**CIC Integration to SFDC**

**Integration Patterns**



Prepared for Bombardier Inc.

September 8, 2015

**Contents**

[**Overview** 4](#_Toc429469852)

[**Objectives** 4](#_Toc429469853)

[**Integration Architecture** 5](#_Toc429469854)

[**Context** 5](#_Toc429469855)

[**The Problem** 5](#_Toc429469856)

[**The Project: define SFDC integration patterns** 5](#_Toc429469857)

[**Development Environment and Technologies** 6](#_Toc429469858)

[**Portlet Integration Patterns** 6](#_Toc429469859)

[**REST versus SOAP** 7](#_Toc429469860)

[**Hacking and Security Concerns** 8](#_Toc429469861)

[**Hacking** 8](#_Toc429469862)

[**Communication Protocol** 8](#_Toc429469863)

[**Passwords** 8](#_Toc429469864)

[**Spoofing/Hijacking** 8](#_Toc429469865)

[**SQL Injection** 8](#_Toc429469866)

[**Denial of Service** 8](#_Toc429469867)

[**Salesforce Applicative Security and Content Filtering** 9](#_Toc429469868)

[**Goals** 9](#_Toc429469869)

[**Query Filtering** 9](#_Toc429469870)

[**Portlet Development Standards** 10](#_Toc429469871)

[**Portlet Standard** 10](#_Toc429469872)

[**Portlet Preferences** 10](#_Toc429469873)

[**Inter-portlet communication** 10](#_Toc429469874)

[**Portlet Namespacing and Name Collisions** 10](#_Toc429469875)

[**Recommended Integration Pattern** 11](#_Toc429469876)

[**SFDC Integration Pattern** 11](#_Toc429469877)

[**Additional Considerations** 13](#_Toc429469878)

[**Security** 13](#_Toc429469879)

[**Multi-lingual Support** 13](#_Toc429469880)

[**Recommended Portlet Development Pattern** 14](#_Toc429469881)

[**Hybrid WCM/Client-side Portlet Pattern** 14](#_Toc429469882)

[**Appendix A** 16](#_Toc429469883)

[**Traditional Server-side Portlet Pattern** 16](#_Toc429469884)

[**Custom XSLT/CSS Portlet Pattern** 18](#_Toc429469885)

[**IFrame/Web Application Bridge Pattern** 20](#_Toc429469886)

[**Digital Data Connector Pattern** 22](#_Toc429469887)

[**IBM Script Portlet Pattern** 25](#_Toc429469888)

[**Forms Experience Builder Pattern** 27](#_Toc429469889)

[**Client-side Portlet Pattern** 28](#_Toc429469890)

# **Overview**

The goal of this project is to define SFDC Integration patterns. The integration patterns will be delivered to demonstrate the following:

## **Objectives**

1. Lower SFDC license costs within **Total Cost of Ownership** (TCO) evaluations
2. Ensure no violation of SFDC license agreement
3. Ensure business agility in modifications (business users must be able to modify portal to match SFDC change without IT involvement)
4. Data displayed must respond WebSphere Portal themes and look and feel

# **Integration Architecture**

## **Context**

Bombardier Business Aircraft (BBA) is migrating its customer service portal (CiC) to the websphere portal solution (WSP). The new platform is based on the Commercial Aircraft (BCA) eServices platform developed in 2012-14 timeframe. The new CiC portal must be in production by end of Q2 2016 and migration work is planned to start in January 2016.

BBA has a Customer Relationship Management (CRM) solution built on the salesforce.com (SFDC) platform and in continuous evolution. BBA is developing (with Deloitte) a case management solution (CASE) on the SFDC platform starting in May 2015, due to be completed by a yet to be determined date. BBA also leverages the Service Max (MAX) solution on the SFDC platform. SFDC thus has become the *de facto* repository for all information regarding customers, orders, and customer support issues. BBA also stores customer information (profile, fleet) in a master customer record database (CDR), built as part of the eServices project.

## **The Problem**

The portal must display information from the SFDC CRM, CASE and MAX solutions in its pages.

Today this is done via custom designed HTML pages developed in SFDC APEX language in the community+ platform and integrated into the portal via a Web Application Bridge (WAB).

