Integrating e-Payment Services with RIA Applications

Mihai Pricope, Radu Constantinescu

Adobe Solutions, Adobe Platform Evanghelist
Bucharest, ROMANIA
Economic Informatics Department,
Academy of Economic Studies Bucharest
Pta. Romana 6, Bucharest, ROMANIA
mpricope@adobe.com, radu.constantinescu@ie.ase.ro

Abstract: Given the expansion of RIA technologies, it is highly important to deliver a set of best practices for developers in order to integrate e-business applications with e-payment systems. In this article we present a solution for the integration of PayPal Express with a RIA application developed on Flex. We focus both on security and functionality issues.

Key-Words: e-payment, e-business security, Flex, RIA, PayPal.

1. Introduction

In the area of web technologies, RIA brings a new perspective regarding user experience. We have arrived at a point in which the initial visible distinctions between classical desktop applications and web applications can be overcame. The classic development approach for web applications has determined a lack of expressivity as the developers were focusing on the benefits brought by the distributed environment. RIA technologies envelope all the benefits of web applications in a framework that delivers interfaces with intuitive behavior and superior graphical capabilities [1], [2].

Flex is a programming platform designated for both developers and designers. The platform is suited for developing web clients which interacts with a large spectrum of server-side technologies. In the distributed application model, Flex supports the development of RIA clients which asynchronously exchange data with server-side applications.

This is a post conference paper. Parts of this paper have been published in the Proceedings of the 2nd International Conference on Security for Information Technology and Communications, SECITC 2009 Conference (printed version).

Flex brings a new paradigm which is state oriented. The classic model assumes that a server waits for client requests, processes the requests and then issues a response. The client has to wait while the server delivers the response. After that interaction the transaction is closed till another request occurs.

Flex implements the actor model. A Flex client basically changes states. The components are sending messages in an asynchronous manner. When an application component included in the client application receives an event, the dedicated event handler is called and then it returns to the waiting state. Events are raised as a response to different inputs which mostly are delivered by users. In the Flex model, there might be several parallel tasks which potentially involves server processing. None of these tasks doesn't leave the application in a blocked state. Therefore the application feedback is likely to be on a real time basis.

As a summary, when a request is to be sent to a server, this is done asynchronously and the application continues executing other tasks. The response is processed when it is received and if it is received.

An inherent issue in developing e-business applications on Flex platform is the integration with e-payment gateways, as PayPal. PayPal is a well-

ISSN 2067 – 4074

known online e-payment system widely used in online transactions. Therefore the security issue is highly important when arriving at a point in which a developer aims to connect a certain web application to the e-payment system.
The challenge of integrating payment services into RIAs is due to the fact that currently, payment services like PayPal are designed to work in request-response paradigm which is the standard paradigm for web-based applications and while RIAs are stateful.
We will briefly describe the e-payment service offered by PayPal to a certain client. As a first step, the client creates an online account in order to issue online payments. The client links the online account to his credit card and then activates the account. PayPal withdraws a small amount form the bank account that was assigned to the card in order to test the account. The amount is returned after the account validation. Once the account is validated, the user can fulfill deposits on the online account in various currencies and therefore issue payments. The PayPal system can be used for e-payments as it is a widely-accepted mechanism. In the article we will assume that the user has already purchased a PayPal account.
The system has several advantages as instant transfer of money, support for payment cancelation, reduced commissions, fast authentication system, various currencies support and high portability.
As a drawback, PayPal's security system is not a highly restrictive one, so the developer should pay attention when he is supposed to integrate the e-payment system in a web application [4].

2. Security Issues

One reason for the success of e-commerce is that the Internet has proven to be a secure medium for transferring money and making payments. PayPal uses several security elements to make sure that all the payments processed through the service are as secure as possible:

- Usage of https protocol for communicating with PayPal Adaptive Payments Web Services ensures that the communication is protected from third party access.
- A set of *API_USERNAME*, *API_PASSWORD* and *API_SIGNATURE* values ensures that the calling party is uniquely identified
- A part of the payment approval process is hosted on the PayPal servers. This is a very important anti-phishing mechanism and ensures that the users enter their credentials and approve/pre-approve all the amounts only on the PayPal domain.

