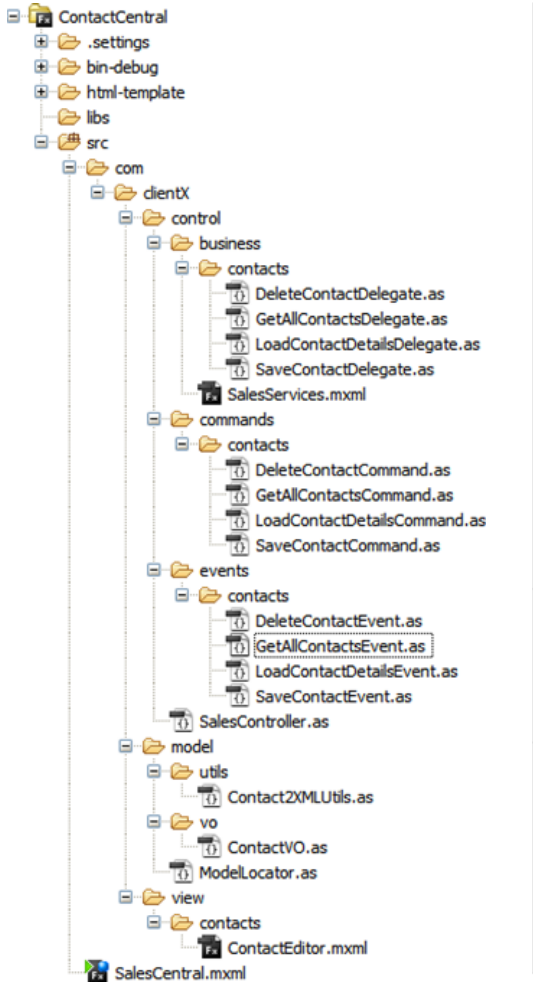
Adobe Cairngorm MVC

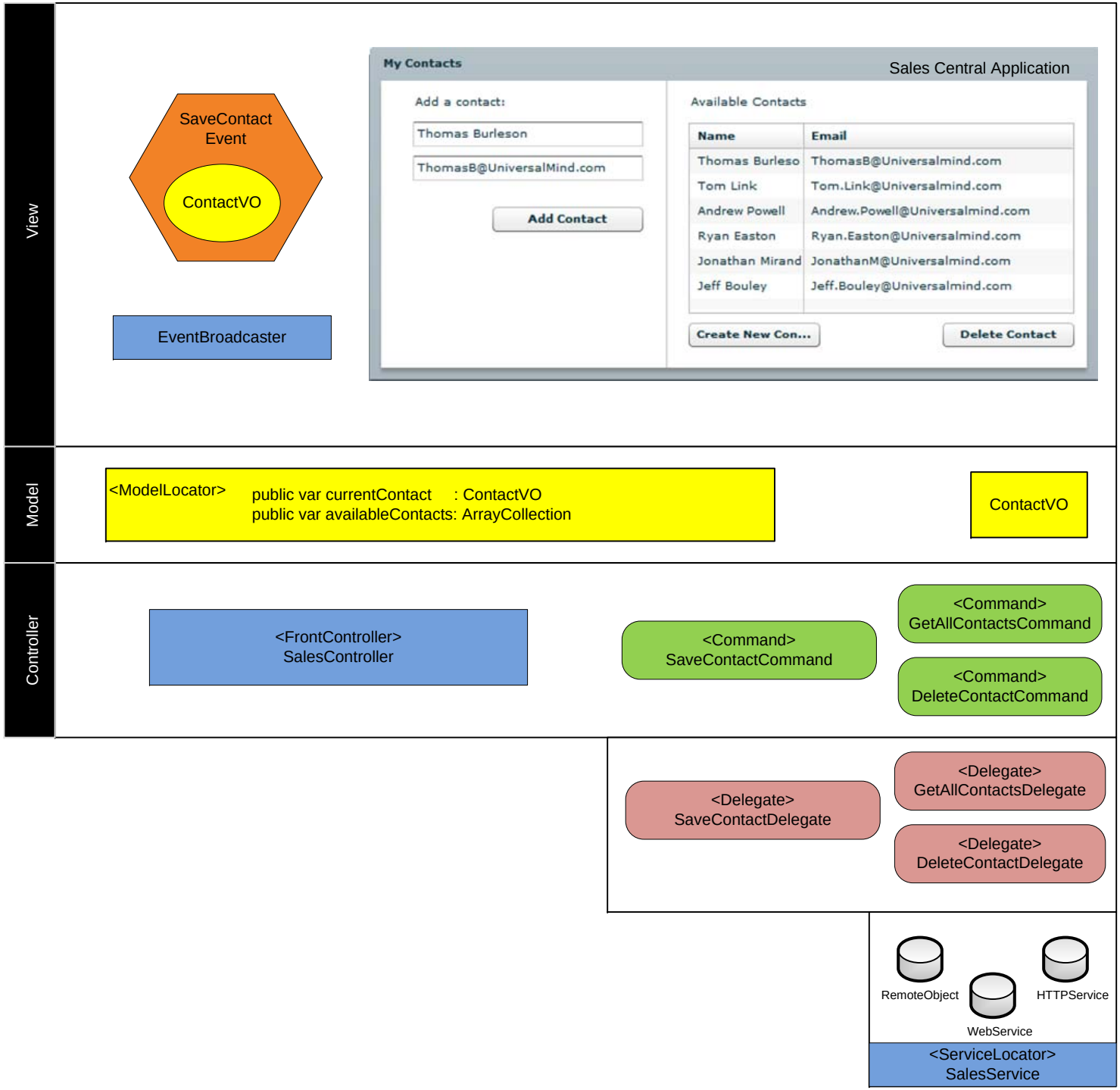
Universalmind Cairngorm MVC

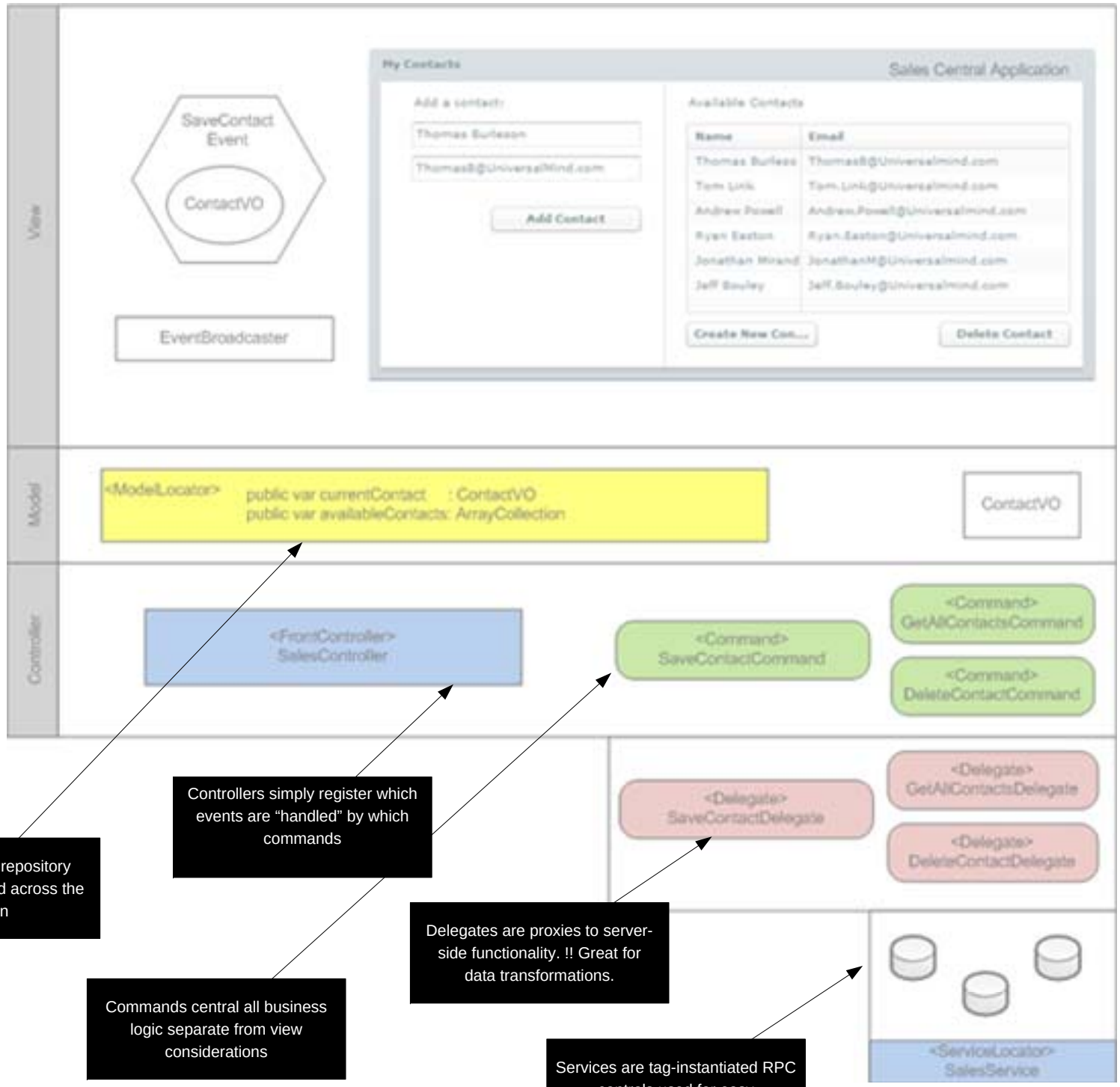
Understanding the conceptual MVC layers, component interactions, and event flows.
Identify the disabilities of classic Adobe Cairngorm to scale with enterprise applications.
Learn about the advantages of Cairngorm MVC using the UniversalMind extensions.