The solution has a number of disadvantages:

* It consumes SFDC community+ licenses;
* It is difficult to evolve as it requires changes to SFDC APEX code, which adds costs and delays;
* WAB integration is not appropriate for all use cases (we want to integrate SFDC data, not existingweb pages)

## **The Project: define SFDC integration patterns**

The project must deliver the integration patterns to integrate SFDC data into pages of the WebSphere CiC portal according to our objectives and use cases.

# **Development Environment and Technologies**

* Rational Application Developer 9.1.1
* WebSphere Portal 8.5
* Maven 3
* Spring 4.1
* Handlebars 3.x
* JSR 286 portlets
* GIT

# **Portlet Integration Patterns**

The portlet integration patterns defined are based upon several assumption:

1. They must leverage the services and adapter patterns as a proxy to the backend SFDC REST-based calls
2. The solution is based on a single portlet that is entirely configurable through the use of the CONFIGURATION and EDIT DEFAULT modes
3. Integration with WCM is the preferred solution as a web content store for XSLT, Handlebar HTML templates, CSS, etc.
4. The displayed data is entirely customizable via CSS (defined in the theme or overridden as required by the portlet)
5. Business users will have the skills necessary to publish and extend content as required
   1. i.e. Writing basic SFDC queries, extending HTML forms, publishing to WCM
6. The User-experience demonstrated by the portlets is not meant to be used to end-users as it’s still only in the prototype stages.
7. The technologies should reduce the TCO and minimize complexity for developers, business users, and administrators

# **REST versus SOAP**

The SOAP and REST based Salesforce APIs include comparable functionality. The decision to use REST over a SOAP-based interface include the following:

|  |  |
| --- | --- |
| Motivation | Description |
| Direct REST-based delivery without the need to serialize and de-serialize objects | Traditionally object serialization is required to convert the resulting data into Java objects on the server and then de-serialize into a format that the client-side code can use. |
| Data Payload | The preferred payload for the requests are provided in JSON format. This offers a human and machine readable format, similar to XML and is directly consumed by client-side browser applications.  The payload is easily extensible and can be directly injected into the HTML templating technologies. It’s significantly easier to navigate over than using the complex XSL and other processing technique for client-side traversal |
| Reduced administrative impact | The administrative setup for the SOAP interface can be considerable for the Application Server as well as any intermediary servers. The REST-based APIs for Salesforce only required configuration to accept the SSL certificate. |
| Significantly reduced development effort to template resulting JSON data sets | The JSON payload can be quickly transformed into HTML. Handlebars allows web developers to build HTML-based templates that include placeholder text and inject the JSON data values.  XML based content must be manipulated using XSLT templates and require considerable expertise, management, and processing to deliver final HTML renderings. |
| REST-based reads may be cached | The data requests may be cached on both the server as well as the client side. |
| REST-based services are becoming the industry standard | i.e. REST-based services for the IBM Connections platform is entirely REST-based and the payload has been shifting from the ATOM-based ‘standard’ to a more flexible JSON payload |

# **Hacking and Security Concerns**

## **Hacking**

The ability to gain access to a system must be prevented at all cost. There are mechanisms built into the Portlet, as well as the Integration model that help to prevent hackers from gaining access.

## **Communication Protocol**

SFDC requires that SSL communication be used. All REST-based calls are invoked between the WebSphere Application Server and SFDC. To facilitate this communication, the SSL certification from SFDC was added to the WebSphere Application Server trust-store to facilitate the secure calls.

## **Passwords**

The service accounts and passwords must be defined for SFDC service accounts. These account details for the prototype are stored in a property file defined in the Enterprise Application and deployed to the Application Server.

Moving forward, these passwords can be externalized and integrated into a secure trust store.

## **Spoofing/Hijacking**

The primary factor is eliminating the opportunity to high-jack the requests between the client and the backend system. The payload from the client may be altered however, the identity of the individual user cannot because it’s injected by the Adapter on the server.

## **SQL Injection**

SQL Injection attacks are not possible because the CRUD operations are handled entirely on the server and the REST-based services cannot be directly access via the client.