Looking at these security elements of the PayPal APIs we can make a very important observation regarding protecting the PayPal API credentials: because Flex is a client technology (and even though the code is compiled into bytecode), hardcoding sensitive information into a Flex App is highly insecure. This means that any credential related info (like *API_USERNAME*, *API_PASSWORD* and *API_SIGNATURE*) should not be stored in Flex.

3. Architectural Approach

Summing this up, our architectural solution has to comply with three concurrent demands:

- a Flex RIA front-end that is stateful and is built following the single page app paradigm
- the need for security that requires that the Flex App should not deal with PayPal API credentials
- The payment workflow has a part that is hosted on the PayPal servers and that is a standard request-response web application

In order to address these we propose the following approach:

- all PayPal API calls should be done on the server side so that API credentials will be protected

- the access to the PayPal web application should be done in a pop-up/new page so that the Flex Application will stay in the Single Page Paradigm and thus preserve its state

The workflow for checkout process assumes that the buyer will take the following six steps:

1. Chooses to checkout using PayPal in the RIA application. Starting from this point it is mandatory that all requests are done through https.
2. Sees a new window/pop-up open and he logs into PayPal to authenticate his/her identity
3. Reviews the transaction on PayPal
4. Confirms the order and pays from your site
5. Reviews the order confirmation on your site
6. Closes the pop-up and returns to Flex Application

4. Problem solution

We will suppose that an user visits a new on demand video site. He selects to watch a certain media entry. After having watched a part of the movie, the user is asked to pay an amount of money as a fee. The user selects to pay the amount using PayPal and then continues to watch the rest of the movie. We will present a possible implementation for the payment process in this scenario.

As we said above the PayPal API credentials need to stay on the server, so PayPal API invocation should be done from the server as well. For these examples we will use PHP as a server language and also the PayPal Name-Value Pair (NVP) API sample code. Of course any server language might be used, the principles and techniques highlighted here remain the same [3].

The first step is to open the pop-up window using the *ExternalInterface* call which brings the ability to control the window appearance.

```
ExternalInterface.call('window.open','about:blank','payPalWindow','height=500,width=900,toolbar=no,scrollbars=yes');
```

The second step is to send a request to a server page in the newly opened window containing the user's choice.

```
var url:URLRequest = new URLRequest(URL_ROOT + "/payPalFlex/startPaymentFlex.php");
url.data = new URLVariables();
var obj:URLVariables = new URLVariables();
url.data.movieId = '1';
url.data.paymentReason = 'Movie';
url.method = "GET";
navigateToURL(url, "payPalWindow");
```

On the server page we follow the PayPal NVP samples and we will generate the URL that will redirect the user to PayPal site. The *getMovieAmount* function can be replaced with a more sophisticated one. The *returnURL* is the location where buyers return when a payment has been succesfully authorized. The *cancelURL* is the location buyers are sent to when they hit the cancel button during authorization of payment during the PayPal flow.

The *$nvpstr* variable contains the parameter string that describes the PayPal payment. The variables were set in the web form. The application makes the call to PayPal to set the Express Checkout token. If the API call succeded, then the application redirects the buyer to PayPal to begin to authorize the payment. If an error has occured, the application will show the resulting errors.

```
$serverName = $_SERVER['SERVER_NAME'];
$serverPort = $_SERVER['SERVER_PORT'];
$url = dirname ( 'http://' . $serverName . ':' . $serverPort . $_SERVER ['REQUEST_URI'] );

function getMovieAmount($movieId)
{
return 1;
}
```

ISSN 2067 – 4074

```
$paymentAmount =
getMovieAmount($_GET['movieId']);
$currencyCodeType = 'USD';
$paymentType = 'Sale';

$returnURL = urlencode ( $url .
'/GetExpressCheckoutDetails.php?cu
rrencyCodeType=' .
$currencyCodeType .
'&paymentType=' . $paymentType .
'&paymentAmount=' . $paymentAmount
);
$cancelURL = urlencode (
"$url/cancel.php?paymentType=$paym
entType" );

$nvpstr = "&Amt=" . $paymentAmount
. "&PAYMENTACTION=" . $paymentType
. "&ReturnUrl=" . $returnURL .
"&CANCELURL=" . $cancelURL .
"&CURRENCYCODE=" .
$currencyCodeType;

$resArray = hash_call (
"SetExpressCheckout", $nvpstr );
$_SESSION ['reshash'] = $resArray;
$ack = strtoupper ( $resArray
["ACK"] );

if ($ack == "SUCCESS") {
$token = urldecode ( $resArray
["TOKEN"] );
$payPalURL = PAYPAL_URL . $token;
header ( "Location: " . $payPalURL
);
} else {
$location = "APIError.php";
header ( "Location: $location" );
}
```

