12-19-2008

Securely Consume Web Services Using PHP

Sonny Tran-Hai Vo
University of New Orleans

Securely Consume Web Services Using PHP

A Thesis

Submitted to the Graduate Faculty of the
University of New Orleans
in partial fulfillment of the
requirements for the degree of

Master of Science
in
Computer Science

by

Sonny Tran-Hai Vo

B.S. in Computer Science University of New Orleans, 2000

December, 2008

Acknowledgments

I would like to express my deeply gratitude toward everyone who has helped me during my graduate study at the University of New Orleans. Especially, I am truly grateful for my advisor, Dr. Shengru Tu, for his constant guidance and continuous supports that go far above and beyond the call of duty.

I would also like to thank Dr. Vassil Roussev and Dr. Adlai DePano for their directions throughout the years.

Finally, I wish to thank my parents for sacrificing so much for me to have a better life. Most importantly, I wish to thank my wife and my children for their patience, their understandings and for always standing by me.

Table of Contents

List of Figures

Abstract

The PHP: Hypertext Preprocessor language (PHP) has evolved to a sophisticated mainstream programming language for rapid development of significant Web applications at major sites including Facebook.com, Wikipedia.org and Yahoo.com. Leading software vendors such as Oracle and IBM are rushing in providing tools that bridge their products to PHP. However, we have observed a gap in facilitating PHP to utilize Web services efficiently.

This thesis reports our efforts in design and implementation of PHP applications that consume Web services. In doing so, I have proposed a framework facilitating PHP programs to utilize Web services with high performance capability. In addition, a number of Web service standards including WS-Addressing and those in WS-Security are integrated into my PHP implementation. Examples of using various Amazon Web Services are provided with details.

Keywords: PHP, SOA, SOAP, Web Services, XML, XPath, WSDL, UDDI, WS-Addressing, WS-Security, Amazon Web Services

Chapter 1 Introduction

With the popularity of Service-Oriented Architecture (SOA), the internet has opened many business opportunities. Companies no longer require large investment in infrastructures since many services could easily delegate to the Service providers. For example, instead of acquiring equipment and man-power to run and maintain one's own servers, one could easily purchase the space and computing power from Web service providers at minimal fee comparing to the cost of new equipments and maintaining them. There are a vast numbers of Web services available from weather forecasting, stock quoting to selling merchandises. For instance, Amazon's Simple Storage Service (S3) and Elastic Computer Cloud (EC2) directly provide users with storage space and computational power. In addition, Amazon.com has a comprehensive set of Web services that allows users to list and sell items on their websites to get commissions.

On the other hand, the Internet has created a large number of domain investors, whose business is buying, selling, monetizing and developing internet domain names. Generally, these investors acquire and hold large number of domain name portfolios, which are often not being developed. In the past several years, many of these investors have turned to domain parking companies to monetize these undeveloped domains since it is much faster and easier than to develop large number of domains. According to the Web Server report from Netcraft.com, a company whose business is to "research data and analysis on many aspects of the Internet" [Netcraft], there are more than 182 millions websites worldwide as of October 2008 (Figure 1). Many of these websites might not be fully active since it is a common practice of domains registrars and domain hosting companies to create a default template as a place holder for the

undeveloped domains according to Netcraft.com. There is a strong desire of a system that could flexibly build contents for large number of web sites with very minimal human intervention.

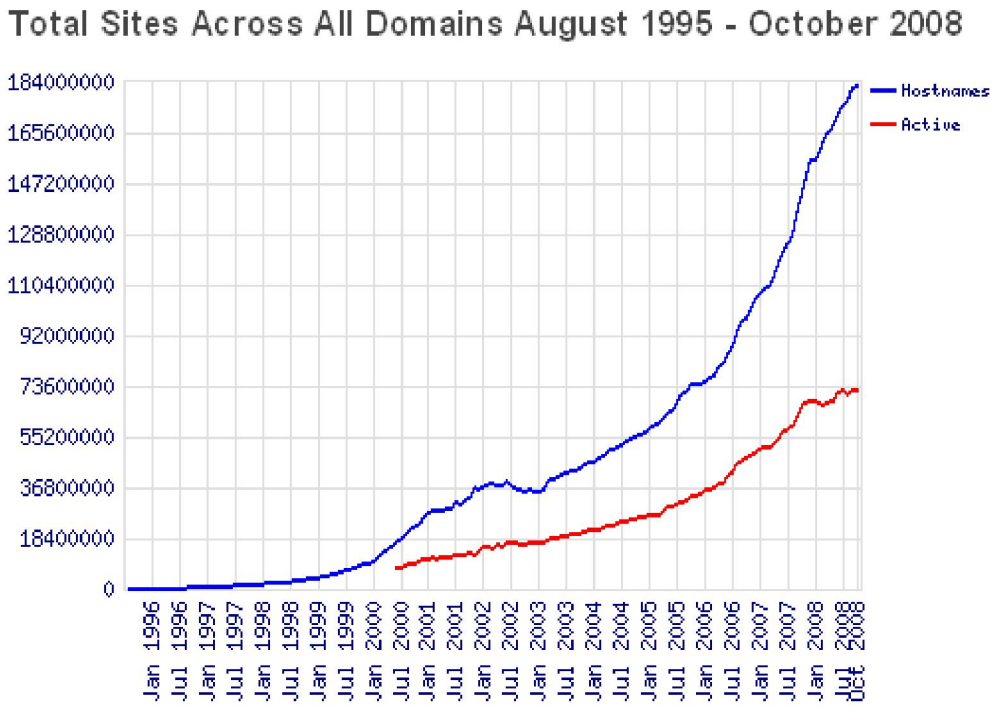

Figure 1: Netcraft.com's October 2008 Web Server Survey

For light-weight Web site development, the PHP: Hypertext Preprocessor (PHP) has become one of the most popular languages [PHP]. It is one of the components of LAMP (a term referring to a set of set of software components used to run web servers: Linux as the operating system, Apache as the web server, MySQL as the backend database, and PHP as the programming language). Evidently, it is used in many web sites including Wikipedia.org, WorldPress.org, Friendster.com, Facebook.com, and Yahoo! Inc. As of Oct of 2008, PHP is the in the top 10 most popular programming languages (behind Java, C, C++, and Visual Basic) [TIOBE].

With the explosion of Web services in recent years, many PHP applications are now consuming Web services published by external or internal providers. Unfortunately, PHP's built-in SOAP extension (PHP SOAP), which is written in C for speed, has somewhat fallen short in many aspects, especially when it comes to consuming complex types and supporting various Web service standards (often referred as WS-* standards). To aid PHP in its newly found role of the SOA world, there have been several frameworks and libraries to help PHP consuming Web services such as PEAR:Soap and NuSOAP, which were popular before PHP's built-in SOAP capabilities came to life. The most notable effort is WSO2's WSF/PHP framework, which can be used to provide and consume Web service. It can also support WS-Addressing and WS-Security specifications, requires installation and server configuration.

1.1 Main Goals

If PHP's SOAP extension could efficiently and easily consume Web services, it would be a much better solution than using external extensions or libraries. Consequently, the main goal of my thesis project is to investigate and implement a method for PHP application to efficiently and securely consume Web services that have complex types and require WS-Addressing and WS-Security specification using PHP's native SOAP extension. An additional constraint of this method is that the Web applications are to be deployed at an external Web hosting site in which available administration capabilities are minimal. For example, the Web servers are not accessible for retooling and re-compilation.

1.2 Objectives

To achieve the stated goals, I will build a set of Web application using PHP's built-in SOAP extension to consume external Web services. There are many Web services out there. I have

chosen a mainstream service provider, Amazon's Web services. They provides a wide range of Web services, some of which only require simple SOAP messages while others require the SOAP messages to be signed using the Public Key Infrastructure (PKI) framework.

The objective of this thesis is to develop a fully functional Web application that will be used to display different contents on different web sites using Amazon's Web service. Other functionalities will be also implemented so the application could easily be expanded to consume other Web services provided by Amazon, such as Amazon Simple Storage Service (S3), Amazon Simple DB, and Amazon Elastic Compute Cloud (EC2).

In addition, the system should be easily extended by adding a SOAP server to serve other websites that are not resided in the same web server. Also, the domain name could be translated into keywords (each keyword is associated with a list of settings) automatically to minimize human interventions.

Chapter 2 Background and Related Works

Web Services is an important new technology, which has enjoyed widespread supports from leading industry players such as IBM, Microsoft, HP, Oracle, Novell and Sun.

According to IBM's definition of Web Services, "Web Services are self-contained, modular applications that can be described, published, located, and invoked over a network". In another word, Web Services are usually services that are offered via the Web. However, the word "Web" in "Web Services" is actually a little bit misleading since it might hint that Web Service sends XML-based messages over HTTP. On the contrary, Web Service does not require HTTP as the transport layer; it could easily be migrated to other transport protocols.

