Real-Life Service-Oriented Architecture (SOA) with D6

EMC Proven™ Professional Knowledge Sharing 2008

Pawan Kumar
Principal
Doquent, Inc.
pk@doquent.com

Table of Contents

Abstract 4

Introduction 5

Business Scenario 6

Outline 6

Key Concepts 7

Web Services 7

Web Services Description Language (WSDL) 7

SOA 7

Orchestration 8

WS-BPEL 8

Solution Architecture 9

Deployment 10

Interaction 11

Implementation Tools 12

DFS SDK 12

WS-BPEL 13

PHP 14

Axis 14

Implementation 15

Payment Document Service 15

API 15

Document Types 16

Payment Service 19

Invoice Payment Service 20

Web Application 23
View Invoice 23
Pay Invoice 24
Payment Confirmation 25
Summary 26
Acknowledgments 26
Bibliography 26
Author Biography 28

Disclaimer: The views, processes or methodologies published in this compilation are those of the authors. They do not necessarily reflect EMC Corporation's views, processes, or methodologies.

Abstract

Service-Oriented Architecture (SOA) facilitates composition of loosely-coupled services into new services and applications. SOA must deliver business and technical agility to be successful. In other words, SOA serves business goals other than the standard "potential cost reduction via technical changes."

This article translates a real-life business goal into an SOA solution that includes a key service provided by D6 (EMC Documentum version 6). While the article is primarily technical, it keeps business drivers and context at the forefront.

Even as SOA is becoming mainstream, Enterprise Content Management (ECM) has become a key piece of the enterprise infrastructure puzzle. Explosive content growth has led Gartner to forecast a 12% annual growth in ECM software revenues for the next three years. EMC Documentum has been a leading player in the ECM space so it should come as no surprise that SOA has been adopted as one of the core principles of D6.

D6 provides Documentum Foundation Services (DFS) for plugging Documentum into an SOA solution. Documentation and articles about D6 and SOA typically focus on DFS features. While they are essential for using Documentum in an SOA solution, they need another complementary capability for completing the SOA picture – *orchestration* of services for composing new services or applications. This article explores implementation of a real-life business scenario via an SOA solution including Documentum as a key component.

Consider a business goal to reduce Days Sales Outstanding (DSO) defined as the average collection period of account receivables over a time period (typically quarterly or yearly). A large DSO value means that it takes longer for the business to collect payments. A process analysis reveals that *invoice presentment and payment* are the primary improvement opportunities. This article describes an SOA-based implementation of an *Invoice Presentment and Payment Application*. Documentum is used to manage payment-related documents such as *invoices* and *payment receipts*.

The article illustrates the following key aspects of such a solution:

1. Using DFS for creating content-oriented services
2. Orchestrating SOA services for composing applications or other services
3. Addressing real-life concerns

Introduction

Documentum v5.3 included Web Services Framework (WSF), a relatively limited set of technologies designed for making Service-based Business Objects (SBO) available as web services. Essentially, a piece of business logic in Documentum could be exposed as a web service using WSF. [CMSW] Documentum v6.0 (D6) introduced Documentum Foundation Services (DFS), replacing WSF to develop and deploy ECM services, and deliver service to consumers using a service-oriented architecture.

DFS provides ready-made services and tools to create custom services and service consumers. The DFS-provided services are called Enterprise Content Services (ECS) and include Object Service, Version Control Service, Query Service, Schema Service, Search Service, and Workflow Service. The DFS Software Development Kit (SDK) facilitates creation of custom services as well as consumers of services. Using the DFS SDK, developers can easily generate new services from regular Java code, and these services can easily utilize the ECS.

DFS is designed to easily plug Documentum into a service-oriented architecture. Therefore, it is helpful to recap the key aspects of SOA [R15M]:

1. SOA is a component-based architecture
2. Components are loosely coupled by leveraging standards
3. Components are used to build composite applications
4. SOA is a joint initiative between IT and business

This article illustrates these SOA aspects while demonstrating how to build an application utilizing SOA and including Documentum as a key component. It describes the architecture at a high level and implementation at a mid level, without getting lost in the code. This approach guides decision-making needed for implementation. The intent is to provide tools, examples, and questions that can help developers address requirements appropriately.

Business Scenario

SOA must drive business agility [BIZA]; one of the key aspects of SOA is that it is a joint initiative between IT and the business. Thus, it is appropriate to start with the business requirement that will be supported by technology.

Consider a scenario where a business discovers that it takes them a long time to collect money owed to them. The business measures this time using a metric known as Days Sales Outstanding (DSO). This is the average collection period of account receivables measured over a time period (typically quarterly or yearly). [DSOW] A large value of DSO means that it takes longer for the business to collect payments.

A process analysis reveals that *invoice presentment* and *payment* processes are prime areas for improvement. The analysis concludes that an Invoice Presentment and Payment Application (IPPA) can facilitate viewing and payment of invoices through an additional channel to overcome the challenges faced in the current processes.

This article describes an SOA-based implementation of such an application; Documentum is used to manage payment-related documents such as *invoices* and *payment receipts*.

Outline

Before looking at implementation details, it is helpful to preview the article's organization. It begins with an overview of key concepts. Next, solution architecture describes the big picture of the solution and the interaction of the components. Implementation of this architecture uses certain tools and technologies, which are discussed next. Equipped with the concepts, tools, and the solution architecture, we will be ready to explore implementation details. The implementation section describes the components, their roles, and how they are implemented. The article concludes with a summary. Acknowledgments give credit where it is due and the Bibliography provides a list of relevant reading material.

Key Concepts

Web Services

A web service is defined as "a software system designed to support interoperable machine to machine interaction over a network." [WSWK] Essentially, a web service makes certain functionality available for other machines to consume. This is in contrast to applications that have a user interface (UI) for human interaction.

Web Services Description Language (WSDL)

The Web Services Description Language is an XML-based language that provides a model for describing Web services. [WSDL] The WSDL file is the standards-based public face of a web service that makes the implementation technology of the web service immaterial for consumers of the service. WSDL is a critical piece in the SOA implementation and facilitates the creation of services and applications using other web services.

SOA

Service-Oriented Architecture enables the creation of robust, standards-based, interoperable solutions that make the differences in platforms and implementation technologies of the various components inconsequential. [BKSO]

Note that an SOA can be based on various technologies, including DCOM and CORBA. However, web services are currently the prevalent choice for implementing SOA.