We should notice that we have stored the result of the API call in session to use it later. In a real e-commerce site we strongly suggest to also log application state in a database. This way you will have access later on to all transaction steps.
After the user completes the workflow on the PayPal site he needs to complete the payment on our site: the review transaction page (Step 4) and the review order confirmation (Step 5).
Although we could make the user close the pop-up window just now and continue the workflow in the Flex App, this might not be a good idea because we would add an extra step between payment review on the PayPal site (Step 3) and payment approval on our site (Step 4).
So we choose to implement the payment approval using standard PHP and HTML and reuse the PHP NVP API Samples from PayPal for calling GetExpressCheckoutDetails API (Step 4) and DoExpressCheckoutPayment API (Step 5).
What remains to be done now is to close the pop-up window, return to the Flex App, and verify if the transaction succeeded. To communicate with the Flex application we will use the *ExternalInterface* mechanism. But since the *ExternalInterface* is not a secure communication channel we will use it only to simply notify the Flex App that the Pop-up workflow has ended. The status will be retrieved by the Flex App from the server. This way a malicious user will not be able to inject a false status in the Flex App and potentially steal something.
After we call the *DoExpressCheckoutPayment API*, we save the results in session. Now in Flex we will have a method that check the status and decided if the transaction failed or not where *getPaymentStatus.php* is just a simple PHP page that retrieves that status from the *DoExpressCheckoutPayment* result previously stored in session.

```
$resArray=hash_call("DoExpressChec
koutPayment",$nvpstr);
$_SESSION ['reshash'] = $resArray;
private function
paymentComplete():void {
var srv:HTTPService = new
HTTPService();
srv.url = URL_ROOT +
"/payPalFlex/getPaymentStatus.php"
;
srv.addEventListener(ResultEvent.R
ESULT,function
(event:ResultEvent):void {
Alert.show(event.result.status.typ
e);
if (event.result.status.type ==
'SUCCESS') {
currentState = 'Succes';
} else {
currentState = 'Fail';
}
});
```

```
srv.addEventListener(FaultEvent.FA
ULT,function
(event:FaultEvent):void {
currentState = 'Fail';
Alert.show(event.message.toString(
));
});
srv.send();
}
```

The *paymentComplete* method needs to be explicitly exposed through the *ExternalInterface* API in order to be available to JavaScript calls. This can be done when the Flex application initializes, the *applicationComplete* event is a good candidate for this.
After that, the only thing that remains now to be done is to close the pop-up window and notify the Flex App.

```
ExternalInterface.addCallback('pay
mentComplete',paymentComplete);

<script type="text/javascript">
function gotoflex() {
window.opener.window.document.getE
lementById('payPalFlex').paymentCo
mplete();
window.close();
</script>
<a    class="home"    id="CallsLink"
href="javascript:gotoflex()">Retur
n to Flex</a>
```

5. Conclusions

The purpose of this article was to describe relevant aspects of a possible implementation of an e-payment system in a RIA application. Even if the Flex client is compiled in bytecode, a good security practice is to avoid including sensitive data on the client-side. Moreover, we strongly recommend using secure transfer protocols and a mechanism that ensures that the calling party is uniquely identified.

References

[1] T. Ahmed, J. Hirschi, F. Abid *"Flex 3 in Action", Manning Publications*, 2009

[2] J. Noble, T. Anderson, "*Flex 3 Cookbook*", O'Reilly, 2008

[3] M. Pricope, http://miti.pricope.com

[4] R. Constantinescu, F. Nastase, "Process Models for Security Architectures", *Informatics in Economy Journal*, no. 4, 2006