Enacted Software Development Process based on Cross Platform Unified Framework in the Context of Investment Banking

Pavitdeep Singh, MIE
Dept. of Credit Risk & Trading Book
Markets & International Banking
Royal Bank of Scotland

Prof. Jatinder Kaur
Dept. of Applied Sciences
University College of Eng. & Tech.
Chandigarh University

ABSTRACT

The demand for the rich web based solution changed the way the application were developed in the investment banking industry specifically for sales and trading (front office activities). Traditionally, simple web application were considered to be enough powerful to provide the user with all the capabilities. Later on developer started implementing mashup technology in order to cater to IBs ever increasing demands, which actually is a technique for building applications that combine data from multiple sources to create an integrated experience. This paper extends the concept of enterprise mashups to provide a new framework called cross platform unified framework (CPUF) which is a cross-technology and delivery platform framework, services, patterns and practices aimed at changing the way IB industry projects are design and delivered to business users. This paper not only explains the various core components of the framework it also highlights the important benefits of the CPUF. It aimed at understanding the user needs from their technical interface(s), identify the systems that can provide these features, provide a technology agnostic framework that can stitch this information together without moving data or processes out of the existing system.

General Terms:

Design, Architecture

Keywords:

CPUF, Investment Banking, Offering, Sales and Trading, Trade Deal Checking System, Context Event Bus, Context Events, Component Registry, Offering repository

1. INTRODUCTION

Traditionally, applications were based upon client/ server architecture which takes the user request and process the data at the server via web services and then return the processed data to the client for GUI rendering. Gradually new applications were developed using component based approach wherein the focus was more on the component rather than the whole application. As the demand for rapid and situational development took place in the recent times, the whole paradigm of development shifted towards the concept of Mashups which actually is aggregation of content, presentation and application functionality from different sources. Currently, there are different types of mashups which are targeted towards specific layer of architecture, namely Web mashups, Services mashups, Data Mashups [3]. The most commonly mashups available in the internet world are Web mashups which actually takes content and services readily available online [6]. But there isn't any structured framework or tools around developing application using web mashups .Moreover, it is limited to the applications that may be hosted through web only. Serivce Mashup provides the communication among different heterogeneous and distributed applications. Various standards which support the web services are Web Services Description Language (WSDL), UDDI, and SOAP [1]. Recently, light weight services are available namely, Web APIs and RESTful (Representational State Transfer). The other kinds of mashups available in the market are business and consumer mashups. Business or enterprise mashups combines their own resources, application functionality and content with various external services. They mainly work on a single presentation which allows for easy collaboration between developers and business. It is best suitable for agile methodology of development. Apart from these, there are few other kinds of mashups available which are getting recongintion. One such is information mashup, which is specifically designed for situational applications where applications are developed "on the fly" for some adhoc requirement [5]. There is a Tool named Damia which actually is a enterprise data integration service used for searching, managing and storing existing data mashups. It mainly consists of three components namely, browser based GUI, a server engine and APIs for searching , managing , debugging and executing mashups [4]. IBM moving one step further have desgined Shared Code online service platform which uses domain-specific language to design mashups and assist in sharing and re-using the mashups components for internet applications [2].

With the advent of different kinds of mashups, industries have started realizing the real power of combining the content, presentation and functionality to provide a rapid solution to ever increasing requirement of their business users. Investment banking is one such industry where the requirement of the business changes more rapidly as compared to other industry as it is quite volatile to the market changes. Regulatory changes also trigger modification to the strategy of existing processes within the bank. One of the key tasks at Front Office (FO) in Investment banking is sales and trading. The bank deals with buying and selling products on the behalf of their customer with the sole purpose of making money in each trade. Moreover, it provides trading ideas for their institutional and high net worth investors. Thus it also demands for situational, unstructured and volatile requirements from the business to address the issues of their clients. Currently, FO applications are developed using traditional integration methodology called component based development. The major disadvantage of this approach is that it may not suitable for applications that for required for short period of time (short lived application) and will require more effort in combining with newer technologies like Web APIs and RESTful for communication apart from the existing web services.

This paper proposes a new framework called Cross Platform Unified Framework (CPUF) for developing applications specifically for Investment banking system which is actually based upon on the concept of enterprise mashups. It provides a standard way of creating visual and non-visual components and services. Additionally, it also defines a comprehensive way of using the existing components / services to meet individual business requirements that may be short lived with minimum development effort. This paper is organized as follows: section 2 discusses the key benefits of CPUF framework. Section 3 talks about the overall architecture of the framework with emphasis on each core element of it, section 5 discusses the case study where the framework has applied to one of the projects in investment banking. Finally, section 6 draws the conclusions and section 7 discusses the future work.