Orchestration

SOA solutions can be composed from services in a number of ways. You can write code that invokes the services and combines the results of invocations. However, *orchestrations* offer a more flexible approach. These are "composite, controller services defining how the services being consumed will interoperate to get the job done." [BKSO]

An orchestration assembles services into a process executed by an orchestration engine. In simpler words, the interaction with the services is defined (declaratively) in a format that the engine understands. The engine interprets this process definition and executes those interactions.

Orchestration controls the interaction with other services. There is also a web services *choreography* specification that provides another way of creating interaction among services. In choreography, the centralized control is replaced by peer-to-peer relationships among the services. [BKSO]

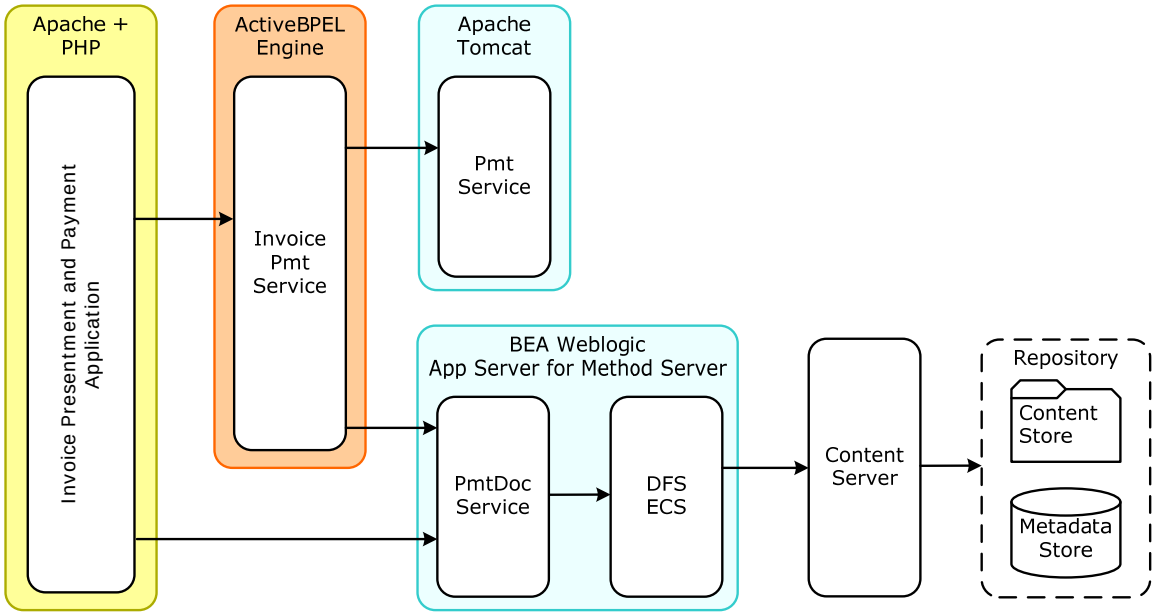
WS-BPEL

WS-BPEL is an orchestration language used to describe execution logic of a process and its interaction with web services. WS-BPEL is based on several specifications such as SOAP (Simple Object Access Protocol), WSDL, and XML schema.

There are other specifications such as XLANG, WSFL, XPDL, and BPML that are alternatives to WS-BPEL as executable business process languages. [BKSO]

Solution Architecture

The IPPA can be architected in many ways even while employing SOA. This article employs a specific architecture and describes the solution using the details associated with this choice. Nevertheless, the concerns raised and various aspects of techniques adopted for implementation will be similar to those encountered in many other choices. The solution architecture:



Solution Architecture for IPPA

There are two key services in this solution – *Payment Service* (`PmtService`) and *Payment Document Service* (`PmtDocService`). The Payment Service processes a payment and provides a confirmation number. In real life, this may be a third-party service or a service wrapping a third-party payment solution. The Payment Document Service manages payment-related documents (invoices, payment receipts) in a D6 repository.

Note the important architectural principles. Payment Document Service builds upon DFS and thus reuses deployed and operational code. It also creates a layer on top of DFS, encapsulating the detail that Documentum is being used for document management. If the interface of Payment Document Service does not expose Documentum-specific details, upgrades and changes to the document management system can be shielded.

The application is a web application and possesses a simple UI. However, the invoice payment process requires several interactions with the Payment Service and the Payment Document Service. This invoice payment process is implemented using a WS-BPEL process that is exposed as a service as well.

By keeping the invoice payment process out of the web application, a new service is created which is available as a service for participating in other solutions as well. This approach also keeps the web application simple and focused on user interaction.

Deployment

IPPA components are deployed as follows. The UI is implemented using PHP and is deployed on an instance of the Apache Web Server. PHP is used for illustrating the heterogeneity of solution components as well as for benefiting from the ease of creating web service clients.

The invoice payment process is implemented as a WS-BPEL process and is deployed on the ActiveBPEL Engine running in an Apache Tomcat instance. [ACTE]

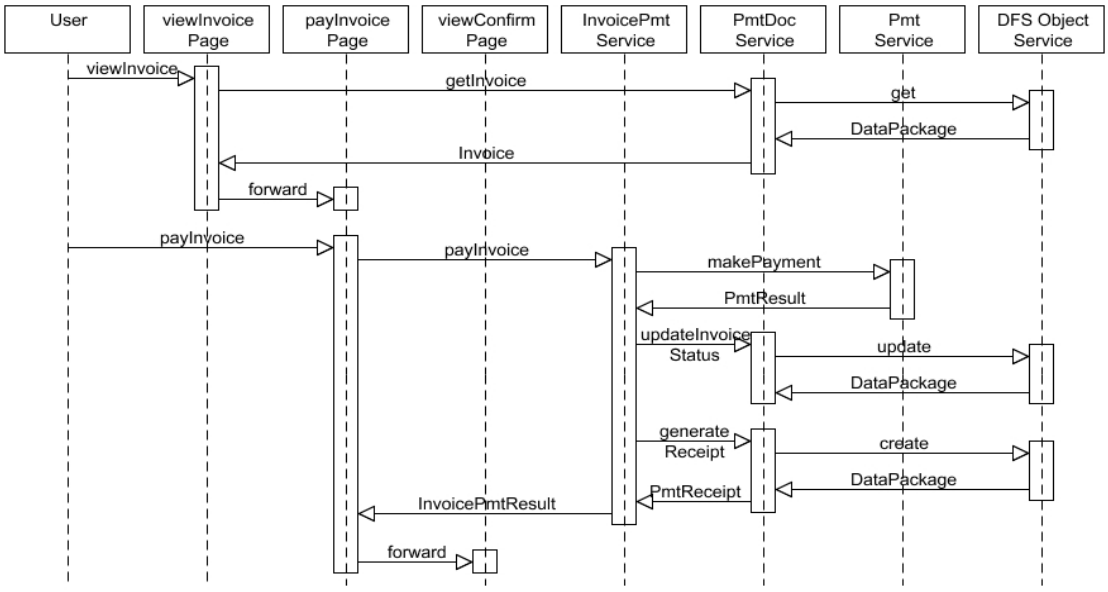
The Payment Service is a Plain Old Java Object (POJO) service deployed in an Axis2 web application instance running on an Apache Tomcat instance.