## **Denial of Service**

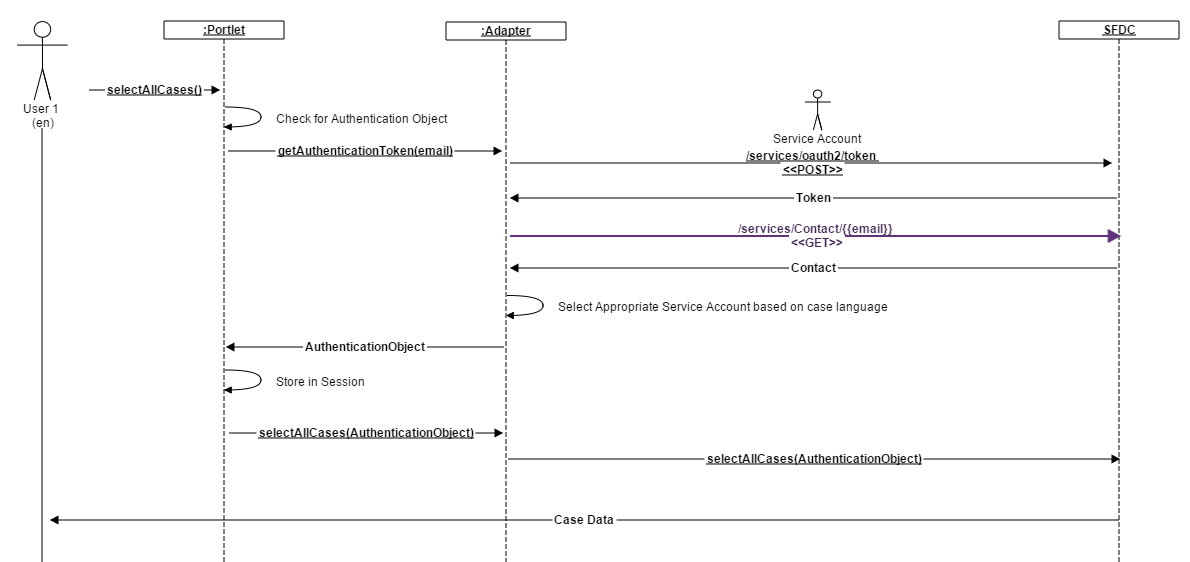
Performance tuning and throttling efforts must be made to avoid unwanted and unwarranted access to the system. There are several opportunities to adjust the web and other thread pools on the Application Server to account for optimal.

The amount of data stored in the HTTP Session objects will be limited as well. The use of the cache (whether the Portlet session or the WebSphere DynaCache) will be explicitly defined and implemented.

# **Salesforce Applicative Security and Content Filtering**

## **Goals**

* One service account will be used to define **language** and **roles**
  + English Service Account
  + French Service Account
* Leverage the SFDC Applicative Security model
  + This will significantly reduce the overhead of duplicating and maintaining the authorization model in both Salesforce and WebSphere Portal
* A new Contact object will be provisioned for each user in SFDC
* The Contact will define the language for the particular user and the Adapter will select the service account based on the contact information



## **Query Filtering**

Authentication is managed by the WebSphere Application Server. When the user logs in, the WebSphere Portal profile information is made available to the portlet. A unique identifier, (i.e. email) will be used to retrieve the Contact details from SFDC.

The Contact object indenter is injected by the adapter for all calls. By injecting the logged in user’s contact id, results will be restricted to fields and operations that a user has access to in SFDC.

All REST-based calls are obfuscated from the client and the web page/JavaScript. This eliminates the ability to intercept and change the requests. Then authentication token is also sent and is only available for the length of the user’s browser session.

For example, provided the following query for cases:

*q=Select Id, Description, CaseNumber, Status From Case*

The Adapter will automatically adjust the query to append the logged in user’s contact id:

*q=Select Id, Description, CaseNumber, Status From Case* **where** ***contactId=123456789***

# **Portlet Development Standards**

## **Portlet Standard**

The JSR 286 (or Portlet 2.0) specification is the latest portlet specification and is fully supported by WebSphere Portal 8.5. It has replaced the JSR 168 specification and corrected many of the limitations. To a large extent, this technology should remain as vendor-agnostic as possible. Not necessarily with the intent of moving away from one vendor over another, but instead of minimize the impact of future migrations.

## **Portlet Preferences**

To facilitate configuration of the portlet, portlet preferences should be leveraged. These preferences leverage the feature defined in the portlet specification to read and store preferences at run time. These preferences can be updated without the need for code deployment and allow administrator, page editors, and possibly users to control the intended behavior of a portlet.

## **Inter-portlet communication**

When multiple portlets are added to a portal page, data may be shared between portlets on the same page or across pages through a number of mechanisms. In addition, a client-side inter-portlet communication model will be leveraged to ensure the best quality user experience.