In a web service transaction, a Web application sends a request message using SOAP to a service at some URI over communication protocols such as HTTP, SMTP, FTP, etc. The service provider then processes the request and returns a response. The URI of the service could be discovered using the UDDI (Fig. 2); however, in real life, URI is often known (Fig. 3). Even if some services are free, the service consumers are typically required to acquire access permissions from the service providers. On the other side of the equation, a web service is a published software application that communicates with other applications using XML-based messages using the above communication protocols. It is available to be integrated into other applications over the Web.

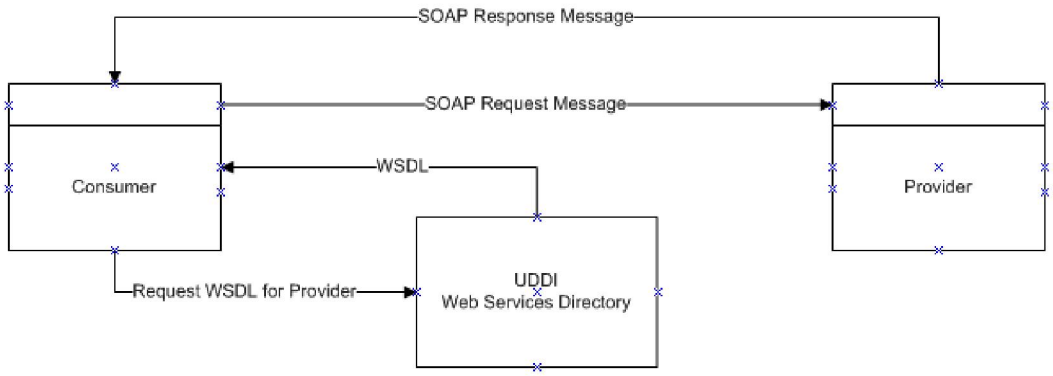

Figure 2: Consumer request WSDL from UDDI and send request to Provider

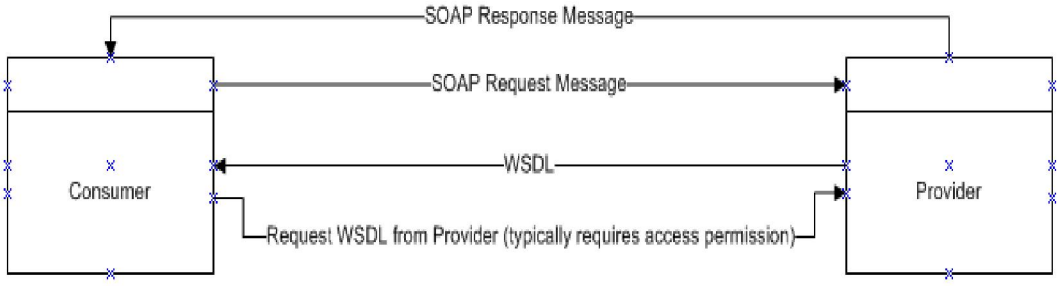

Figure 3: In real life, URI is usually known

Since Web Services are loosely-coupled and not tied to any particular implementation, they allow application programmers to share codes that are well-tested and could be used across platforms and programming languages. For instance, SOAP clients written in PHP, Java or Python should still be able to consume the same Web service without any problem. Ultimately, new features could be seamlessly integrated into websites with minimal costs.

As the popularity of Web Services rises, many associated standards have emerged so businesses could easily and securely publish and consume Web Services, some of which are XML, XPath, SOAP, WSDL, UDDI. We will exam each of those associated standards and their

roles in Web Services. We will further discuss the role of Web Services in Service-Oriented Architecture.

2.1 XML

eXtensible Markup Language (XML) is a specification, defined by the W3C [W3C], which provides syntax to structurally represent data using markup. XML, however, usually refers to the entire family of related-technologies. The W3C recommends not abbreviating XML elements and attribute names. For example, the following listing shows how a student record might be marked up using XML (Fig. 4):

```
<StudentRecord>
        <StudentId>123456</StudentId>
        <StudentName>John Doe</StudentName>
        <Address>123 College Dr</Address>
        <City>New Orleans</City>
        <State>LA</State>
        <PostalCode>70130</PostalCode>
</StudentRecord>
```

Figure 4: XML sample

A well-formed XML must also adhere to certain requirements such as the start tag and end tag must be the same, no overlapping tags, and element and attribute names must be surrounded by quotes.

In another word, XML documents have a structure format which allows data to send across network in an interoperable manner. Naturally, XML has widespread support to become the messaging standard.

2.2 XPath

XML Path Language (XPath) is another XML-related specification used to address certain part of a XML document [XPath]. For example, "/StudentRecord/PostalCode/" will return

"70130" using XPath. Since XPath easily pinpoint to attributes and elements of a XML document, it is often used in Web Service security.

2.3 SOAP

Simple Object Access Protocol (SOAP) defines a structure of XML-based messages that are being sent between Web services applications [SOAP]. It essentially provides a functionality to enclose messages insides envelopes. The figure below illustrates a SOAP envelope (Fig. 5):

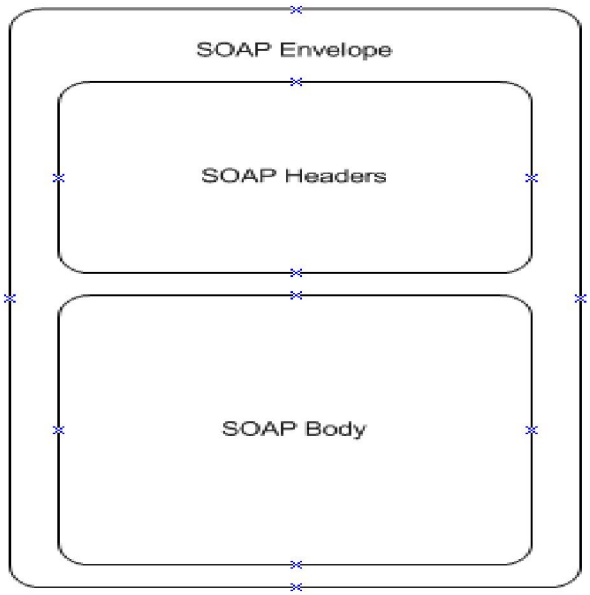

Figure 5: SOAP Envelop

Even if SOAP is often used to send messages from one application to another, SOAP can be used for both messaging (Documented-Based SOAP) and Remote Procedure Calls (RPC SOAP).

2.4 WSDL

Web Service Definition Language (WSDL) is an XML format to describe the service provider's operations and the interactions between the service consumer and the service

provider. It acts as the contract between the service provider and the service consumer [WSDL].

A WSDL has well defined basic structure, which includes the following main elements (Fig. 6):

```
<definitions>
    <types>
    ……
    </types>
    <message>
    ……
    </message>
    <portType>
    ……
    </portType>
    <binding>
    ……
    </binding>
    <service>
    ……
    </service>
</definitions>
```

Figure 6: WSDL Structure

Any complex and array data types should be declared in the types element. For example, the following figure defines a DescribeImageType (Fig. 7):

```
<xs:complexType name="DescribeImagesType">
    <xs:sequence>
        <xs:element name="executableBySet"
        type="tns:DescribeImagesExecutableBySetType" minOccurs="0"/>
        <xs:element name="imagesSet" type="tns:DescribeImagesInfoType"/>
        <xs:element name="ownersSet" type="tns:DescribeImagesOwnersType"
        minOccurs="0"/>
    </xs:sequence>
</xs:complexType>
```

Figure 7: Example of a Complex Type

The message element is used to assign data types as parameters of the Web service's operations in the portType block. The following listing illustrates the message (Fig. 8):

```
<message name="DescribeImagesRequestMsg">
        <part name="DescribeImagesRequestMsgReq" element="tns:DescribeImages" />
</message>
```

Figure 8: WSDL's message element

The portType element lists the Web service's operations. For example, the following fragment illustrates a "Request-Response" (or In/Out) operation (Fig. 9).

```
<operation name="DescribeImages">
        <input message="tns:DescribeImagesRequestMsg" />
        <output message="tns:DescribeImagesResponseMsg" />
</operation>
```

Figure 9: A Web Service operation

We clearly see that the operation "DescribeImages" will take a "DescribeImagesRequestMsg" as an input and will response to the client with a "DescribeImagesResponseMsg".

The binding element is where the declared portType are tied into actual SOAP actions. For example, the following listing (Fig. 10) shows how the operation "DescribeImages" listed above is tied into the SOAP action:

```
<binding name="AmazonEC2Binding" type="tns:AmazonEC2PortType">
        <soap:binding style="document"
        transport="http://schemas.xmlsoap.org/soap/http" />
                <operation name="DescribeImages">
                        <soap:operation soapAction="DescribeImages" />
                         <input>
                                <soap:body use="literal" />
                        </input>
                        <output>
                                <soap:body use="literal" />
                        </output>
                  </operation>
        </binding>
```

Figure 10: WSDL's binding element

The style attribute specifies the Binding type, which is either RPC or Document; whereas the transport attribute specifies the transport protocol.