2. STRATEGIC BENEFITS OF CPUF

The key principle behind CPUF is the separation of functionality into distinct services that are made available over the network, described in a consistent and standardized manner. Consumers can subsequently combine and reuse these visual and non-visual services in the production of business-aligned offerings.

CPUF's main focus is on taking all existing components and putting them together in one architecture, thus reducing the number of needed frameworks. As a result, this simplifies creation of new offerings for other developers.

A large amount of duplication is avoided and reuse of existing resources is enforced by having all components shared within one framework,. Therefore, this greatly saves the time for the development of new offerings.

In order to achieve the same level of understanding and give similar user experience from the application, it is essential to have the same tools on both sides.

Infrastructure independence helps to avoid specific bottlenecks in components deployment, as well as their maintenance. This helps developers create and add new components without affecting the system as a whole.

Providing only specific functionality to the end user greatly increases effective use of application and simplifies the use of the offering.

Broader support in terms of environments and paradigms greatly simplifies the use of our framework and increases its accessibility among all clients.

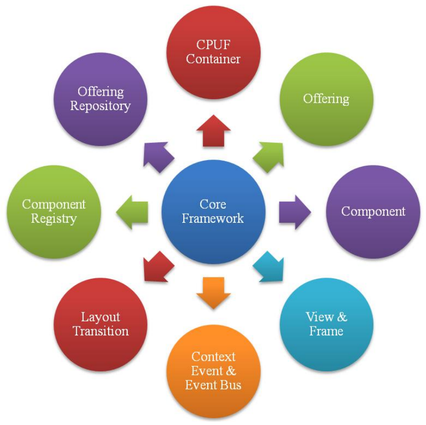

Fig. 1. Overview of CPUF Architecture

3. CPUF ARCHITECTURE

A technology agnostic presentation framework designed to bind loosely-coupled interfaces together to support the specific needs or views required by individual business units and roles. It's based on the architecture concept of enterprise mashups. A collection of GUI components (called CPUF core components) and services providing common capabilities to be used by other services. These components provide no direct business value until combined with other business-aligned services. A set of standards policies that define how services should be designed to allow for consistent, standardized integration with other services and the mashup of presentation interfaces.

From architect's perspective CPUF has taken many of its principals and approaches from Service Oriented Architecture.

Following are the key elements which constitute the core framework.

3.1 CPUF Container

The CPUF Container or runtime is the software that runs, reads the offering definition, performs the authentication, loads and hosts the components, routes context events and context properties. It is written in Java and runs in a Java VM. The initial download of CPUF is done via Java web start. Once the container is downloaded from web start, it connects to the Offering Repository and downloads the Offering. The container then uses the Offering to download and run the components from the endpoints specified.

3.2 Offering

An offering is an application that is runs on CPUF and is made up of one or more components. An offering is described by an XML document which defines the components, their interactions via context events and properties, the layout and layout transitions.

3.3 Components

In CPUF, a component is a reusable piece of software. Components come in three flavors, HTML, Silverlight and Java which reflect the programming language they were written in. Components are assembled together to form an offering.

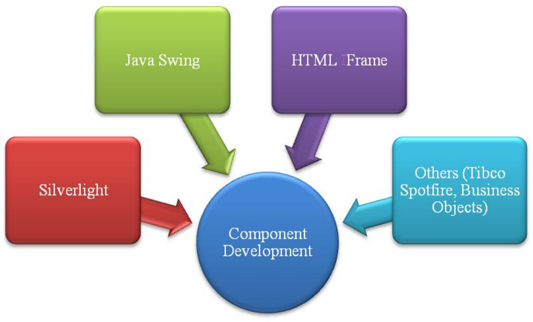

Fig. 2. Component Development Languages

Table 1. Different Component Types

Component Type	API	End Point
HTML	End	Web Page
Java	Java	ZIP
Silverlight	Silverlight	ZAP

Components ultimately run within the CPUF runtime, which is a container component. The container exposes functionality to the components via an API. For java components this is a Java API. For Silverlight, Silverlight and for HTML it is java script. Among other things, the APIs allow components to create and respond to context events, retrieve and provide context properties and access the security context.

Components are deployed to endpoints. An endpoint is a url. Components will be deployed to an endpoint per environment. Components interact with other components via context events and context properties. They are registered in the component registry. This includes the details of the context events they publish and subscribe to and the context properties they publish and consume. This makes up the interface of the component and allows offering writers to assemble them into offerings.