The Payment Document Service is a custom DFS service and is deployed in the Weblogic server instance that hosts the Documentum Java Method Server as well as the DFS ECS.

In real life, deployment choices depend on several factors such as the technologies in place, IT staff skill set, and performance and security requirements.

Interaction

This diagram illustrates invoice payment interaction among the application components.



Sequence Diagram for IPPA

The user paying an invoice retrieves the `viewInvoice` page. The user enters information to identify the invoice to be paid and submits the information. The page invokes the `PmtDocService`, which utilizes the DFS Object Service to obtain the invoice information. Finally, invoice information is returned to the page. The page then forwards the information to the `payInvoice` page. The `payInvoice` page presents the invoice information to the user. The user adds the payment information, such as credit card details, and submits the payment request.

The invoice payment request is sent to the `InvoicePmtService`. This is the BPEL service that orchestrates the interaction with the `PmtService` and `PmtDocService` to complete the payment. In addition to completing the payment, it updates the invoice status and generates a receipt before returning payment confirmation information. Note how the application makes only one request for paying the invoice and the multiple interactions with the underlying services are encapsulated within the BPEL process.

Finally, the user is forwarded to the `viewConfirm` page that shows payment confirmation information.

Implementation Tools

This section describes the tools for implementing this solution. This information will be helpful when discussing the implementation of the solution components later.

DFS SDK

DFS enables a Documentum deployment to participate in an SOA infrastructure. DFS ECS consist of core web services for interacting with the content server. The operations made available by ECS are relatively fine-grained and reflect the existing API for interacting with Documentum Content Server. The DFS SDK includes tools for generating consumers of DFS services where the consumers themselves can be services. The SDK also contains sample code to jumpstart development. [DQRS]

Note that the SDK can create Java clients for DFS services. These clients utilize included libraries, leaving only the business logic to be added. On the other hand, a DFS service is a standard web service with its interface published through a WSDL file. Any client that is capable of understanding WSDL and interacting with the underlying service can interact with such services.

Creating clients purely based on WSDL requires a better understanding of the Web Services infrastructure. There are some details about interaction with DFS services that are automatically handled by the generated Java client. Such details need to be explicitly addressed when a client is generated using WSDL but not the DFS SDK. For example, the DFS service classes generated by the SDK make the service an authorized service. A DFS SDK-generated Java client takes care of this aspect transparently. However, a pure WSDL-based client either needs to handle this authorization explicitly or the service must be made anonymous. [DQAF]

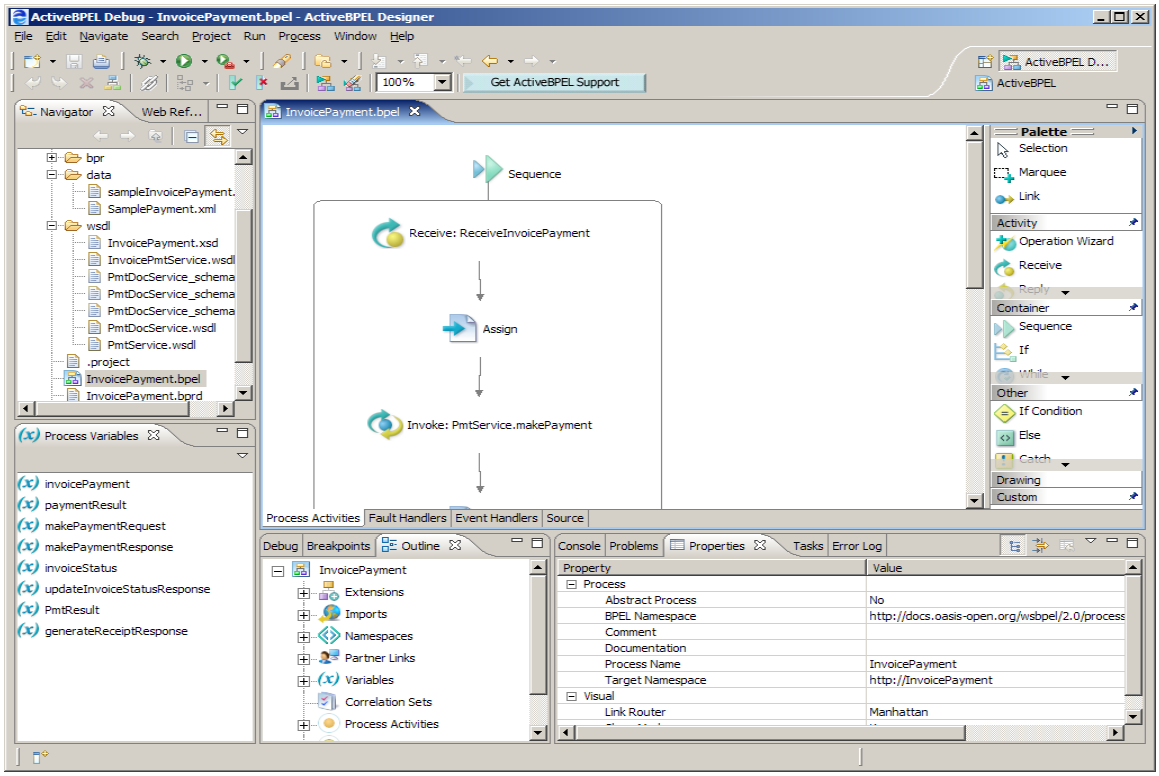
The Payment Document Service uses the Object Service from DFS ECS. Understanding the DFS data model supports understanding the sample code provided in the DFS SDK, as well as understanding the creation of services and consumers. At a minimum, we need to understand the classes used for identifying objects, passing arguments, and returning results. The Documentum Foundation Service Development Guide [DFSD] describes the SDK and the core services in detail and includes examples.

WS-BPEL

As explained earlier, WS-BPEL is a language for specifying business process behavior based on Web Services. Processes in WS-BPEL export and import functionality by exclusively using Web Service interfaces. [BPEL] A BPEL process declaratively implements a process that executes on a BPEL engine. BPEL development tools enable developers to design processes, generate related artifacts, and deploy them on a BPEL engine instance conveniently.