Developer Package Structures

Client-Side Flex Application







ModelLocator is a repository of global data shared across the application

Controllers simply register which events are “handled” by which commands

Commands central all business logic separate from view considerations

Delegates are proxies to server-side functionality. !! Great for data transformations.

Services are tag-instantiated RPC controls used for easy maintenance.

User clicks the “Add Contact” button:

(1) Contact data is gathered from the TextInput controls.

(2) A custom event is created to announce the “intent” to save the contact. The contact data is “packaged” as part of the event

(3) The Cairngorm dispatcher is used to “send” the event to the controller layers.

(4) The Controller registers listeners for each “registered” event.

(5) When a event listener is fired, the command registered with that event is instantiated and “executes” the event.

(6) A custom delegate is used to “proxy” the call to the remote data services

(7) The specific service for that custom delegate proxy is used as the RPC mechanism for remote data services

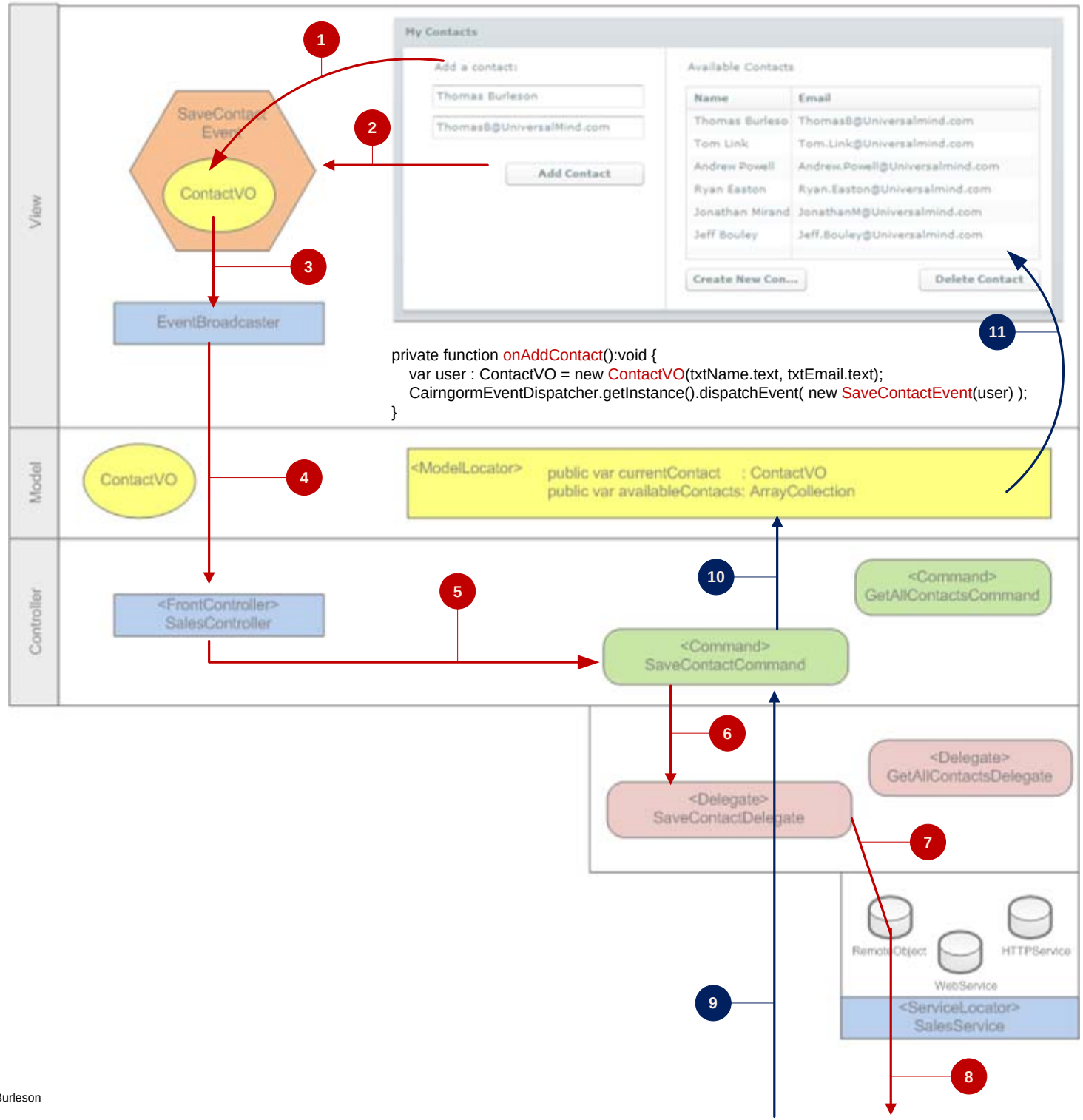
(8) The RPC control coordinates with the Flash player and executes an ASYNCRHONOUS call to the server-tier.

