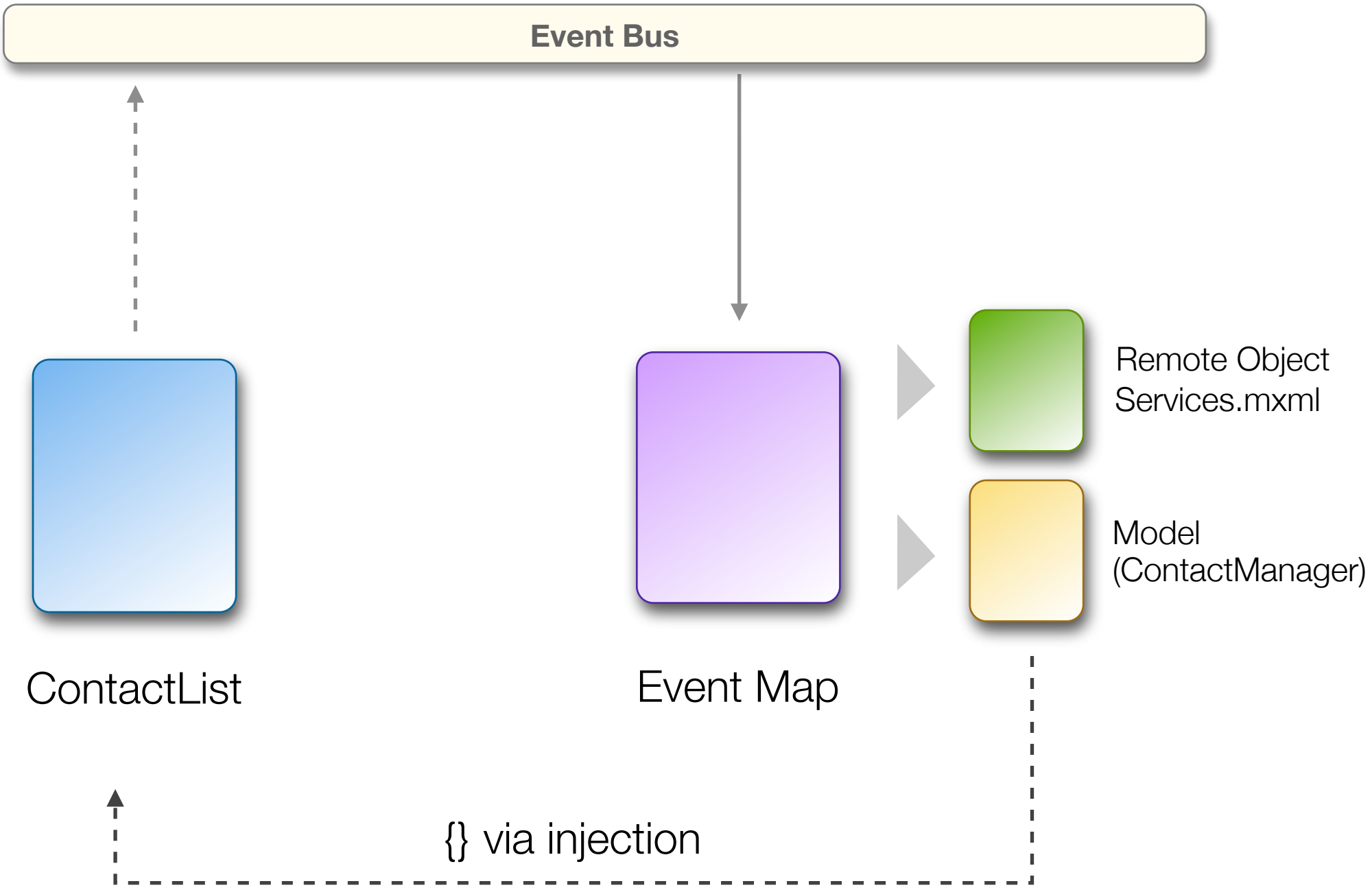
```
class

ContactListPresentationModel


[Bindable]
public var contacts:ArrayCol

[Bindable]
public var state:String;

public function
itemSelected(contact:Contact):
```

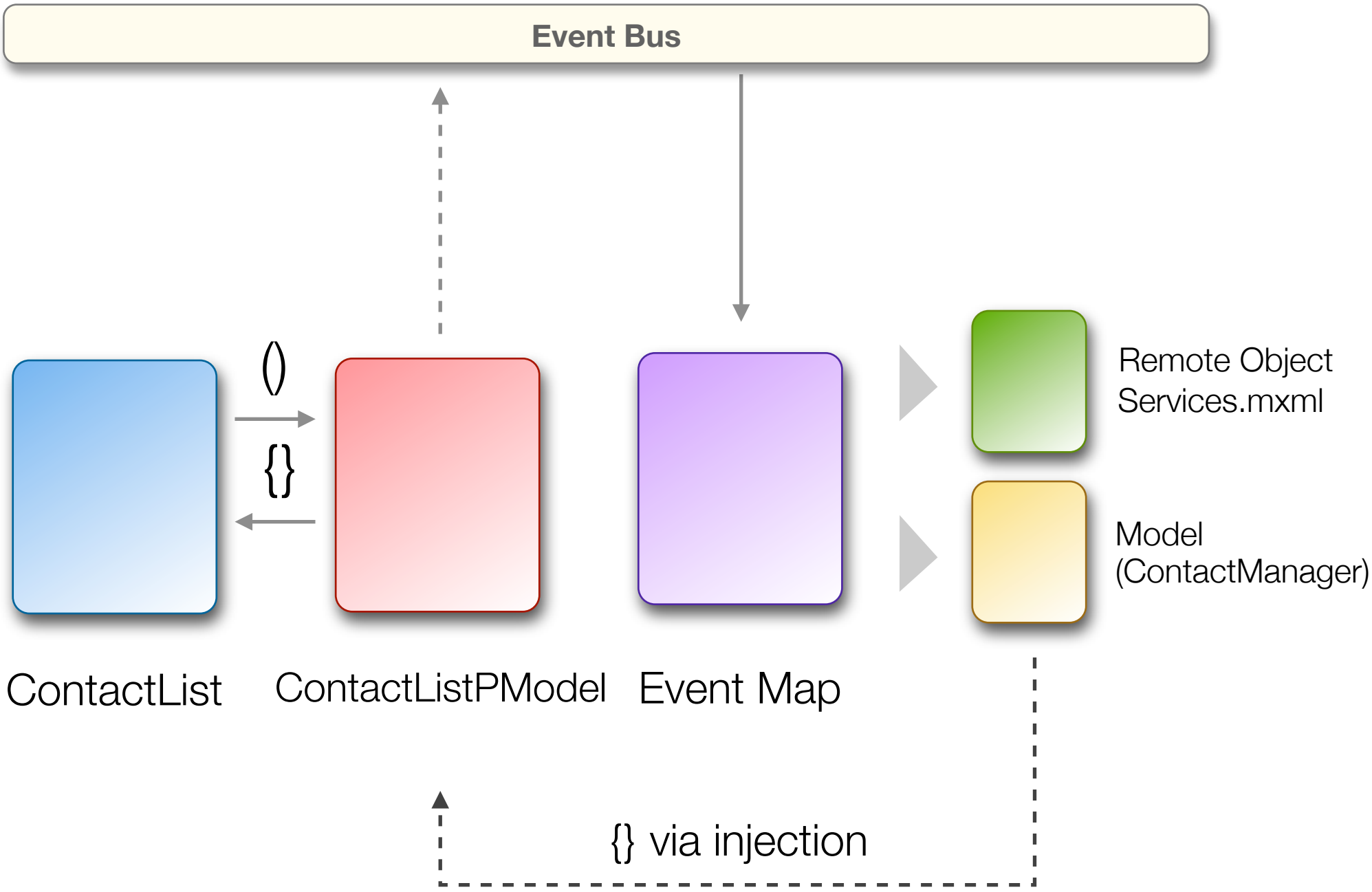
```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
      currentState="{ model.state }" >

   <mx:Script>
   [Bindable]
   public var model:ContactListPresentationModel;

    </mx:Script>

    <mx:List id="list" dataProvider="{model.contacts}"
                                                  change="model.itemSelected(
                                                      list.selectedItem as
                                                      Contact)" />

    <views:LoadingIndicator id="loader" />

     <mx:states>
            <mx:State name="{ContactListPresentationModel.LOADING_STATE}"/>
            <mx:State name="{ContactListPresentationModel.LOADED_STATE}">
                <mx:RemoveChild target="{ loader }"/>
            </mx:State>
     </mx:states>
 </mx:Panel>
```

