

# Petroselinum crispum



# PARSLEY

## Application Framework

by Serge Stepuro



# Parsley

is an

Application Framework for Flex and Flash Applications

built upon an

IOC Container and Messaging Framework

that can be used to create

highly decoupled architectures.



# Features

IOC Container

Dependency Injection

Decoupled Bindings

Messaging

Object Lifecycle

Dynamic View Wiring

Support for Modular Applications

Localization

Extensibility



# Features

IOC Container

Dependency Injection

Decoupled Bindings

Messaging

Object Lifecycle

Dynamic View Wiring

Support for Modular Applications

Localization

Extensibility



# Configuration and Initialization



# Configuration and Initialization

Define managed classes



MXML  
XML  
AS



Configure DI & Messaging



MXML  
XML  
Metadata tags



Initialize IOC



# Configuration and Initialization

## Step #1: MainConfig.xml

<Objects

xmlns:fx="http://ns.adobe.com/mxml/2009"

xmlns:services="com.bookstore.services.\*"

xmlns:actions="com.bookstore.actions.\*"

xmlns="http://www.spicefactory.org/parsley">

<fx:Declarations>

**<services:LoginServiceImpl timeout="3000"/>**

**<actions:LoginAction/>**

</fx:Declarations>

</Objects>



## Configuration and Initialization

### Step #2: LoginAction.mxml

```
package com.bookstore.actions {
```

```
class LoginAction {
```

```
    [Inject]
```

```
    public var service:LoginService
```

```
    [MessageHandler]
```

```
    public function handleLoginEvent (event:LoginEvent) : void {  
        service.login(event.username, event.password);
```

```
    }
```

```
}
```

```
}
```



## Configuration and Initialization

### Step #3: SampleApplication.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<s:WindowedApplication >

    <fx:Declarations>
        <parsley:ContextBuilder config="{MainConfig}"/>
    </fx:Declarations>

    <sample:MainView width="100%"/>
</s:WindowedApplication>
```



# Dependency Injection



# Dependency Injection

## Constructor Injection

### **[InjectConstructor]**

```
class LoginAction {  
  
    private var service:LoginService;  
    private var manager:UserManager;  
  
    function LoginAction (service:LoginService, manager:UserManager = null) {  
        this.service = service;  
        this.manager = manager;  
    }  
  
}
```



# Dependency Injection

## Method Injection

### [Inject]

```
public function init (service:LoginService, manager:UserManager = null) : void {  
    this.service = service;  
    this.manager = manager;  
}
```



# Dependency Injection

## Property Injection

**[Inject]**

```
public var service:LoginService;
```

**[Inject(id="defaultLoginService")]**

```
public var service:LoginService;
```



# Dependency Injection

## Configure Injection in MXML

```
<fx:Declarations>
```

```
  <Object id="loginService" type="{LoginServiceImpl}">  
    <Property name="timeout" value="3000"/>  
  </Object>
```

```
  <Object id="userManager" type="{UserManager}"/>
```

```
  <Object type="{LoginAction}">  
    <ConstructorArgs>  
      <ObjectRef idRef="userManager"/>  
    </ConstructorArgs>  
    <Property name="service" idRef="loginService"/>  
  </Object>
```

```
</fx:Declarations>
```



# Dependency Injection

## Configure Injection in XML

```
<object type="com.bookstore.actions.LoginAction">  
  <constructor-args>  
    <object type="com.bookstore.services.UserManager"/>  
  </constructor-args>  
  <property name="service">  
    <object type="com.bookstore.services.LoginServiceImpl">  
      <property name="timeout" value="3000"/>  
    </object>  
  </property>  
</object>
```



# Decoupled Bindings



# Decoupled Bindings

**[Publish]**[Bindable]

```
public var selectedContact:Contact;
```

**[Subscribe]**

```
public var selectedContact:Contact;
```



```
[Publish(objectId="selectedContact")][Bindable]  
public var selectedContact:Contact;
```

```
[Subscribe(objectId="selectedContact")]  
public var selectedContact:Contact;
```



# Messaging



# Messaging

## Dispatching Messages

### Managed Events

if the dispatching object is a regular  
EventDispatcher.

### Injected MessageDispatchers

if your messages are not subclasses of Event.



# Messaging

## Dispatching Messages: **Managed Events**

```
[Event(name="loginSuccess",type="com.bookstore.events.LoginEvent")]  
[Event(name="loginFailed",type="com.bookstore.events.LoginEvent")]  
[Event(name="stateChange",type="flash.events.Event")]
```

**[ManagedEvents("loginSuccess,loginFailure")]**