## **Portlet Namespacing and Name Collisions**

One of the advantages of the Portlets over traditional web development and Single Page Architecture (SPA) is that multiple portlets may be added to the page. This offers the ability to include many disparate but related windows into business services.

To facilitate multiple portlet instances on the same page, particular care must be taken to uniquely identifier (aka namespace) HTML elements on the page. For example. Form names and IDs, JavaScript variable names, and style elements must be considered.

The Hybrid Portlet will ensure that namespace is implemented by wrapping the entire portlet view under a <div> element and using a specialized JavaScript encapsulation technique to avoid method name collisions.

# **Recommended Integration Pattern**

## **SFDC Integration Pattern**

|  |  |
| --- | --- |
| **Description** | A custom service adapter pattern is used as an intermediary between custom Java-based portlet calls and the backend SFDC platform. |
|  |  |
| **Problem** | How can custom Java-based portlets communicate with the SFDC (or any REST-based API) platform while maintaining extensibility, scalability, and customizability? |
|  |  |
| **Solution** | The SFDC Integration Pattern offers an effective and repeatable solution to execute **read**, **create**, **update**, and **delete** operations to SFDC.  The solution integrates through intermediary sources and is extensible enough to support caching, paging, and other extensions. The solution supports responses for both **JSON** and **XML** result sets.  At this time, the prototype facilitates communication between the SFDC REST-based API and Java. The client invokes service calls through a set of service and adapter interfaces. In the future, additional adapters may be added and enhanced and integrated through configuration.  Since the solution is Spring based, the service and adapter classes can be swapped out with new or updated classes as required using configuration files.  For example, a SalesforceMockService may be created and defined to provide sample Salesforce data and facilitate Unit Test checking prior to compilation and deployment of the code.  To streamline the services and adapter, JSON has been selected as the mechanism to transfer data from the client portlet through to Salesforce.  The Spring Rest Templates have been leveraged to provide communication between Java Web-based applications and Salesforce.  A custom ClientHttpRequestInterceptor was created to pass the required headers and authentication tokens between the Portlet and Salesforce. |

|  |  |
| --- | --- |
|  |  |
| **Sequence Diagram** | Note: The **SERVICE\_URI** is a configurable as part of the service query  The Authentication Token should be persisted in the user session. (i.e. Portlet Session) |
|  |  |
| **Results** | |  |  | | --- | --- | | Skills | Advanced Java EE, Java, Spring, and Web Development skills are required | | Integration | The service adapter pattern can be extended to support any REST-based backend systems | | User experience | N/A | | Deployment | A continuous integration environment should be setup to simplify deployment  The service pattern is externalized from the portlet model | | Business Agility | Technically not a direct mapping between the business user and the backend systems, however as requirements changes, adapters may be altered and adjusted to meet any business need. | | Look and Feel | N/A | |

|  |  |
| --- | --- |
|  |  |
| **Working Code Reference** | [**https://github.com/spece/leoneinc/tree/master/leone-connections**](https://github.com/spece/leoneinc/tree/master/leone-connections)  The services and adapters are defined in the **leone-connections** project. |
|  |  |
| **Cost Analysis** | Required license: **Salesforce licenses are required for the Service Accounts**  Maintenance:   * Refactoring is required to facilitate true SSO and principal user definitions are currently defined in a property file due to the scope of the project * Code deployment is required as adapters are added or enhanced * The SFDC queries and object mappings are controlled through configuration * JSON or XML result sets largely eliminate the need to perform any type of data-binding between Portlet and SFDC   + Typical solutions bind the results between the portlet, service, and adapter calls * Visual representations and web content components is entirely controlled through Portlet configuration and WCM |

## **Additional Considerations**

### **Security**

* Service accounts must be created to facilitate multi-lingual support for queries
* Adapter will inject query filters on the server-side to ensure results are specific for each user call
* The Portlet will inject the user credentials prior to invoking the service calls
* Authentication token must be cached in the user session

### **Multi-lingual Support**

* Multiple service accounts must be defined on SFDC (one for each language to support)
* A Picklist REST-based service extension must be created to support the multi-language for dropdowns
* Result sets will be returned in the appropriate language specified for each service account