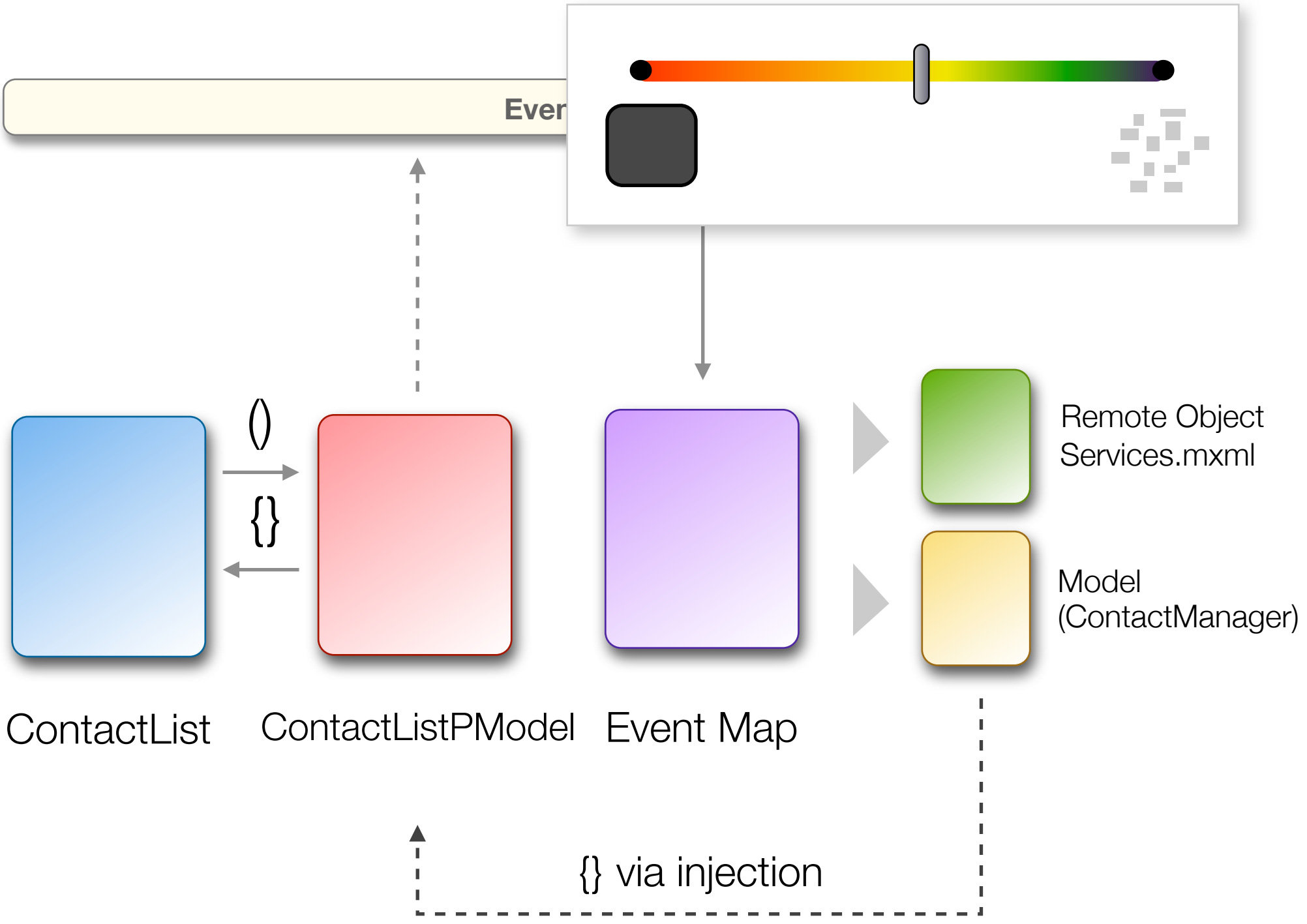
```
class

ContactListPresentationModel


[Bindable]
public var contacts:ArrayCol

[Bindable]
public var state:String;

public function
itemSelected(contact:Contact):
```

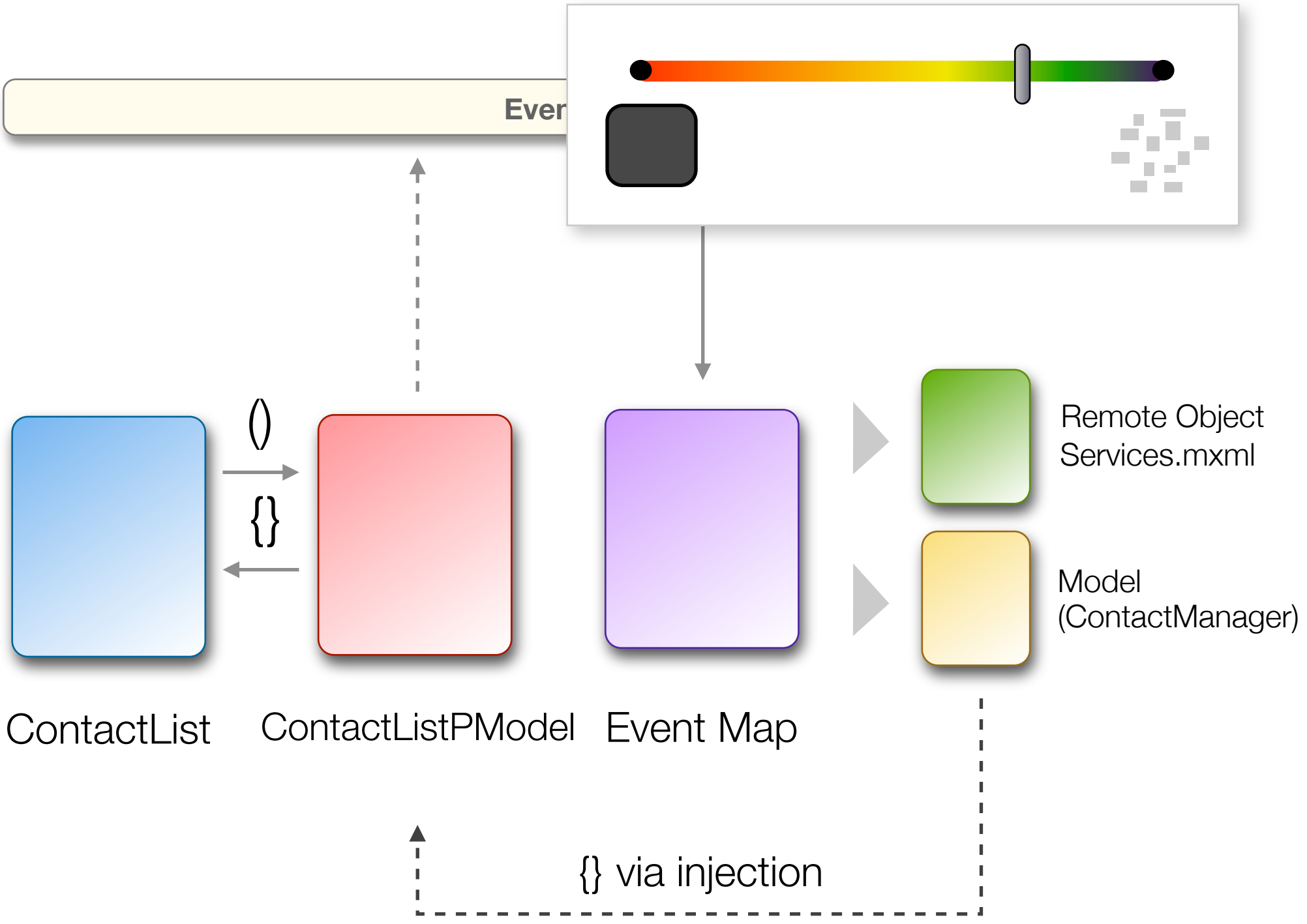
```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
      currentState="{ model.state }" >

   <mx:Script>
   [Bindable]
   public var model:ContactListPresentationModel;

    </mx:Script>

    <mx:List id="list" dataProvider="{model.contacts}"
                                                  change="model.itemSelected(
                                                      list.selectedItem as
                                                      Contact)" />

    <views:LoadingIndicator id="loader" />

     <mx:states>
          <mx:State name="{ContactListPresentationModel.LOADING_STATE}"/>
          <mx:State name="{ContactListPresentationModel.LOADED_STATE}">
              <mx:RemoveChild target="{ loader }"/>
          </mx:State>
     </mx:states>
</mx:Panel>
```

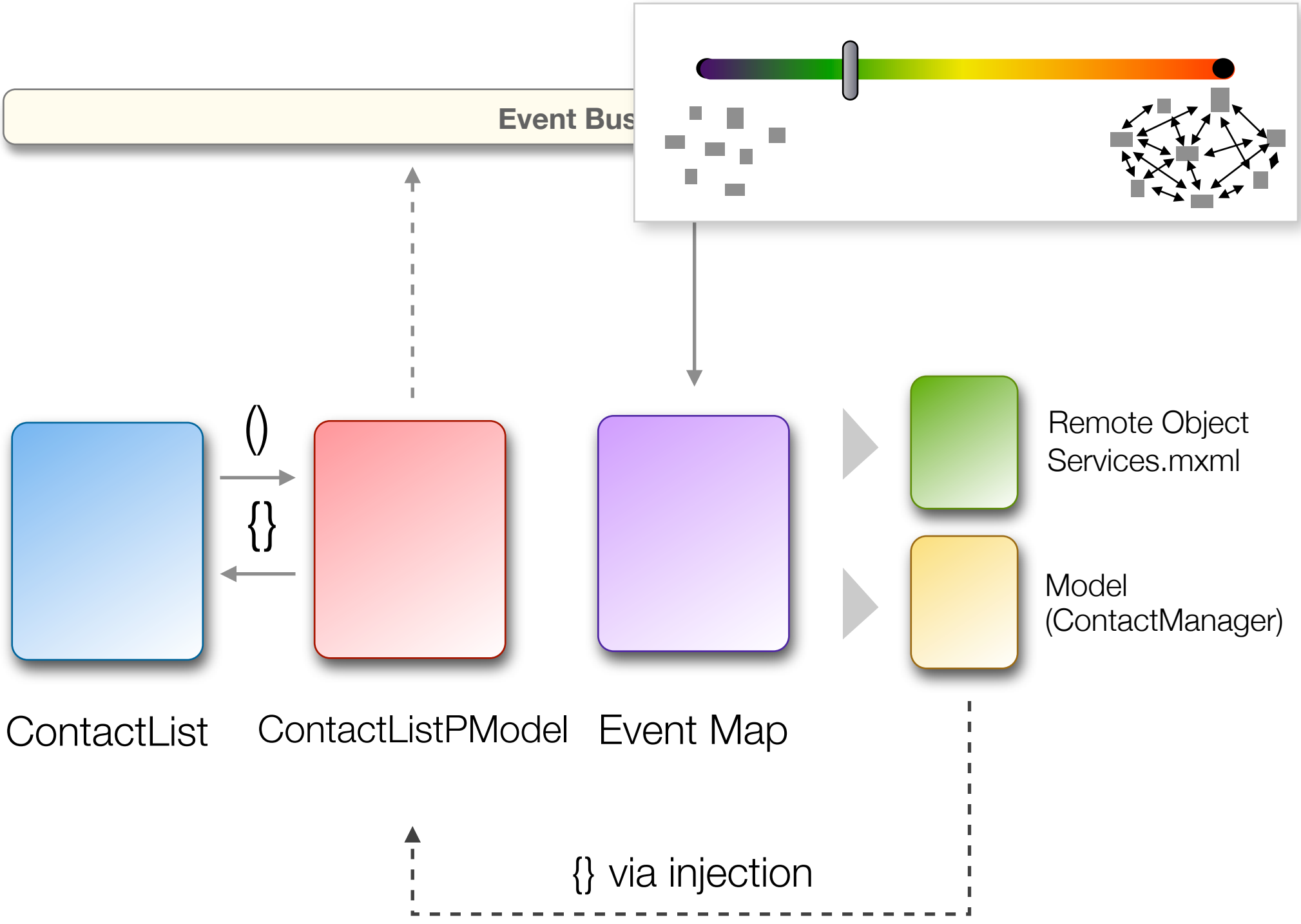
```
class

ContactListPresentationModel

[Bindable]
public var contacts:ArrayCol

[Bindable]
public var state:String;

public function
itemSelected(contact:Contact):
```