```
public class LoginServiceImpl extends EventDispatcher implements LoginService  
{  
  
    [...]  
  
    private function handleLoginResult (user:User) : void {  
        dispatchEvent(new LoginEvent("loginSuccess", user));  
    }  
  
}
```



# Messaging

## Dispatching Messages: **Injected MessageDispatchers**

```
public class LoginServiceImpl implements LoginService {
```

```
    [MessageDispatcher]
```

```
    public var dispatcher:Function;
```

```
    [...]
```

```
    private function handleLoginResult (user:User) : void {  
        dispatcher(new LoginMessage(user));  
    }
```

```
}
```



# Messaging

## Receiving Messages

### MessageHandlers

for methods that should be invoked when a particular message is dispatched

### MessageBindings

for properties that should be set when a particular message is dispatched

### MessageInterceptors

for intercepting, and optionally cancelling or deferring then redispersing a message before it is processed by handlers or bindings

### Error Handlers

for handling errors thrown by other handlers, bindings or interceptors.



# Messaging

## Receiving Messages: `MessageHandlers`

**[`MessageHandler(selector= "loginSuccess")`]**

```
public function loginSuccessHandler(event:LoginEvent ):void {
```

**[`MessageHandler`]**

```
public function handleLogin (message:LoginMessage) : void {
```



# Messaging

## Receiving Messages: `MessageBindings`

```
[MessageBinding(messageProperty="user",  
type="com.bookstore.events.LoginMessage")]  
public var user:User;
```



# Messaging

## Receiving Messages: `MessageInterceptors`



```
public class DeleteItemInterceptor {
```

```
    [MessageInterceptor(type="com.bookstore.events.ShoppingCartDeleteEvent")]
```

```
    public function interceptDeleteEvent (processor:MessageProcessor) : void {
```

```
        var listener:Function = function (event:CloseEvent) : void {
```

```
            if (event.detail == Alert.OK) {
```

```
                processor.proceed();
```

```
            }
```

```
        };
```

```
        Alert.show("Do you really want to delete this item?", "Warning",
```

```
            Alert.OK | Alert.CANCEL, null, listener);
```

```
    }
```

```
}
```



# Messaging

## Receiving Message: Error Handlers

**[MessageError(type="com.bookstore.LoginEvent")]**

```
public function handleError (processor:MessageProcessor, error:IOException) : void;
```

**[MessageError]**

```
public function handleError (processor:MessageProcessor, error:Error) : void;
```



# Object Lifecycle



# Object Lifecycle

## [Init]

```
public function init () : void {  
    [...]  
}
```

## [Destroy]

```
public function dispose () : void {  
    [...]  
}
```

## [Observe]

```
public function enhance  
(panel:ShoppingCartPanel) : void;
```



# Commands



# Commands

## Asynchronous Command Methods

[Command(selector="save")]

```
public function saveUser (event:UserEvent) : AsyncToken {
```

[CommandResult(selector="save")]

```
public function handleResult (user:User, event:UserEvent) : void {
```

[CommandResult(type="com.foo.events.UserEvent" selector="save")]

```
public function handleResult (user:User) : void {
```

[CommandComplete]

```
public function handleResult (event:UserEvent) : void {
```

[CommandError(selector="save")]

```
public function handleResult (fault:FaultEvent, trigger:UserEvent) : void {
```

[CommandStatus(type="com.foo.events.UserEvent", selector="save")]

```
public var isSaving:Boolean;
```



# Commands

## Short-lived Command Objects

```
public class SaveUserCommand {  
  
    [Inject]  
    public var service:RemoteObject;  
  
    public function execute (event:UserEvent) :  
AsyncToken {  
        return service.saveUser(event.user);  
    }  
  
    public function result (user:User) : void {  
        // do something with the result  
    }  
  
    public function error (fault:Fault) : void {  
        // do something with the result  
    }  
  