# **Recommended Portlet Development Pattern**

## **Hybrid WCM/Client-side Portlet Pattern**

|  |  |
| --- | --- |
| **Description** | A custom portlet solution that’s meant to leverage best-of-breed technologies. The goal of this solution is to reduce complexity, development and deployment costs by allowing developers to quickly managed and extend object queries trough configuration.  HTML-like template mappings to SFDC object code are created by HTML and JavaScript web developers and these templates are stored in WCM to simplify code promotion. |
|  |  |
| **Problem** | How can a custom portlet offer a simplified and customizable set of queries and presentation templates to retrieve data and easily create, update, and delete objects in salesforce without developing server-side components? |
|  |  |
| **Solution** | This solution attempts to simplify the development model into the two logical areas:   1. **Integration** – provided through the *SFDC Integration Pattern* 2. **Presentation** – provided through a client-side framework   New instances of the portlet allow the following configurations:   1. List query statements 2. Details query statements 3. HTML Template selection    1. i.e. Point to the location of the WCM Content Item that contains the rendering templates for the list, details, and form templates 4. Content type for data result sets    1. i.e. XML or JSON |
|  |  |
| **Sequence Diagram** | C:\Users\Sal\SkyDrive\Business\Leone Enterprise Consulting Inc\Clients\Bombardier\Work\Sequence Diagrams\Final\Hybrid Portlet Pattern.png |
|  |  |
| **Results** | |  |  | | --- | --- | | Skills | Client-side web development skills (HTML, JavaScript, JQuery, CSS)  WCM Authoring training for business users | | Integration | The SFDC Integration Pattern is leveraged and configured through the Portlet preferences | | User experience | A client-side solution is leveraged to provide an easily extensible framework to build solutions  At this time **Handlebars** is used to quickly render any data returned to the client | | Deployment | The Portlet is deployed once and multiple instances are configured to render the appropriate content.  WCM libraries are coupled through configuration.  Either WCM Syndication or Portal Application Archive (PAA) deployments will satisfy environment promotion requirements | | Business Agility | Business users may quickly update and create forms | | Look and Feel | The theme will contain the core CSS and styling elements for the site, however each portlet may override the styling with its own styling as required. | |
|  |  |
| **Working Code Reference** | [**https://github.com/spece/leoneinc/tree/master/leone-portlet**](https://github.com/spece/leoneinc/tree/master/leone-portlet)  The **leone-portlet** project works in conjunction with WCM as the repository for the Handlebar templates used to render the SFDC JSON data.  It uses a set of Portlet preferences to define properties such as:   * WCM Library * Site area for templates * Salesforce Object types * Salesforce list queries * Salesforce detail object query * WCM CSS components * Content Type to render (JSON or XML) |
|  |  |
| **Cost Analysis** | Required license: N/A  Maintenance:   * Deployments will only be required as changes are made to the portlet itself * Adding pages, rendering templates, styling, and extending queries will be configurable * WCM syndication or Portal Application Archive deployment may be leveraged |

# **Appendix A**

## **Traditional Server-side Portlet Pattern**

|  |  |
| --- | --- |
| Description | Custom portlet where business logic is hard-coded in Java within the portlet  A flexible option but requires deployment to production |
|  |  |
| Problem | How do we develop a simple server-side portlet based solution to integrate with SFDC and leverage the skills of traditional portlet developers? |
|  |  |
| Solution | The Server-side portlets will leverage the Service Integration Pattern to query, create, update, and delete SFDC data  Custom JSP Tag libraries are used to create the rendering views for the queried result sets, and HTML forms must be created to invoke actions in the portlet controller class |
|  |  |
| Sequence Diagram |  |
|  |  |
| Results | |  |  | | --- | --- | | Skills | Advanced Java EE, Java, HTML, CSS, and Web Development skills are required | | Integration | To integrate with the backend SFDC, the SFDC Integration Pattern would still be leveraged | | User experience | Less than desirable server-side and page refreshes for actions | | Deployment | A continuous integration environment should be setup to simplify deployments | | Business Agility | Requires direct IT involvement | | Look and Feel | Fully integrated with theme | |
|  |  |
| Working Code Reference | [**https://github.com/spece/leoneinc/tree/master/leone-portlet**](https://github.com/spece/leoneinc/tree/master/leone-portlet)    The Portlet project can be transformed to support the traditional server-side model  The Spring-based Portlet MVC model has become a standard when developing server-side portlets |
|  |  |
| Cost Analysis | Required license: **N/A**  Maintenance:   * Not easily extensible * Tightly coupled to Java objects for rendering * Page-refreshes on actions results in poor user experience * Introduces a higher cost development model, requiring many more developers, and requires more architect and project manager involvement |