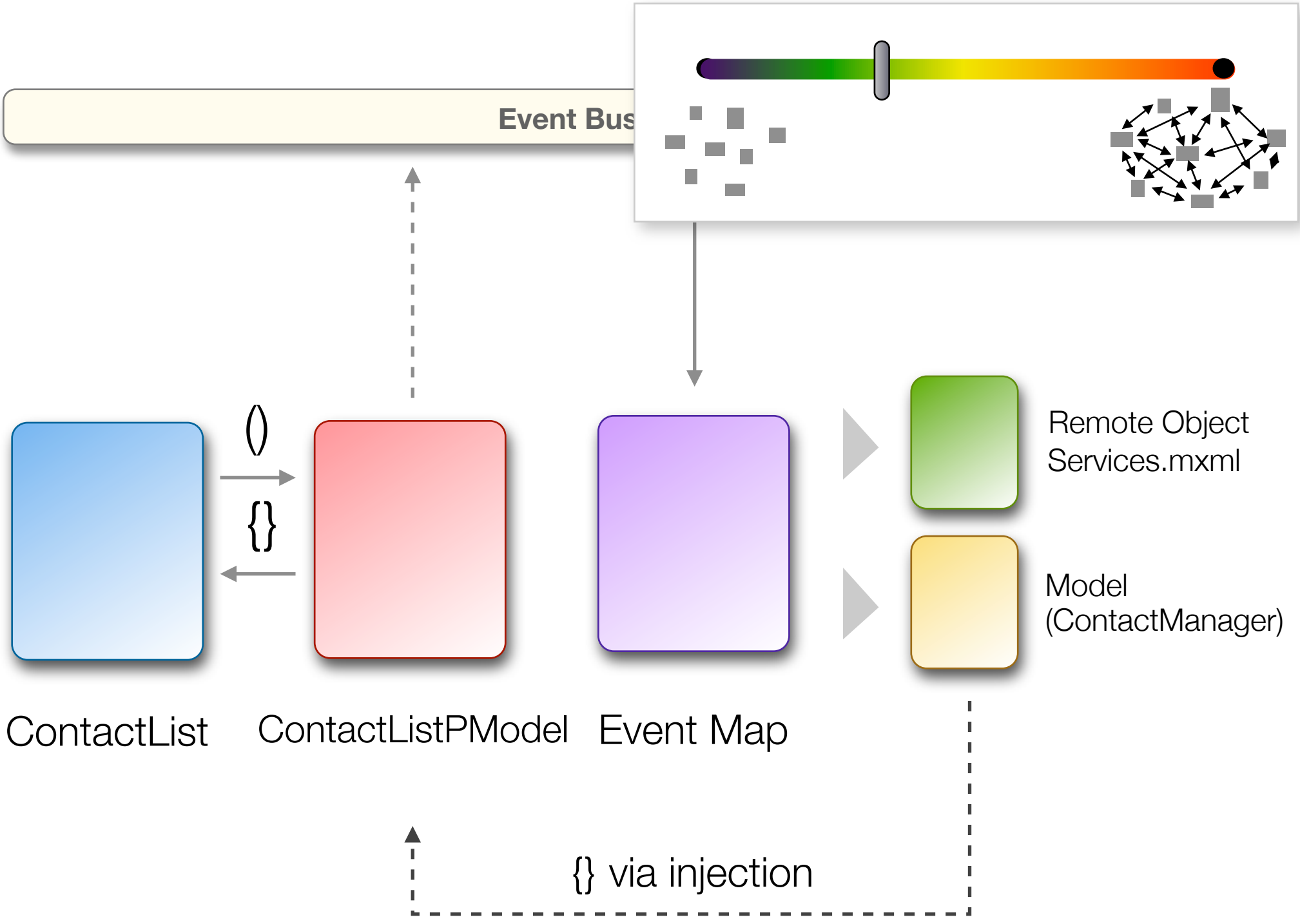

Event Bus
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
Event Map
{} via injection


Event Bus
()
{}
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
ContactListPModel
Event Map
{} via injection


Ever
()
{}
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
ContactListPModel
Event Map
{} via injection


Ever
()
{}
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
ContactListPModel
Event Map
{} via injection


Event Bus
()
{}
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
ContactListPModel
Event Map
{} via injection


Event Bus
()
{}
Remote Object
Services.mxml
Model
(ContactManager)
ContactList
ContactListPModel
Event Map
{} via injection

Testing views

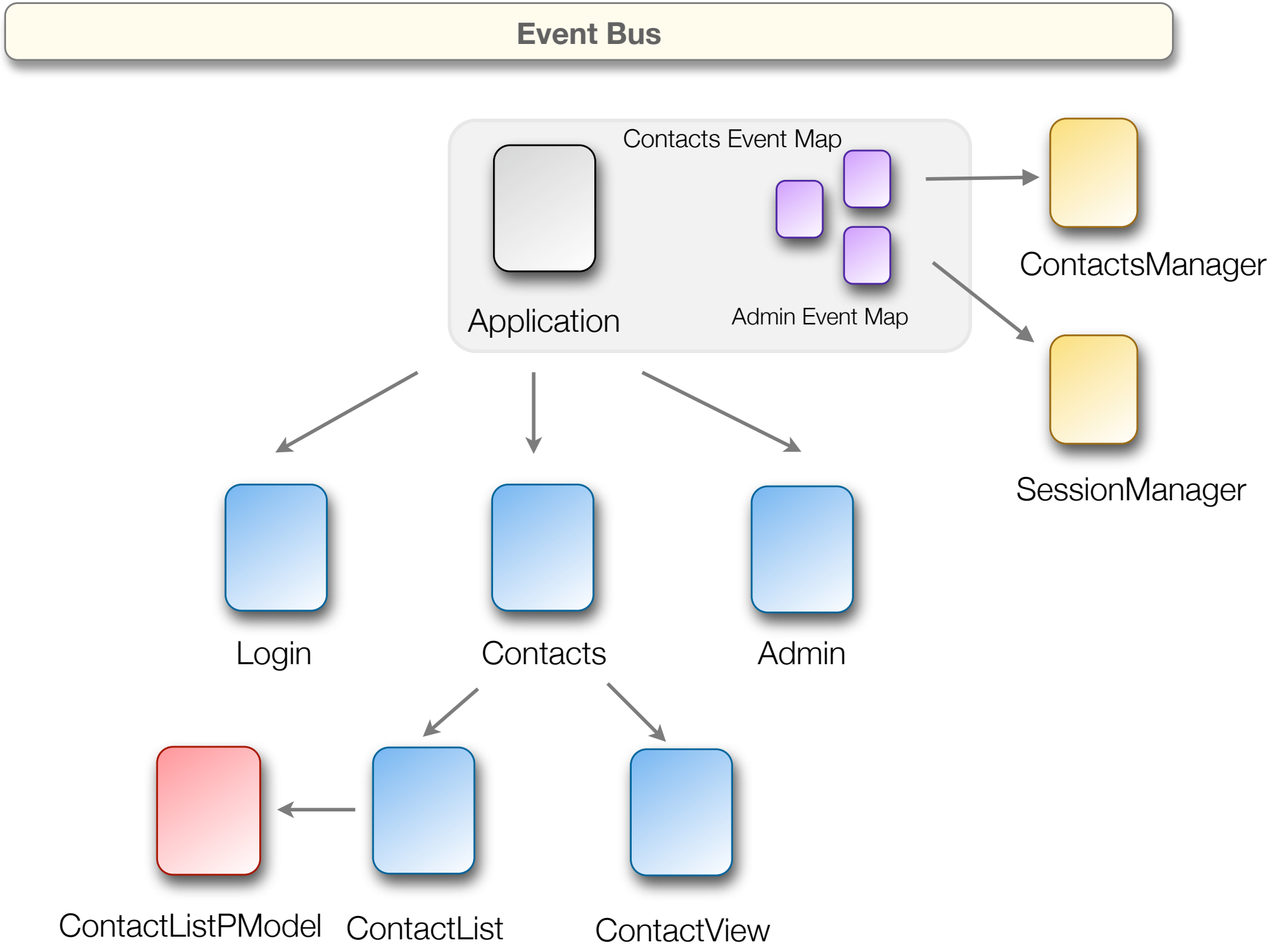
```
package com.asfusion.examples.intranet.admin.login.ui.presenters {

    public class LoginPresentationModelTest extends TestCase
    {
        private var model:LoginPresentationModel;
        private var dispatcher:IEventDispatcher;

        // This method will be called before every test function
        override public function setUp():void
        {
            super.setUp();

            dispatcher = new EventDispatcher();
            model = new LoginPresentationModel( dispatcher );
        }

        // This method will be called after every test function
        override public function tearDown():void
        {
            super.tearDown();
            model = null;
        }

        public function testLogin():void
        {
            model.login( "","" );
            assertEquals( model.errorMessage, "Username and Password are required fields.");

        }
    }
}
```