Various tools are available to implement BPEL processes. ActiveBPEL is an open-source WS-BPEL engine created by Active Endpoints. [ACTE] Oracle offers Oracle BPEL Process Manager & Designer for WS-BPEL development and deployment. [ORBP] Sun offers WS-BPEL capabilities in the NetBeans IDE. [NETB] Microsoft offers BPEL support in Windows Workflow Foundations. [MWWF] This list is not exhaustive since most players in the application server business have BPEL offerings. This article uses ActiveBPEL for implementing the BPEL process in the solution.

Active BPEL Designer is an Eclipse-based Integrated Development Environment (IDE) that includes tools and wizards for designing a BPEL process, its deployment descriptor, and its business process archive for deployment to the ActiveBPEL Engine. It includes an instance of the ActiveBPEL Engine. The ActiveBPEL Engine is a web application deployed on Apache Tomcat application server by default. However, wizards allow deployment to another instance on the engine. The following figure shows a snapshot of the designer workbench.



PHP

This application uses PHP to illustrate how diverse technologies can interact with each other primarily using WSDL as the interface. The PHP SOAP extension [PHPS] makes it easy to create consumer and web services with PHP. In this application, the PHP client interacts with services and presents information on the UI.

Axis

The Axis library has become the de facto standard to implement web services in Java. A large number of tools and libraries related to web services utilize Axis. [AXIS] Axis includes development and deployment tools. It is fairly straightforward to generate WSDL from Java code or the client stub/server skeleton from a WSDL file using Axis. The Axis web application contains the runtime for hosting Java web services as well as an administration application for managing the deployed services.

Implementation

The solution architecture includes the following key components:

1. Payment Document Service – custom DFS-based service
2. Payment Service
3. Invoice Payment Service
4. Web application

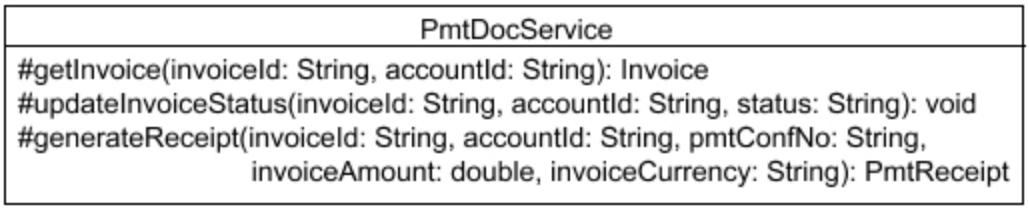
Note that DFS ECS is available out-of-the-box and does not need to be implemented. Implementation of the above-mentioned components and related concerns are discussed below.

Payment Document Service

The Payment Document Service provides document management functionality for documents related to invoice payment processing. Implementation aspects of this service are discussed below.

API

The API for the Payment Document Service is shown below:



`PmtDocService` API

This service was created using the DFS SDK as a custom service. The SDK samples and the DFS Development Guide facilitated this implementation. This service is deployed on the Java Method Server (running on Weblogic), along side DFS ECS. The service was tested using the SDK-generated Java client. It was also tested using an Axis2-based client, generated from the service WSDL.

A snippet of the WSDL file `PmtDocService.wsdl` reflecting the API is shown below:

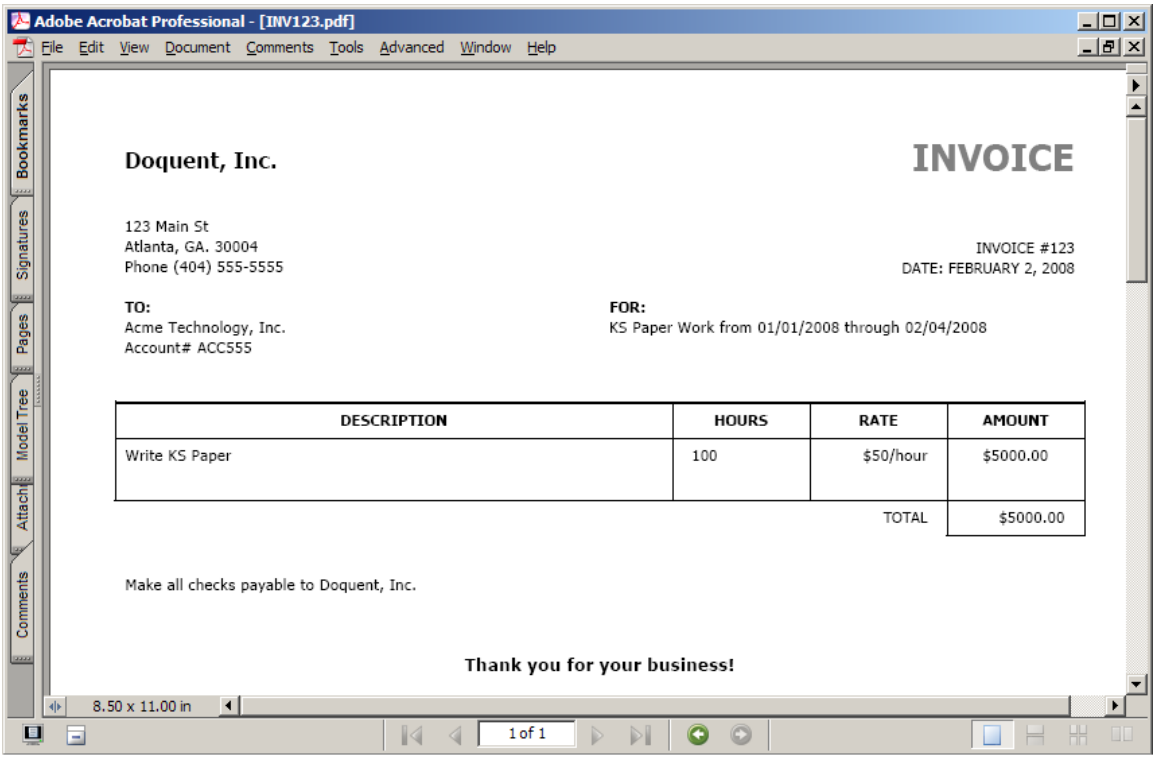
```
<portType name="PmtDocServicePort">
  <operation name="getInvoice">
    <input message="tns:getInvoice"/>
    <output message="tns:getInvoiceResponse"/>
    <fault name="SerializableException"
           message="tns:SerializableException"/>
  </operation>
  <operation name="updateInvoiceStatus">
    <input message="tns:updateInvoiceStatus"/>
    <output message="tns:updateInvoiceStatusResponse"/>
    <fault name="SerializableException"
           message="tns:SerializableException"/>
  </operation>
  <operation name="generateReceipt">
    <input message="tns:generateReceipt"/>
    <output message="tns:generateReceiptResponse"/>
    <fault name="SerializableException"
           message="tns:SerializableException"/>
  </operation>
</portType>
```

IPPA expects the invoice to be present in the document repository before a payment can be made on that invoice. Therefore, invoices are not added to the repository by this application, another process must add the invoices along with the metadata to the repository. If an operation is needed for generating or adding an invoice, `PmtDocService` would be the perfect place for it.