}
```

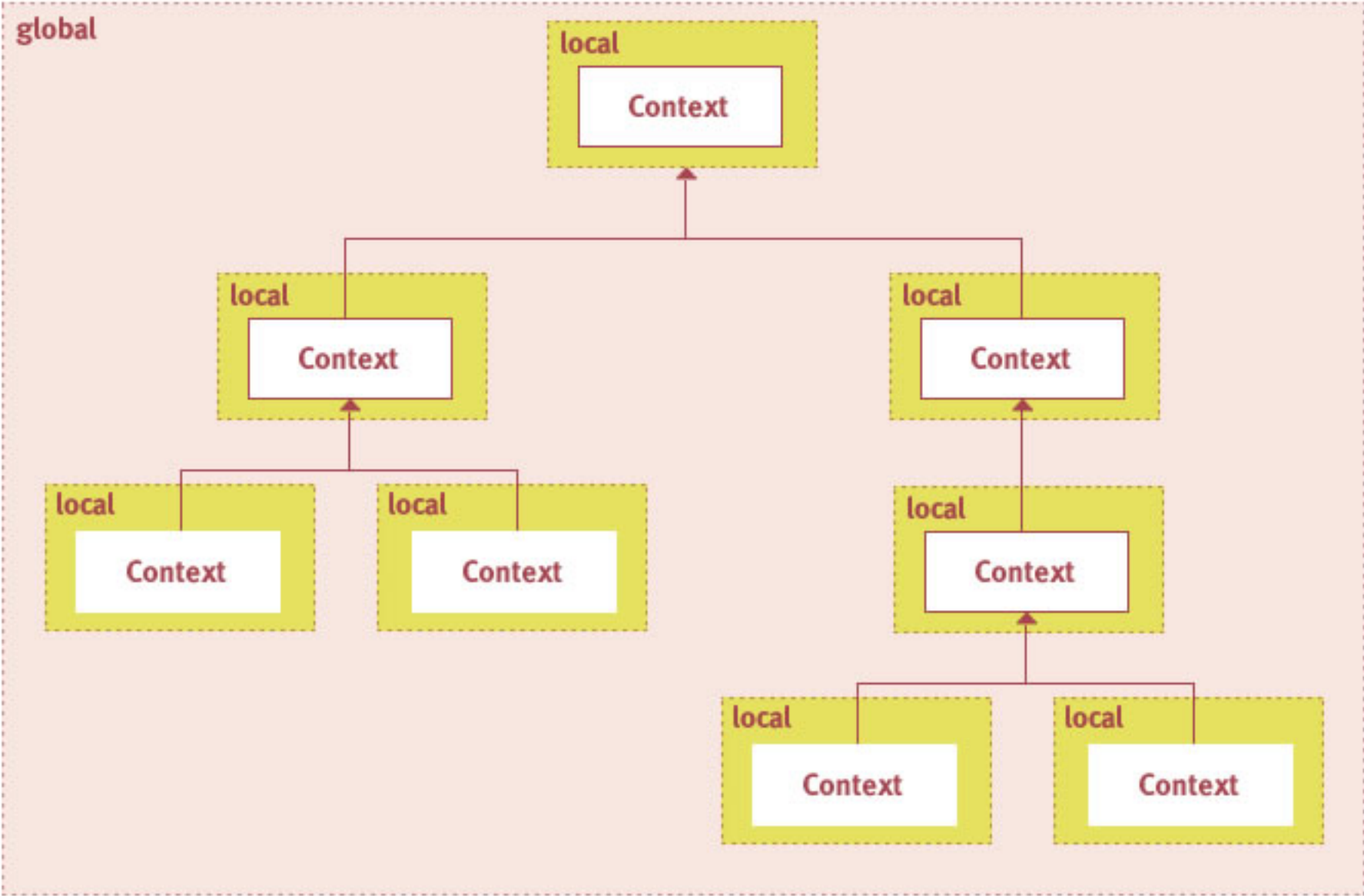
```
<DynamicCommand  
    type="{SaveUserCommand}" "  
    messageType="{UserEvent}" "  
    selector="save"  
    execute="execute" "  
    result="result" "  
    error="error" "  
>
```



# Using Scopes



# Scopes



# Scopes

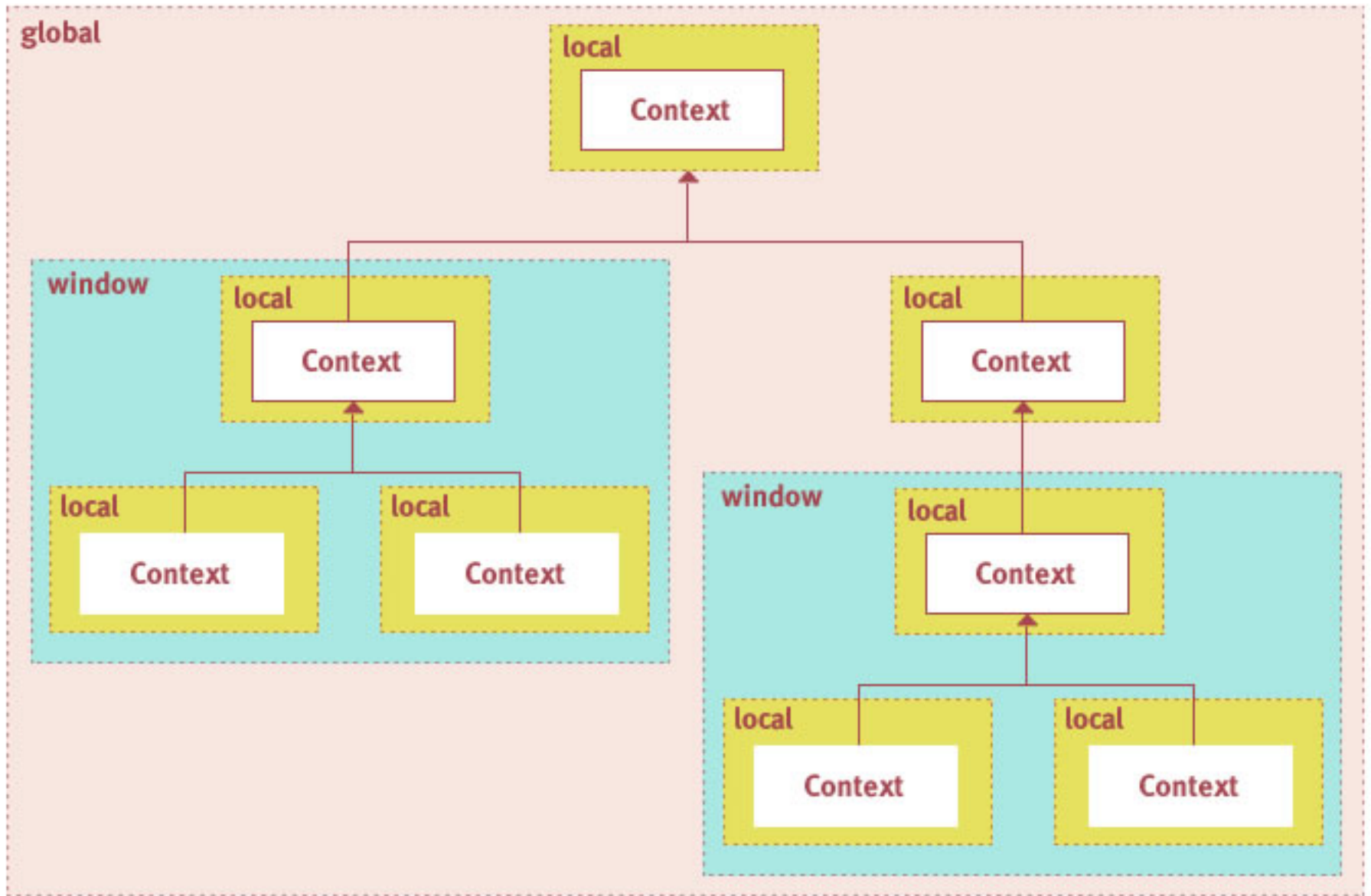
```
[MessageHandler(selector="save", scope="local")]  
public function save (event:ProductEvent) : void {
```