Sometime later...

(9) The response arrives from the asynchronous call and the responder is fired to handle the Result or Fault event.

(10) The command uses the data packaged within the response to create a new contact value object. That value object is “added” to the ModelLocator::availableContacts array.

(11) Databinding fires to “magically” update the grid in the “My Contacts” view.



Issues with Adobe Cairngorm MVC

1) Non-trivial applications become difficult to manage due to large # of class files required; here each event dispatched requires 1 distinct event, 1 distinct command, and [possibly] 1 delegate.

e.g.
An application with 15 events need 45 class files… Yikes!

2) Views are coupled to the CairngormEventDispatcher… thus reducing their usability within other applications.

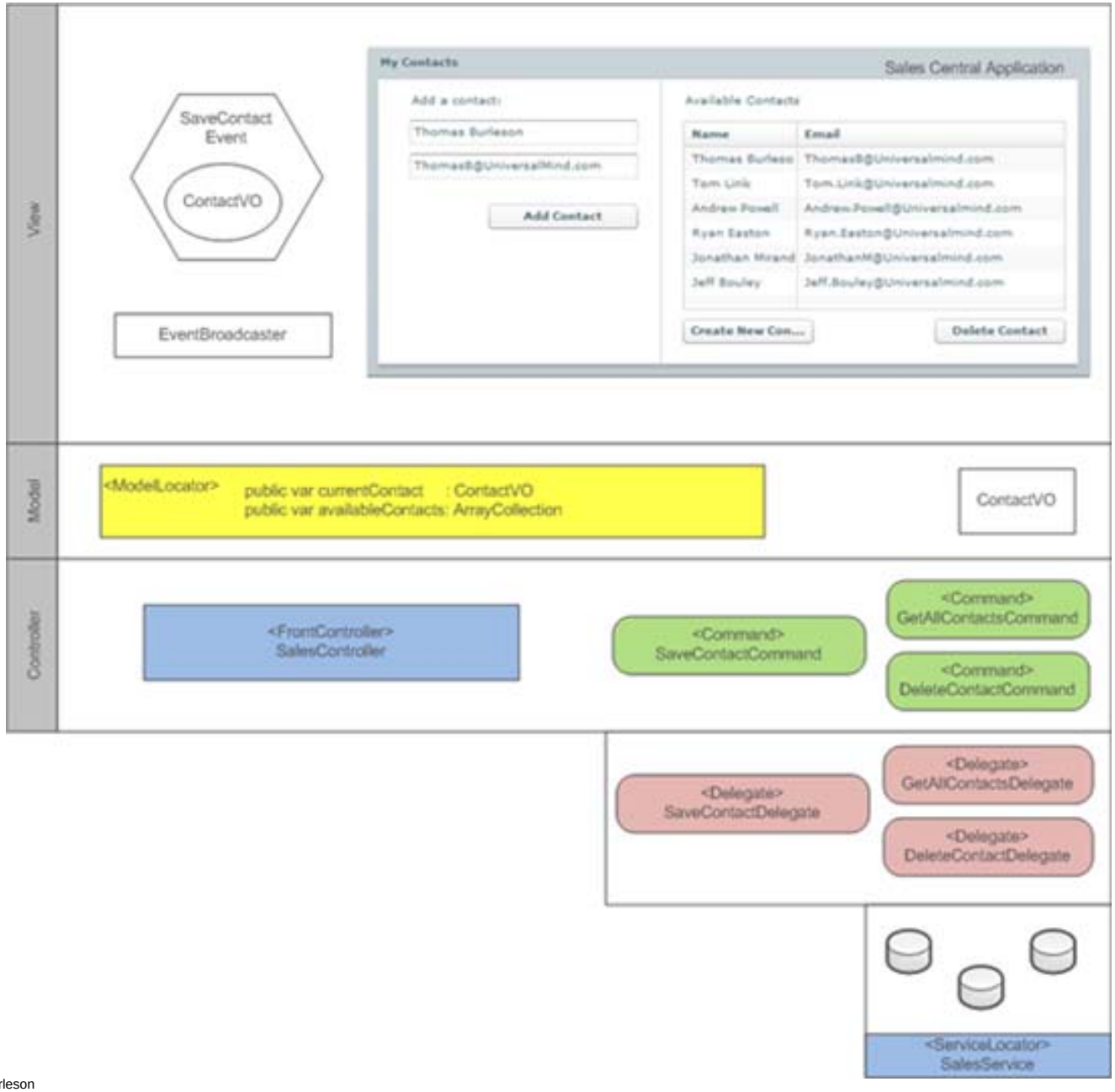
3) It is non-trivial for views to "know" when the asynchronous event has succeeded or failed.

