Generic Designer Support

Overview

SDC is a Design platform, as such, we want to create an easy way to add new design capabilities to SDC.

The SDC design capabilities are provided by the main design tools, onboarding (VNF design), VF design/import and the service design. In addition to the main designe capabilities, SDC also provides the DCAE-DS blueprint design studio.

At the current state, SDC does not have a standard for adding new designers. because of this, any new integration requires the SDC team to create a specific logic in the SDC code base to support such an integration.

The code changes needed are mostly found in the UI since until now we had to create a designer specific logic to support each integration

Solution concept

We want to create a way where the integration of a new designer is seamless to the SDC and will allow new designers integration without requiring any code addition on the SDC side.

The solution will provide several options for showing the designer as part of SDC. The communication between the SDC and the designers on the UI side will be event based and on the back-end side It will be based on the external API's that SDC exposes.

Designer Configuration

As part of the integration between SDC and the different plugins, we uphold the concept that SDC can function without any of the different plugins.

The designers enhance the capabilities of SDC they do not prohibit the SDC core functionality.

As a result, we want to create a mechanize that is configured based to decide what designers are shown in the SDC.

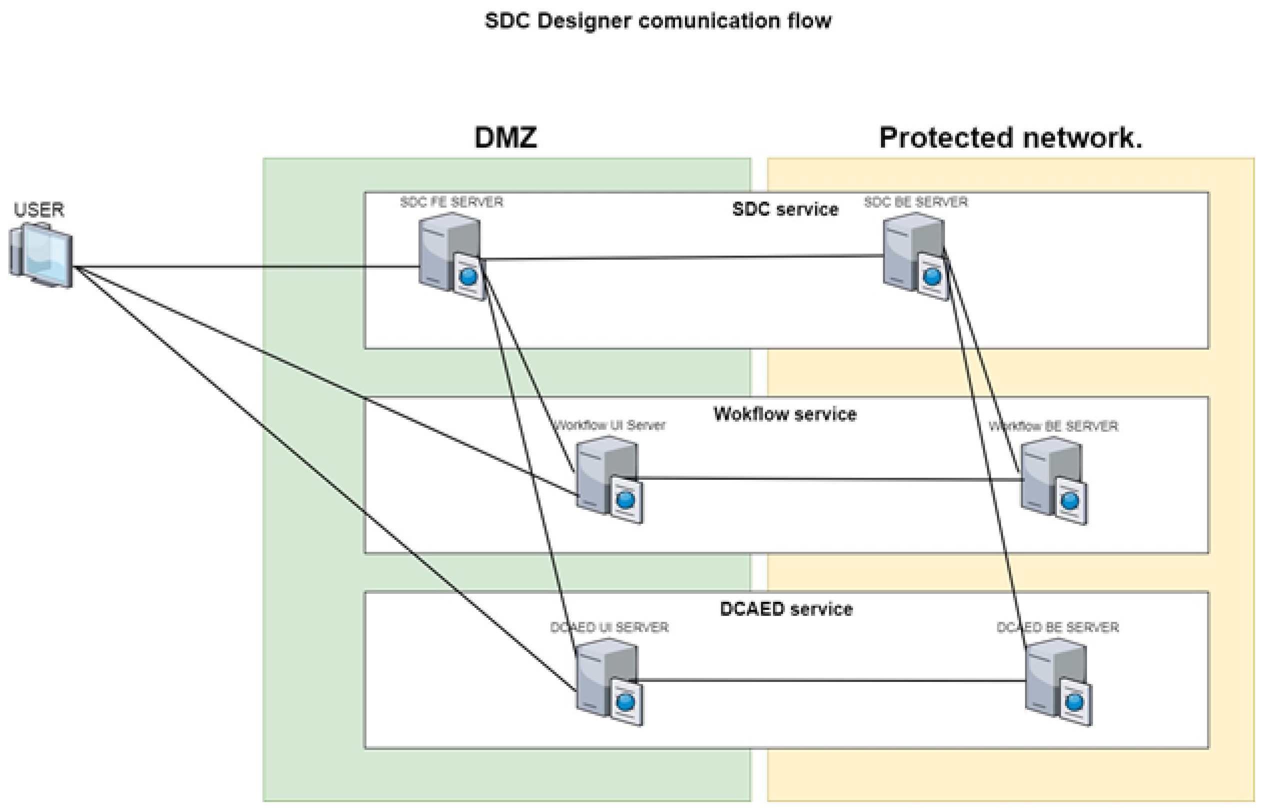
Solution

The availability will be based on a REST HEAD call to the plugin UI server to check the availability of the server.