Document Types

`PmtDocService` manages invoices and payment receipts in this solution. These documents also need to store some associated metadata in the underlying repository. The implementation details for supporting these document types are discussed below.

A sample invoice document may look like the following:



Sample Invoice Document

To store the associated metadata in the Documentum repository, a custom object type was created – `dq_ksinvoice`, which extends `dm_document` and is described below.

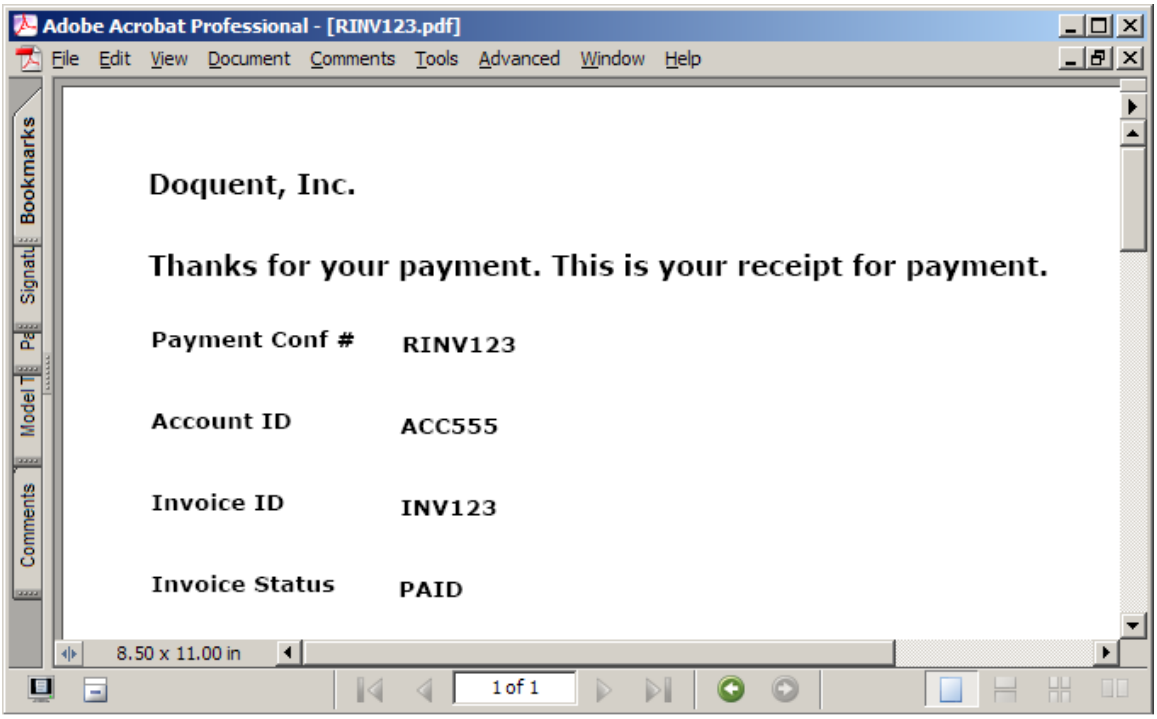
	attr_name	type	label	repeating
1	account_id	Char(32)	Account ID	
2	invoice_amount	Double	Invoice Amount	
3	invoice_currency	Char(3)	Invoice Currency	
4	invoice_id	Char(32)	Invoice ID	
5	invoice_status	Char(32)	Invoice Status	

Custom Type `dq_ksinvoice`

Note that this metadata serves three purposes:

1. identify the invoice – `account_id` and `invoice_id`
2. capture amount due – `invoice_amount` and `invoice_currency`
3. track status in the payment process – `invoice_status`

A receipt is generated and added to the repository with appropriate metadata after a payment has been made. A sample receipt corresponding to the invoice:



Sample Receipt Document

The receipt document is stored as an object of a custom type – `dq_kspmtreceipt`, that extends `dq_ksinvoice` and is described below. Thus, a receipt object also has the invoice custom attributes shown above.

	attr_name	type	label	repeating
1	pmt_conf_no	Char(32)	Payment Confirmation Number	

Custom Type `dq_kspmtreceipt`

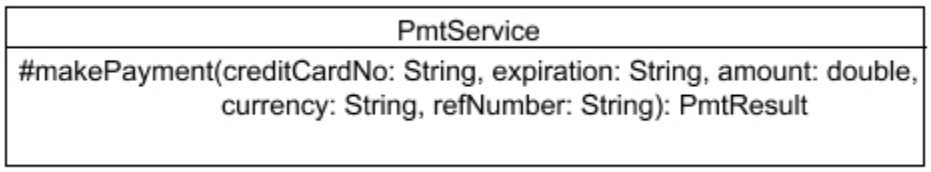
The receipt stores a payment confirmation number in addition to the attributes associated with an invoice.

Note: a receipt is not *a kind of* invoice so direct inheritance is probably not appropriate in a production solution. Here `dq_kspmtreceipt` extends `dq_ksinvoice` just for acquiring its attributes.

Payment Service

The Payment Service is a simple Axis2-based Java service with only one operation – `makePayment`. This operation creates a confirmation number and returns. In a real-life solution, the Payment Service would perform secure authentication, validate the payment information, process payment, and then generate a confirmation number.

The API for the Payment Service is shown below.



`PmtService` API

The API is published through a WSDL file – `PmtService.wsdl`. A snippet of this file:

```
    <wsdl:portType name="PmtServicePortType">
        <wsdl:operation name="makePayment">
            <wsdl:input message="axis2:makePaymentRequest"
                   wsaw:Action="urn:makePayment"/>
            <wsdl:output message="axis2:makePaymentResponse"
                   wsaw:Action="urn:makePaymentResponse"/>
        </wsdl:operation>
    </wsdl:portType>
```

`PmtService` is deployed within the Axis web application.

