Online Souvenir Store using Service Oriented Architecture (SOA) Hands on Project Report

Neon Ngo and Tracy Tran
George Mason University
neonngo@yahoo.com, tracy.tran13@gmail.com

Abstract

Service-Oriented Architecture (SOA) is a software architecture that supports communication between services. An ecommerce application was built using this architecture. The application is an online Souvenir Store that allows users to search and browse through the store's catalog, add items to a shopping cart, as well as purchasing the items in the cart. Users are able to self register to the system and maintain their shipping address and credit card information. A database service is used to maintain the store's inventory and user information, a credit card web service is used to validate the users' credit card information, and an email service is used to send confirmation emails to the users.

1 Introduction

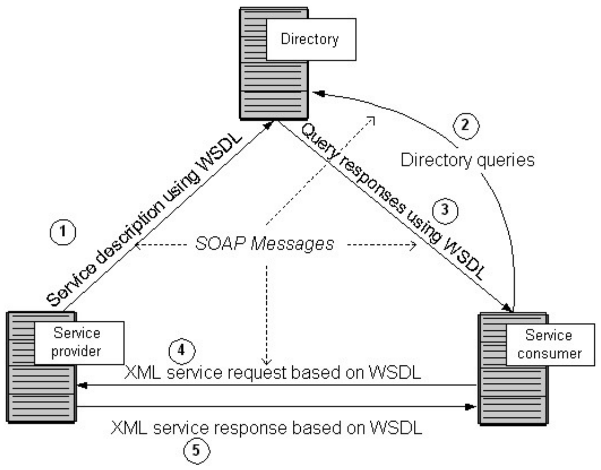
1.1 Service Oriented Architecture

Service-Oriented Architecture (SOA) is a software architecture that supports communication between services. A service is a unit of work that is to be performed on behalf of a client. In a SOA environment, a node on a network makes its resources/service available to other nodes on the network. These services are loosely coupled and highly interoperable. These services interoperate based on a formal definition (or contract) which is independent from the underlying platform and programming language. SOA is not a totally new architecture; rather, it is an evolution that captures the best practices of other distributed architectures that came before it by stressing the importance of using standards, which are well-defined and highly interoperable.

1.1.1 Web Services

Web Services is one of the types of services that can facilitate a SOA application. Web Services achieve interoperability because the services are based on a formal contract which is independent from the underlying platform or languages. In order to encapsulate the vendor and language specific implementation, each service provides its clients a Web Service Description Language (WSDL) file which describes what type of services are provided. The WSDL is published to a service directory and can be located by the client. Web services are accessible via standard protocols, such as SOAP, over Hyper-Text Transfer Protocol (HTTP). SOAP is a protocol for exchanging Extensible Markup Language (XML) based messages over a network even in the presence of a network firewalls.

Figure 1 below demonstrates the basic Service-Oriented Architecture implementing Web Services. The figure depicts the service provider publishing its WSDL to a directory of services. The service consumer then queries the directory to locate a service and determine how to communicate with that service. The directory then passes the WSDL to the consumer. Using the WSDL, the consumer determines what type of requests it wants to send to the service provider and the service provider in turn responses with an appropriate message. Via the SOAP protocol, requests and responses are sent using SOAP envelops, which are XML based.



Directory
Service description using WSDL
Query responses using WSDL
Directory queries
SOAP Messages
Service provider
Service consumer
XML service request based on WSDL
XML service response based on WSDL
1
2
3
4
5

Figure 1. SOA implementing Web Services

1.1.2 Other Services

As mentioned earlier, a service is a unit of work that is to be performed on behalf of a client. A service within a SOA environment doesn't necessarily have to be a Web Service. A service can be a database in which the client communicates with to retrieve and manipulate data. A database service is highly interoperable as it normally uses the ANSI/ISO Structured Query Language (SQL) to create, modify, retrieve and update data from a database management system (DBMS). A service can also be a mail service which routes and receive emails. The Simple Mail Transfer Protocol (SMTP) is the standard for email transmission on the Internet. All these different interoperable services communicating with each other over a distributed network to support the requirements of its users make up a Service-Oriented Architecture environment.