The SDC UI will execute the call when accessing the plugin page. This way we're making sure that only when the user wants to access the plugin page in the UI the actual online state of this particular plugin will be checked.

High-level overview

The solution will position SDC as a hosting application providing a platform for plugins to enhance the existing capabilities. Each plugin will be treated as a separate service in the eyes of SDC. From the SDC perspective, the plugin can be position anywhere. Because SDC is a multi-tier application we expect the different plugins to uphold the same architecture. There should be a separation between the FE server which is in the DMZ and serves UI pages to the BE server which is in the Protected network, that handles the DB access and the Businesses logic operations. The designer service is loosely integrated with SDC. The two integration points are the plugin UI server location, which is configurable in the FE server configuration and a rest based communication between the plugins backend and SDC backend.



Flow diagram fix TBD

The diagram describes the discovery process from the point when a user access SDC.

Rest endpoint

Request (GET)

The used resource is:

https://{serverRoot}/sdc1/config/ui/plugins

Name	Description
serverRoot	Server base URL hostname+port path Hostname shall contain the FQDN of the SDC FE server or VIP

Special Request Headers

Header Name	Mandatory?	Description

X-ECOMP-RequestID	N	According to the general agreement the "X-ECOMP-RequestID" header with the transaction UUID should be published by component calling an exposed by other component API in order to make possible the transaction traceability across ECOMP. If it is not sent it will be automatically generated by SDC on request receipt.
Accept	N	Determines the format of the body of the response. Valid values are : "application/json"

Request Body

N/A

Request Sample

Request Sample

```
GET  sdc1/config/ui/plugins  HTTP/1.1

Accept : application/json

X-ECOMP-RequestID: AA97B177-9383-4934-8543-0F91A7A02836
```

Response

Special Response Headers

Header Name	Description
Content-Type	Determines the format of the response body. Valid value is : "application/json"
Content-Length	Length of the response body

Response body

Response body is returned as JSON object (Content-Type: application/json).

Parameter Name	Parameter Type	Description
pluginsList	List of objects	List describing each plugin location and configuration for the UI integration.
connectionTimout	Integer	Time in milliseconds for trying to establish a connection to when executing an HEAD request to each plugin

Plugins list:

Parameter Name	Parameter Type	Description
pluginId	String	The id of the plugin
pluginDiscoveryUrl	String	The URL which will be used to check the plugin availability
pluginSourceUrl	String	The URL which will be used to load the plugin page
pluginStateUrl	String	The state URL for the UI router
pluginDisplayOptions	Map of objects	An map of object representing the plugin options, the currently supported options are **tab** and **context**.

PluginDisplayOptions tab:

Parameter Name	Parameter Type	Description
displayName	String	The name to show for this plugin in the tab area.

displayRoles	List of string	The user roles for which this plugin will be shown. Supported roles are: ADMIN, DESIGNER, TESTER, OPS, GOVERNOR

PluginDisplayOptions context:

Parameter Name	Parameter Type	Description
displayName	String	The name to show for this plugin in the context area.
displayRoles	List of string	The user roles for which this plugin will be shown. Supported roles are ADMIN, DESIGNER, TESTER, OPS, GOVERNOR.
displayContext	List of string	A list of contexts in which the plugin will be shown: VFC, VL, CP, VF, PNF, SERVICE.