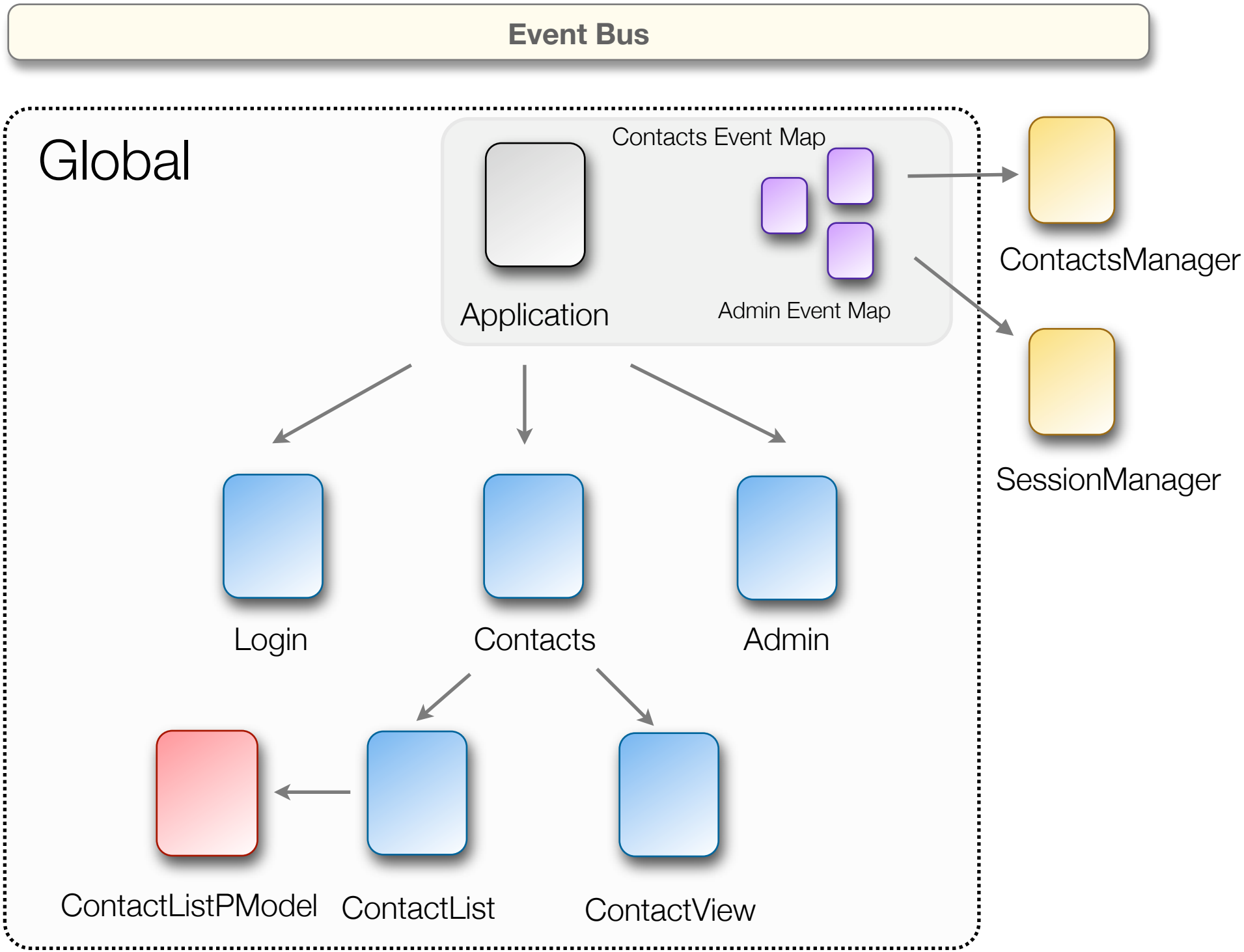
```
package com.asfusion.examples.intranet.admin.login.ui.presenters {

    public class LoginPresentationModelTest extends TestCase
    {
        private var model:LoginPresentationModel;
        private var dispatcher:IEventDispatcher;

        // This method will be called before every test function
        override public function setUp():void
        {
            super.setUp();

            dispatcher = new EventDispatcher();
            model = new LoginPresentationModel( dispatcher );
        }

        // This method will be called after every test function
        override public function tearDown():void
        {
            super.tearDown();
            model = null;
        }

        public function testLogin():void
        {
            model.login( "","" );
            assertEquals( model.errorMessage, "Username and Password are required fields.");

        }
    }
}
```

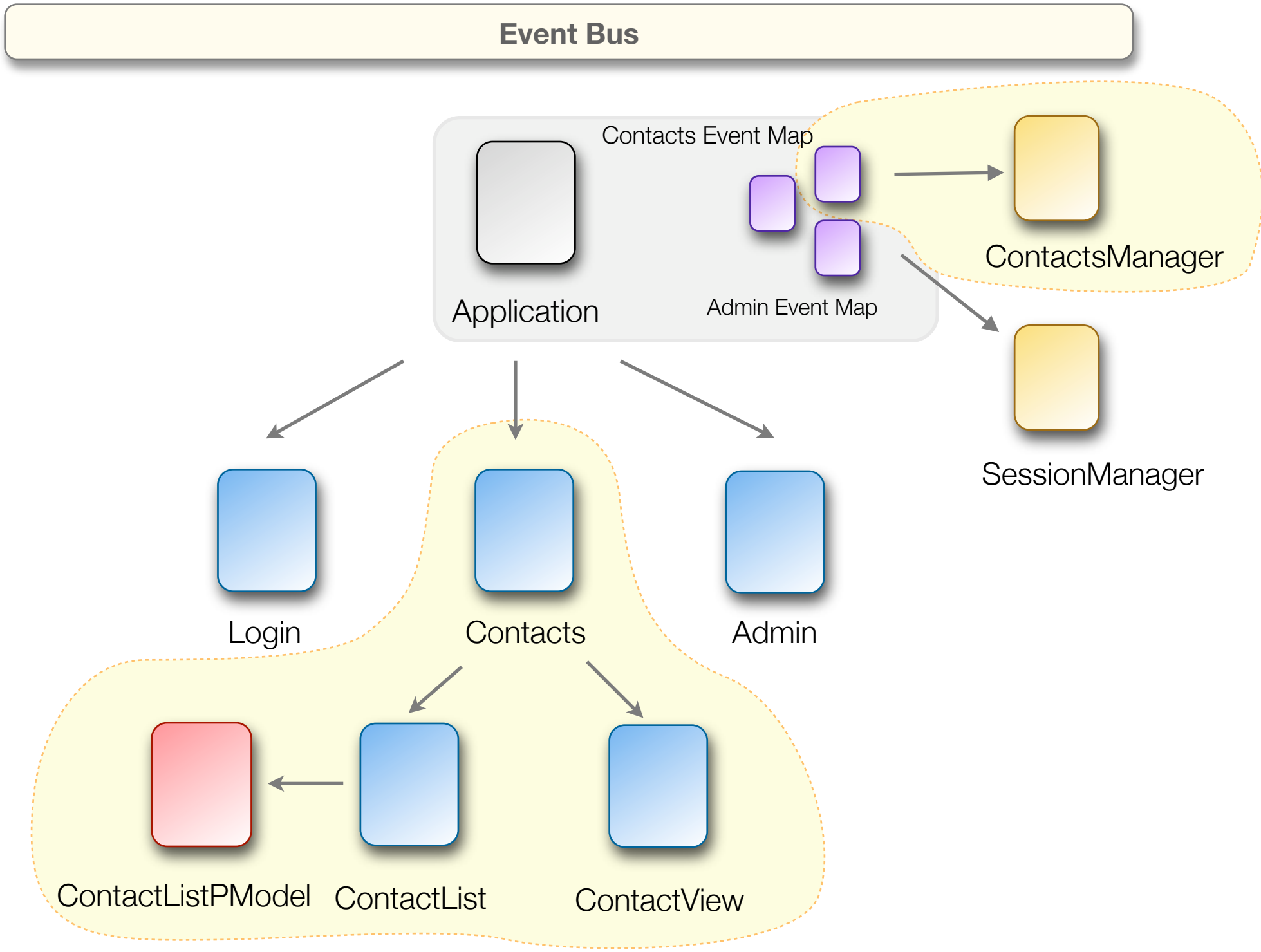
```
package com.asfusion.examples.intranet.admin.login.ui.presenters {

    public class LoginPresentationModelTest extends TestCase
    {
        private var model:LoginPresentationModel;
        private var dispatcher:IEventDispatcher;

        // This method will be called before every test function
        override public function setUp():void
        {
            super.setUp();

            dispatcher = new EventDispatcher();
            model = new LoginPresentationModel( dispatcher );
        }

        // This method will be called after every test function
        override public function tearDown():void
        {
            super.tearDown();
            model = null;
        }

        public function testLogin():void
        {
            model.login( "","" );
            assertEquals( model.errorMessage, "Username and Password are required fields.");

        }
    }
}
```