1.2 Rationale behind Proposed Project

Our goal was to build a specialized online shopping "store" application, the Souvenir Store, which implements the Service-Oriented Architecture. As opposed to other existing ecommerce sites such as Amazon.com and Buy.com, the Souvenir Store sells only souvenir items specifically related to Washington D.C. Washington D.C. souvenirs are generally available only within the Washington D.C. area itself. By making these souvenirs available for purchase online, souvenir vendors can make more money from travelers who may have forgotten to purchase souvenirs while they were visiting Washington D.C.

Our online Souvenir Store allows the users to browse, search, and add items to a shopping cart. The application also allows the users to purchase the added items. Users can self register and maintain their account information, such as shipping and credit card information, online.

2 Solutions/Analyses

2.1 Application Architecture

In building the Souvenir Store that that consists of a Web Service as well as a database service and mail service, the architecture of the application needed to be defined. The application consists of two clients and four services. The first client is the web browser in which the customers interface with to perform their online shopping transactions.

The browser interfaces with our "store" service/server via HTTP when servicing the users' requests. The "store" service is a web application that returns standards based documents such as Hyper-Text Markup Language (HTML), Cascading Style Sheets (CSS), JavaScript, and Joint Photographic Experts Group (JPEG) images. The "store" service is also a client in itself. It is a client that interfaces with our remaining three services, the Credit Card service (Web Service), the Store Database service (MySQL Database), and the Email service (Mail Service). The Credit Card Validation service is used by the "store" during the checkout process to authenticate the customer's credit card. The Credit Card Service is also used to validate the user's credit card during self-registration or account updates when the user provided credit card information. The Store Database service keeps track of our store inventory and catalog as well as registered users. It is used by the "store" when the user browses or search through the store's catalog. Lastly, the "store" interfaces with our fourth service, the Email service, via SMTP to send email confirmations to the customers after a user registration is completed, user account information is updated, and after orders are successfully placed.

Figure 2 depicts the Souvenir Store application architecture.

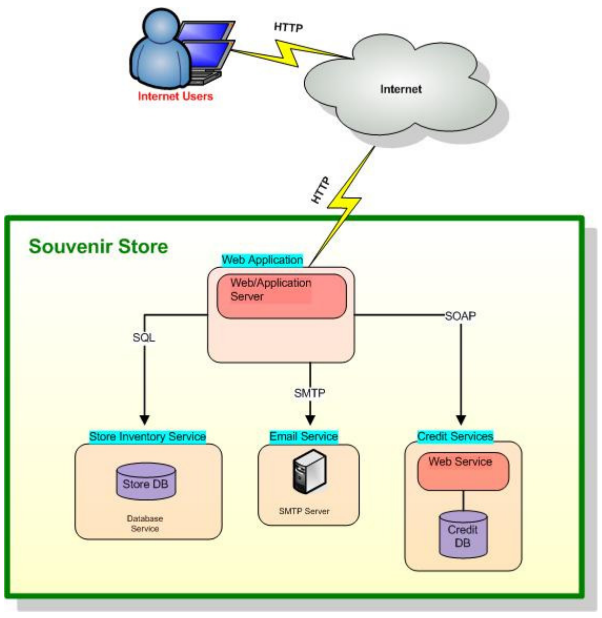


Figure 2. Application Architecture

2.2 Framework Architecture

The application was built using Ruby on Rails (RoR). Rails is a framework on top of the Ruby programming language for developing database-backed web applications. The rationale for using this framework is that Ruby on Rails simplifies and saves time on web development by providing a full stack framework that uses "Convention Over Configuration" (COC), "Don't Repeat Yourself" (DRY), and "Keep It Simple Stupid" (KISS) principles.

2.2.1 Model-View-Controller Architecture

Rails follows the Model-View-Controller (MVC) architecture. The model layer consists of data and business rules that govern access to and manipulation of this data. Rails uses Object/Relational Mapping (ORM) to map the database tables to classes. For example, if we have a table called *orders*, a Rail's application will have a class called *Orde*r. Active Record, part of the Rails' library, is used through out our application to interface with the database. The View layer is responsible for generating the user interface based on data in the model. In Rails, dynamic contents can be generated by embedding Ruby code within the HTML, which are known as RHTML templates. The controller layer receives events from the view and determines the actions that need to be performed by the model by handling the interaction between the browser, the view and the model. Rails handles this interaction behind the scene so the code written within this layer can focus more on the application-level functionality. Rails have internal support which links the three layers together seamlessly without much configuration or programming. Essentially, Ruby on Rails reduces a large amount of plumbing that developers have to do either programmatically and via configuration.