E.g.
We need to disabled the buttons until the contact has been saved on the server…

4) How do views load data that should not be shared / stored within the ModelLocator?

5) How are view-level events [events without any business domain logic] handled and mixed with Cairngorm events?

6) How can the delegates easily perform data transformations before the command responders are invoked?



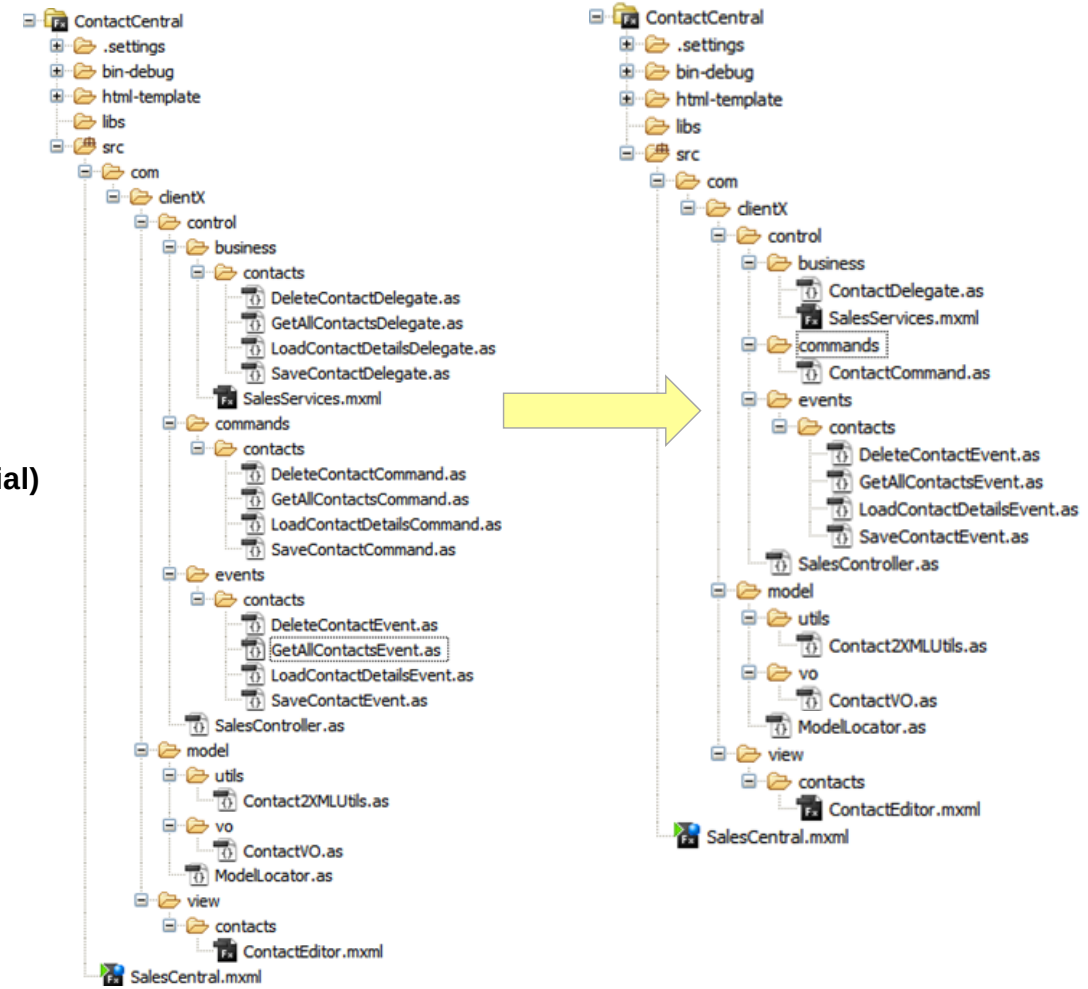
Flex Cairngorm MVC

Class [Adobe] Cairngorm

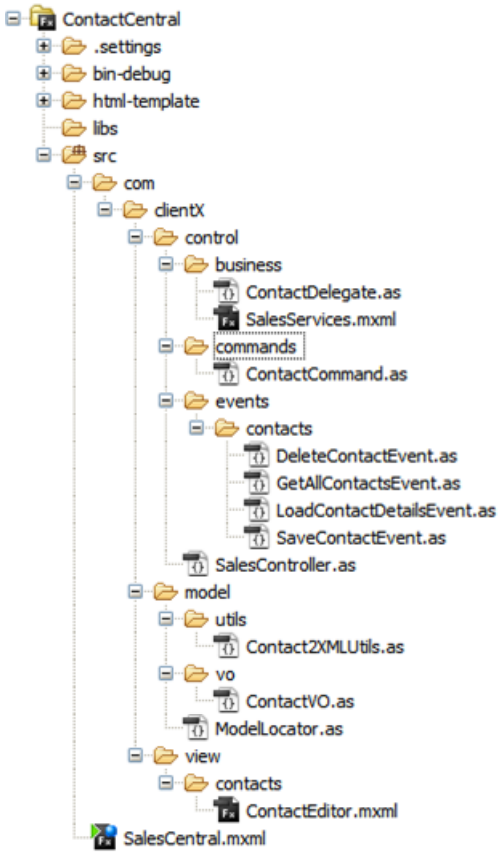
1. Uses MVC pattern
2. Poor support of Webservices initializations
3. Enterprise issues view notifications
4. Modules not supported

Extended with UM Cairngorm

1. Support for view responders
2. Support for command event dispatching
3. Superior support for delegate queues
4. Support for batch events (parallel and/or sequential)
5. Support for delegate-level data transformations
6. Support for command logic aggregations
7. Mini-MVC apps/modules supported
8. Usable for full non-ui FlexUnit testing



Developer Package Structures



Client-Side Flex Application

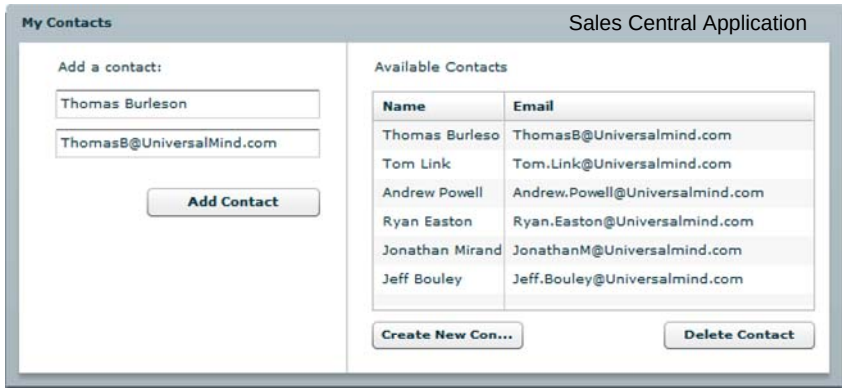
View

Model

Controller

SaveContact Event

ContactVO




```
private function onAddContact():void {
   var user     : ContactVO = new ContactVO(txtName.text, txtEmail.text);
   var handlers : CallBacks  = new CallBacks (onResult_addContact);

   dispatchEvent( new SaveContactEvent(user, handlers) );
}
```

<ModelLocator> public var currentContact : ContactVO
public var availableContacts: ArrayCollection

ContactVO

<FrontController>
SalesController

<Command>
ContactCommand

<Delegate>
ContactDelegate

All business logic for any events related to Contacts is here!

Any data transformations for any Contact data is “hidden” here.

Only Delegates “know” about services.

RemoteObject

HTTPService

WebService

<ServiceLocator>
SalesService

Event Flow w/ UM Cairngorm

(1) Contact data is gathered from the TextInput controls.

(2) A custom event is created to announce the “intent” to save the contact. The contact data is “packaged” as part of the event

(3) The event is dispatched just like mx events and “bubbles” to controller.

(4) The Controller registers listeners for each “registered” event.

(5.1) When a event listener is fired, the command registered with that event is instantiated and “executes” the event.

(5.2) Use the event ID, the appropriate business function is called with specific function as the response handler.

(6) A custom delegate is used to “proxy” the call to the remote data services

(7) The specific service for that custom delegate proxy is used as the RPC mechanism for remote data services

(8) The RPC control coordinates with the Flash player and executes an ASYNCRHONOUS call to the server-tier.

Sometime later...