```
[ManagedEvents("save,delete", scope="local")]
```



# Scopes

## Custom Scope: Window



# Scopes

```
<parsley:ContextBuilder>  
  <parsley:FlexConfig type="{ServiceConfig}"/>  
  <parsley:FlexConfig type="{ControllerConfig}"/>  
  <parsley:XmlConfig file="logging.xml"/>  
  <parsley:Scope name="window" inherited="true"/>  
</parsley:ContextBuilder>
```

```
[MessageHandler(selector="save", scope="window")]  
public function save (event:ProductEvent) : void {
```

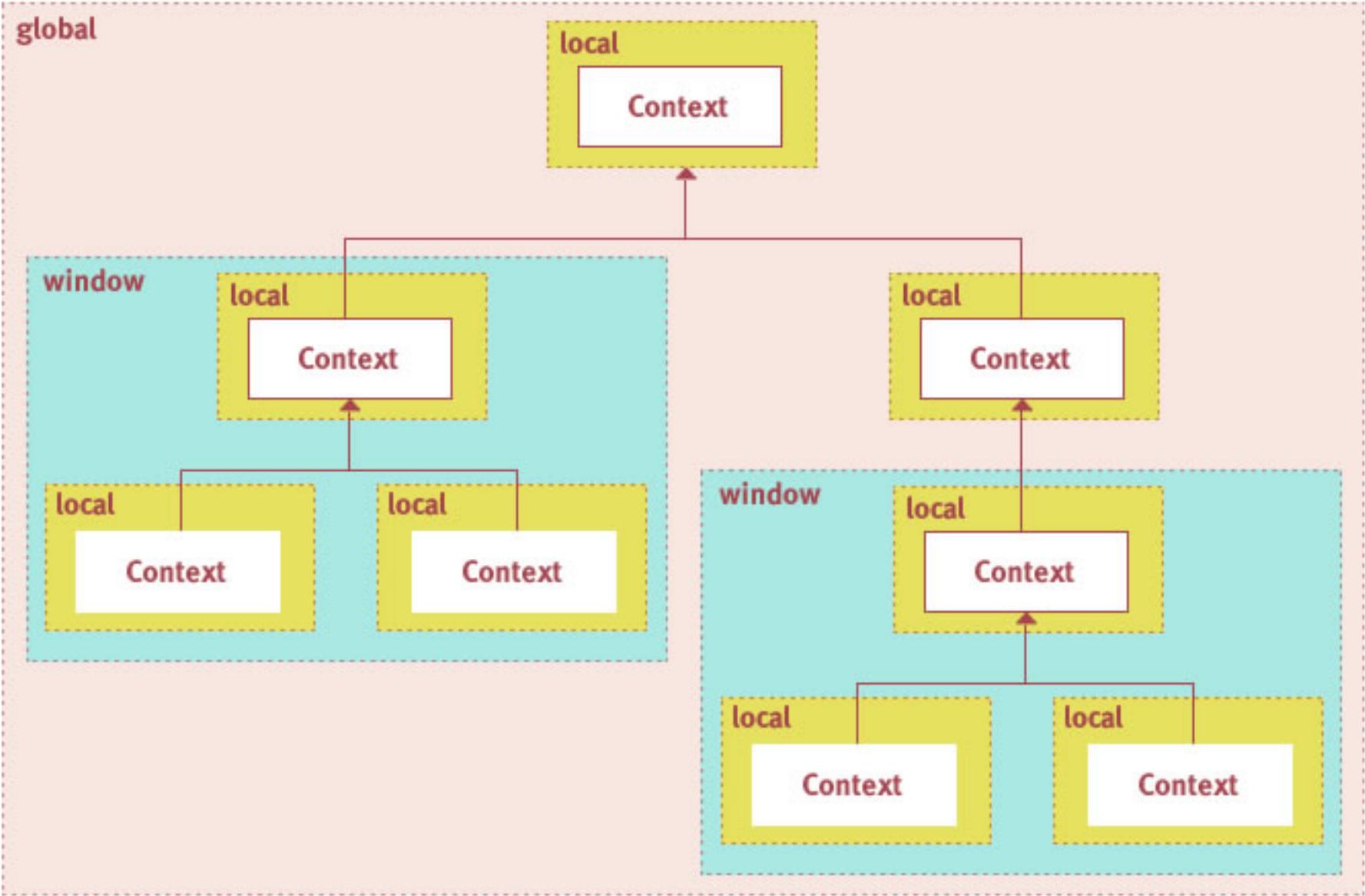
```
[ManagedEvents("save,delete", scope="window")]
```



# Building Modular Applications



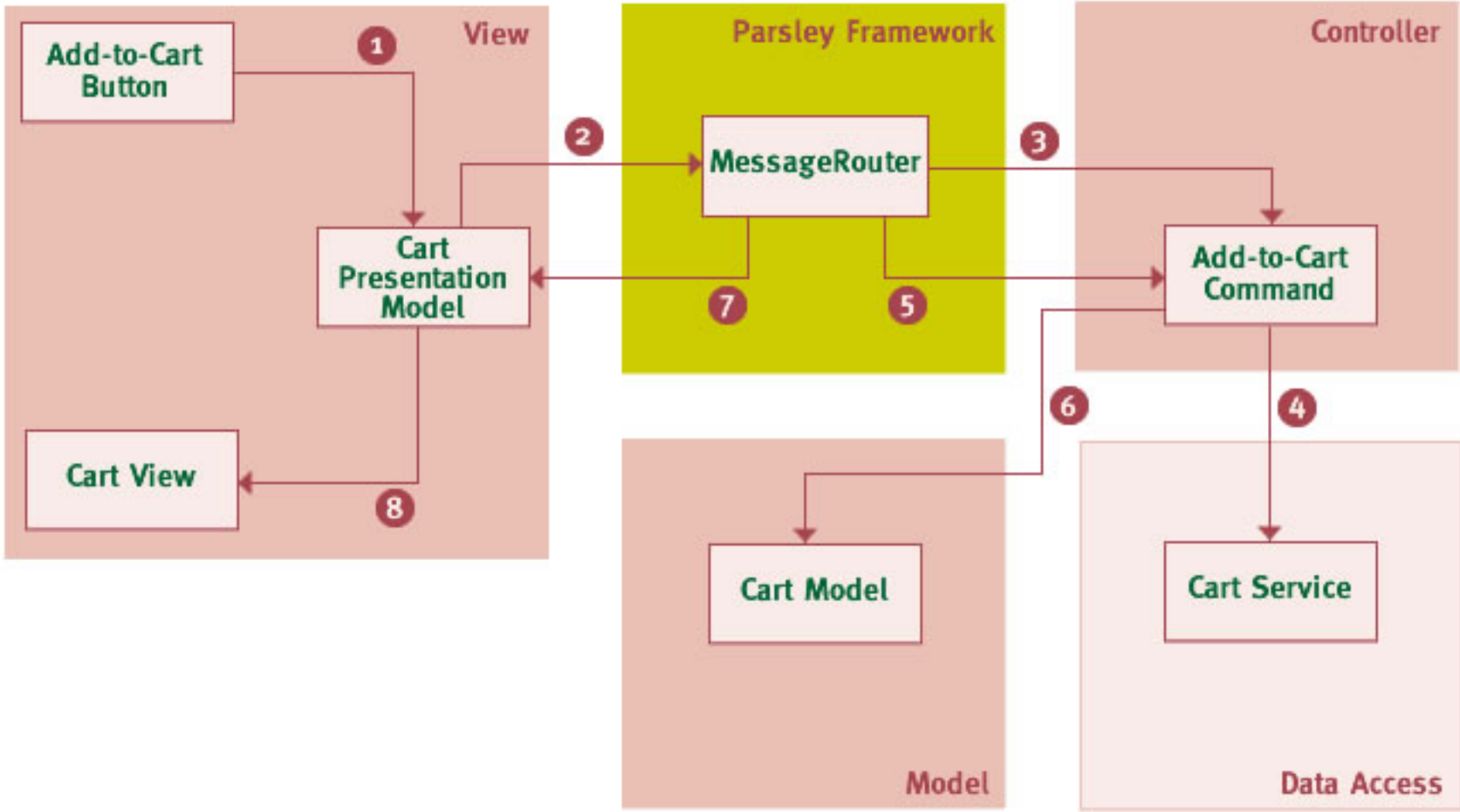
# Building Modular Applications



# Building MVC Architectures



# Building MVC Architectures



# Building MVC Architectures

## **[Observe]**