Figure 3 below depicts the MVC architecture within Rails.

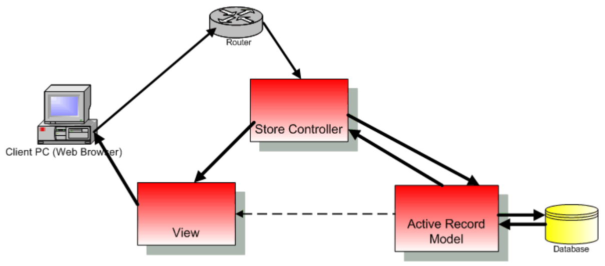
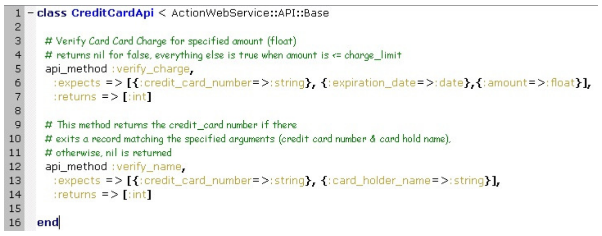


Figure 3. Rails and Model-View-Controller Architecture

2.3 Web Services

Rails' library provides support for implementing Web Services. To implement the Credit Card web service, Rails' Action Web Service was used. Action Web Service is a part of the Rail's library which supports the use of SOAP and XML-RPC web service protocols. Using Action Web Service, we declared and published the Credit Card service APIs via WSDL with little effort. It also allows a Action Web Services client to easily import the much simpler Credit Card Service API definition and use directly by the Web Services client without having to go through steps of generating stub classes, that most other web service implementations require (Apache Axis, Java Web Services Developer Pack (WSDP), Microsoft .NET framework, and Codehaus XFire)

Figure 5 depicts the Credit Card Web Service WSDL generated by Rails' Action Web Service framework based on the simple API definition shown in Figure 4



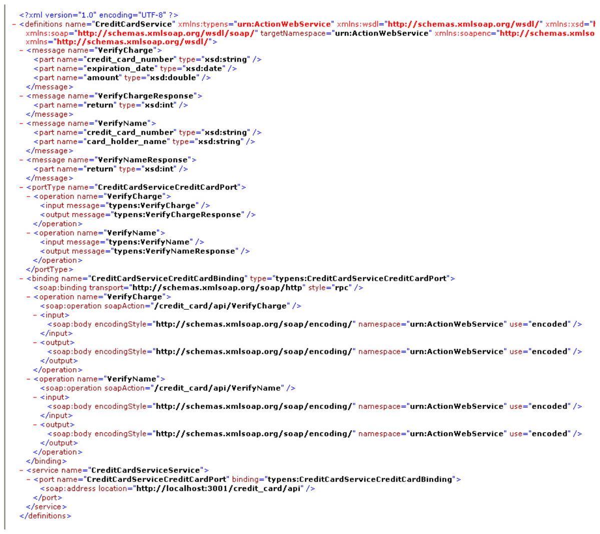
```ruby
- class CreditCardApi < ActionWebService::API::Base

    # Verify Card Card Charge for specified amount (float)
    # returns nil for false, everything else is true when amount is <= charge_limit
    api_method :verify_charge,
      :expects => [{:credit_card_number=>:string}, {:expiration_date=>:date},{:amount=>:float}],
      :returns => [:int]

    # This method returns the credit_card number if there
    # exits a record matching the specified arguments (credit card number & card hold name),
    # otherwise, nil is returned
    api_method :verify_name,
      :expects => [{:credit_card_number=>:string}, {:card_holder_name=>:string}],
      :returns => [:int]

  end
```