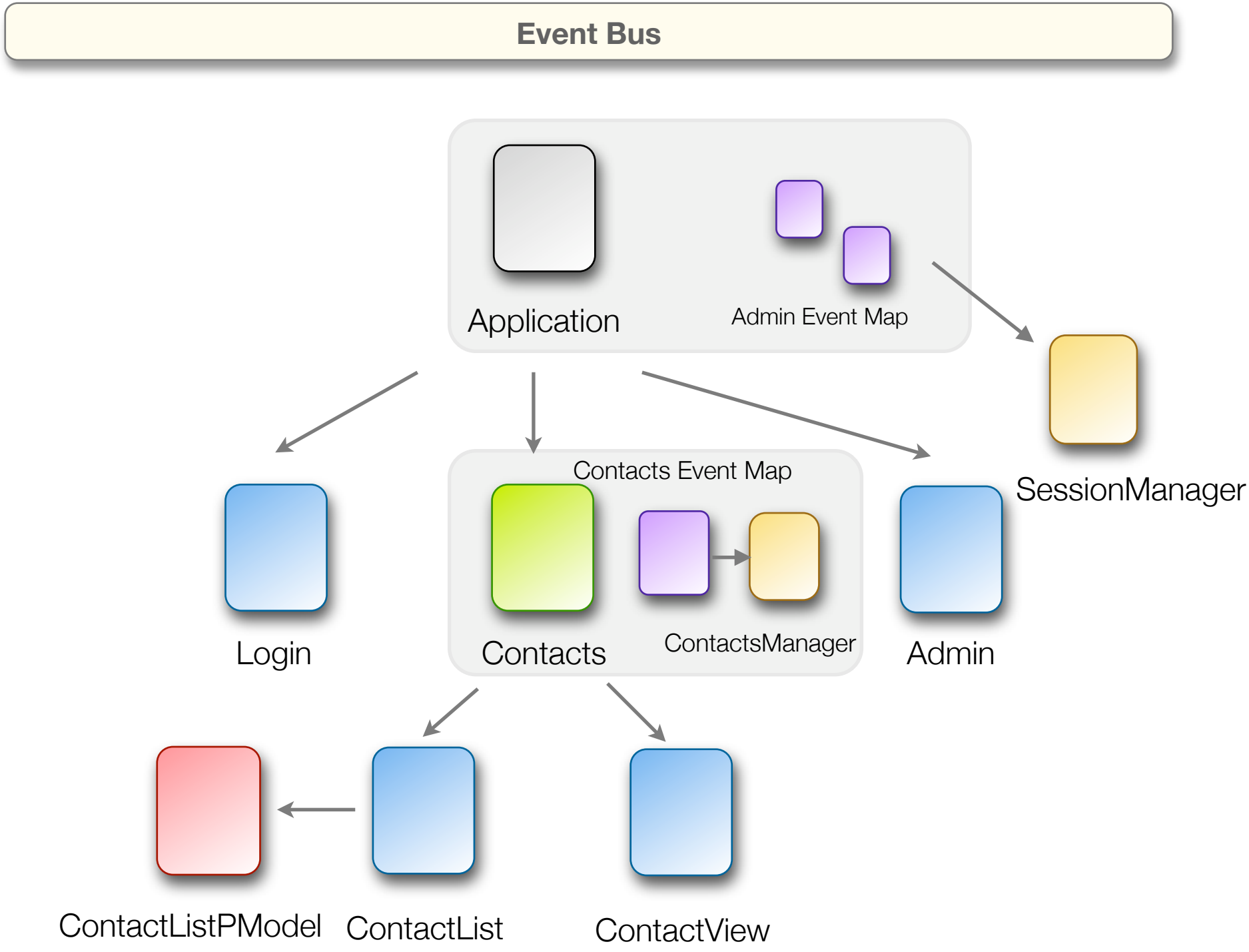
Modules

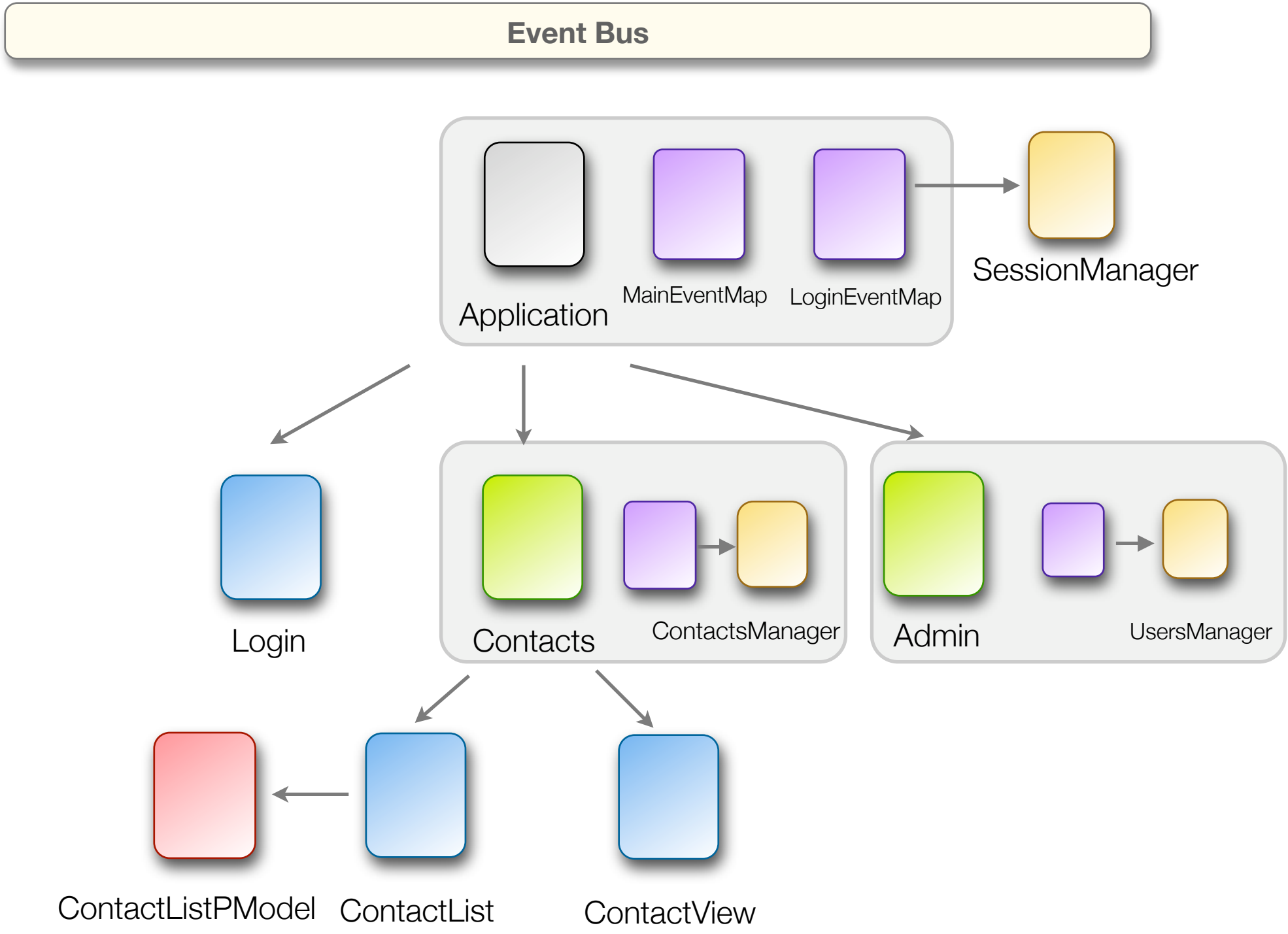
```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml">


    <maps:MainEventMap />
    <maps:LoginEventMap/>
    <maps:ContactEventMap />
    <maps:AdminEventMap />

   <views:MainUI width="100%" height="100%"/>

</mx:Application>
```

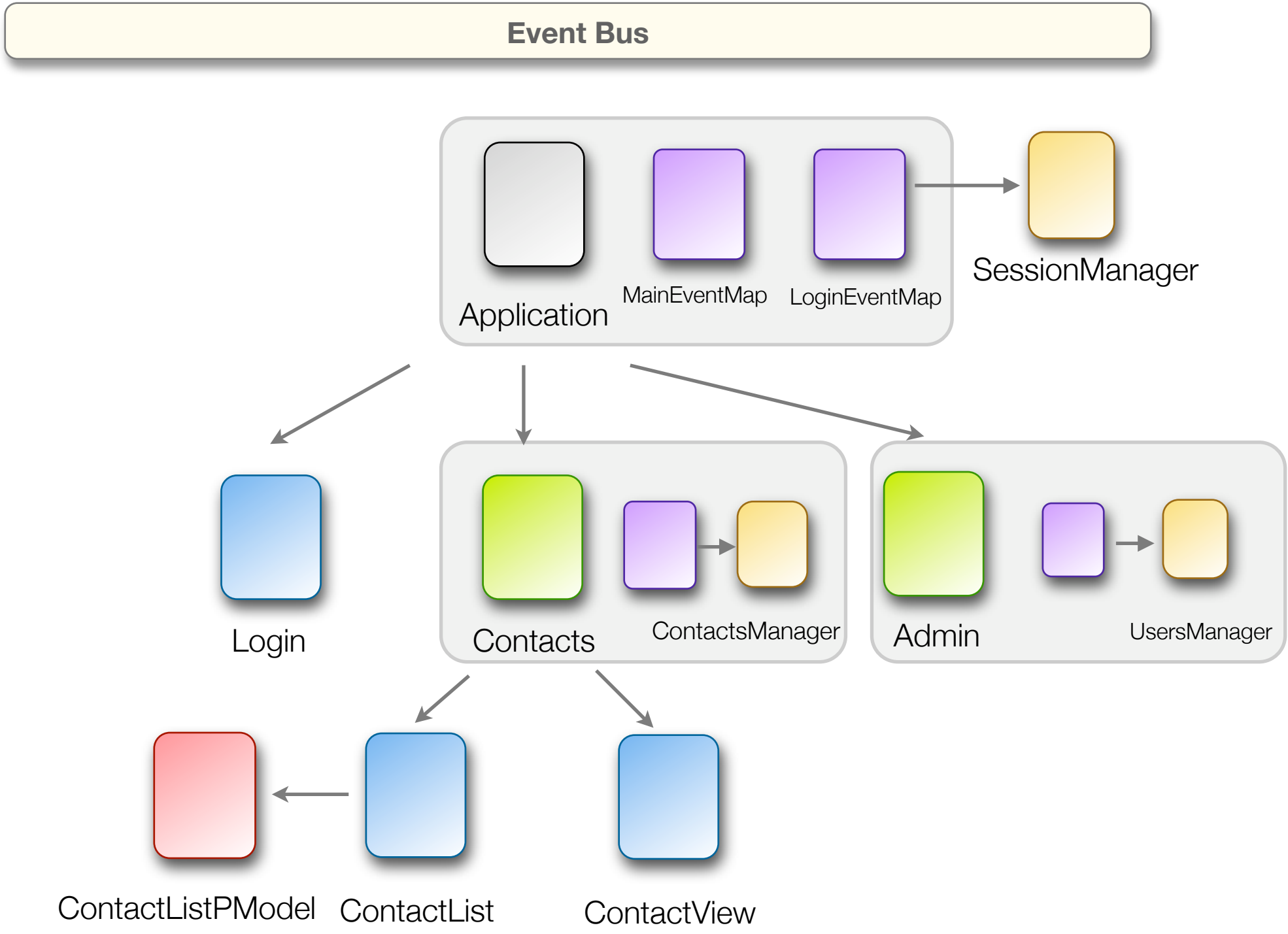

Event Bus
Contacts Event Map
Application
Admin Event Map
ContactsManager
SessionManager
Login
Contacts
Admin
ContactListPModel
ContactList
ContactView