Response Sample

Response Sample

```
[ {    "pluginId": "DCAE",
    "pluginDiscoveryUrl": "http://<ip>:8702/dcae/#!/index",
    "pluginSourceUrl": "http:// <ip>:8702/dcae/#!/index",
    "pluginStateUrl": "dcae",
    "pluginDisplayOptions": {
      "tab": {
        "displayName": "DCAE",
        "displayRoles": [ "TESTER" ]
      },
      "context": {
        "displayName": "Monitor",
        "displayContext": [ "VF", "SERVICE" ],
        "displayRoles": ["DESIGNER" ]
      }
    }
  },  {
    "pluginId": "WORKFLOW",
    "pluginDiscoveryUrl": "http://<ip>:9527/",
    "pluginSourceUrl": "http:// <ip>:9527/",
    "pluginStateUrl": "workflowDesigner",
    "pluginDisplayOptions": {
      "tab": {
        "displayName": "WORKFLOW",
        "displayRoles": ["DESIGNER","TESTER" ]
      },
      "context": {
        "displayName": "Workflow Designer",
        "displayContext": ["VF"],
        "displayRoles": ["DESIGNER", "TESTER" ]
      }
    }
  }]
```

HTTP response codes

Response code	Service/Policy Exception	Reason /Description
200	n/a	plugin information is returned.

Fe server configuration definition

The plugin configuration is used to define the location where the plugin Front-end server is located.

Based on the defined configuration SDC will provide their information to the SDC UI.

The configuration allows SDC to control where the plugin UI will be displayed, tab mode or context mode.

Configuration fields

The designer configuration is located under {app base}/config/catalog-fe/plugin-configuration.yaml

Field	Type	Description
connectionTimeout	Integer	This is the number of milliseconds, for the head request to wait for a response from the plugin.
pluginsList	List of objects	A list of plugins's configurations.
pluginId	String	The name of the plugin.
pluginSourceUrl	String	The plugin URL passed to the UI to retrieve the page.
pluginDiscoveryUrl	String	The plugin URL used for the availability check.
pluginStateUrl	String	The plugin state for UI router.
pluginerDisplayOptions	String	We currently support two work modes for plugins: 1. tab – opens the designer as a tab, for this type we need to set displayName field to define what the tab text and the display roles which define what roles will see the plugin. example: tab: displayName: "Workflow Designer" displayRole: ["TESTER"] 1. context – opens the designer as a context, for this type we need to set displayName field, displayRole and define the displayContext in which the plugin will open, options VF/SERVICE/CP/VL/VFC. example: context: displayName: "monitor" displayRole: ["TESTER"] displayContext: ["VF", "SERVICE"]

Example

Example

```
pluginsList:
  - pluginId: DCAE
    pluginDiscoveryUrl: <%= @dcae_discovery_url %>
    pluginSourceUrl: <%= @dcae_source_url %>
    pluginStateUrl: "dcae"
    pluginDisplayOptions:
      tab:
        displayName: "DCAE"
        displayRoles: ["TESTER"]
      context:
        displayName: "Monitor"
        displayContext: ["VF", "SERVICE"]
        displayRoles: ["DESIGNER"]
  - pluginId: WORKFLOW
    pluginDiscoveryUrl: <%= @workflow_discovery_url %>
    pluginSourceUrl: <%= @workflow_source_url %>
    pluginStateUrl: "workflowDesigner"
    pluginDisplayOptions:
      tab:
        displayName: "WORKFLOW"
        displayRoles: ["DESIGNER", "TESTER"]
      context:
        displayName: "Workflow Designer"
        displayContext: ["VF"]
        displayRoles: ["DESIGNER", "TESTER"]
connectionTimeout: 1000
```

Health Check Integration - TBD

SDC-1568 - Getting issue details... STATUS

UI Integration

SDC needs to onboard plugins in a way that will be decoupled from the SDC UI implication.

The solution needs to be agnostic to the language in which the plugin UI is developed.

The communication between the hosted and the hosting applications needs to be language agnostic and extendable to allow the addition of more functionality and logic on top of this communication method.