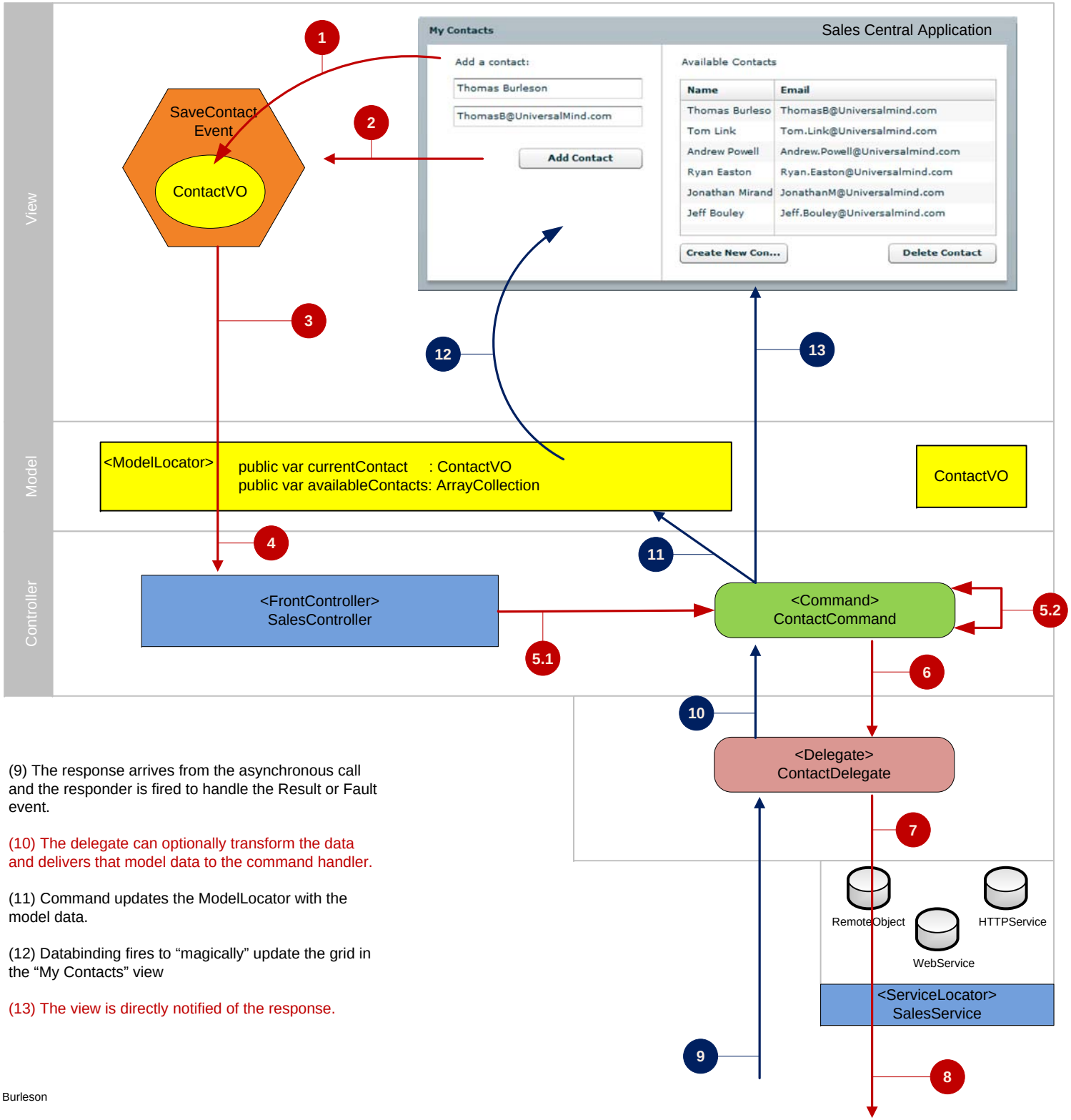
(9) The response arrives from the asynchronous call and the responder is fired to handle the Result or Fault event.

(10) The delegate can optionally transform the data and delivers that model data to the command handler.

(11) Command updates the ModelLocator with the model data.

(12) Databinding fires to “magically” update the grid in the “My Contacts” view

(13) The view is directly notified of the response.



UM Cairngorm Extras

(1) Any UIComponent can use CallBack instances as a “proxy responder” and be used as a Responder to asynchronous calls.

(2) CallBack proxies can be queued. Examples are callback queues from Delegate -> Command -> View method.

(3) EventGenerators can be used to specify queues of parallel or sequential events that must fire in specific orders or dependencies.