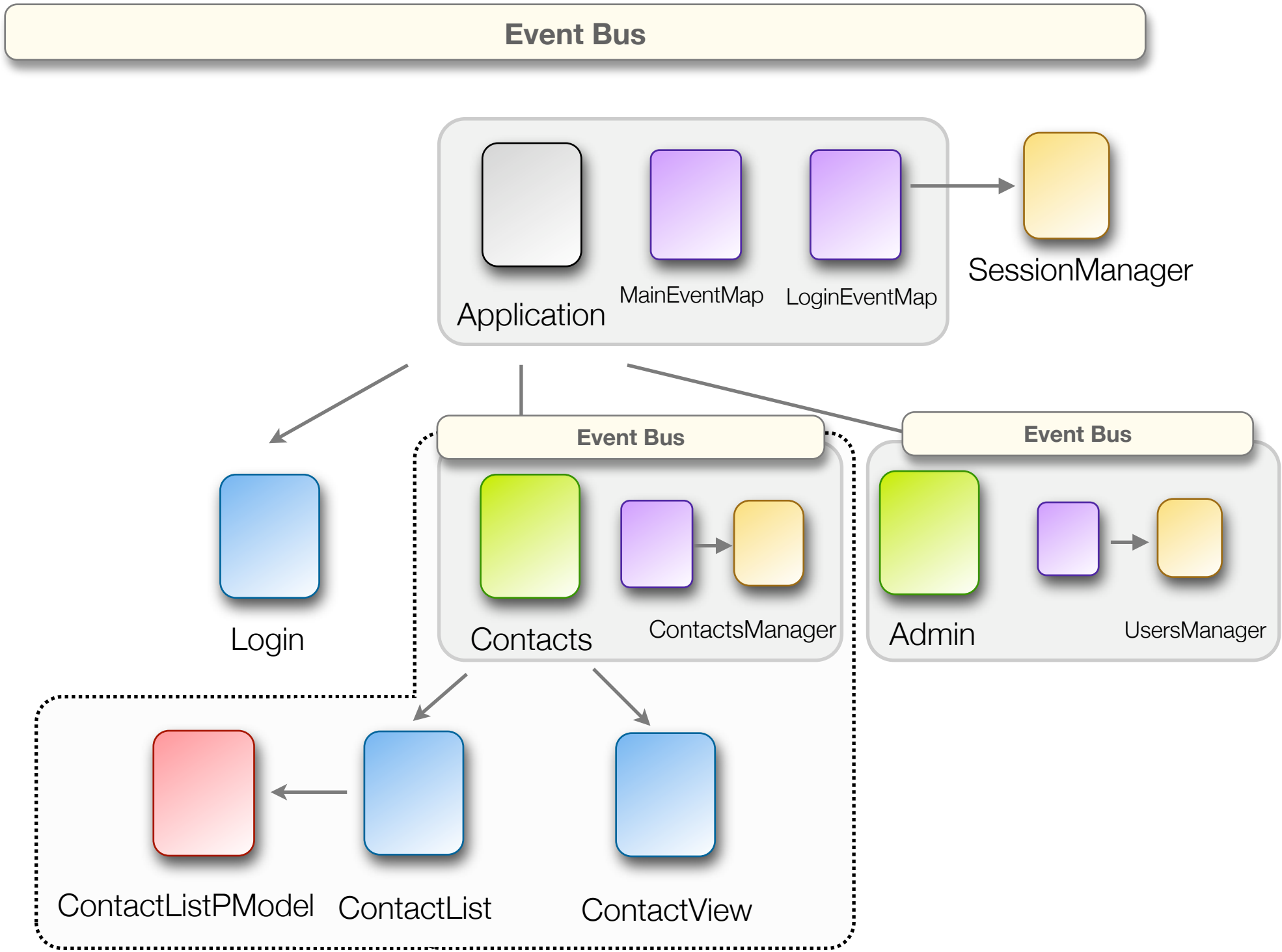

Event Bus
Global
Contacts Event Map
Application
Admin Event Map
ContactsManager
SessionManager
Login
Contacts
Admin
ContactListPModel
ContactList
ContactView


Event Bus
Contacts Event Map
Application
Admin Event Map
ContactsManager
SessionManager
Login
Contacts
Admin
ContactListPModel
ContactList
ContactView


Event Bus
Application
Admin Event Map
SessionManager
Contacts Event Map
Login
Contacts
ContactsManager
Admin
ContactListPModel
ContactList
ContactView


Event Bus
Application
MainEventMap
LoginEventMap
SessionManager
Login
Contacts
ContactsManager
Admin
UsersManager
ContactListPModel
ContactList
ContactView


Event Bus
Application
MainEventMap
LoginEventMap
SessionManager
Login
Contacts
ContactsManager
Admin
UsersManager
ContactListPModel
ContactList
ContactView


Event Bus
Application
MainEventMap
LoginEventMap
SessionManager
Login
Event Bus
Contacts
ContactsManager
Event Bus
Admin
UsersManager
ContactListPModel
ContactList
ContactView

Event Map

```
<EventMap>

    <EventHandlers type="{ ContactEvent.GET_ALL }">
        <RemoteObjectInvoker instance="{ services.contacts }"
            method="getAll">
            <resultHandlers>
                <MethodInvoker generator="{ ContactManager }"
                    method="storeContacts" arguments="{ resultObject }"/>
            </resultHandlers>
        </RemoteObjectInvoker>
    </EventHandlers>

    <Injectors target="{ ContactList }">

        <PropertyInjector targetKey="model"
        source="{ ContactListPresentationModel }"/>

    </Injectors>

</EventMap>
```

Event Map

```
<LocalEventMap>

    <EventHandlers type="{ ContactEvent.GET_ALL }">
        <RemoteObjectInvoker instance="{ services.contacts }"
            method="getAll">
            <resultHandlers>
                <MethodInvoker generator="{ ContactManager }"
                    method="storeContacts" arguments="{ resultObject }"/>
            </resultHandlers>
        </RemoteObjectInvoker>
    </EventHandlers>

    <Injectors target="{ ContactList }">

        <PropertyInjector targetKey="model"
        source="{ ContactListPresentationModel }"/>

    </Injectors>

</LocalEventMap>
```

Contacts Module

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%">

    <maps:ContactsEventMap dispatcher="{ this }"/>

    <view:ContactUI />


</mx:Module>
```

Contacts Module

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%">

    <maps:ContactsEventMap dispatcher="{ this }"/>

    <view:ContactUI />


</mx:Module>
```