## **Custom XSLT/CSS Portlet Pattern**

|  |  |
| --- | --- |
| Description | Custom portlet where XSLT transforms the received XML data uses CSS to style the result. Portlet exposes configuration fields to select the XSLT and CSS, allowing a single portlet to be reused without new deployment to production. |
|  |  |
| Problem | How can XML-based data retrieved from SFDC be rendered as HTML data? |
|  |  |
| Solution | The introduction of an XML based solution offers the ability to introduce XSL transformations to convert the payload from the SFDC queries into HTML. However there are several limitations to this solution.   * The XSL must be stored in a web content repository or database. * The XSLT transformations require a mostly server-side transformation model, consuming resources from the server rather than leveraging the underutilized client-side browser technologies * The IBM implementation of the standard XSLT Transform has difficulties performing the transformations as defined by the Java EE specifications   **Note**: DataPower could be leveraged to solve the performance and XSL warehouse issues, but complicates the architecture and adds cost |
|  |  |
| Sequence Diagram |  |

|  |  |
| --- | --- |
|  |  |
| Results | |  |  | | --- | --- | | Skills | Advanced Java EE, Java, HTML, CSS, and Web Development skills are required.  XSLT skills are required to adequately construct the presentation layer | | Integration | To integrate with the backend SFDC, the Integration Service Layer will be leveraged | | User experience | Less than desirable server-side and page refreshes for actions | | Deployment | A continuous integration environment should be setup to simplify the deployment | | Business Agility | Requires direct IT involvement. Business users are not well versed in XSLT | | Look and Feel | Fully integrated with theme | |
|  |  |
| Working Code Reference | [**https://github.com/spece/leoneinc/tree/master/leone-portlet**](https://github.com/spece/leoneinc/tree/master/leone-portlet)  The portlet is configurable and allows the administrator to select XML as the content type to be returned by queries |
|  |  |
| Cost Analysis | Required license: **N/A**  Maintenance:   * XSL files must be stored and updated on their context store * Portlet configuration will be leveraged to indicate the location of the XSLT files and define SFDC queries * The SFDC Integration Pattern will be leveraged to integrate with SFDC |

## **IFrame/Web Application Bridge Pattern**

|  |  |
| --- | --- |
| **Description** | Current solution. Web pages created in VisualForce and integrated in the portal as iFrames. May lose some flexibility in display look and feel. |
|  |  |
| **Problem** | How can pages and content developed through VisualForce be quickly exposed via the Portal? |
|  |  |
| **Solution** | The existing Web Application Bridge has several disadvantages.   * SFDC community + user licenses are consumed * It is difficult to evolve as it requires changes to SFDC APEX code, which adds costs and delays   **Note: WAB Integration is not appropriate for all use cases defined** |
|  |  |
| **Sequence Diagram** |  |
|  |  |
| **Results** | |  |  | | --- | --- | | Skills | Portal administration skills are required to manage the portlet deployment and configuration | | Integration | To integrate with the backend SFDC, the Integration Service Layer would still be leveraged | | User experience | Less than desirable, not easily customized, unable to leverage many of the portlet features such as finer-grained security, user credentials, inter-portlet communication, etc. | | Deployment | N/A | | Business Agility | Changes must be implemented on the Salesforce side | | Look and Feel | Unable to leverage the styling in the existing theme | |

|  |  |
| --- | --- |
|  |  |
| **Working Code Reference** | N/A : **Currently implemented in the existing solution** |
|  |  |
| **Cost Analysis** | Required license:  **Consumes SFDC community+ licenses**  Maintenance:   * Changes must be implemented in Salesforce |