The last element is the service element, which specifies the actual location for each port. For example, the following service element indicates that the listed operations for "AmazonEC2Port" are located at "https://ec2.amazonaws.com/" (Fig. 11).

```
        <service name="AmazonEC2">
                <port name="AmazonEC2Port" binding="tns:AmazonEC2Binding">
                        <soap:address location="https://ec2.amazonaws.com/" />
                </port>
        </service>
```

Figure 11: WSDL's service element

2.5 UDDI

Universal Description, Discovery and Integration (UDDI) is a XML-based registry for businesses worldwide who have web services to register and also allow searching for specific

web services[UDDI]. In another word, like a Yellow Pages phone book, the goal of UDDI is to assist business to list and find suitable services.

2.6 SOA

Service-Oriented Architecture (SOA) provides methods to facilitate deployment of interoperable services; to some extends, SOAP decouple the services with the underlying programming languages, platforms, etc. In a SOA, the Web service is **published** on a network where it can be discovered (ability to **find** the service) and bound (**binding**: the ability to connect to the service). In another word, these three activities are corresponding to Web Service Provider, Web Service Requestor and Web Service Broker in Web Service Architecture [O'Neill].

2.7 WS-Addressing

According to the W3C, "WS-Addressing specification defines XML elements to identify Web service end points and to secure end-to-end end point identification in message" [WS-Addressing]. Basically, to comply with WS-Addressing specification, a SOAP message will have the following headers (Fig. 12):

```
<wsa:MessageID>...</wsa:MessageID>
<wsa:ReplyTo>
        <wsa:Address>...</wsa:Address>
</wsa:ReplyTo>
<wsa:FaultTo>
        <wsa:Address>...</wsa:Address>
</wsa:FaultTo>
<wsa:To>...</wsa:To>
<wsa:Action>.../wsa:Action>
```

Figure 12: WS-Addressing Headers

The following figure (Fig. 13) depicts how WS-Addressing might be used:

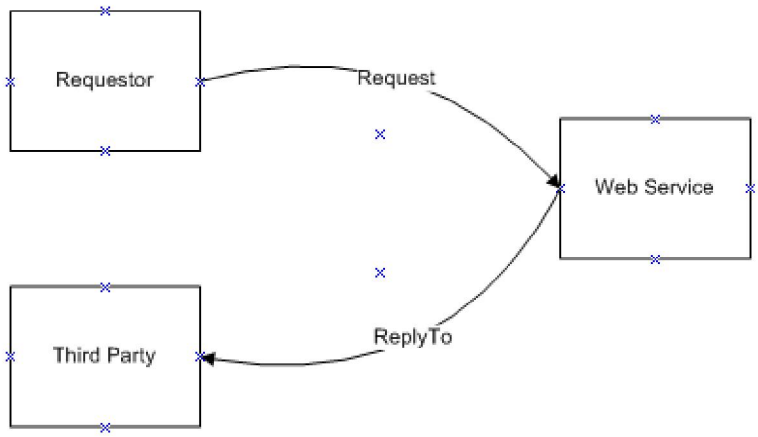

Figure 13: WS-Addressing

Ultimately, we should take precaution to prevent End Point References from being tempered, which lead us to another specification, which is WS-Security.

2.7 WS-Security

WS-Security specification provides encryption, integrity and authentication support for securing SOAP messages [WS-Security]. In another word, WS-Security aims to deliver the following:

- A message from the Web Service consumer to the Web Service provider should not be viewed by third party while it travels in the network.
- The provider should be able to ensure that the message from the consumer has not been tempered with.
- The provider should be able to determine from whom the message was from and be able to verify if the sender is who the sender claims to be.

To achieve the above goals, WS-Security defines three token types:

- UsernameToken Profile
- X.509 Certificate Token Profile
- SAML (Security Assertion Markup Language Token Profile) [SAML]

The listing below shows SOAP headers for WS-Security UsernameToken profile (Fig. 14):

```
<S11:Envelope xmlns:S11="..." xmlns:wsse="..." xmlns:wsu="..." xmlns:ds="...">
  <S11:Header>
    <wsse:Security xmlns:wsse="...">
      <wsse:UsernameToken wsu:Id="Example">
        <wsse:Username> ... </wsse:Username>
        <wsse:Password Type="..."> ... </wsse:Password>
        <wsse:Nonce EncodingType="..."> ... </wsse:Nonce>
        <wsu:Created> ... </wsu:Created>
      </wsse:UsernameToken>
    </wsse:Security>
  </S11:Header>
```

Figure 14: WS-Security Username Token Profile Headers

Web Service has become a very important distributed computing technology, and it will become even more important since it has widely support from many industry leading companies.

Chapter 3 Design for PHP Web Service Consumer Components

The Web application will be shared among a large number of web sites to automatically create their contents with very minimal human intervention. Each web site will be assigned with a keyword; the contents will be created based on that given keyword (Fig. 15). This application will be implemented using PHP as the programming language and should be able to consume Amazon Web Associates service and facilitate the consumption of other Amazon Web services as well.

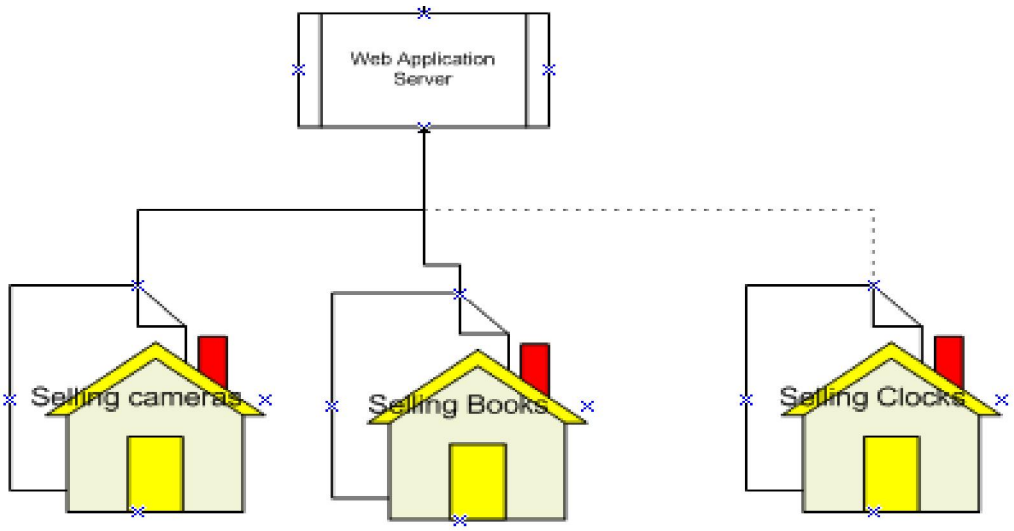

Figure 15: Web application overview

3.1 System Architecture

When users access one of the websites, it will send a request to the Web application server with parameters such as the keyword, the search index, and the Browser Node Id, the Web application server will then make a SOAP request to Amazon's Web service with the given information. Once the data is returned from Amazon, it will be formatted and displayed to the end users (Fig. 16).

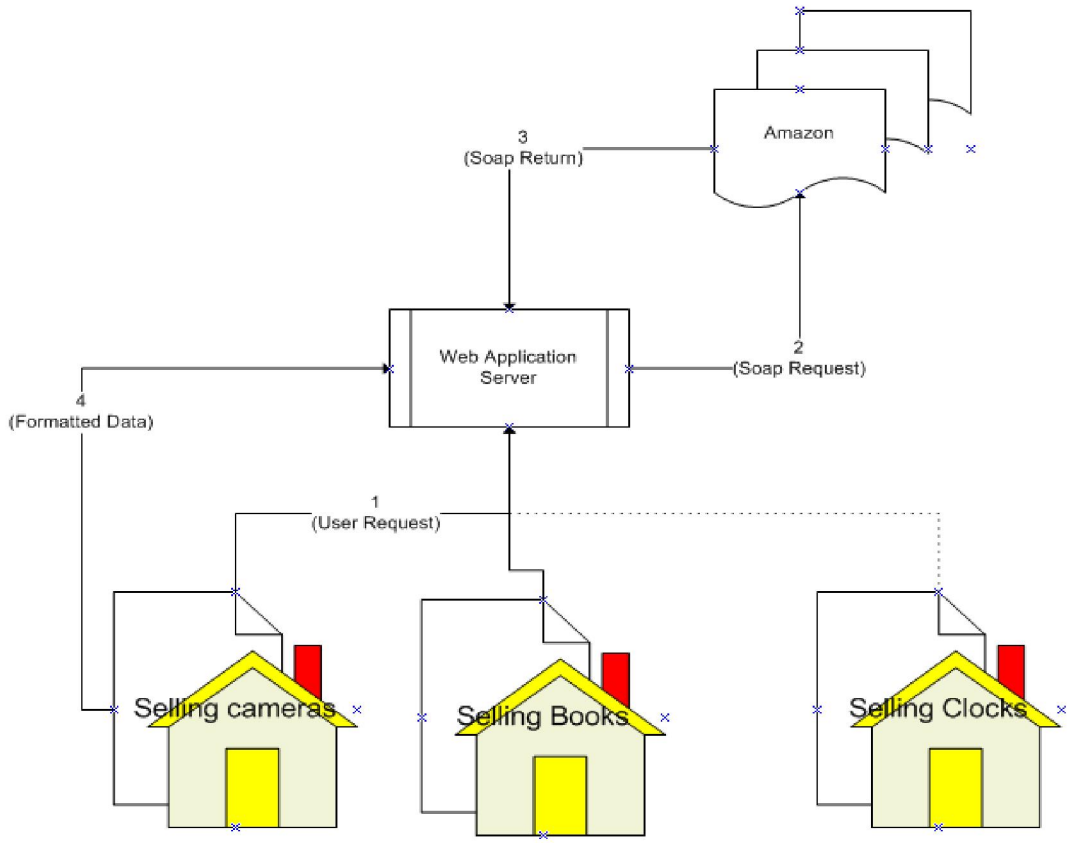

Figure 16: System overview

There are numerous choices to consume a Web service using pure PHP. Four of them are described in the following because I evaluated the pro and con of using each of them.