Solution

The solution chosen was to use IFrame to host the plugins. The iFrame provides a decoupled window that can load any UI content as a separate window. The window contains the Html JS and CSS in such a way that they will not interfere with the hosting and hosted application.

The communication between the application will be done based on events.

SDC will hold an event bus in the application, we will define events for operations done in SDC that the plugins need to be notified about. On each defined operation, an event will be placed on the event bus for the plugins to consume.

Supported use cases

Based on the current plugin in SDC we identified two main use cases of how the plugins are used. The first is the presentation of the plugin as a tab on SDC top bar, this mode is used for Plugin that wants to design an object that can later be reused across different SDC elements, the second use case is opening the plugin as part of the service or resource where the designer will model objects in the context and using the properties of that resource/service.

Standalone use case

Use in a context of a service or resource

Iframe initiation

When the Iframe is initiated SDC pass initial information to the Iframe so that the correct plugin screen will be presented. The information will assist the iframe to define the context in which it is currently being opened.

The information will be passed as part of the designer URL as a query param.

Plugin tab mod

In case of a TAB mode, SDC will pass the logged in user.

Param	Description
userId	The id of the user logged into SDC
userRole	The role of the connected user, options ADMIN, DESIGNER, TESTER, OPS, GOVERNOR.
displayType	The type of view the plugin is opened in tab/context.
parentUrl	The URL of the SDC the plugin is opened in. used to subscribe to the event bus in SDC.
eventsClientId	The name with which the plugin needs to register with to the event bus.

Example

http://<IP>:<PORT>/designer?userId=cs0008&userRole=DESIGNER&displayType=tab&parentUrl=http://localhost:9000/&eventsClientId=plugin

Plugin context mod

In case of context mod SDC will pass the following info:

Param	Description
userId	The id of the user logged into SDC.
userRole	The role of the connected user, options ADMIN, DESIGNER, TESTER, OPS, GOVERNOR.
displayType	The type of view the plugin is opened in tab/context.
contextType	The type of the context where the plugin is opened: VF/CP/VL/VFC/SERVICE
uuid	Unique id of the context the plugin is opened in.

lifecycleState	The state the context /resource or service are currently in: READY_FOR_CERTIFICATION/ CERTIFICATION_IN_PROGRESS/ CERTIFIED/ NOT_CERTIFIED_CHECKIN/ NOT_CERTIFIED_CHECKOUT;
isOwner	A Boolean value describing if the logged user is the owner/lastUpdator of the resource/service.
version	The version of the item displayed.
parentUrl	The URL of the SDC the plugin is opened in. used to subscribe to the event bus in SDC.
eventsClientId	The name with which the plugin needs to register with to the event bus.

Example

http://<IP>:<PORT>/designer?userId=cs0008&userRole=DESIGNER&displayType=context&contextType=VF&uuid=a21af8a1daa948f78e30f9b269a253ba&lifecycleState= NOT_CERTIFIED_CHECKOUT&isOwner=true&version=1.0&&parentUrl=http://localhost:9000/&eventsClientId=plugin

UI communication

UI communication is based on event processing. SDC creates an event bus, any plugin in the system and SDC itself will subscribe to this event bus. Events will be generated according to pre-defined events in the system. Any plugin can subscribe to events that are relevant to them.

The plugin can be enhanced so that in the future different plugins can communicate between them using the event bus. The event bus will use javascript for its implementation so that any UI language used will be able to use it.

SDC provides a mechanism to define if there is a need to wait for a response from a plugin or not.

SDC-pubsub library

before you can integrate with SDC you will need to add our pubsub librery in to your application.

SDC will expose a base class in JS as part of the SDC UI project. Any plugin will be able to communicate with SDC after they instantiate this class.

The class will define a set of method that will allow the communication with SDC.

NPM Repository:

https://www.npmjs.com/package/sdc-pubsub

Using the librery

Loading It Up

CommonJS

```
import {PluginPubSub} from 'sdc-pubsub'
```

Global Variable

```
<!-- index.html -->
<script src="./node_modules/sdc-pubsub/dist/sdc-pubsub.js"></script>


// script.js
var pubsub = window.sdcPubSub.PluginPubSub;
```

BasePubSub API

Base class holding all the basic operations needed for using the event hub.