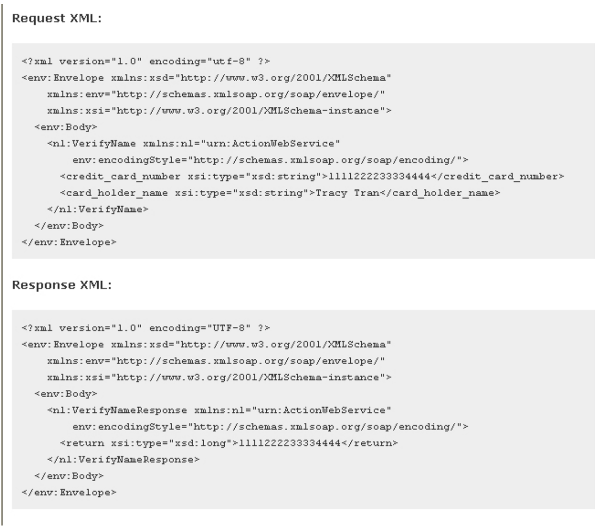
Figure 4 Credit Card Web Service API definition



```xml
  <?xml version="1.0" encoding="UTF-8" ?>
- <definitions name="CreditCardService" xmlns:typens="urn:ActionWebService" xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/" xmlns:xsd="
    xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/" targetNamespace="urn:ActionWebService" xmlns:soapenc="http://schemas.xmlso
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  - <message name="VerifyCharge">
      <part name="credit_card_number" type="xsd:string" />
      <part name="expiration_date" type="xsd:date" />
      <part name="amount" type="xsd:double" />
    </message>
  - <message name="VerifyChargeResponse">
      <part name="return" type="xsd:int" />
    </message>
  - <message name="VerifyName">
      <part name="credit_card_number" type="xsd:string" />
      <part name="card_holder_name" type="xsd:string" />
    </message>
  - <message name="VerifyNameResponse">
      <part name="return" type="xsd:int" />
    </message>
  - <portType name="CreditCardServiceCreditCardPort">
    - <operation name="VerifyCharge">
        <input message="typens:VerifyCharge" />
        <output message="typens:VerifyChargeResponse" />
      </operation>
    - <operation name="VerifyName">
        <input message="typens:VerifyName" />
        <output message="typens:VerifyNameResponse" />
      </operation>
    </portType>
  - <binding name="CreditCardServiceCreditCardBinding" type="typens:CreditCardServiceCreditCardPort">
      <soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="rpc" />
    - <operation name="VerifyCharge">
        <soap:operation soapAction="/credit_card/api/VerifyCharge" />
      - <input>
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:ActionWebService" use="encoded" />
        </input>
      - <output>
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:ActionWebService" use="encoded" />
        </output>
      </operation>
    - <operation name="VerifyName">
        <soap:operation soapAction="/credit_card/api/VerifyName" />
      - <input>
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:ActionWebService" use="encoded" />
        </input>
      - <output>
          <soap:body encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" namespace="urn:ActionWebService" use="encoded" />
        </output>
      </operation>
    </binding>
  - <service name="CreditCardServiceService">
    - <port name="CreditCardServiceCreditCardPort" binding="typens:CreditCardServiceCreditCardBinding">
        <soap:address location="http://localhost:3001/credit_card/api" />
      </port>
    </service>
  </definitions>
```

Figure 5. Credit Card Service WSDL

One of the great things about using Rails to develop Web Services is that it generates a view that can be used to test the service's API. Figure 6 depicts test request and response SOAP messages that are being passed between the client (Store Service) and the Credit Card service.



Request XML:

```xml
<?xml version="1.0" encoding="utf-8" ?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Body>
    <n1:VerifyName xmlns:n1="urn:ActionWebService"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <credit_card_number xsi:type="xsd:string">1111222233334444</credit_card_number>
      <card_holder_name xsi:type="xsd:string">Tracy Tran</card_holder_name>
    </n1:VerifyName>
  </env:Body>
</env:Envelope>
```

Response XML:

```xml
<?xml version="1.0" encoding="UTF-8" ?>
<env:Envelope xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:env="http://schemas.xmlsoap.org/soap/envelope/"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <env:Body>
    <n1:VerifyNameResponse xmlns:n1="urn:ActionWebService"
        env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
      <return xsi:type="xsd:long">1111222233334444</return>
    </n1:VerifyNameResponse>
  </env:Body>
</env:Envelope>
```

Figure 6. Credit Card Service SOAP Messages

2.4 Mail Services

Any standard SMTP server can be the Mail Service used by the Store service via the Rails Action Mailer library to facilitate the sending of emails across the network. Since SMTP is a de facto standard for email, its service is well defined and readily available from any network connected client. The email architecture and infrastructure is vast, wide and provides an effective and efficient electronic communication mechanism from any entity to another, whether they are people or systems such as the online souvenir store.

