conga®

TASKRAY

aircall

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

absi
far beyond system integration

Texei

Building better components: reusable and generic

Christian Menzinger
Product Developer
cmenzinger@appero.com
@appero_com

“Please allow me to introduce myself”

Sympathy for the Devil - Rolling Stones

Christian Menzinger

Working for ISV partner appero

3,5 years Salesforce Developer

Munich Developer Group Co-Leader

appero
smart use of the force

salesforce registered ISV partner

http://www.appero.com

@appero_com

appero-com

SALESFORCE CERTIFIED Administrator

SALESFORCE CERTIFIED Force.com Developer

salesforce developers

Developer Group Leader - Munich, Germany

French Touch Dreamin’

Objectives

What can you expect

Climbing up!

Leveraging the framework

Fundamental thinking for Awesome Admins and Users

Code for Developers

Example from a Developer's Life

Calling a server side action using Apex

How to? Google!

Example from a Developer's Life

Google Search Result

Lightning Components Developer Documentation

https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/controllers_server_actions_call.htm

Ever copy & paste this code?

Calling a Server-Side Action

Call a server-side controller action from a client-side controller. In the client-side controller, you set a callback, which is called after the server-side action is completed. A server-side action can return any object containing serializable JSON data.

A client-side controller is a JavaScript object in object-literal notation containing a map of name-value pairs.

Let's say that you want to trigger a server-call from a component. The following component contains a button that's wired to a client-side controller `echo` action. `SimpleServerSideController` contains a method that returns a string passed in from the client-side controller.

```
<aura:component controller="SimpleServerSideController">
    <aura:attribute name="firstName" type="String" default="world"/>
    <lightning:button label="Call server" onclick="{!c.echo}"/>
</aura:component>
```

This client-side controller includes an `echo` action that executes a `serverEcho` method on a server-side controller.

Use unique names for client-side and server-side actions in a component. A JavaScript function (client-side action) with the same name as an Apex method (server-side action) can lead to hard-to-debug issues. In debug mode, the framework logs a browser console warning about the clashing client-side and server-side action names.

```
({
    "echo" : function(cmp) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = cmp.get("c.serverEcho");
        action.setParams({ firstName : cmp.get("v.firstName") });

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                // Alert the user with the value returned
                // from the server
                alert("From server: " + response.getReturnValue());

                // You would typically fire a event here to trigger
                // client-side notification that the server-side
                // action is complete
            }
            else if (state === "INCOMPLETE") {
                // do something
            }
            else if (state === "ERROR") {
                var errors = response.getError();
                if (errors) {
                    if (errors[0] && errors[0].message) {
                        console.log("Error message: " +
                                 errors[0].message);
                    }
                } else {
                    console.log("Unknown error");
                }
            }
        });

        // optionally set storable, abortable, background flag here

        // A client-side action could cause multiple events,
        // which could trigger other events and
        // other server-side action calls.
        // $A.enqueueAction adds the server-side action to the queue.
        $A.enqueueAction(action);
    }
})
```

Is this productional code?

Still some work to do

```
({
    "echo" : function(cmp) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = cmp.get("c.serverEcho");
        action.setParams({ firstName : cmp.get("v.firstName") });

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                // Alert the user with the value returned
                // from the server
                alert("From server: " + response.getReturnValue());

                // You would typically fire a event here to trigger
                // client-side notification that the server-side
                // action is complete
            }
            else if (state === "INCOMPLETE") {
                // do something
            }
            else if (state === "ERROR") {
                var errors = response.getError();
                if (errors) {
                    if (errors[0] && errors[0].message) {
                        console.log("Error message: " +
                                 errors[0].message);
                    }
                } else {
                    console.log("Unknown error");
                }
            }
        });

        // optionally set storable, abortable, background flag here

        // A client-side action could cause multiple events,
        // which could trigger other events and
        // other server-side action calls.
        // $A.enqueueAction adds the server-side action to the queue.
        $A.enqueueAction(action);
    }
})
```