Accessing global data

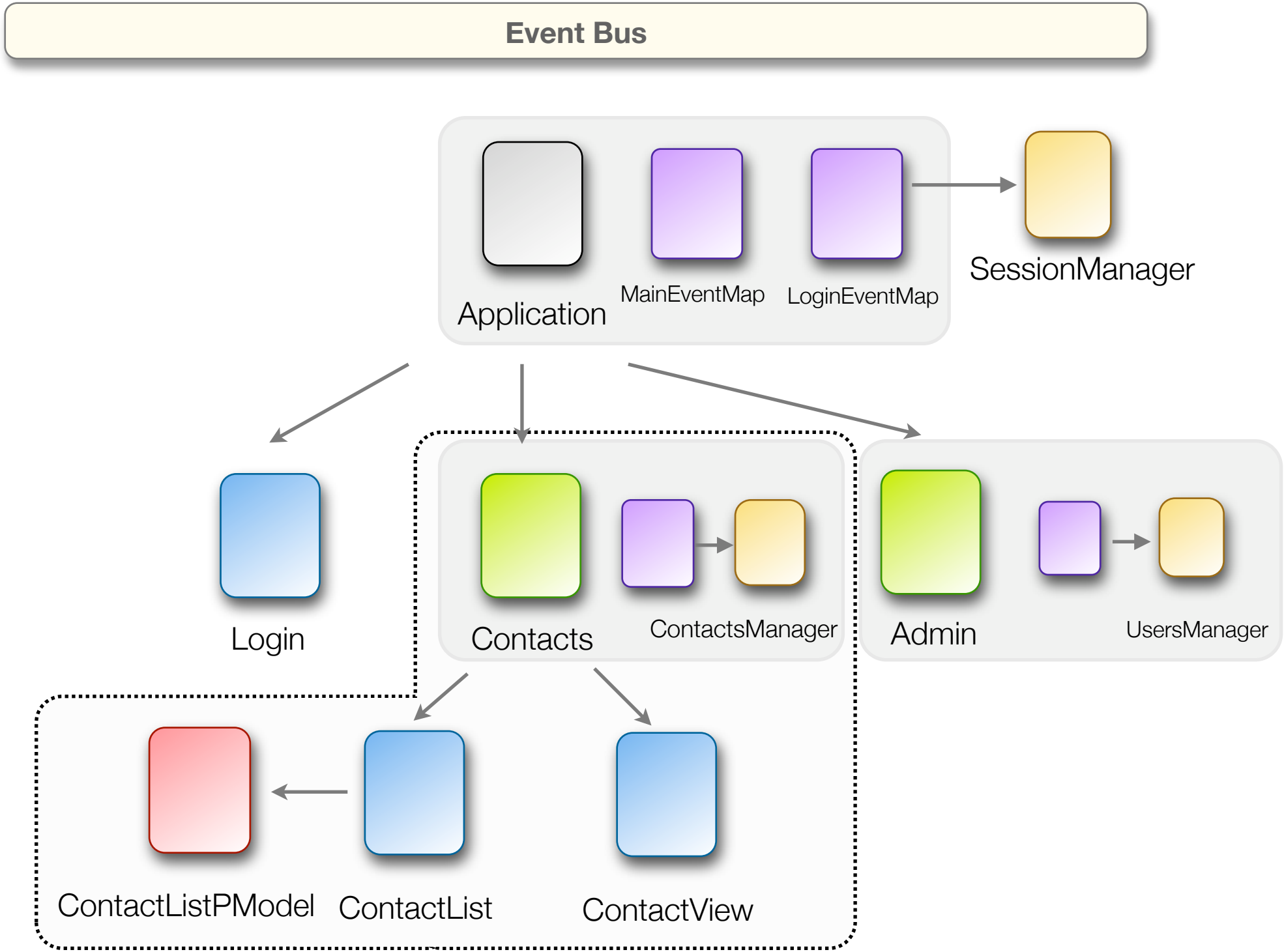
```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    currentState="{ model.state }">

    <mx:Script>

            [Bindable]
            public var model:ContactUIPresentationModel

    </mx:Script>

    <view:ContactList />
    <view:ContactView id="viewMode" />

    <mx:states>
        <mx:State name="{ ContactUIPresentationModel.EDIT_DISABLED }"/>
        <mx:State name="{ ContactUIPresentationModel.EDIT_ENABLED }">
            <mx:AddChild>
                <mx:Button label="Add Contact" click="model.addContact();"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>

</mx:Panel>
```

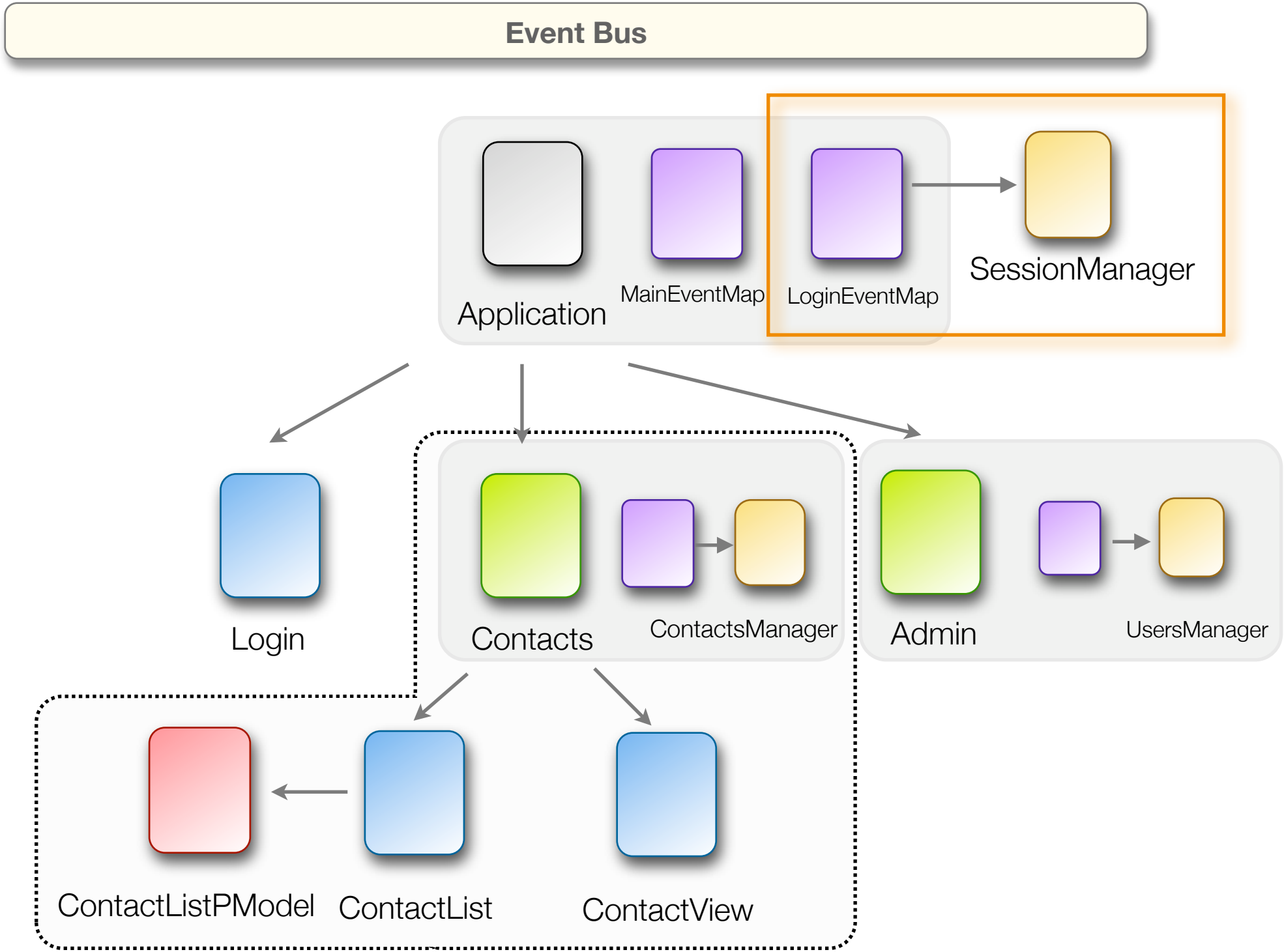
```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    currentState="{ model.state }">

    <mx:Script>

        [Bindable]
        public var model:ContactUIPresentationModel

    </mx:Script>

    <view:ContactList />
    <view:ContactView id="viewMode" />

    <mx:states>
        <mx:State name="{ ContactUIPresentationModel.EDIT_DISABLED }"/>
        <mx:State name="{ ContactUIPresentationModel.EDIT_ENABLED }">
            <mx:AddChild>
                <mx:Button label="Add Contact" click="model.addContact();"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>

</mx:Panel>
```

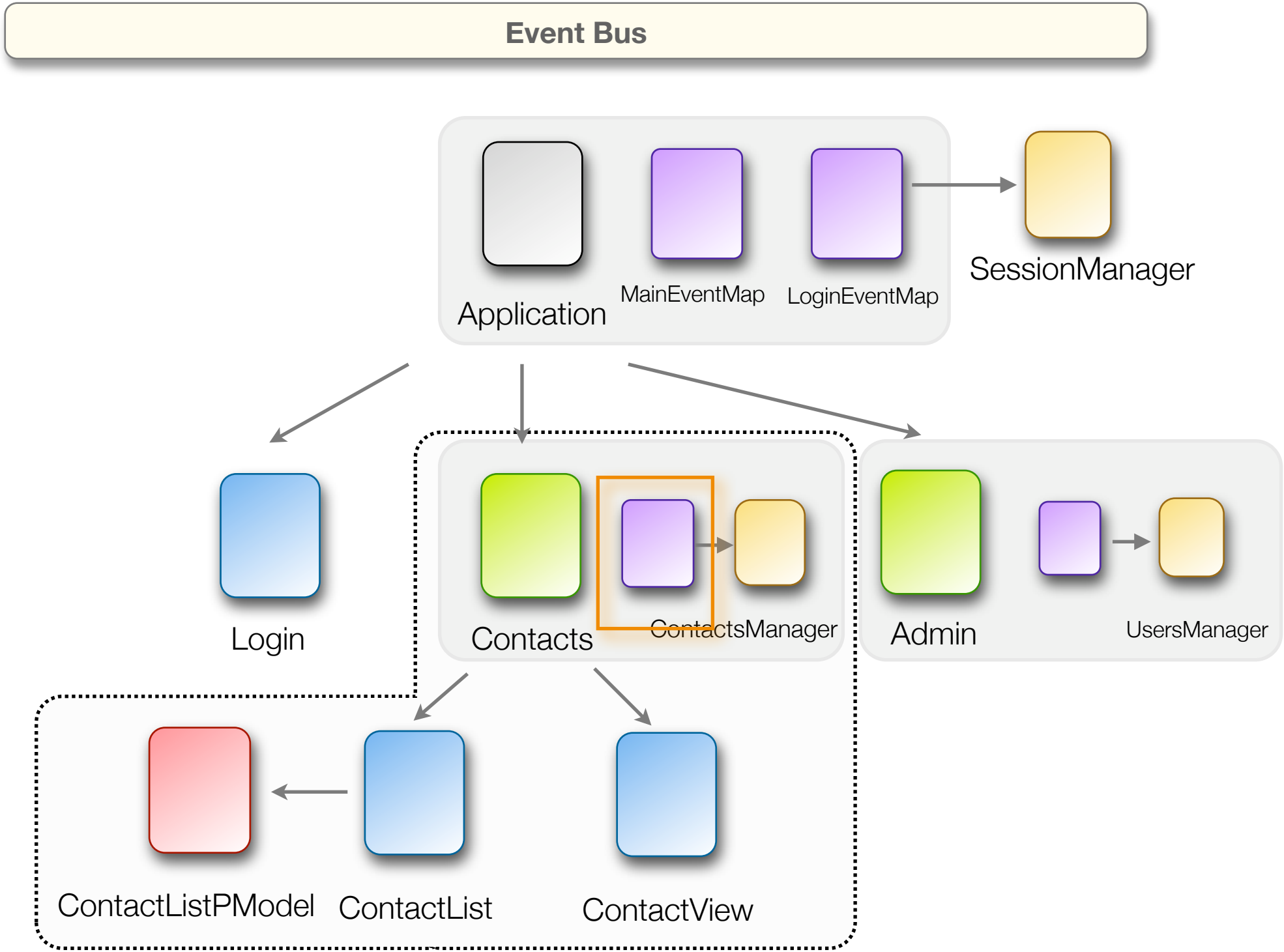
```
<mx:Panel xmlns:mx="http://www.adobe.com/2006/mxml"
    currentState="{ model.state }">

    <mx:Script>

            [Bindable]
            public var model:ContactUIPresentationModel

    </mx:Script>

    <view:ContactList />
    <view:ContactView id="viewMode" />

    <mx:states>
        <mx:State name="{ ContactUIPresentationModel.EDIT_DISABLED }"/>
        <mx:State name="{ ContactUIPresentationModel.EDIT_ENABLED }">
            <mx:AddChild>
                <mx:Button label="Add Contact" click="model.addContact();"/>
            </mx:AddChild>
        </mx:State>
    </mx:states>

</mx:Panel>
```

```
public class ContactUIPresentationModel extends EventDispatcher {

    public function set currentUser( user:User ):void{

            _currentUser = user;

            if( user == null || user.permissions == UserPermissions.READ_ONLY )
            {
                newState = EDIT_DISABLED;

            }
            else if( user.permissions != UserPermissions.READ_ONLY
                    && state == EDIT_DISABLED)
            {
                newState = EDIT_ENABLED;
            }
        }

.....
}
```

```
public class ContactUIPresentationModel extends EventDispatcher {

    public function set currentUser( user:User ):void{

            _currentUser = user;

            if( user == null || user.permissions == UserPermissions.READ_ONLY )
            {
                newState = EDIT_DISABLED;

            }
            else if( user.permissions != UserPermissions.READ_ONLY
                    && state == EDIT_DISABLED)
            {
                newState = EDIT_ENABLED;
            }
        }

.....
}
```