2.5 Database Architecture

A MySQL database was used to support the Store database service and the Credit Card Validation service. MySQL is a free SQL Database Management System (DBMS), which has the available APIs that allow applications written in various languages to access the database from many platforms. Figure 7 depicts the Store service's database schema, and Figure 8 depicts the Credit Card service's database schema.

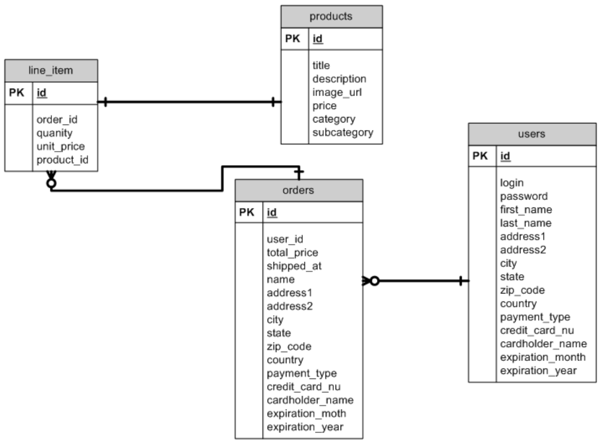


Figure 7. Store Service Database Schema

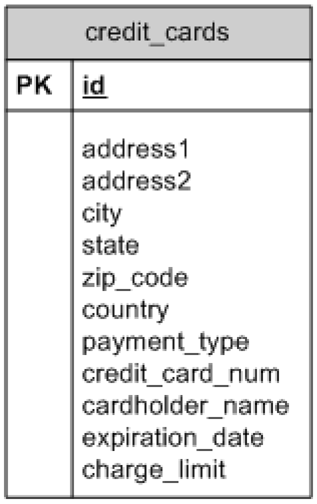


Figure 8. Credit Card Service Database Schema

3 Related Project Works

Numerous online stores have emerged through out the recent years. One very well known store is Amazon.com. Unlike our Souvenir Store application, Amazon's store catalog contains a large number of categories from books to music to house hold items. For such a large application, Amazon for sure implements SOA and uses various services to support their business. Whether using Web Services or other types of services, Amazon uses different services for handling credit card validation, user authentication, store inventory, ordering and shipping.

Another related work is Rich Solutions' Credit Card Service. This is a credit card Web Service that validates and verifies credit card information and transaction by providing a service gateway to all the major credit card vendors. Registered clients can use the service's WSDL to make credit card requests and this service claims to provide real-time authorization over the Internet. Online businesses can subscribe and connect to this service for processing credit card transactions instead of rolling their own implementation to connect to each individual credit card vendor's services.

4 Summary

Our goal was to build a specialized Souvenir Store web application using a Service-Oriented Architecture. We achieved this by using the Ruby on Rails framework for development. Rails provides a mechanism for interfacing with our MySQL databases while also providing a mechanism for creating a Web Service (Credit Card service), and an Email Service client. We chose to implement the Credit Card service as a Web Service as there are no standards based mechanism or WSDL for credit card validation. Our "store" uses the credit service to validate credit card information using the Action Web Service and Rails framework stack to encapsulate the complex details of working with web services. Communication between the "store" and the Credit Card service is achieved via the SOAP stand which is XML based. The communication between the "store" and the Mail service is achieved via SMTP to send email to the end users of the store service.

For future work, we plan to use or define a standard set of security services for user authentication, registration, and single sign-on (SSO) from multiple participating web sites or services. This service will handle all transaction relating to the users. The "store" will use this service for any user related needs such as user authentication and single sign-on.

5 References

Thomas, Dave, Hansson, David, Agile Web Development with Rails, Pragmatic Bookshelf, 2005.

http://www.service-architecture.com/

http://rubyonrails.org/