1. Pear:SOAP, a package of PEAR (PHP Extension and Application Repository), is a framework to build many applications in PHP, normally requires Pear download manager to be installed [Pear:SOAP].
2. NuSOAP is a library used to covert PHP data types to proper XML Schema types, had enjoyed huge attention in the previous PHP version (version 4) since there was not much option when it came to consume web service. However, even its latest version (0.7.3) supports neither SOAP 1.2 nor any WS-* specification [NuSOAP].

3. WSO2's WSF/PHP is an external extension of PHP that could be used to provide and consume Web services [WSF/PHP]. This extension could be used in providing and consuming Web Services in PHP. WSO2 WSF/PHP is a complete solution for building and deploying Web services and is the only PHP extension with most extensive implementations for the widest range of WS-* specification. Its key features include, secure services and clients with WS-Security support, binary attachments with MTOM, automatic WSDL generation (code first model), WSDL mode for both services and clients (contract first model) and interoperability with .NET and J2EE (WSO2.org). However, this external extension will not meet the business requirement for this project because it is not feasible to compile or reconfigure a Web server in an external shared hosting environment.
4. PHP's native SOAP extension (PHP SOAP), which is only available if the SOAP option is turned on if the PHP installation is configured with " **--enable-soap**" (for example, **./configure --with-xml --with-mysqli --enable-soap**), is written in C and could be used to write SOAP server and client. Fortunately, most shared hosting environments that support PHP5 usually have this SOAP feature enabled. However, this built-in extension is short in many aspects.

   The SoapClient class, which enables PHP to consume Web services, is not well-documented. For example, as of today, the function __setSoapHeaders() is not even listed (Fig. 17).

Figure 17: PHP's SoapClient

Using generic classes provided by the PHP SOAP extension, a Soap Header could be created with complex SOAP types to accommodate WS-Security's UsernameToken. The code segments shown below (Fig. 18) illustrate how to create a SOAP header in PHP along with the resulting header (Fig. 19):

```php
class WSSecurity {
    private $wssens = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd";
    private $wsuns = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd";
    private $wsunametoken = "http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-profile-1.0";
    private $UsernameToken = null;
    private $mustUnderstand = true;

    function __construct($mustUnderstand=null) {
        if ( !empty($mustUnderstand) )
            $this -> mustUnderstand = $mustUnderstand;
    }


    public function addUsernameToken($username, $password = null, $isDigest = false) {
        if ( is_null($password) && $isDigest ) {
            throw new Exception("Password is required if PasswordDigest is used.");
        }
        $params = array();
        $params[] = $this -> _wrapSoapVar($username, XSD_STRING, "Username");
        $nonce = md5(uniqid(rand(), true));
        if ( $password ) {
            if ( $isDigest ) {
                $passwordType = $this -> wsunametoken . "#PasswordDigest";
                $password = base64_encode(sha1($nonce . $dateCreated . $password));
            } else {
                $passwordType = $this -> wsunametoken . "#PasswordText";
            }
            $params[] = $this -> _wrapSoapVar($password, XSD_STRING, "Password", $passwordType);
        }
        $params[] = $this -> _wrapSoapVar(base64_encode($nonce), XSD_BASE64BINARY, "Nonce");
        $this -> UsernameToken = $this -> _wrapSoapVar($params, SOAP_ENC_OBJECT);
    }

    private function _wrapSoapVar($param, $param_type, $node_name, $type_name=null) {
        if ( !empty($param) ) {
            return (new SoapVar($param, $param_type, $type_name, null, $node_name));
        }
    }

    public function getWSSHeader() {
        try {

            return ( new SoapHeader($this -> wssens, "Security", new SoapVar(array('UsernameToken' => $this -> UsernameToken), SOAP_ENC_OBJECT),
                    $this -> mustUnderstand));

        } catch (Exception $ex) {
            throw $ex;
        }
    }

