

Plug-In SFA Example

Table of Contents

- Section A: Business Level Product Overview (High Level)
- Section B: Architecture and Design
- Section C: New approaches
- Section D: Potential / future plans for the platform

Section A:

Business Level Product Overview (High Level)

The product is a plug-able platform, which is highly and easily scalable and customizable. It can be dropped into laptops or high memory bearing smart phones.

The platform defines clear process for delivering cost-effective, quality solutions with architecture and common features similar to existing SFA platform.

Plug-ins are based on a stable and thoroughly tested Plug-In Sales Force **Platform** and as result they can easily support a number of critical application features:

- displaying data in table format (list screen)
- sorting capability
- filtering capability
- global search capability
- history navigation
- reporting
- printing
- storing any data
- simple, generic GUI with tabs (similar screen organization as in existing SFA)
- data synchronization with server

The plug-ins can be easily and independently created, modified, added and removed from the application.

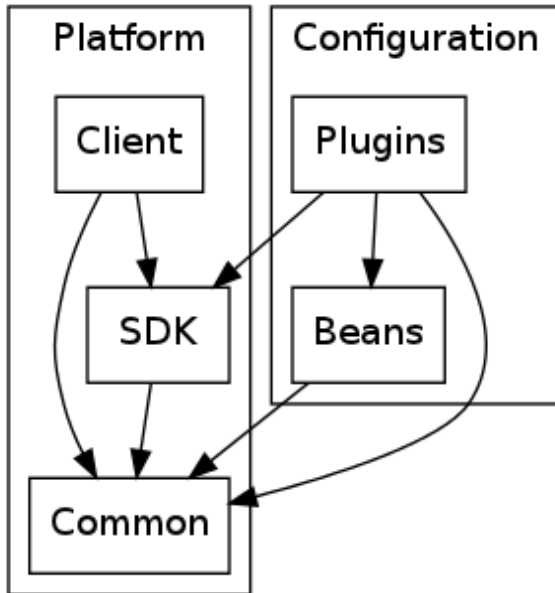
Current implementation of the Plug-in platform focuses mainly on the client side.

Section B: Architecture and Design

The system (client side) is composed of two major parts:

- Plug-In Platform (Core)
- Specific business domain Configuration

* Please refer to details below



PluginSalesForce Projects

1. Plug-In Platform (Core)

- Common Client - Java Swing standalone application, which, along with SDK, provides a skeleton (browsing mechanism) for loading, presenting and managing plug-ins (screens).

The Common Client allows for:

- Launching the application
- Presenting the information from business domain as a set of screens
- Navigating between screens
- Integrating Help
- Performing data synchronization
- Reporting and Printing
- Interacting between plug-ins
- Dividing single screens into logical set of screens (perspectives)

- SDK – set of platform core services (APIs)
 - Data Provider API and Query API
 - Printing, Reporting and Dynamic Reporting API
 - Common API (Filter API, Dialogs, Common UI parts such as Table layout, utility classes)
 - Plug-able mechanism (API)
- Common server / client part

2. Specific business domain Configuration

- Business domain Configuration is a class implementing AppConfigurable interface (part of platform SDK)

It provides the following information for the platform engine:

- List of plug-ins to be used (plug-in registration)
- References between plug-ins (in other words, it specifies how plug-ins interact with each other)
- Logical groups of plug-ins (perspectives)
- Synchronization rules (synchronization rules are specific to data beans, not plug-ins, since same data beans might be used by a multiple plug-ins)
- Plug-able Global Data Lookup

- Number of Plug-Ins implementing specific business logic

The benefit of implementing new business logic as a plug-in is that we have clearly defines processes / mechanisms for many, critical functions:

- **Data Synchronization** - Plug-In SDK provides a set of synchronization methods which can be selected for the plug-in (or rather for data bean used by plug-in). Different methods support different types of synchronization approaches i.e. one-way data transfer, both-way data synchronization, full data replacement etc.
- **Mechanism of data storage** with ability to perform CRUD operations on business entities.
- **Common mechanisms to visualize/work with data in GUI**, such as DefaultTable class used for list screens with incorporated custom, sophisticated filtering, record count etc.
- Data Beans

Section C: New approach

1. Extracting core services from existing VIP application

We created three major projects based on the existing VIP application. Each project has its distinct purpose:

- **Common-Client** – Java Swing client application providing a skeleton (navigation mechanism) for plug-ins
- **SDK** – number of APIs (Data Provider API, Query API, Reporting / Printing, Sync API, set of interfaces supporting plug-in mechanism)
- **Plug-Ins** – composed of Configuration class (registration of plug-in, info on interactions between plug-in etc.); implementation of data synchronization and Plug-In classes (implementing specific business logic)

The task was challenging since it required major reorganization of existing projects.

We also introduced a number of interfaces clearly defining actions and behaviors of classes implementing them.

The example could be client synchronization mechanism. In SFA all related code exists in one project. In Plug-In platform we divided the mechanism between 3 projects:

- a) Visual synchronization module located in Common-Client
- b) Data Synchronization API located in SDK
- c) Implementation of data synchronization for specific data types located in Plug-In project (using methods from Sync API).

2. Implementing mechanism for plug-in interactions.

Each plug-in should be an independent component without direct references to each other.

All rules for interactions between plug-in are specified / implemented in external Configuration class. The actual plug-in is independent and does not know anything about other plug-in (i.e. does not know its class name)

In Configuration class we can easily specify:

- list of plug-in to be used / registered in the application
- specification of plug-in access (adding buttons on main and sidebar navigation bars activating the plug-in)
- rules for interaction between specific plug-in (events activating related plug-ins, passing data between plug-ins)

3. Implementing Navigation Manager mechanism

In SFA we used array List and current Index approach to keep track of navigation history (back and forth steps).

In Plug-In platform we use three distinct data structures storing:

- the current view name
- stack of 'Back' view names
- stack of 'Forward' view names

Current implementation is easier to work with and less error prone.

4. Creating system documentation (still in the process)

The ultimate goals of the documentation are::

- clearly and thoroughly describe technical architecture
- clearly define process of adding new and modifying existing plug-ins
- clearly describe the benefits of using the platform for implementing specific business logic

The documentation is meant for the developers not having previous experience and knowledge of SFA project

The task is quite difficult since the platform is composed from many projects and classes.

Section D: Potential / future plans for the platform

Important features which might be added in the future

1. Utility (with interface) for creating new data entities and corresponding code (i.e. server side sync, java beans etc.)

We could take it even further and create a utility for creating new plug-ins (GUI based on some common templates; adding / removing fields; generating code behind)

2. More common GUI components (client side), kind of templates supporting typical business screens and layouts. The template would be easy to customize.
3. Easier and better standardized way of communicating between plug-ins (especially data passing). Currently, during development time, we need to check related plug-in code in order to figure out what data needs to be passed. We would like to eliminate that.
4. More Wizards.