## **Digital Data Connector Pattern**

|  |  |
| --- | --- |
| Description | Proposed integration pattern by IBM. Appears complex and difficult to create and maintain. |
|  |  |
| Problem | How can content from Salesforce be consumed through WebSphere Portal and presentation rendered through the use of customizable components? How can this solution be extended to support create, update, and delete operations? |
|  |  |
| Solution | The Digital Data Connector Pattern allows an intermediary service to be configured to act as the integration layer between Salesforce and the portlet.  Presentation templates, authoring templates and content stored in WCM, can be leveraged to render the required data  Additional HTML forms must be created in WCM to create, update, and delete content from Salesforce. The construction of these forms may be complex.  It might be recommended to leverage DDC for visual representation of queries, and Form Builder to create and update content. |
|  |  |
| Sequence Diagram |  |
|  |  |
| Results | |  |  | | --- | --- | | Skills | Portal administration skills are required to manage the portlet deployment and configuration.  Portlet Development, CSS, HTML and many Web Content Management experience is required to adequately deliver this solution. | | Integration | Integration is accomplished via the **Digital Data Connector** implementation, but may not satisfy all the SFDC REST-based requirements | | User experience | Less than desirable, not easily customized, and uses a page-refresh paradigm to perform actions | | Deployment | N/A | | Business Agility | Business users could potentially generate their own forms, but this scenario is not realistic given the complexity involved in creating these forms | | Look and Feel | Presentation templates and other authoring tools may be leveraged to meet any look and feel requirements.  Generally all styles defined in the theme can be reused. | |
|  |  |
| **Working Code Reference** | N/A |

|  |  |
| --- | --- |
|  |  |
| **Cost Analysis** | Required license: **Service accounts must be defined.**  **This *should* be able to automatically add the additional filtering piece to ensure that applicative security model is maintained.**  Maintenance:   * Advanced WCM and programming experience is required to maintain and extend forms * Syndication/Deployment is required to push changes to production   + Alternatively the Portal Application Archive package may be leveraged * This is a relatively new technology, requires very specific knowledge to use, and even more specific knowledge to implement properly   Limitation:  The SFDC REST-based services require that the PATCH HTTP Method be used to perform updates. I have been unable to locate information whether or not this is supported by the DDC.  I did find the following:  **The data sink supports simple HTTP PUT, POST, and DELETE operations for XML-based remote APIs** |

Reference: [**http://www-01.ibm.com/support/knowledgecenter/SSYJ99\_8.5.0/social/plrf\_use\_gen\_xml\_ddc\_datasink.dita?lang=en**](http://www-01.ibm.com/support/knowledgecenter/SSYJ99_8.5.0/social/plrf_use_gen_xml_ddc_datasink.dita?lang=en)

Creating the forms – Document complexity

[**http://www-01.ibm.com/support/knowledgecenter/SSYJ99\_8.5.0/social/plrf\_sendata2wcv\_createhtmlform.html?lang=en**](http://www-01.ibm.com/support/knowledgecenter/SSYJ99_8.5.0/social/plrf_sendata2wcv_createhtmlform.html?lang=en)

## **IBM Script Portlet Pattern**

|  |  |
| --- | --- |
| **Description** | Variation on the Java portlet using JavaScript as the main language. Evaluate if appropriate, performance is adequate and if not deprecated. |
|  |  |
| **Problem** | How can client-side only technologies be used to develop custom portlet applications and simplify code promotion? |
|  |  |
| **Solution** | The proposed solution offered by the Script Portlet is to eliminate the complexity of writing server-side Java code and client-side JavaScript and HTML.  Client-side web developers should be able to leverage their existing skillset to create functional web based applications |
|  |  |
| **Sequence Diagram** |  |

|  |  |
| --- | --- |
|  |  |
| **Results** | |  |  | | --- | --- | | Skills | Portal administration skills are required to manage the portlet deployment and configuration. Web Development and WCM skills are required to develop this solution effectively. Although a RAD/Eclipse based project exists, it is not fully supported (i.e. will not function correctly with Virtual Portals) | | Integration | Integration is accomplished via the Digital Data Connector implementation, but may not satisfy all the SFDC REST-based requirements | | User experience | Client-side rendering will permit an improved overall user experience | | Deployment | WCM syndication or Portal Application Archive deployment allow code to be promoted between environments | | Business Agility | Business users could potentially generate their own forms, but this does not represent a realistic scenario as the development paradigm is complex. | | Look and Feel | Fully integrated with theme | |
|  |  |
| **Working Code Reference** | N/A |
|  |  |
| **Cost Analysis** | Required license: **Salesforce licenses are required for the Service Accounts and likely user based accounts because server-side injection of applicative security cannot be defined on the server**  Maintenance:   * Development methodologies and tools are new and definitive best practices have yet to be defined * Code is largely maintained in WCM, however issues have been documented when working with virtual portals * Interactions with back-end components require services similar to those defined in the SFDC Integration Pattern * Security concerns exist over having to persist the access token in cookies on the client-side |
|  |  |