API	Description
constractor(pluginId)	Defines the name of the plugin in the event bus.
register(subscriberId, subscriberWindow, subscriberUrl)	adds the subscriber to the subscriber list.
unregister (subscriberId)	remove the subscriber from the subscriber list.
notify(eventType, eventData)	send with post message the data to all the subscriber after creating the data structure(type, actualData, originId). Returns an object with a subscribe function that receives a callback. This call back will be executed when all subscribers completed their actions regarding the event.
on(eventCallback)	add the callback to the eventsCallbacks map.
off(eventCallback)	removes the callback from the eventsCallback map.
onMessage(event)	Protected method, check if the origin is present in the clients list. Executes all the callbacks from the eventsCallback map.

Call back definition

Each callback function that will be added to the eventsCallbacks map should have the following signature – callback(eventData, event).

eventData – The actual data the will be passed into the callback (The data that will be created in the notify function).

event – The JavaScript event object that was received in the onMessage function.

PluginPubSub API

the client class for communicating with the hub.

API	Description
constructor (pluginId, perentUrl, eventsToWait)	Defines the communication with the hub. PluginId: defines the id of the plugin with which it will be recognized this must correlate to the id defined in the SDC configuration. parentUrl: this is the URL of the parent pub sub hub. This is populated based on the input received in the query param. eventsToWait: this parameter is optional. This defines the events that the plugin wants SDC to wait on before continuing the flow. The regular flow only waits for the call back execution, if the event is defined here sdc will wait for the plugin to respond before continuing the flow.
subscribe (eventsToWait)	The method allows to subscribe again after unsubscribing. the subscriber method is called automatically during the constructor. eventsToWait: this parameter is optional. The method is called automatically by executing the contractor. This defines the events that the plugin wants SDC to wait on before continuing the flow. The regular flow only waits for the call back execution if the event is defined here SDC will wait for the plugin to respond before continuing the flow.
unsubscribe()	The method is used to unsubscribe from the hub.

EventBusService API

API	Description
constructor()	Defines the name of the hub in the event bus.
handlePluginRegistration (eventData, event)	Protected method, handles registration of plugins to the HUB and their un registration.
unregister(pluginId)	Notifies that a plugin is being closed before preregistering it.

onMessage(event)	Protected method, calls the handlePluginRegistration in case of a plugin register or unregister event while passing the event to its subscribers

Plugin API Usage examples

This is an example of how plugin should use the API's defined in SDC event bus.

Plugin initialization

As part of the request SDC does to the plugin to retrieve the index.html we pass query prams.

From the query prams you will need to use eventsClientId this is the id defined which will be passed to the contractor of the class for registration, this allows SDC to understand what plugin is sending the events and parentUrl which is the URL of SDC which is needed to send massages to SDC in a system working cross site.

The second example shows how to set the event the plugin wants sdc to wait for a response.

In this case SDC will continue the flow only after the ACTION_COMPLETED event is received.

Example

```
eventsClientId=<received from query params>
parentUrl=<received from query params>
PluginPubSub client = new PluginPubSub(eventsClientId, parentUrl)
```

Example with events to wait for

```
eventsClientId=<received from query params>
parentUrl=<received from query params>
eventsToWaitFor = [ “CHECK_IN” ]
PluginPubSub client = new PluginPubSub(eventsClientId, parentUrl, eventsToWaitFor)
```

Send ready event

As part of the plugin flow, the plugin will need to send the ready event to SDC so that SDC will know the plugin has loaded successfully.

Example

```
client.notify(“READY”)
```

Listen to events

The plugin will need to define a call back that will be called when an event is triggered in SDC.

The plugin call back can decide using the event type what and if any logic needs to be executed.