Problem: “I did it my way”

My Way - Frank Sinatra

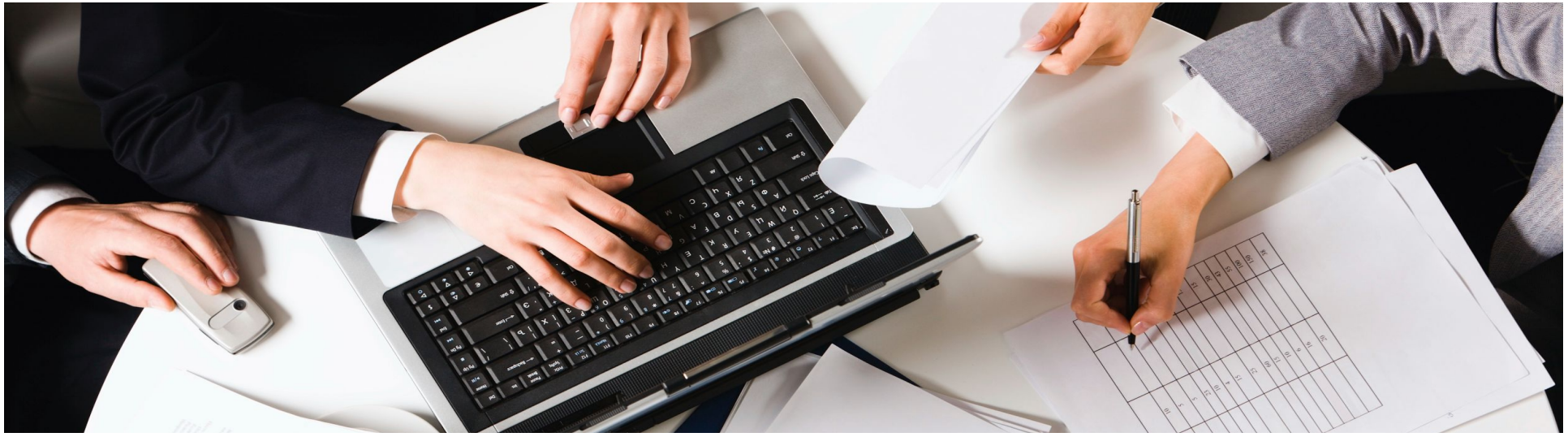
Several developers

Multiple Apex calls

Different code locations

Error-handling: console.log()? alert()? Toast messages? Modal component?

Have to re-Google template code after a while



Idea: “One ring to rule them all”

The Lord of the Rings - J.R.R. Tolkien

Need a consistent behavior

Need one code location

Don’t want to fiddle around with the same code again and again!

Thinking about behavior

OOP: behavior = method

Prototyping: using a helper function

What changes in different scenarios?

Identify what changes

What are the essentials?

focus on:

- Apex function name
- calling parameters
- return values

```
({
    "echo" : function(cmp) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = cmp.get("c.serverEcho");
        action.setParams({ firstName : cmp.get("v.firstName") });

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                // Alert the user with the value returned
                // from the server
                alert("From server: " + response.getReturnValue());

                // You would typically fire a event here to trigger
                // client-side notification that the server-side
                // action is complete
            }
            else if (state === "INCOMPLETE") {
                // do something
            }
            else if (state === "ERROR") {
                var errors = response.getError();
                if (errors) {
                    if (errors[0] && errors[0].message) {
                        console.log("Error message: " +
                                 errors[0].message);
                    }
                } else {
                    console.log("Unknown error");
                }
            }
        });

        // optionally set storable, abortable, background flag here

        // A client-side action could cause multiple events,
        // which could trigger other events and
        // other server-side action calls.
        // $A.enqueueAction adds the server-side action to the queue.
        $A.enqueueAction(action);
    }
})
```

Prototyping: “v1UsingHelperMethodHelper.js”