3.4 View & Frame

A view if for structuring how your components are laid out in your offering. A view can be used as docked or floatable and can have frames added to it at different locations - side-center, side-north, side-south, side-east, side-west.

A frame is for displaying visual components within a panel in your offering. A frame can be used as docked or floatable and can have components added to it at different locations - side-center, side-north, side-south, side-east, side-west. Frames are lower down the visual hierarchy than Views. A view can have frames in it, but a frame cannot have views in it.

3.5 Context Event & Event Bus

Context events are the primary mechanism that components use to communicate. Context events consist of two parts, the topic and the payload. The topic describes the source of the event, the nature of the event and the version. The payload is the content of the event.

The context event bus allows the integration of multiple client-side components within a single container, providing a client-side framework that allows trusted and untrusted components to co-exist and communicate with each other. The primary function of the context event bus is the ability for components to publish and subscribe to topics that are used to send messages asynchronously to each other.

3.6 Layout Transition

A Layout Transition is the idea that visual components can have different layout states and switch between them e.g. open, close, hidden. This is normally used in conjunction with View/Frame. An example is when a Layout Transition is used to target opening an instance of a view template with a payload based on a particular context event.

3.7 Component Registry

It contains all the components that have been developed to the current date. The records store the current version, developing team and business stream of the project. When developing, it is not mandatory to register your components however as you progress through to unit acceptance & testing phase you will be required to do. This will provide the necessary information to other offering developers to develop and validate their offerings.

3.8 Offering Repository

It stores all the offerings that are available. When you run CPUF it loads the specified offering from the Offering Repository when launching your offering.

4. LAYOUT

Clients can provide their own layouts (custom layouts) according to their individual needs. Custom layouts could be configured and reused. To enable this, as well as the layout being stored in the CPUF Store, we also store an xml schema and usage instructions. When the layout is specified, xml can be specified under it, we can validate this with what is in the CPUF Store and then pass the configuration xml to the layout code which will understand it and lay out appropriately.

```
<view id="viewId">
     <visualComponents>
         <swingComponent id="Component1"/>
         <swingComponent id="Component2"/>
         <swingComponent id="Component3"/>
         <swingComponent id="Component4"/>
    </visualComponents>
    <layout>
         <custom id="fourWindowLayout">
             <windows top="Component1"
                      bottomLeft="Component2"
                      bottomCenter="Component3"
                      bottomRight="Component4"/>
         </custom>
    </layout>
</view>
```

Below is the sample layout generated by loading the custom layouts for various components mentioned in the xml file.

5. CPUF IN THE DEVELOPMENT OF INVESTMENT BANKING SYSTEMS

An example of usage of CPUF in the investment banking domain is a component based solution for Trade Deal Checking System for a

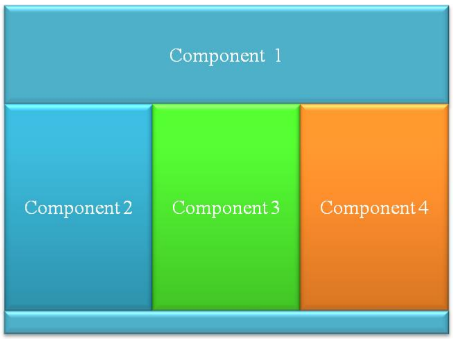

Fig. 3. Component Layout Design

relatively complex business problem. The solution involve developing the components in such a way that other application can make use of these components when required. Moreover, the components can be written in different languages supported by the framework. Figure 4 shows the overall architecture diagram for Trade Deal Checking System.

As depicted, there are 4 GUI components namely, Counterparty Search, Product Selection, Trade Attributes and Notional Availability Information. These independent components interact with each other through context events. For example, Trade Attributes components has subscribed to 2 events, one from each Counterparty Search and Product Selection component. Whenever there is a change in counterparty, the event corresponding to counterparty search get triggered and the components which have registered for this event will be able to listen. In our case, Trade Attributes will get the new selection for counterparty. The components can invoke internal or external service depending upon the requirement. These services can further interact with different databases to provide the data. These components needs to be registered in the component registry. On the other hand, a classical approach would have involved all the components to be developed irrespective of their availability might be with little re-usability. Moreover, the components already available may not be utilized in case it's not developed with the differnt technology. This would have resulted in higher development effort and which in turn requires high cost for the project. However, a CPUF based solution with the Client, Product Selection and Trade Entry components already developed (irrespective of any technology) and registered with the component registry this would simply mean developing the core component for Trade Deal Checking application.