Event Bus
Application
MainEventMap
LoginEventMap
SessionManager
Login
Contacts
ContactsManager
Admin
UsersManager
ContactListPModel
ContactList
ContactView


Event Bus
Application
MainEventMap
LoginEventMap
SessionManager
Login
Contacts
ContactsManager
Admin
UsersManager
ContactListPModel
ContactList
ContactView

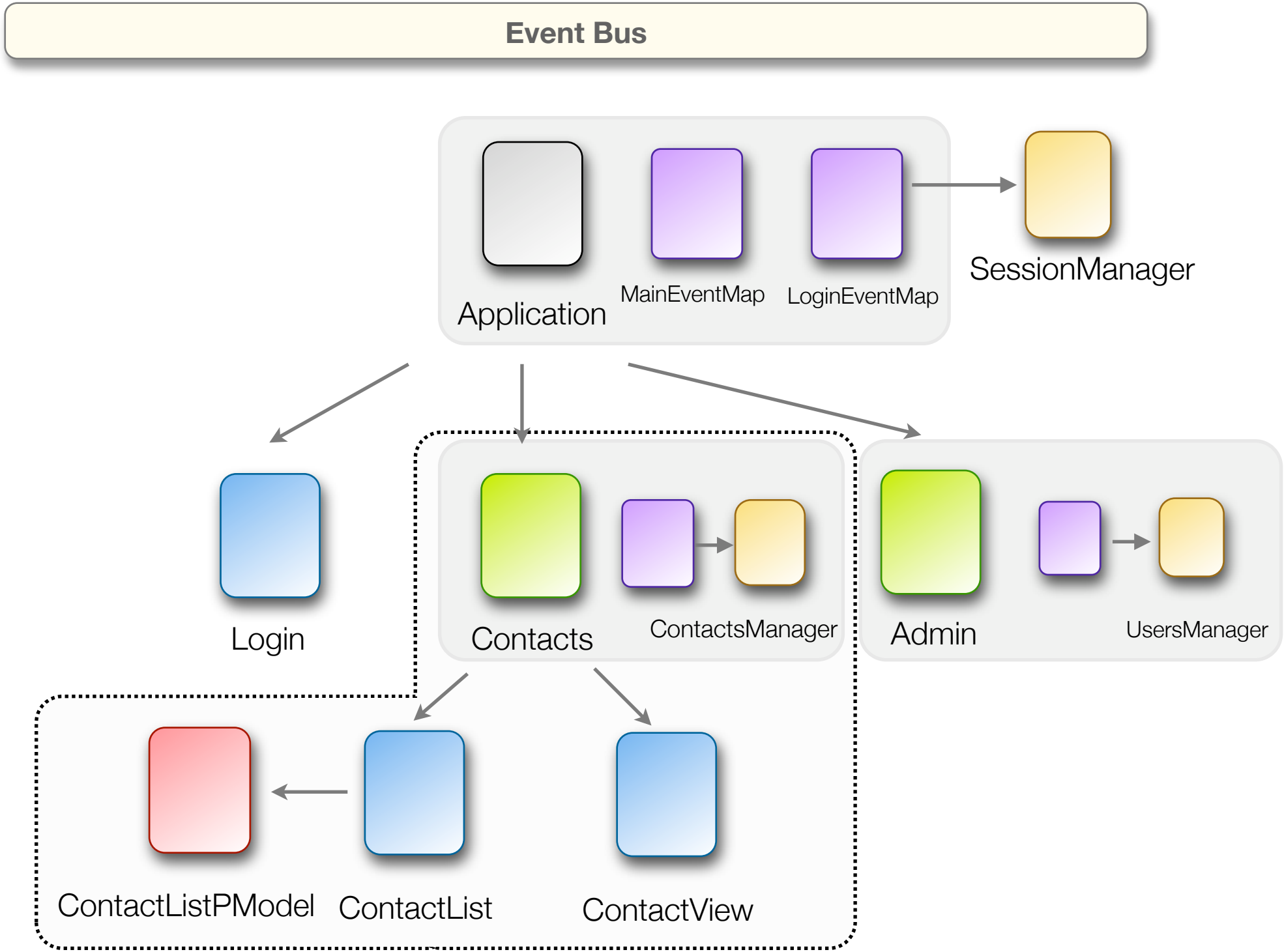

Event Bus
Application
MainEventMap
LoginEventMap
SessionManager
Login
Contacts
ContactsManager
Admin
UsersManager
ContactListPModel
ContactList
ContactView

ContactMap

```
<LocalEventMap xmlns:mx="http://www.adobe.com/2006/mxml">


    <Injectors target="{ ContactUIPresentationModel }">

        <PropertyInjector targetKey="currentUser" source="{ SessionManager }"
                          sourceCache="global" sourceKey="user"
                          softBinding="true"/>

    </Injectors>

</LocalEventMap>
```

ContactMap

```
<LocalEventMap xmlns:mx="http://www.adobe.com/2006/mxml">

    <Injectors target="{ ContactUIPresentationModel }">
        <PropertyInjector targetKey="currentUser" source="{ SessionManager }"
                          sourceCache="global" sourceKey="user"
                          softBinding="true"/>
    </Injectors>

</LocalEventMap>
```


Event Bus
Application
MainEventMap
LoginEventMap
SessionManager
Login
Contacts
ContactsManager
Admin
UsersManager
ContactListPModel
ContactList
ContactView

MainEventMap

```
<EventMap xmlns:mx="http://www.adobe.com/2006/mxml">


    <Injectors target="{ IRestrictedModule }" includeDerivatives="true">
        <PropertyInjector targetKey="currentUser" source="{ SessionManager }"
                          sourceKey="user" softBinding="true"/>
    </Injectors>

</EventMap>
```

MainEventMap

```
<EventMap xmlns:mx="http://www.adobe.com/2006/mxml">

    <Injectors target="{ IRestrictedModule }" includeDerivatives="true">
        <PropertyInjector targetKey="currentUser" source="{ SessionManager }"
                          sourceKey="user" softBinding="true"/>
    </Injectors>

</EventMap>
```

Module implements interface

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%"
     implements="com.asfusion.examples.intranet.IRestrictedModule">


.......


</mx:Module>
```

Module implements interface

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Module xmlns:mx="http://www.adobe.com/2006/mxml"
    width="100%" height="100%"
     implements="com.asfusion.examples.intranet.IRestrictedModule">


.......


</mx:Module>
```

Module to module communication

Module 1 sends an event:

```
<EventAnnouncer generator="{ ContactEvent }"
        type="{ ContactEvent.LOADED }"
        dispatcherType="global"/>
```

Module 2 handles it:

```
<EventHandlers type="{ ContactEvent.LOADED }"
dispatcherType="global">

            <!-- do something with the event -->

</EventHandlers>
```

Module to module communication

Module 1 sends an event:

```
<EventAnnouncer generator="{ ContactEvent }"
    type="{ ContactEvent.LOADED }"
    dispatcherType="global"/>
```

Module 2 handles it:

```
<EventHandlers type="{ ContactEvent.LOADED }"
dispatcherType="global">

        <!-- do something with the event -->

</EventHandlers>
```

Module to module communication

Module 1 sends an event:

```
<EventAnnouncer generator="{ ContactEvent }"
        type="{ ContactEvent.LOADED }"
        dispatcherType="global"/>
```

Module 2 handles it:

```
<EventHandlers type="{ ContactEvent.LOADED }"
dispatcherType="global">

        <!-- do something with the event -->

</EventHandlers>
```

Learn more at

http://mate.asfusion.com