The following figure shows the Axis2 web application view of the deployed service:

`PmtService` Deployed on Axis2

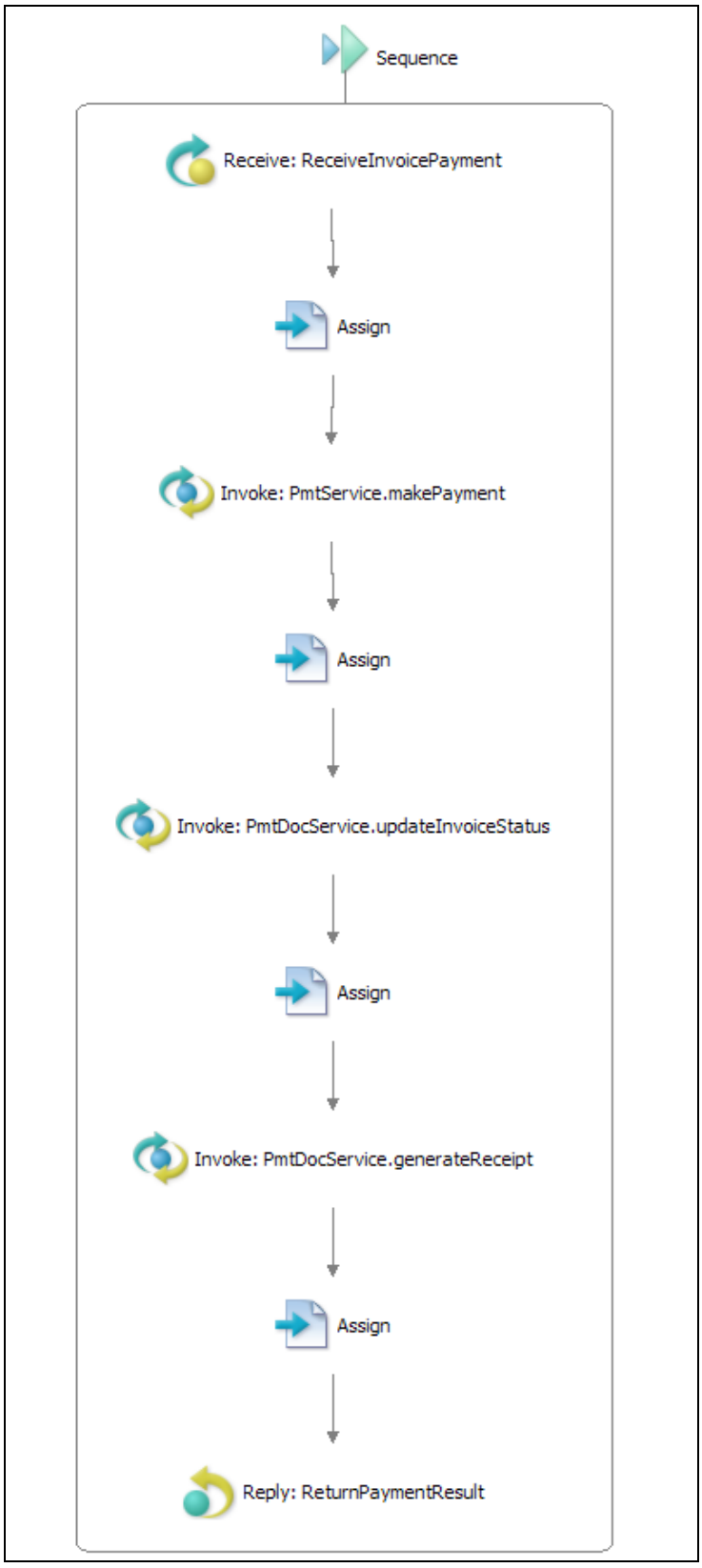
Invoice Payment Service

The invoice payment process includes the following steps:

1. Make payment
2. Update invoice status
3. Create receipt and store in the Documentum repository

In this process, data needs to be passed from one step to another. Since these steps are available as operations on other services, a BPEL process is composed using the other services. This process is exposed as a web service, Invoice Payment Service.

The following figure shows the BPEL process being designed in the ActiveBPEL Designer:



Invoice Payment BPEL Process

This BPEL process is a simple sequential interaction with other web services. Active BPEL Designer (as well as the BPEL specification) supports complex processes that use conditional flow control.

The visually designed process definition is persisted as an XML file. The following snippets show parts of this BPEL process definition in XML source form:

```
<bpel:sequence>
    <bpel:receive createInstance="yes" name="ReceiveInvoicePayment"
          operation="payInvoice" partnerLink="invoicePmtProviderLT"
          portType="ns1:invoicePmtServicePT"
          variable="invoicePayment"/>
    <bpel:assign>
       <bpel:copy>
          <bpel:from>
                $invoicePayment.body/creditCard/number
          </bpel:from>
          <bpel:to>
                $makePaymentRequest.parameters/ns3:creditCardNo
          </bpel:to>
```

```
    <bpel:invoke inputVariable="makePaymentRequest"
          name="PmtService.makePayment" operation="makePayment"
          outputVariable="makePaymentResponse"
          partnerLink="ccPaymentLT"
          portType="ns2:PmtServicePortType"/>
```

Note that the last two invocations in this process are to the same service, `PmtDocService`. This setup shows a “chatty” interaction that can be wrapped in a coarse-grained operation on `PmtDocService`, say `postProcessPayment`. This new operation can replace two web service calls with one, potentially reducing overhead and improving performance.

The API for Invoice Payment Service is published through a WSDL file. The following snippet from the WSDL file shows the available operation:

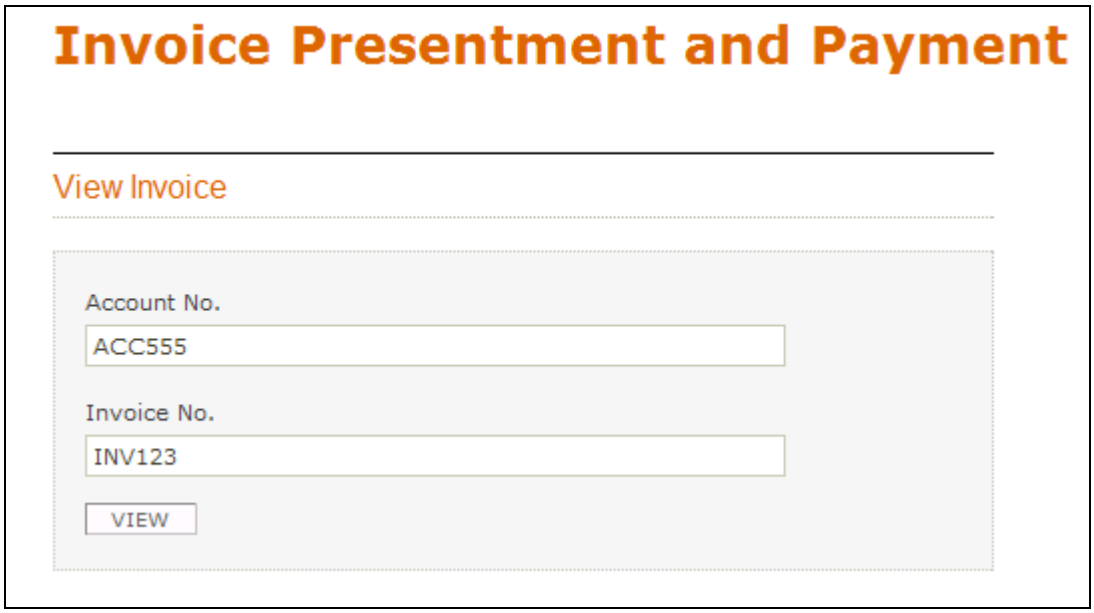
```
 <wsdl:portType name="invoicePmtServicePT">
     <wsdl:operation name="payInvoice">
         <wsdl:input message="tns:invoicePayment"/>
         <wsdl:output message="tns:paymentResult"/>
     </wsdl:operation>
```

Web Application

The web application is a direct or indirect consumer of the services discussed above. As mentioned earlier, the web application is implemented using PHP though it could be implemented with any other technology capable of interacting with web services. The web application itself is not the focus of this article; however, the UI provides a tangible way to bring the solution together. We will keep the web application very simple with three pages – View Invoice, Pay Invoice, and Payment Confirmation.

View Invoice

The View Invoice Page retrieves an invoice by providing an account number and an invoice number. If an invoice exists in the document repository with the matching metadata, it will be retrieved and the user will be forwarded to the Pay Invoice page.



View Invoice Page

Clicking on VIEW submits the request to retrieve the invoice.

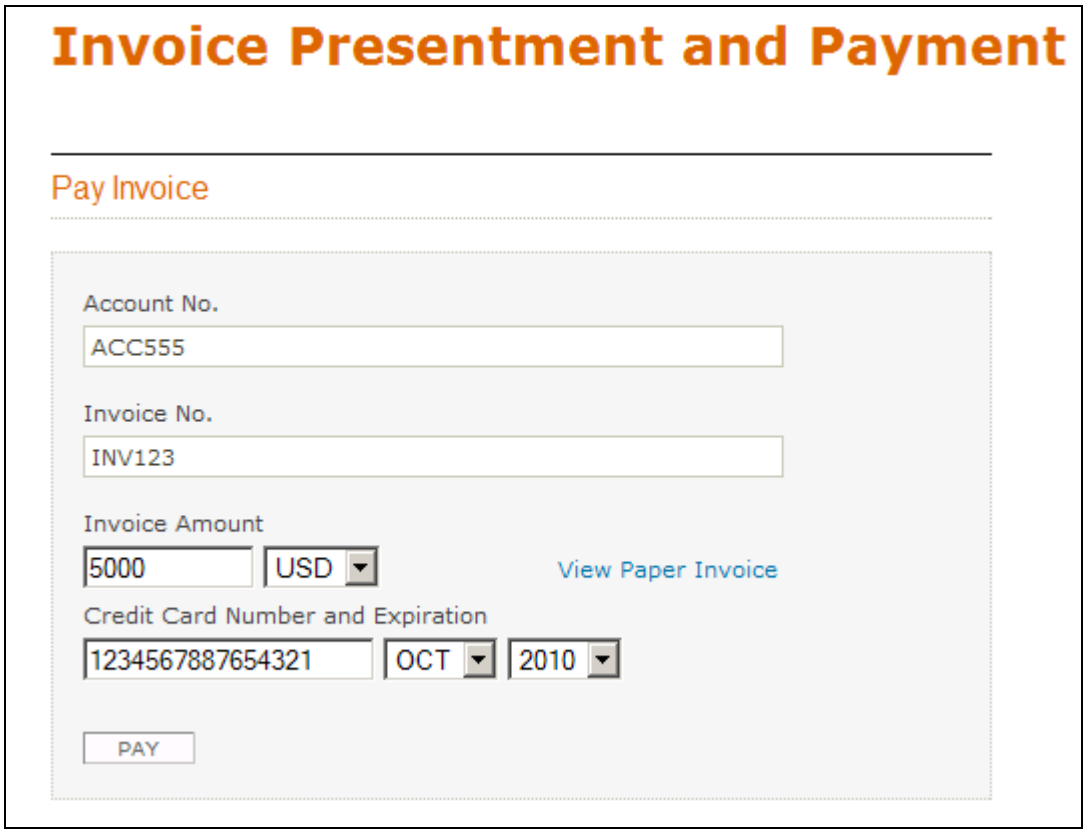
Note that in real life, such an application would have stringent security requirements throughout the solution. The UI will require authentication and encryption (HTTPS). It is likely that authentication and encryption may also be required by the web services. Additional security measures may also be needed on the content and data stores.

Pay Invoice

The invoice retrieved by the request from View Invoice Page is displayed on the Pay Invoice Page. This page populates the amount and the currency of the amount due in the invoice.

Note that there is a link, View Paper Invoice, to retrieve the invoice PDF if the user wishes to view it.

There are additional fields on the page for accepting payment information. This page only shows credit card as a means of payment. In a real system, there may be other options (such as PayPal, bank debit) for making a payment. Further, additional information may be required to accept payment, e.g. credit card payment may require credit card type (Visa, MasterCard, etc.), billing address, security code etc.



