Portal Syndication with Web Services and Cocoon (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. What Is Web Syndication?

Web Site Syndication has gained popularity as more and more web sites cross reference each other, not only by a single hyperlink, but also by embedding parts of their content. The idea was pioneered by Netscape with their Rich Site Summary (RSS) (http://www.oasis-open.org/cover/rss.html) XML format. RSS was developed in early 1999 to populate Netscape's My Netscape portal with external newsfeeds ("channels"). Since then RSS has taken on a life of its own and now thousands of Web sites use RSS as a "what's new" mechanism to drive traffic their way.

The current RSS 1.0 standard is an application of Resource Description Framework (RDF) (http://www.w3.org/TR/rdf-schema/). RDF is a framework for describing and interchanging metadata. The RDF framework is extensible and allows adding new types of entities. It also gives meaning to resources to enable automated processing of Web resources.

RSS is unarguably an example of an organically grown and widely accepted standard. For long it was not endorsed by any of the popular standards committees. Even so it quickly became popular and found a large number of creative uses. Lately though it has reached its limits. There is a demand for more advanced portal syndication which RSS cannot satisfy.

2. Going beyond RSS with Web Services

Latest generation web portals demand more than simply posting cross linked news stories from RSS. Embedding and personalizing rich content and behavior from remote portals is becoming necessity. Limited success has been achieved through complex and sophisticated backend integration via proprietary or Web Services compliant protocols. Recognizing the growing demand, influential organizations have attempted to develop new languages such as:

2.1. Web Services Experience Language (WSXL)

(http://www.webservices.org/index.php/article/articleview/345/)

"WSXL is a Web services centric component model for interactive Web applications. WSXL is designed to achieve two main goals: enable businesses to distribute Web applications through multiple revenue channels and enable new services or applications to be created by leveraging existing applications across the Web."

2.2. Web Services Inspection Language (WSIL)

(http://www.webservices.org/index.php/article/articleview/85/)

"The specification allows a Web services provider to publish a WS-Inspection (WSIL) document which lists the services on offer and their corresponding WSDL (Web services description language) files. The convention is that the WSIL document should be called "inspection.wsil" and be located at a common entry point to the web site. This paves the way for future Web services "crawlers" to locate and parse WSIL documents for Web service search engines."

2.3. Web Services for Remote Portals (WSRP)

(http://www.oasis-open.org/committees/wsrp/)

"Defining an XML and Web services standard that will allow the plug-n-play of visual, user-facing Web services with portals or other intermediary Web applications"

2.4. Web Services for Interactive Applications

(http://www.oasis-open.org/committees/wsia/)

"Create an XML and web services centric framework for interactive web applications. The designs must achieve two main goals: enable businesses to distribute web applications through multiple revenue channels, and enable new services or applications to be created by leveraging existing applications across the Web."

While these efforts are certainly worthwhile and promising, it will most likely take years before they pass the filters of real life use before they can claim widespread adoption. All of them ask for a thick infrastructure layer to support implementations. While possible, it is unlikely that mainstream deployment will be achieved instantly.

Not all is lost though. Fortunately, there is way to satisfy a large portion of the syndication requirements by applying already established technologies and tools. We will illustrate the architecture of a possible solution using an open source framework for XML Publishing - Apache Cocoon.

3. Apache Cocoon

(http://cocoon.apache.org/index.html)

"Apache Cocoon is an XML publishing framework that raises the usage of XML and XSLT technologies for server applications to a new level. Designed for performance and scalability around pipelined SAX processing, Cocoon offers a flexible environment based on a separation of concerns between content, logic and style. To top this all off, Cocoon's centralized configuration system and sophisticated caching help you to create, deploy and maintain rock-solid XML server applications".

First, let's describe a typical use case scenario: User logs in to a familiar portal and happily surfs about. At some point the user clicks on a link which leads to a strange page. It has the portal logo, even shows the same login id but still looks very different and unfriendly ... After some time and frustration the user gets used to switching back and forth between the two faces of the portal ... while looking for another provider which offers both services in a coherent graphical interface.

For those who have never had similar experience, we will give a popular example. Yahoo! Autos (http://autos.yahoo.com/finance.html?refsrc=autos/insurance) offers an easy to use interactive catalog of cars. However when it comes to insuring an automobile, applying for a loan or buying a car, the web site hyperlinks to a co-branded page of another company. For example Lending Tree (https://www.lendingtree.com/newauto/.....) will show Yahoo! Autos logo at the top of the screen, however the rest of the page looks very different than any other Yahoo! page. All the personalization spoils that a Yahoo! user enjoys are lost as soon as the application for a loan begins. Not only the colors and layout are different. A login session with Yahoo! does not carry over to Lending Tree. On top of that a pop-up window appears when switching between the two sites, which reads "You are about to view pages over a secure connection ...". When added up these "negligible" inadequacies, lead to an overall poor experience, which is certainly not the original intent of the Yahoo! content producers.

Now as we have an idea of how things are not supposed to work, we will show that outsourcing interactive components to a third party site, while preserving the look & feel of the original portal is

still possible when done right. As we mentioned Cocoon offers a solution. Since Cocoon is a very sophisticated framework, an indepth analysis of its features is beyond the scope of this text to cover.

4. Web Services Proxy to the rescue

The latest version of Cocoon is 2.1 and it has a new Web Service Proxy component. It is this component which we shall focus on for the reminder of the text. To follow the rest of the article, it will be useful (but not essential) to have a basic knowledge of Cocoon 2.