```
public function mediateLoginView(view:LoginView):void
```



# Remoting



# Remoting

## Flex Remoting

```
<fx:RemoteObject  
    id="loginService"  
    destination="loginService"  
    showBusyCursor="true"/>
```

```
[Inject(id="loginService")]  
public var service:RemoteObject;
```



# Competitors



# [S] Swiz Framework

The brutally simple micro-architecture for  
Enterprise ActionScript development



BAFPUG New Wave 2011

Thanks very much.

Any Questions?



# BAFPUG New Wave 2011

## Useful Links

Developers manual:

<http://www.spicefactory.org/parsley/docs/2.3/manual/>

Good articles with some theory:

<http://artinflex.blogspot.com/2010/09/quick-dive-into-parsley-intro-why.html>

Articles with simple examples.

You can learn main concepts and best practices of Parsley applications. Sometimes parsley or SDK versions are too old so don't try to launch that projects.

<http://blog.neosavvy.com/wordpress/?p=143>

<http://coenraets.org/blog/2009/07/building-a-flex-application-with-the-parsley-framework/>

<http://joelhooks.com/2009/07/14/inversion-of-control-and-dependency-injection-with-flex-using-the-parsley-application-framework-part-2/#comments><http://www.sammur.com/?p=16>

<http://www.sammur.com/?p=16>

Several interesting ideas:

<http://www.rialvalue.com/blog/2010/05/12/properties-file-based-configuration-mechanism-for-parsley/>

<http://blogs.adobe.com/tomsugden/parsley/>