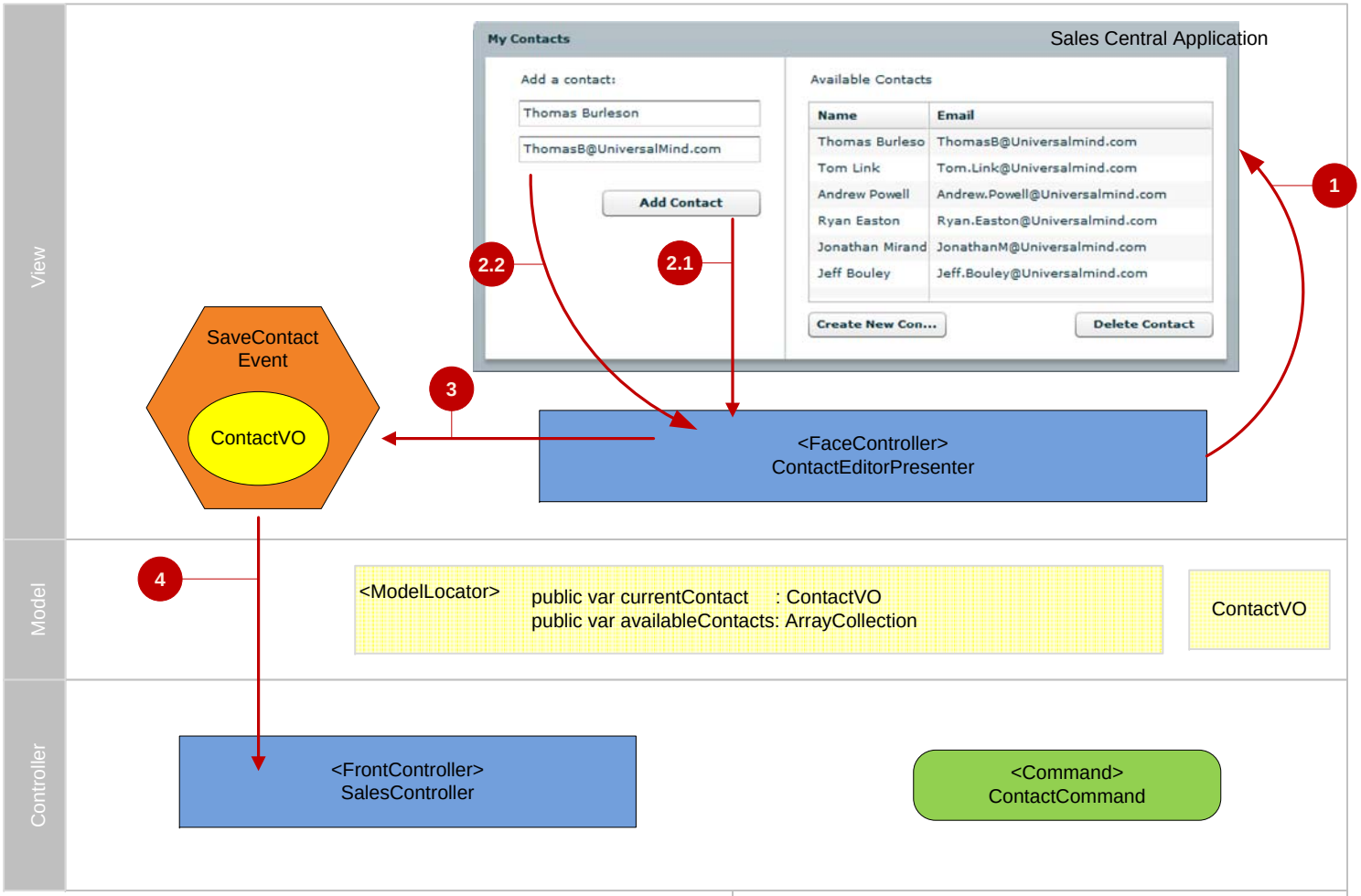
(4) ValueObjects are forced to implement copyFrom(), clone() and equals() methods.

(5) FaceControllers can be used to manage view level events and child relationships. This allows views to be completely isolated from component-interaction and business logic constraints.

(6) Commands can dispatch other events

(7) Common fault handling is supported in the UM Command and fires an AnnounceFaultEvent for any “fault” response.

(8) Modules can have their own proprietary Model-view-controller mini-frameworks… these are easily glued into the primary MVC.



Event Flow w/ UM Cairngorm

(1) Presenter registers listeners for events (click, mouseDown, change, etc.) in ContactEditor view.

(2.1) Click on button triggers listener in Presenter.

(2.2) Contact information is gathered from TextInput controls.

(3) SaveContactEvent is created

(4) Event is dispatched to Controller