```

Figure 18: Code fragment to create Soap headers

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://webservices.amazon.com/AWSECommerceService/2008-10-06"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:ns2="http://docs.oasis-
open.org/wss/2004/01/oasis-200401-wss-wssecurity-secext-1.0.xsd">
<SOAP-ENV:Header><ns2:Security SOAP-ENV:mustUnderstand="1">
<UsernameToken><Username>Blahal</Username>
<Password xsi:type="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-username-token-
profile-1.0#PasswordDigest">
Y2ExMmEwOTFhN2IxNGFjNTcyNDNjMGRkMTU3NWM4MjYzZmNhYzM0MQ==</Password>
<Nonce>TVdRMk9XUTFOamhsTW1ObE1EWTFabVUwTURnek1EQmlPREJsT1RkbE1EST0=</Nonce>
</UsernameToken></ns2:Security></SOAP-ENV:Header>
```

Figure 19: Soap headers created with generic PHP classes

Even though PHP's SoapClient class could add the required Header information to accommodate WS-Security's UsernameToken, there is no easy way to add attributes to the SOAP's body, which would be required if the SOAP message needs to be signed. Thus, to accomplish our stated goals, we will need to find a better solution. Fortunately, the SoapClient class could be extended and has its methods overridden to conform to one's specifications. For example, we override the __doRequest() method in order to support WS-Addressing and WS-Security features.

3.2 Technology Overview

As mentioned in the previous section, PHP SOAP could support complex SOAP types; however, in order to support wider range of WS-Security standards, there will be more than just wrap complex types inside the SoapVar class. We will override the __doRequest() method in SoapClient and modify the request message before sending it off to the Web service provider. We will use the PHP classes written by Robert Richards to support WS-Addressing and WS-Security [Richards].

Moreover, the SoapClient class has additional useful features in consuming Web service in WSDL mode with the options array of flags:

1. cache_wsdl option: allow for the Soap client to disable caching the WSDL file (WSDL_CACHE_NONE), to cache the WSDL file in memory (WSDL_CACHE_MEMORY), to cache the WSDL file on disk (WSDL_CACHE_DISK) or to do both memory and disk caching (WSDL_CACHE_BOTH).
2. classmap option: allow mapping Schema Elements to PHP Classes as an associative array (Schema Elements => name of PHP Classes). Mapping Schema Elements to PHP classes will ease the implementation process since additional functionalities could be added directly to the classes.

3.3 System Design

Amazon's Associates Web service is very flexible; it allows applications to retrieve a list of sale items for displaying Web pages. When a website visitor clicks on a listed item, there are two options:

a. The visitor will be redirected to Amazon website with the associate's referral tag. When the visitor purchases that item, the associate will receive commission for that purchase.
b. Another option is to use the Remote Shopping Cart, which allow the visitor to stay on the website instead of being redirected back to Amazon. The Remote Shopping Carts are hosted at Amazon servers.

Whichever method is chosen, the associate will get paid the same using the following payout structure according to Amazon's "classic fee structure" [AmazonAssociates]. For this

thesis project, the first option is selected, which means the links on the websites will redirect visitors back to Amazon website. Consequently, the overall system will look like the following figure (Fig.20):

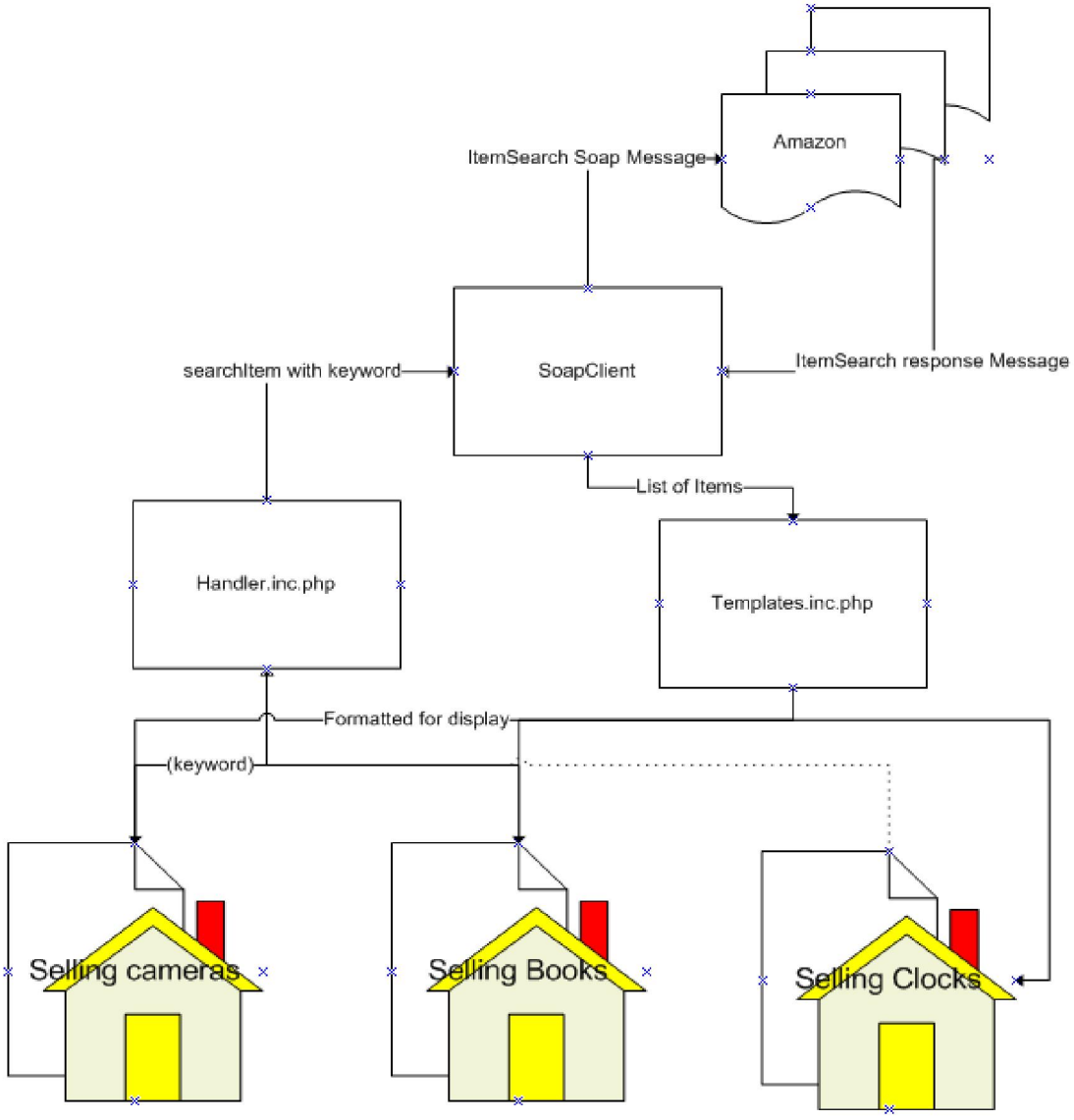

Figure 20: System Details

The Sequence Diagram for a visitor's request is included in Fig. 21:

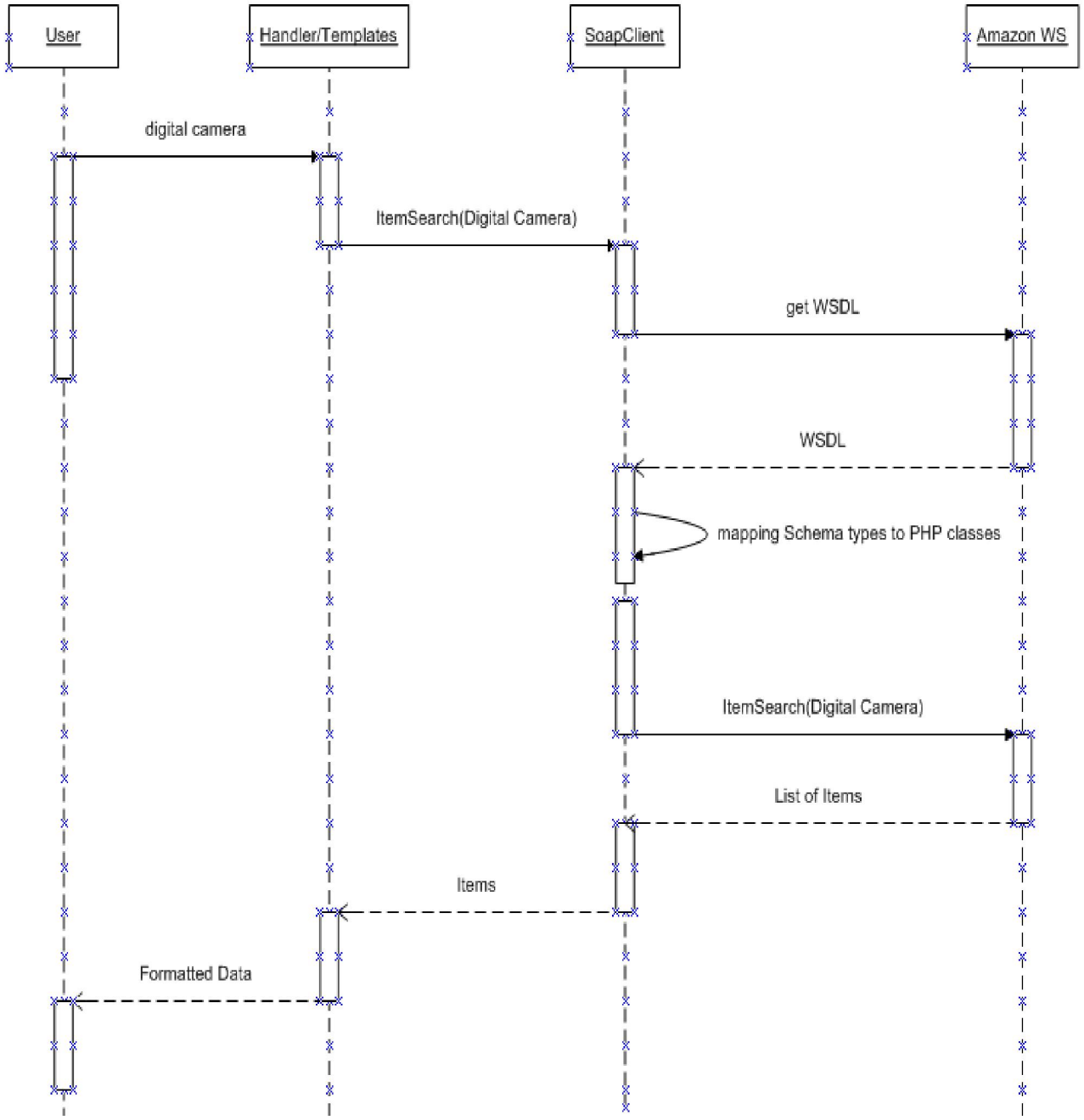

Figure 21: Sequence Diagram

In addition, Amazon provides many more other services such as Amazon Simple Storage Service (S3) [S3], Amazon Elastic Compute Cloud (EC2) [EC2] and Amazon Simple DB [SimpleDB], which could greatly enhance this application; this application will need to be able to consume those other services as well. Consequently, the Class Diagram (Fig. 22) will include classes required to consume other Services. In addition, we will illustrate how PHP Soap client consuming complex types.

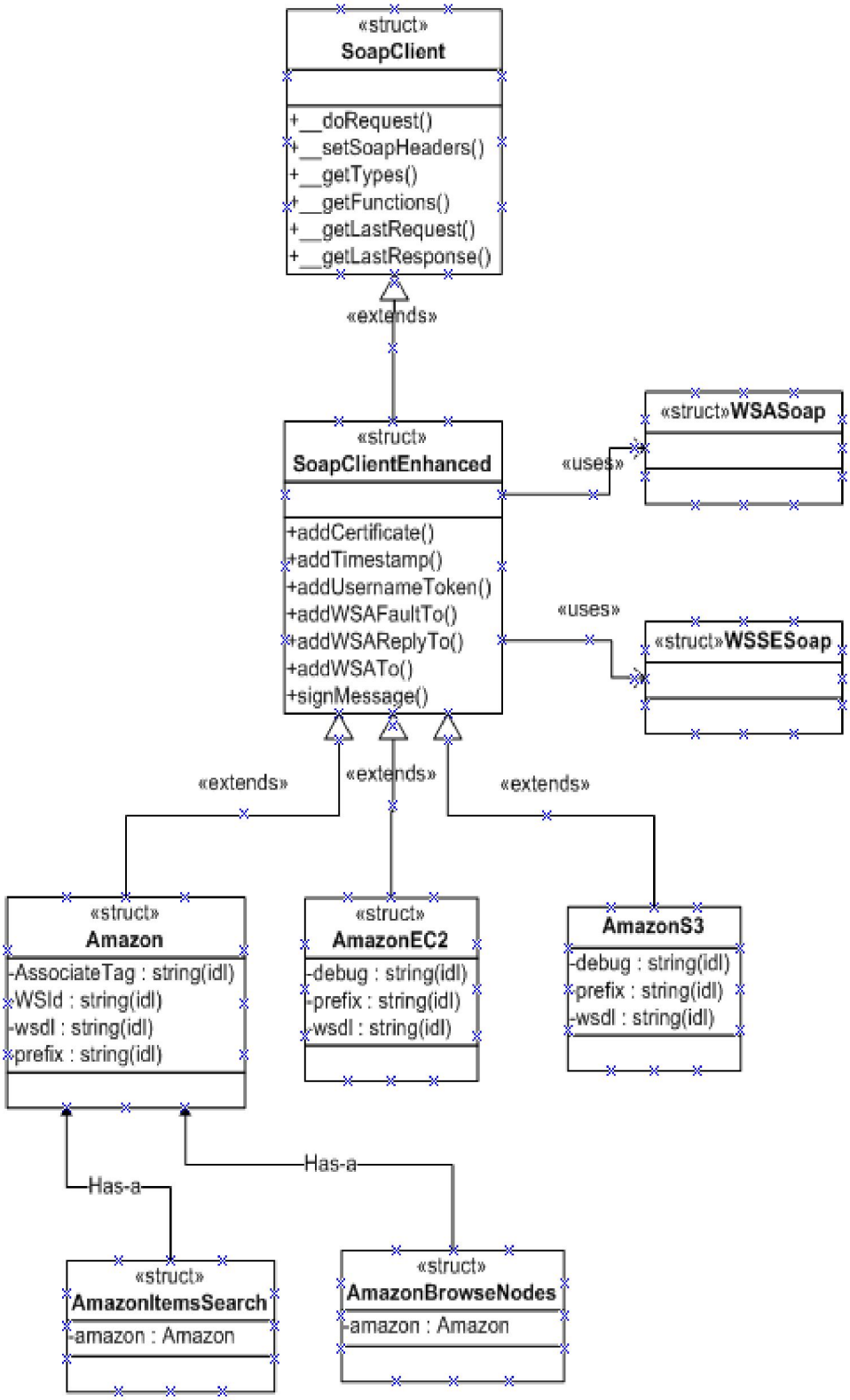

Figure 22: Class Diagram

There are five more components that are not listed in the Class Diagram:

a. index.php: takes the request and pass the request to the Controller (handler.inc.php file).

b. handler.inc.php: the controller of the application. It processes requests from the index.php file, invoke appropriate classes to retrieve data from Amazon Web services, and pass the data to the View (Templates.inc.php) for displaying.

c. templates.inc.php: embed the data into HTML for displaying

d. Amazon_Item.class.php: a custom PHP class will be mapped to Amazon Item object with additional methods to display data.Amazon_BrowseNode.class.php: a custom PHP class will be mapped to Amazon BrowseNode object with additional method to display data.

Chapter 4 System Implementation

In the implementation of this project, we emphasized on reusing existing libraries and frameworks. For example, instead of implement functionalities supporting WS-Addressing and WS-Security standards, we used a PHP library written by Robert Richards and is released with Open Source license [Richards] and codes example written by Adam Delves [Delves].

4.1 The Soap Client

In the heart of this application is the SoapClientEnhanced class, which extends PHP native SoapClient class. As mentioned before, the native SoapClient class has very limited functionalities; its methods are listed in Fig. 23. Another method that was not document is **__setSoapHeaders()**.

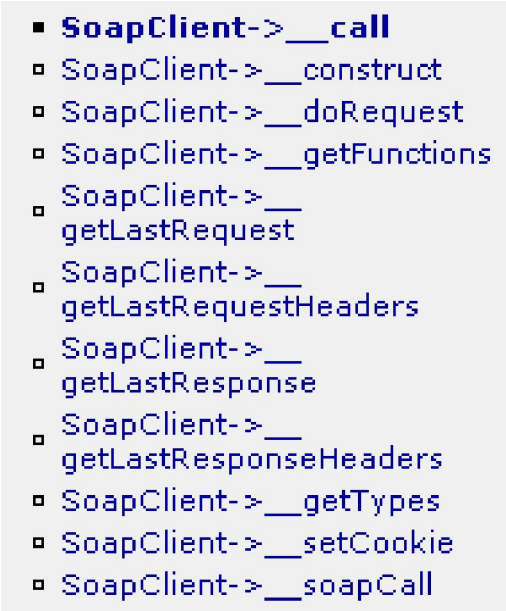

Figure 23: PHP native SoapClient class methods

We will now closely exam the SoapClient class and its methods to further understand its capabilities.

- The __call() method has been deprecated.

- The __construct($wsdl [, array $options]) method is the constructor. It takes in the URI or the WSDL file itself as the first parameter; however, it could be set to NULL in Non-WSDL mode. The second parameter, the optional $options array, is an associative array of options and is optional in the WSDL mode. It is required in Non-WSDL mode with the **location** and **uri** options, where **location** is the URL to request and **uri** is the target namespace of the SOAP service. Since our main focus is on the WSDL mode, we will only concentrate on options that pertain to WSDL mode.
    - The **soap_version** could be set to specify if to use SOAP 1.1 or 1.2.
    - The **login** and **password** options could be set for HTTP authentication.
    - The **proxy_host**, **proxy_port**, **proxy_login** and **proxy_password** could be used to make HTTP connection via a proxy server.
    - The **local_cert** and **passphrase** are used to support HTTPS client certificate authentication.
    - The **compression** option is used to use compression of SOAP requests and responses.
    - The **encoding** option defines internal character encoding.
    - The **classmap** option can be used to map WSDL types to PHP classes. This option must be an array with WSDL types as keys and names of PHP classes as values.
    - The **trace** option enables use the use of the methods __getLastRequest(), __getLastRequestHeaders(), __getLastResponse() __getLastResponseHeaders.

- The **exceptions** option defining whether or not exceptions of type SoapFault are thrown when SOAP error occurs.
- The **connection_timeout** option specifies a timeout in seconds for the connection to the SOAP service.
- The **typemap** option is an array of type mappings. Type mapping is an array with keys type_name, type_ns (namespace URI), from_xml (callback accepting one string parameter) and to_xml (callback accepting one object parameter).
- The **cache_wsdl** option is one of WSDL_CACHE_NONE (no WSDL cache), WSDL_CACHE_DISK (cache WSDL on Disk), WSDL_CACHE_MEMORY (cache WSDL in Memory) or WSDL_CACHE_BOTH (cache WSDL both in Memory and on Disk).
- The **user_agent** option specifies string to use in User-Agent header.
- The **stream_context** option is a resource for context.
- The **features** option is a bitmask of SOAP_SINGLE_ELEMENT_ARRAYS, SOAP_USE_XSI_ARRAY_TYPE, SOAP_WAIT_ONE_WAY_CALLS.

- The __doRequest (string $request, string $location, string $action, int $version [, int $one_way]), which could be overridden by subclass to add additional XML processing, returns a string. The $request is the XML SOAP request; the $location is the URL to request; the $action is the SOAP action; the $version is the SOAP version.

- The __getFunctions(), which works only in WSDL mode, returns an array of SOAP functions. For example, this method returns all functions of Amazon S3 as shown in the following listing:

```
Supported Functions:
Array
(
        [0] => CreateBucketResponse CreateBucket(CreateBucket $parameters)
        [1] => DeleteBucketResponse DeleteBucket(DeleteBucket $parameters)
        [2] => GetObjectAccessControlPolicyResponse
        GetObjectAccessControlPolicy(GetObjectAccessControlPolicy $parameters)
        [3] => GetBucketAccessControlPolicyResponse
        GetBucketAccessControlPolicy(GetBucketAccessControlPolicy $parameters)
        [4] => SetObjectAccessControlPolicyResponse
        SetObjectAccessControlPolicy(SetObjectAccessControlPolicy $parameters)
        [5] => SetBucketAccessControlPolicyResponse
        SetBucketAccessControlPolicy(SetBucketAccessControlPolicy $parameters)
        [6] => GetObjectResponse GetObject(GetObject $parameters)
        [7] => GetObjectExtendedResponse GetObjectExtended(GetObjectExtended
        $parameters)
        [8] => PutObjectResponse PutObject(PutObject $parameters)
        [9] => PutObjectInlineResponse PutObjectInline(PutObjectInline $parameters)
        [10] => DeleteObjectResponse DeleteObject(DeleteObject $parameters)
        [11] => ListBucketResponse ListBucket(ListBucket $parameters)
        [12] => ListAllMyBucketsResponse ListAllMyBuckets(ListAllMyBuckets $parameters)
        [13] => SetBucketLoggingStatusResponse
        SetBucketLoggingStatus(SetBucketLoggingStatus $parameters)
        [14] => GetBucketLoggingStatusResponse
        GetBucketLoggingStatus(GetBucketLoggingStatus $parameters)
        [15] => CopyObjectResponse CopyObject(CopyObject $parameters)
)
```

Figure 24: Sample of return of __getFunctions method

- The __getLastRequest() and __getLastRequestHeaders(), which only work if the trace option is on, will return the request and the headers of the last SOAP request respectively.
- The __getLastResponse() and __getLastResponseHeaders(), which also only work if the trace option is on, will return the response and the headers from the last SOAP response respectively.

  The __getTypes() method, which only works in WSDL mode, will return the list of SOAP types. For example, the following listing is a sample of types returned by Amazon EC2 (Fig. 25). Complex types are identified by the “struct” keyword.