For example:

Example

```
client.on((eventData,event) => {
      if(eventData.type == ”WINDOW_OUT”) {
            //do logic
       }
   }
)
```

Event Types

The events used in the communication will have the same structure to allow easy use.

Each event will include its type the generating entity and a body.

In this way, new events can be easily added to the system and the event originator can be tracked.

Event structure

```
{

      type: "eventType"

      data: {}

      originId: "originName"

}
```

SDC generated events

Name	Type	When	Data	UI state	Description
Window out	WINDOW_OUT	Before SDC closes plugin, The event is sent on all events below except VERSION_CHANGE and check out	none	Moves out of the plugin scope	The event is posted by SDC once the plugin window is going to be closed because of user action.
Version change	VERSION_CHANGE	After SDC retrieves item	New item version and UUID	The plugin is displayed on screen.	The event is posted by SDC once context version is changed by the user.
Check in	CHECK_IN	Before SDC closes plugin	none	SDC opens the catalog.	The event is posted by SDC once the context is being checked by the user.
Check out	CHECK_OUT	After SDC successful checkout the item	New item version and UUID	The plugin is displayed on screen.	The event is posted by SDC once the context is being check out by the user.
Submit for testing	SUBMIT_FOR_TESTING	Before submitting for testing is executed	none	SDC opens the catalog.	The event is posted by SDC once the context is being submitted for testing by the user.
Undo check out	UNDO_CHECK_OUT	After SDC executed the undo check out.	item version and UUID	SDC opens the catalog.	The event is posted by SDC once the context is being undone by the user.

Window out event example

```
{
    type: "WINDOW_OUT"
    data:
    originId: "sdc-hub"
}
```

Version change event example

```
{
    type: "VERSION_CHANGE"
    data: {
        uuid: a21af8a1daa948f78e30f9b269a253ba ,
        version:1.1
    }
    originId: "sdc-hub"
}
```

Check in event example

```
{
    type: "CHECK_IN"
    data:
originId: "sdc-hub"
}
```

Check out event example

```
{
    type: "_CHECK_OUT"
    data: {
        uuid: a21af8a1daa948f78e30f9b269a253ba ,
        version:1.1
    }
    originId: "sdc-hub"
}
```

Submit for testing event example

```
{
type: "SUBMIT_FOR_TESTING"
data:
originId: "sdc-hub"
}
```

Undo check out event example

```
{
    type: "UNDO_CHECK_OUT"
    data: {
        uuid: a21af8a1daa948f78e30f9b269a253ba ,
        version:1.1
    }
    originId: "sdc-hub"
}
```

Designer events

Name	Type	Description
Ready	READY	The event is posted by the plugin once it is ready.
Action completed	ACTION_COMPLETED	The event is sent by the plugin after receiving an event from SDC that the plugin added to the eventsToWait list in order to notify SDC to continue with its flow.

Ready event example

```
{
    type: "READY"
    data:
originId: "pluginName"
}
```

Action completed event example

```
{
    type: "ACTION_COMPLETED"
    data:
originId: "pluginName"
}
```

Security TBD

Need to add solution for the iframe authorization and authentication.

SDC-1569 - Getting issue details... STATUS

SDC-UI

to allow a single look and feel in SDC we need to create a single design style definition.

The user using the system should not distinguish between the different designer studios in SDC, the experience should be unified across the different design activities.

SDC UI Style and component library

SDC developed a UI library include the style guide fonts and components used in SDC.

The library components were developed in angular 2 and react to allow easy integration of the component in different design application.

The library allows the different designers to align their style to the SDC to create a single look and feel.

The library is located here: the library is in the process of moving to LF.

SDC Designer infra structure

SDC will aim to provide a robust infrastructure to allow new designers to easily integrate with SDC. SDC will to create a generic way for designers to add them self's to SDC without the need to add designer specific code to SDC.

SDC external API's

Designer will leverage the existing external API's provided by SDC. The API's are protected by basic authentication.

Any new designer will need to request a user for their designer to interact with the SDC external API's.

more information in the Appendix.

Appendix A – References

SDC external API's

SDC consumer definition