Only little changes from original code

```
({
    getContactList: function(component, filterString) {
        this.callApex(component, "c.getContactsByFilter", {"filter": filterString}, this.getContactListSuccess);
    },

    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
        action.setParams(actionParameters);

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                //->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
                successCallback(component, response.getReturnValue())
            }
            else if (state === "INCOMPLETE") {
```

Prototyping: “v1UsingHelperMethodHelper.js”

Only little changes from original code

```
31                          console.log("Unknown error");
32                      }
33                  }
34          });
35
36          // optionally set storable, abortable, background flag here
37          // A client-side action could cause multiple events,
38          // which could trigger other events and
39          // other server-side action calls.
40
41          // $A.enqueueAction adds the server-side action to the queue.
42          $A.enqueueAction(action);
43      },
44
45      getContactListSuccess: function(component, returnValue) {
46          //process result in some way
47          component.set("v.contactList", returnValue);
48      }
49  })
```

How far do we get?

“callApex” can be reused for a second server call:

```
({
    getContactList: function(component, filterString) {
        this.callApex(component, "c.getContactsByFilter", {"filter": filterString}, this.getContactListSuccess);
    },

    getOpportunityList: function(component, stageFilter) {
        this.callApex(component, "c.getOpportunityByStage", {"stage": stageFilter}, this.getOpportunityListSuccess);
    },

    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
```

But:

Only possible within the **same** helper

One Apex controller per component

Still different code locations

Reusability

Encapsulate functionality

How can other components use this **abstraction** of an Apex service?

OOP: **class** = **extensible** program-code-template

Idea:

Create an **abstract** base **class** where other components can **extend** from!

Prototyping: “v2AbstractComponent.cmp”

Abstract base class

```
1  <aura:component extensible="true" abstract="true">
2
3
4
5
6
7
8
9
10
11
12 </aura:component>
```

Prototyping: “v2AbstractComponent.cmp”

Behavior

```
1  <aura:component extensible="true" abstract="true">
2
3      <aura:method name="callApex" action="{!c.onCallApex}">
4          <aura:attribute name="component" type="Aura.Component" />
5          <aura:attribute name="controllerMethod" type="String" />
6          <aura:attribute name="actionParameters" type="Object" />
7          <aura:attribute name="successCallback" type="Object" />
8      </aura:method>
9
10     {!v.body}
11
12 </aura:component>
```

Prototyping: “v2AbstractComponentController.js”

Controller invocation from method

```
({
    onCallApex : function(component, event, helper) {
        //get the method parameters
        var params = event.getParams().arguments;
        var controllerMethod = params.controllerMethod;
        var actionParameters = params.actionParameters;
        var successCallback = params.successCallback;
        helper.callApex(component, controllerMethod, actionParameters, successCallback);
    }
})
```

Prototyping: “v2AbstractComponentHelper.js”

Copy v1-Helper-JavaScript: callApex() did not change

```
({
    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
        action.setParams(actionParameters);

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                //->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
                successCallback(component, response.getReturnValue())
            }
            else if (state === "INCOMPLETE") {
```

Prototyping: “v2ConcreteComponent.cmp”

Extend the abstract component

```
1 <aura:component extends="c:v2AbstractComponent"
2     [...]
3 </aura:component>
```

Prototyping: “v2ConcreteComponentHelper.js”

Invoking the method

```
({
    getContactList: function(component, filterString) {
        //call aura method from abstract base component:
        component.callApex(component, "c.getContactsByFilter", {"filter": filterString}, this.getContactListSuccess);
    },

    getContactListSuccess: function(component, returnValue) {
        //process result in some way
        component.set("v.contactList", returnValue);
    }
})
```

How far do we get?

“v2AbstractComponent” is **generic** - no hardcoded dependencies, all information is passed in

“v2AbstractComponent” is **reusable** - even on other projects

Problem:

Problem: One does not simply extend multiple classes

Inheritance is one to one

Sometimes already “used”: `extends=”force:slds”`

OOP: “Favor composition over inheritance”!