Components can be developed in any one of the technologies namely java, Silverlight or html. Once the component is developed, component is registered with the component registry and entry is made in the offering file to add the component to this offering. Custom layouts as defined earlier can be used to position the component in the mail GUI screen. The whole process of software development of the component would have resulted in drastically less effort. Moreover the deployment process would have required the deployment of this core component only (if other dependent components are already deployed and currently in available in application).Thus, CPUF has drastically reduced the effort required for development and deployment of the application components.

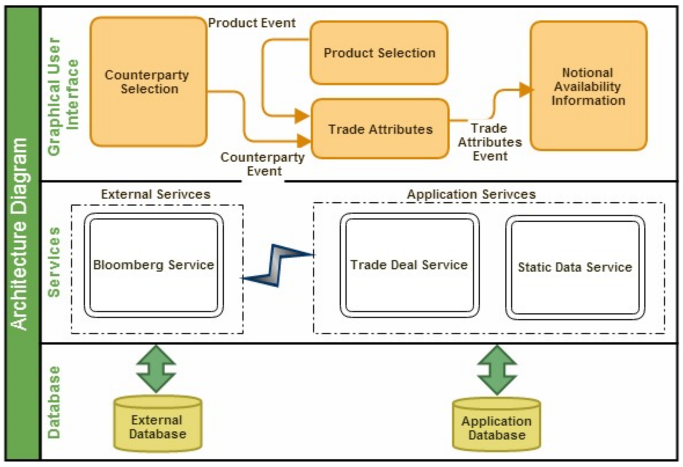

Fig. 4. Architecture of Trade Deal Checking System

6. CONCLUSIONS

This paper presents a brief description of CPUF framework and its core elements and its application in the context of investment banking Trade Deal Checking System. Due care is given to present the main ways in which CPUF based framework is used to develop application (mainly components) and how these components are laid out in the GUI with the help of various custom layouts. The best part is components can be developed in differnt technologies (technology agnostic), it can be Java, Silverlight or html. This paper shows that CPUF is most suitable for IB application as it involves hosting of various components for a particular offering. Data among the various components can be passed with the help of context bus and context events (through subscription and publication mechanisms). This framework is well suitable for large scale applications in investment banking domain specifically FO which requires the interaction among various parts (or components) of the application. This also reduces the development life cycle of various components and helps in quick testing and deployment of various components developed. Due to time to market criticality of IB projects CPUF framework emerge as a very strong and workable paradigm that investment banking industry tend to adopt. This paper shows that various components developed at global level are consistent within IB organization.

7. FURTHER RESEARCH CHALLENGES

This section summarizes advanced features and improvements that we are currently exploring as enhancements to CPUF:

(1) Performance Improvement: CPUF relies heavily on components defined in the offering file. More the number of components more the time it take to load them in memory and this would result in decrease in performance in case a business user if interested in very few components. There should a way in which only those components should be loaded in memory for which the user has registered rather than loading all the components defined in offering file (xml file).
(2) Updating Custom Layout at Run time: Changing the layout of various components at run time and then allowing the business user to save those changes will not only help in organizing the various components across his/her screen according to his choice but will also help in removing (or hiding) those com-

ponents which does not interest him. The next time user runs the application on the desktop, components should be shown as per the saved preferences.

(3) System Memory Constraints: Memory is finite in any system so special attention needs to the given to this parameter. The more the numbers of components the more the memory it would requires. There should be some mechanism to circumvent this problem.

8. REFERENCES

[1] Schahram Dustdar Djamal Benslimane and Amit Sheth. Services mashups: The new generation of web applications. *Internet Computing IEEE*, 12(5):13–15, Oct 2008.

[2] A. Ranabahu E.M. Maximilien and K. Gomadam. An online platform for web apis and service mashups. *Internet Computing IEEE*, 12(5):32–43, Oct 2008.

[3] Hakim Hacid Giusy Di Lorenzo and Hye young Paik. Data integration in mashups. *SIGMOD Record*, 38(1):59–66, May 2009.

[4] Susan Cline Rajesh Kartha Eric Louie Volker Markl Louis Mau Yip-Hing Ng David Simmen Ashutosh Singh Hmet Altinel, Paul Brown. Damia - a data mashup fabric for intranet applications. *ACM*, pages 1370–1373, Sept 2007.

[5] A. Jhingran. Enterprise information mashups: Integrating information, simply. *Proc. 32nd Int'l Conf. Very Large Databases (VLDB 06), VLDB Endowment*, pages 3–6, 2006.

[6] Fabio Casati Jin Yu, Boualem Benatallah and Florian Daniel. Understanding mashup development. *Internet Computing IEEE*, pages 44–52, Oct 2008.