**Issues with Virtual Portals and script portlet**

References: https://www-304.ibm.com/support/docview.wss?uid=swg21686096

## **Forms Experience Builder Pattern**

|  |  |
| --- | --- |
| **Description** | The Forms Experience Builder offers an easy to use interface for business users to create forms that can be added easily to Portal pages |
|  |  |
| **Problem** | How can business users quickly create forms that can be easily integrated within Portal? |
|  |  |
| **Solution** | The solution introduces the IBM Forms server to allow business users to generate forms to create and update content in SFDC. There is additional maintenance, infrastructure and administration cost associated with setting up this additional server. |
|  |  |
| **Sequence Diagram** | N/A |
|  |  |
| **Results** | |  |  | | --- | --- | | Skills | Portal administration skills are required to manage the portlet deployment and configuration  Training is required to teach business users how to create forms but it should be relatively simple | | Integration | Integration is accomplished via the Digital Data Connector implementation, but may not satisfy all the SFDC REST-based requirements | | User experience | The form based solution will support the create and update actions, however another mechanism must be put in place to display list and detail data | | Deployment | Form rendering portlets must be added to Portal pages and the IBM Forms Server manages the state of the forms | | Business Agility | Business users could generate their own forms | | Look and Feel | Fully integrated with theme | |
|  |  |
| **Working Code Reference** | N/A |
|  |  |
| **Cost Analysis** | Required license: N/A  Maintenance:   * Additional patterns must be leveraged to render and extend list and detail based queries * Additional server licensing, hardware, and maintenance costs are required to run the IBM Forms Server   Note: it’s important to consider the impact when integrating the Forms Experience Builder with Salesforce. Authentication must be facilitated between the systems and additional services may need to be built to provide the necessary meta-data to the Form Builder and construct a form that meets the Salesforce applicative security model. |

## **Client-side Portlet Pattern**

|  |  |
| --- | --- |
| **Description** | Custom Portlet solution that leverages client-side only technologies to invoke calls to SFDC. Integration with the Outbound HTTP Connection is used to facilitate SSO between Portal and Salesforce and to work around the Web Browser single-domain policy. |
|  |  |
| **Problem** | How can we integrate with SFDC using strictly client-based technologies, independent of any 3rd party frameworks? (i.e. IBM Script Portlet) |
|  |  |
| **Solution** | To facilitate AJAX REST-based calls and avoid the Single Origin Policy, the Outbound HTTP Connection (formerly Ajax Proxy) must be configured to pass the appropriate HTTP Headers, including the authentication token and allow queries to connect to the SFDC services.  The handlebars templates could transform the JSON results from the SFDC and render the data as HTML. |
|  |  |
| **Sequence Diagram** |  |

|  |  |
| --- | --- |
|  |  |
| **Results** | |  |  | | --- | --- | | Skills | HTML, CSS, JavaScript and other client-side technologies skills are required for this solution.  To manage the complexity of the user interactions, a client-side MVC framework should also be used (i.e. AngularJS) | | Integration | Direct calls to the SFDC REST-based service could be leveraged.  **NOTE: Concern exists whether PATCH HTTP Method can be leveraged through the Outbound HTTP Connection** | | User experience | Generally favorable. Advanced UI rendering is entirely possible through this pattern. (Comparable to the Hybrid and Script Portlet Patterns) | | Deployment | Changes would require a deployment of the Portlet | | Business Agility | Direct IT involvement would be required in this model unless integrated with Web Content Management | | Look and Feel | Presentation templates and other authoring tools may be leveraged to meet any look and feel requirements | |
|  |  |
| **Working Code Reference** | [**https://github.com/spece/leoneinc/tree/master/leone-portlet**](https://github.com/spece/leoneinc/tree/master/leone-portlet)  **Note**: Much of the client-side implementation provided in this solution can be leveraged, however the Outbound HTTP Connection is not leveraged. |
|  |  |
| **Cost Analysis** | Required license: **Salesforce licenses are required for the Service Accounts and likely user based accounts because server-side injection of applicative security cannot be defined on the server**  Maintenance:   * A solution reliant entirely on client-side code can become incredibly hard to maintain, unit test, and enhance especially as new features are added to SFDC * Enhancements and changes still require a production deployment and will limit the opportunity * Single-Page Architecture (SPA) paradigms should be used to structure code and minimize complexity * IT involvement is required to maintain the Outbound HTTP Connection * Integrating with intermediary service buses could be cost prohibitive * This model is tightly bound to the SFDC REST-based API * Highly scalable as all the advanced processing is completed client-side |