Now: “My XYZ-component **IS** a service component”

Better: “My XYZ-component **HAS** a service component”

Use as many other components as you like - that’s how the framework is designed for!

Prototyping: “v3ServiceComponent.cmp”

Copy & modify V2-Component-markup

```
1  <aura:component extensible="true" abstract="true">
2
3      <aura:method name="callApex" action="{!c.onCallApex}">
4          <aura:attribute name="component" type="Aura.Component" />
5          <aura:attribute name="controllerMethod" type="String" />
6          <aura:attribute name="actionParameters" type="Object" />
7          <aura:attribute name="successCallback" type="Object" />
8      </aura:method>
9
10     {!v.body}
11
12 </aura:component>
```

Prototyping: “v3ServiceComponentController.js”

Copy & modify V2-Component-markup

```
({
    onCallApex : function(component, event, helper) {
        //get the method parameters
        var params = event.getParams().arguments;
        var callerComponent = params.component;
        var controllerMethod = params.controllerMethod;
        var actionParameters = params.actionParameters;
        var successCallback = params.successCallback;
        helper.callApex(callerComponent, controllerMethod, actionParameters, successCallback);
    }
})
```

Prototyping: “v2AbstractComponentHelper.js”

Copy V2-Helper-JavaScript: callApex() did not change

```
({
    callApex: function(component, controllerMethod, actionParameters, successCallback) {
        // create a one-time use instance of the serverEcho action
        // in the server-side controller
        var action = component.get(controllerMethod);
        action.setParams(actionParameters);

        // Create a callback that is executed after
        // the server-side action returns
        action.setCallback(this, function(response) {
            var state = response.getState();
            if (state === "SUCCESS") {
                //->HERE WE CALL THE CALLBACK RATHER PROCESS THE RESPONSE
                successCallback(component, response.getReturnValue())
            }
            else if (state === "INCOMPLETE") {
```

Prototyping: “v3JustAnotherComponent.cmp”

Use the service component

```
1  <aura:component>
2
3      <c:v3ServiceComponent aura:id="service" />
4
5      [...]
6
7  </aura:component>
```

Prototyping: “v3JustAnotherComponentHelper.js”

Invoke the service component

```
({
    getContactList: function(component, filterString) {
        //call aura method from service component:
        component.find("service").callApex(component, "c.getContactsByFilter",
    },

    getContactListSuccess: function(component, returnValue) {
        //process result in some way
        component.set("v.contactList", returnValue);
    }
})
```

“Nobody's gonna slow me down”

Highway to Hell - AC/DC

Improve the service component

Example: show a spinner, lock GUI, use storable actions

Reusable also means: other developer have to know how to use your component - **Documentation**!

Conclusion

What you have to do now

Build more components!

Identify recurring tasks / code!

Be **generic** to be **reusable!**

Fiddle around!

Do Trailheads!

RTFM!

Share your problem!

Join a local Developer Group!

Resources

Calling a Server-Side Action
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/controllers_server_actions_call.htm

Using Object-Oriented Development
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/oo_intro.htm

Calling Component Methods
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/js_cmp_methods.htm

Providing Component Documentation
https://developer.salesforce.com/docs/atlas.en-us.lightning.meta/lightning/components_documentation.htm

Code Examples
https://github.com/appero-com/FTD17-Building-better-components-reusable-and-generic

Join a local Developer Group
https://developer.salesforce.com/dugs

conga®

TASKRAY

aircall

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

absi
far beyond system integration

Texeï

Q & A

conga®

TASKRAY

aircall

Capgemini
CONSULTING.TECHNOLOGY.OUTSOURCING

absi
far beyond system integration

Texei

Thank You

Remember to tell us what you think in the event survey
www.frenchtouchdreamin.com/survey/

Christian Menzinger
Product Developer
cmenzinger@appero.com
@appero_com