Pay Invoice Page

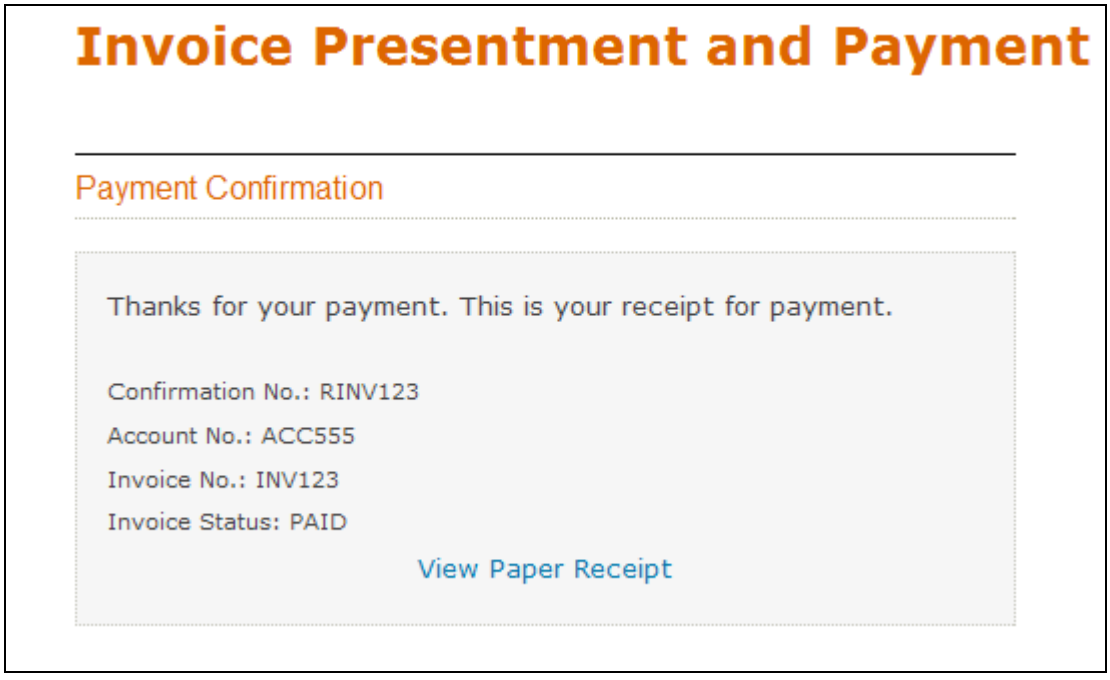
Clicking on PAY submits the information to complete the payment. In a real application, this would take the user to a verification page with an option to cancel payment.

Content transfer is a key concern for a document management service. An unnecessary content transfer can have a significant impact on performance. Further, content transfer logic may require special attention to get the content to the final consumer. Architectural concerns, such as caching, also apply when content transfer is involved. The link for retrieving content on this page frees up other document-related logic to be implemented without retrieving content.

Note that a real-life invoice payment system may have an additional step to notify the user. This could be accomplished by an additional operation named `emailReceipt` on `PmtDocService`. This operation would retrieve the receipt from the repository and email it to the user as an attachment. In terms of the UI impact, this would add notification option fields on the Pay Invoice Page.

Payment Confirmation

Once the invoice payment completes successfully, the user is shown a payment confirmation. A link is provided for retrieving the payment receipt PDF file.



Payment Confirmation Page

Summary

This article guided you through the process of solving a business problem by implementing an SOA solution, with Documentum providing a key component service. Key concepts, technologies, and tools were introduced and discussed. While the solution implementation was simple, real-life concerns were discussed along with the implementation. The key takeaways from this article are:

1. How Documentum can participate in an SOA solution
2. How WS-BPEL technology can be used to implement SOA

Acknowledgments

This article would not have been possible without the unwavering support of my wife Rashmi, who had to manage our two young children longing for my attention while I worked on this article. I am also thankful for the patience of my friends who understood my lack of availability during this period.

Bibliography

[ACTE] Active Endpoints http://www.active-endpoints.com/

[AXIS] Apache Axis2 http://ws.apache.org/axis2/

[BKSO] Book: SOA and WS-BPEL http://www.packtpub.com/SOA-WS-BPEL-PHP-Open-Source-ActiveBPEL/book/mid/081107agjf48

[BIZA] The promise of SOA is business agility http://www.ebizq.net/blogs/it_directions/archives/2006/04/the_promise_of.php

[BPEL] Business Process Execution Language http://en.wikipedia.org/wiki/BPEL

[CMSW] D6 and Your SOA http://www.cmswire.com/cms/enterprise-cms/documentum-6-d6-and-your-services-oriented-architecture-soa-001846.php

[DFSD] Documentum Foundation Service Development Guide, Version 6

[DQAF] “Authorization Failed” in DFS-based Service (D6) http://doquent.wordpress.com/2008/01/26/authorization-failed-in-dfs-based-service-d6/

[DQRS] Running DFS Sample Code for D6 http://doquent.wordpress.com/2008/01/24/running-dfs-sample-code-for-d6/

[DSOW] Daily Sales Outstanding http://en.wikipedia.org/wiki/Days_Sales_Outstanding

[GART] Worldwide Enterprise Content Management Software Market Will Reach $4.2 Billon in 2010 http://www.gartner.com/it/page.jsp?id=506302

[IBMR] Patterns: Service Oriented Architecture and Web Services http://www.redbooks.ibm.com/redbooks/pdfs/sg246303.pdf

[IBMS] Service-Oriented Modeling and Architecture http://www-128.ibm.com/developerworks/webservices/library/ws-soa-design1/

[MWWF] BPEL for Windows Workflow Foundation http://www.microsoft.com/downloads/details.aspx?FamilyID=6D0DAF00-F689-4E61-88E6-CBE6F668E6A3&displaylang=en

[NETB] NetBeans Enterprise Pack 5.5 http://developers.sun.com/jsenterprise/nb_enterprise_pack/

[ORBP] A Hands-on Introduction to BPEL http://www.oracle.com/technology/pub/articles/matjaz_bpel1.html

[PHPS] PHP SOAP Extension http://us3.php.net/soap

[R15M] Review of “A 15-Minute Guide to Service-Oriented Architecture and ECM” http://wordofpie.wordpress.com/2008/01/03/a-15-minute-guide-to-soa-and-ecm/

[SOAD] http://blogs.ittoolbox.com/eai/business/archives/soa-design-service-decomposition-7813

[SOAL] Bringing SOA to Life http://webservices.sys-con.com/read/164560.htm

[WSBP] OASIS Web Services Business Process Execution Language http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel

[WSDL] WSDL – Wikipedia http://en.wikipedia.org/wiki/Web_Services_Description_Language

[WSWK] Web Services – Wikipedia http://en.wikipedia.org/wiki/Web_service

Author Biography

Pawan Kumar is the author of the book - Documentum Content Management Foundations: EMC Proven Professional Certification Exam E20-120 Study Guide. He is a Technical Architect with experience in solution delivery and product development on the Documentum platform. His experience spans solution architecture, document management, systems integration, web content management, business process management, imaging and input management, and custom application development. He holds an MS in Computer Science from University of North Carolina at Chapel Hill and a BS in Electrical Engineering from the Indian Institute of Technology, New Delhi (India).

Pawan believes in giving back to the community and he practices this belief in several ways. He is an active contributor to the Documentum-users Yahoo! user group and dm_developer discussion forums. He founded the online community dm_cram to help community members prepare for Documentum certification exams. He frequently shares his thoughts and experiences on Content Management etc.