Combined with the XMLForm (which is not part of Cocoon anymore) component of Cocoon 2 and XSLT, the Web Service Proxy component allows vendors to share interactive content with little effort. The Web Service Proxy takes advantage of the fact that a Cocoon web application produces XML content, which is later translated into multiple presentation formats, like HTML or WML. Once the proxy is plugged in the Cocoon sitemap, it transparently pipes browser requests to a remote web application and returns the response back to the sitemap for local styling. Receiving a client independent XML format, allows the local site to pull content and style it with XSLT with the desired Look & Feel.

Q. Ok, styling presentation is easy to understand, but how is a form submitted to the original site?

The XMLForm component is the answer. It uses W3C XForms included in the XML content which allows the end user to directly interact with the remote server through the embedding site. The form markup in the XML content of an embedded page uses relative URL address for the target action, when the end user submits, the form data is sent to the containing site, which captures the form data and the relative URL. The Web Service Proxy then takes this information and re-submits it to the original site. It then reads the XML response and makes it available to the sitemap for styling again.

Q. Hmm ... a typical web application maintains a user session while navigating. How is the containing site propagating the end user session to the embedded site?

The answer is simple. The Web Service Proxy simply hooks to the end user session and automatically starts its own session with the remote site. If the remote site requires authentication, then the developer of the local web site has to pass the user credentials as parameters to the WebServiceProxyGenerator.

Q. What transport protocols are supported?

HTTP 1.0, HTTP 1.1, HTTPS.

Below we will illustrate the architecture of the solution with some example code and figures.

Figure 1 - Traditional Http Proxy vs Cocoon Web Service Proxy

Figure 1 - Architecture of the Web Service Proxy Solution. As opposed to a traditional proxy server, the Web Services Proxy captures user input and allows the web site to remain coherent even when the functionality for some of its components is delivered remotely.

Figure 2 - Illustration of the data flow for a composite page

Figure 2 - Illustration of the data flow for a composite page. Some of the content is locally constructed, the rest is obtained remotely. Finally the same styling is applied and the user facing page appears consistent.

Now we will show a snippet of the sitemap which employs the Web Service Proxy. Notice its brevity! The Web Service Proxy completely handles the content and navigation logic between the two portals. Only stylesheets are additionally required to translate the remotely retrieved documents into a user friendly format.

<?xml version="1.0"?> <map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0"> <!-- ========================== Components ================= --> <map:components> <map:generators default="file"> <map:generator name="wsproxy" logger="sitemap.generator.wsproxy" src="org.apache.cocoon.generation.WebServiceProxyGenerator"/> </map:generators> </map:components> <!-- ========================== Pipelines ================== --> <map:pipelines> <map:pipeline> <!-- Interactive Web Application Syndication --> <map:match pattern="*"> <map:generate type="wsproxy" label="xml" src="http://{request:serverName}:{request:serverPort}{request:contextPath}/samples/xmlform/wizard?
<map:transform src="stylesheets/newWizard2html.xsl"/> <map:transform src="context://samples/stylesheets/xmlform/xmlform2html.xsl"/> <map:serialize type="html"/> </map:match> </map:pipeline> </map:pipelines> </map:sitemap>

Figure 3 - sequence diagram

Figure 3 - Above is a sequence diagram outlining the interaction between the key participants in a syndication session.

screen shot 1

Figure 4 - Sample screenshot from a remotely enabled application as it appears standalone.

screen shot 2

Figure 5 - Sample screenshot from the same application embedded in another web application.

The content of the original XML page behind these two screenshot follows:

<?xml version="1.0"?> <document xmlns:xf="http://apache.org/cocoon/xmlform/1.0"> <xf:form id="form-feedback" view="userIdentity" action="wizard" method="GET"> <xf:caption>Personal Information</xf:caption> <error> <xf:violations class="error"/> </error> <xf:textbox ref="firstName"> <xf:caption>First Name</xf:caption> <xf:violations class="error"/> </xf:textbox> ... <xf:selectMany ref="role" selectUIType="listbox"> <xf:caption>Professional roles</xf:caption> <xf:item> <xf:caption>Geek</xf:caption> <xf:value>Geek</xf:value> </xf:item> <xf:item> <xf:caption>Hacker</xf:caption> <xf:value>Hacker</xf:value> </xf:item> ... </xf:selectMany> ... <!-- hidden model attribute --> <xf:hidden ref="hidden"> <xf:value>true</xf:value> </xf:hidden> ... <xf:submit id="next" class="button"> <xf:caption>Next</xf:caption> </xf:submit> </xf:form> <xf:output ref="count" id="show_count" form="form-feedback" class="info"> <xf:caption>Visits Count</xf:caption> </xf:output> </document>

The listing above contains markup in the XMLForm namespace. It is a presentation independent way to specify input controls. Being XForms compliant it is easy to learn and use. The XSLT stylesheets used to convert the XML above are very simple and will not be listed here. They can found in the Cocoon 2.1 distribution.

5. Conclusion

The Web Service Proxy component is tightly integrated with the Cocoon framework and is particularly convenient to use in combination with XMLForm to enable syndication of website functionality. With the presented sample, we only scratched the service of the possible applications. It is easy to see though for a creative mind how it can be extended in multiple directions. Although the solution we offered is conveniently applied with Cocoon, the concepts are generally applicable outside the framework as well. Exposing a Web Application functionality via XML is not just a "neat" feature any more. It opens the gates to a constellation of opportunities, not possible with the classical Model-2

approach where the business logic is directly tied to a graphical output like HTML.

5.1. Have more questions?

Look at the online demo available in the Cocoon distribution in the samples: http://{host}:{port}/{contextPath}/samples/proxy/.

Then study the source code and if you still have questions, join the cocoon users email list and ask. If you have ideas for improvement then you are more than welcome to discuss it on the cocoon development email list and eventually submit a patch through the Apache bug tracking system.

1. Comments

add your comments