```
[6] => struct DeleteKeyPairType {
        string keyName;
}
[7] => struct DeleteKeyPairResponseType {
        string requestId;
        boolean return;
}
[8] => struct DescribeKeyPairsType {
        DescribeKeyPairsInfoType keySet;
}
[9] => struct DescribeKeyPairsInfoType {
        DescribeKeyPairsItemType item;
}
[10] => struct DescribeKeyPairsItemType {
        string keyName;
}
[11] => struct DescribeKeyPairsResponseType {
        string requestId;
        DescribeKeyPairsResponseInfoType keySet;
}
[12] => struct DescribeKeyPairsResponseInfoType {
        DescribeKeyPairsResponseItemType item;
}
[13] => struct DescribeKeyPairsResponseItemType {
        string keyName;
        string keyFingerprint;
}
[14] => struct RunInstancesType {
        string imageId;
        int minCount;
        int maxCount;
        string keyName;
        GroupSetType groupSet;
        string additionalInfo;
        UserDataType userData;
        string addressingType;
        string instanceType;
        PlacementRequestType placement;
        string kernelId;
        string ramdiskId;
        BlockDeviceMappingType blockDeviceMapping;
}
[15] => struct GroupSetType {
        GroupItemType item;
}
```

Figure 25: example of __getTypes() function

- The __setCookie (string $cookieName [, string $cookieValue]), which will affect all subsequent call to the SoapClient's methods, will send a cookie along with the SOAP request.
- The __soapCall (string $function_name , array $arguments [, array $options [, mixed $input_headers [, array &$output_headers]]]) method is a low level API function that make a SOAP call. It will return simple type if the SOAP function return only one value; otherwise, an associative array will be returned.

Thus, we will extend the SoapClient class with a subclass named SoapClientEnhanced to add additional functionalities.

First, we override the parent class __doRequest() method to intercept the SOAP request to inject the necessary SOAP headers to support WS-Addressing and WS-Security standards. The listing below shows the codes fragment of how the __doRquest() method was overridden to support the above standards (Fig.26):

```
function __doRequest($request, $location, $action, $version) {
        try {
                if ( $this -> useWSA || $this -> useWSSE ) {
                        $dom = new DOMDocument();
                        $dom->loadXML($request);
                        if ( $this -> useWSA ) {
                            $objWSA = new WSASoap($dom);
                            $objWSA->addAction($action);
                            if ( empty($this -> to )) {
                                $objWSA->addTo($location);
                            } else {
                                $objWSA->addTo($this -> to);
                            }
                            $objWSA->addMessageID($this -> messageId);
                            $objWSA->addReplyTo($this -> replyTo);
                            $objWSA->addFaultTo($this -> faultTo);
                            $dom = $objWSA->getDoc();
                }
                if ( $this -> useWSSE ) {
                        $objWSSE = new WSSESoap($dom);
                        $objWSSE->addTimestamp($this -> secondToExpires);
                            if ( $this -> addUsernameToken ) {
                                $objWSSE->addUserToken($this -> tokenUsername, $this ->
                                tokenPassword, $this -> tokenDigest);
                            }
                            if ( $this -> signSoapDoc ) {
                                $objWSSE->signAllHeaders = $this -> signAllHeader;
                                 $objKey = new XMLSecurityKey(XMLSecurityKey::RSA_SHA1,
                                array('type'=>'private'));
                                 $objKey->loadKey($this -> pathToPrivateKey, true);
                                $objWSSE->signSoapDoc($objKey);
                            }
                            if ( $this -> addCertificate ) {
                                $token = $objWSSE->addBinaryToken(file_get_contents($this ->
                                pathToCert));
                                $objWSSE->attachTokentoSig($token);
                            }
                        $request = $objWSSE->saveXML();
                    } else {
                        $request = $objWSA->saveXML();
                    }
                    $this -> modifiedRequest = $request;
                }
                return parent::__doRequest($request, $location, $action, $version);
        } catch (Exception $e) {
                throw new Exception("Unable to perform __doRequest: " . $e -> getMessage() . " : " .
$e->getFile() . " [Line:" . $e->getLine() . "]");}     }
```

Figure 26: Override SoapClient's __doRequest()

As a result, the SoapClientEnhanced class now has additional methods that support WS-Addressing and WS-Security specifications (Fig. 27).

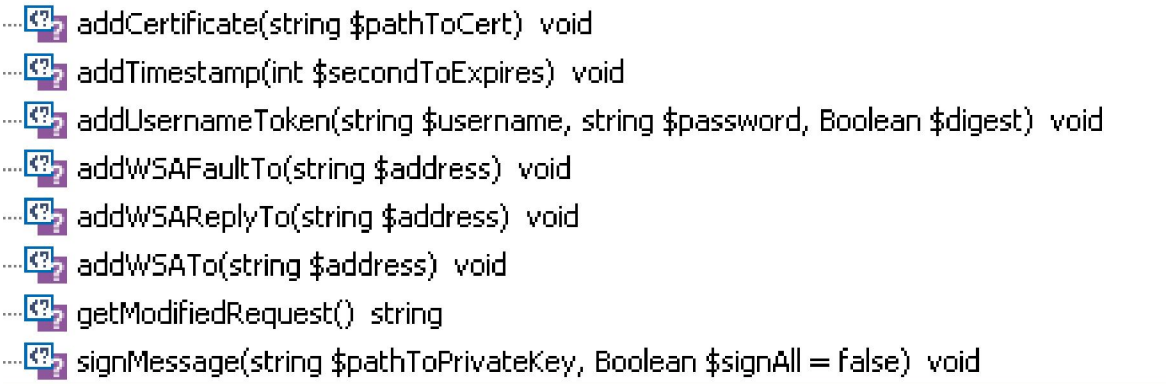

Figure 27: WS-Addressing and WS-Security support

Secondly, using the __getTypes() function, we retrieve the complex types from the WSDL file provided by the Service provider and then map them to the corresponding PHP classes [Delves]. In our case, we prefix the PHP classes with "Amazon_". At the same time, we use the __getFunctions() function to retrieve the Response types to avoid mapping those types since they are not needed. The following codes segment below illustrates this process (Fig. 28):

```
private function _createClassmap() {
        $type = null;
        $name = null;
        $classname = null;
        if ( is_array($funcArray = ($this -> __getFunctions())) ) {
                foreach (new ArrayObject($funcArray) as $func ) {
                        preg_match("/([a-z0-9_]+)\s+([a-z0-9_]+(\[\])?)(.*)?/si", $func,
                        $fMatches);
                        if ( !is_null($fMatches[1]) ) {
                                $this -> returnedObjects[] = trim($fMatches[1]);
                        }
                }
        }
        if ( is_array($typesArray = ($this -> __getTypes()) ) ) {
                foreach (new ArrayObject($typesArray) as $i => $type ){
                        preg_match("/([a-z0-9_]+)\s+([a-z0-9_]+(\[\])?)(.*)?/si", $type,
                        $matches);
                        $type = $matches[1];
                        $name = trim($matches[2]);
                        switch (strtolower($type)) {
                                case 'struct':
                                        $this -> _createClass($name);
                                        break;
                        }
                }
        }
}
private function _createClass($name) {
        try {
                if ( !is_null($name) && !in_array($name, $this -> returnedObjects)){
                        $name = trim(strip_tags($name));/* Remove any potential executable
                        codes */
                        $classname = $this -> classname_prefix . $name;
                        $this -> types[$name] = trim($classname);/*Schema Element => PHP
                        Class name */
                        /* check the class does not exsits before creating it */
                        if(!(class_exists($classname) &&  in_array($classname,
                        get_declared_classes()))) {
                                eval("class $classname {}");
                        }
                }
        } catch (Exception $ex) {
                throw new SoapFault('Server Error', "Unable to create class $classname: " . $ex
                -> getMessage() . " [" . $ex ->getTraceAsString() . "]");}}
```

Figure 28: Mapping Complex types to PHP classes

After the classmap array is created, we will need to initiate the SOAP client one more time with the array of options. At a first glance, this is not a very good solution since the SOAP client will suffer in performance due to the fact that it will have to parse the WDSL file again. However, PHP native SOAP client has another very useful feature: it could be set to cache the WSDL in Memory, on Disk or Both. For our project, we set it to cache the WSDL file for both in Memory and on Disk. The resulting constructor is listed below (Fig. 29):

```
function __construct($wsdl, $classname_prefix='', $debug=false) {
        try {
                $this -> wsdl = trim($wsdl);
                $this -> classname_prefix = trim($classname_prefix);
                $this -> debug = $debug;
                parent::__construct($wsdl);
                $this -> _createClassmap();
                $this -> options['trace'] = ($debug) ? 1 : 0;
                $this -> options['cache_wsdl'] = WSDL_CACHE_BOTH;/*Cache the
                WSDL in memory and on disk */
                $this -> options['classmap'] = $this -> types;
                parent::__construct($wsdl, $this -> options);
        } catch (SoapFault $ex) {
                throw $ex;
        }
}
```

Figure 29: SoapClientEnhanced constructor

As the result of the above actions, now we could easily create an Amazon_Item class, which will have all properties of the original Amazon's Item complex type, but with additional functionalities. The sample code is listed below (Fig. 30):

```
<?php
/**
* @name Amazon_Item.class
 * Wrap Amazon's Item
 */
class Amazon_Item {

	/**
	 * Wrap a URL from Amazon in A tag to display on the site
	 *
	 * @return string
	 */
	public function getDetailPageURL() {
		$url = "<a href=\"" . (string)$this -> DetailPageURL . "\"><b>Click for more
		Detail</b></a>";
		return $url;
	}

	/**
	 * Display item in detail
	 * Brand and Features information could be null,
	 * so we need to make sure to handle cases in which
	 * information is empty
	 *
	 * @return string
	 */
	public function getDisplayURL() {
		$tpl_vars['PRODUCT_URL'] = $this -> DetailPageURL;
		$tpl_vars['IMG_SRC'] = $this -> SmallImage -> URL;
		$tpl_vars['DESCRIPTION'] = $this -> ItemAttributes -> Title;
		$tpl_vars['LIST_PRICE'] = !is_null($this -> OfferSummary -> LowestNewPrice ->
		FormattedPrice)
		?  $this -> OfferSummary -> LowestNewPrice -> FormattedPrice
		: $this -> ItemAttributes -> ListPrice -> FormattedPrice;
		$tpl_vars['BRAND'] = !is_null($this -> ItemAttributes -> Brand) ? $this ->
		ItemAttributes -> Brand : "<i>n/a</i>";
		$features = $this -> ItemAttributes -> Feature;
		if ( is_array($features) )
		{
			$features = implode('<br>', $features);
		}
		$tpl_vars['FEATURES'] = !(is_null($features)) ? $features : "<i>n/a</i>";
		return ( template('/index.php/product.tpl', $tpl_vars, 1));
	}
}//end of Amazon_Item
?>
```

Figure 30: Using complex types from Web Service

4.2 Using the Enhanced Soap Client

Once the SoapClientEnhanced has been completed, to actually consume a Web service, we just need to extend it and provide it with the location to the WSDL file (or the WSDL file itself), the optional class prefix used to name the PHP classes for the classmap option and the debug option (to turn the trace option on). For instance, to consume Amazon S3, we just simply extend the SoapClientEnhanced and add additional functionalities required for Amazon S3 as listed below (Fig. 31):

```
<?php
/**
* @name AmazonS3.class
*/
require_once('SoapClientEnhanced.class.php');
class AmazonS3 extends SoapClientEnhanced {
        private $prefix = 'Amazon_';
        private $debug = true;
        private $wsdl = 'http://s3.amazonaws.com/doc/2006-03-01/AmazonS3.wsdl';
        const _AMAZON_WS_KEY_ = 'WS Access Key Id';//Amazon Web Service Key
        const _AMAZON_SEC_KEY_ = 'Secret Key';//Amazon Secret
        const _AMAZON_S3_PREFIX_ = 'AmazonS3';
        function __construct() {
                parent::__construct($this->wsdl, $this->prefix, $this->debug);
        }
        /**Amazon S3 requires the signature, which is a HASH_HMAC of the combination
         * of ("AmazonS3" + OPERATION + Gmdate timestamp)
         */
        public function createSignature($timestamp, $operation) {
                return ( base64_encode(hash_hmac('sha1', AmazonS3::_AMAZON_S3_PREFIX_ .
                trim($operation) . $timestamp, AmazonS3::_AMAZON_SEC_KEY_, true)));
        }
        public function createTimestamp() {
                return (gmdate("Y-m-d\TH:i:s.B", time()) .'Z');
        }
}
?>
```

Figure 31: AmazonS3 class

The following codes fragment (Fig. 32) illustrates how to initiate a request using the AmazonS3 class:

```
/**
* List all buckets in AmazonS3 that are associated with the given key
 *
 */
public function listBuckets() {
        try {
                $request = new Amazon_ListAllMyBuckets();
                $request -> AWSAccessKeyId = AmazonS3::_AMAZON_WS_KEY_;
                $timestamp = $this -> s3 -> createTimestamp();
                $request -> Timestamp = $timestamp;
                $request -> Signature = $this -> s3 -> createSignature('ListAllMyBuckets',
                $timestamp);
                $result = $this -> s3 -> ListAllMyBuckets($request);
                $this -> buckets = $result -> Buckets;
        } catch (SoapFault $ex) {
                throw new Exception("Unable to retrieve all buckets: " . $ex->faultstring);
        }
}
```

Figure 32: create S3 message

The following is the WSDL fragment for Amazon Associates Web service's ItemSearch (Fig. 33).

```
- <xs:element name="ItemSearch">
  - <xs:complexType>
    - <xs:sequence>
        <xs:element name="MarketplaceDomain" type="xs:string" minOccurs="0"/>
        <xs:element name="AWSAccessKeyId" type="xs:string" minOccurs="0"/>
        <xs:element name="SubscriptionId" type="xs:string" minOccurs="0"/>
        <xs:element name="AssociateTag" type="xs:string" minOccurs="0"/>
        <xs:element name="XMLEscaping" type="xs:string" minOccurs="0"/>
        <xs:element name="Validate" type="xs:string" minOccurs="0"/>
        <xs:element name="Shared" type="tns:ItemSearchRequest" minOccurs="0"/>
        <xs:element name="Request" type="tns:ItemSearchRequest" minOccurs="0" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
- <xs:complexType name="ItemSearchRequest">
  - <xs:sequence>
      <xs:element name="Actor" type="xs:string" minOccurs="0"/>
      <xs:element name="Artist" type="xs:string" minOccurs="0"/>
    - <xs:element name="Availability" minOccurs="0">
      - <xs:simpleType>
        - <xs:restriction base="xs:string">
            <xs:enumeration value="Available"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>
      <xs:element ref="tns:AudienceRating" minOccurs="0" maxOccurs="unbounded"/>
      <xs:element name="Author" type="xs:string" minOccurs="0"/>
      <xs:element name="Brand" type="xs:string" minOccurs="0"/>
      <xs:element name="BrowseNode" type="xs:string" minOccurs="0"/>
      <xs:element name="City" type="xs:string" minOccurs="0"/>
      <xs:element name="Composer" type="xs:string" minOccurs="0"/>
      <xs:element ref="tns:Condition" minOccurs="0"/>
      <xs:element name="Conductor" type="xs:string" minOccurs="0"/>
    - <xs:element name="Count" type="xs:positiveInteger" minOccurs="0">
      - <xs:annotation>
        - <xs:appinfo>
          - <aws-se:restricted>
              <aws-se:excludeFrom>public</aws-se:excludeFrom>
              <aws-se:excludeFrom>partner</aws-se:excludeFrom>
```

Figure 33: Item Search WSDL fragment

The sample code below illustrates how to send a SOAP message according to the above WSDL fragment (Fig. 34).

```
/**
 * Search for the list of Items from Amazon Web service with the given search index,
 * search keyword, responsegroup, page number and optional node id.
 * The page number and totalPages number are used in the paginating system (to display
 * the next and previous link when applicable.
 *
 */
public function search() {
        try {
                $itemSearch = new Amazon_ItemSearch();/* the same as using
                        ItemSearch->AWSAccessKeyId = $accessKey */
                $itemSearch -> AWSAccessKeyId = Amazon::_AMAZON_WS_KEY_;
                $itemSearch -> AssociateTag = Amazon::_AMAZON_ASSOCIATE_TAG_;
                //Associate Tag to get paid
                $request = new Amazon_ItemSearchRequest();
                $request -> SearchIndex = $this -> searchIndex;
                $request -> Keywords = $this -> searchKeyword;
                $request -> ResponseGroup = $this -> responseGroup;
                $request -> ItemPage = $this -> pageNumber;
                if ( $this -> nodeId > 0 )
                        $request -> BrowseNode = $this -> nodeId;
                $itemSearch -> Request = $request;
                $response = $this -> amazon -> ItemSearch($itemSearch);
                $itemsArray = $response -> Items;
                $this -> totalPages = $itemsArray -> TotalPages;
                $this -> totalResults = $itemsArray -> TotalResults;
                $this -> items = $itemsArray -> Item;
        } catch (SoapFault $e) {
                throw $e;
        }
}
```

Figure 34: Create SOAP message for Amazon's ItemSearch operation

However, not all methods are straight forward. For example, Amazon EC2 requires extra information in order to support WS-Security specification (Fig. 35). The paths to the Private Key and the X.509 Certificates are required.

```
try {
        $client = new AmazonEC2();
        $client -> addWSAFaultTo("http://stockcamera.com/replyto");
        $client -> signMessage("/hsphere/local/home/sonny/phpinclude/ws/pkey.pem",
true);
        $client -> addCertificate("/hsphere/local/home/sonny/phpinclude/ws/cert.pem");
        $request = new Amazon_DescribeImagesType();
        $request -> imagesSet -> item -> imageId = 'ami-be3cd8d7';
        $test =  $client -> DescribeImages($request);
} catch (SoapFault $ex) {
        echo "SOAPFAULT: " . $ex->faultstring;
}
```

Figure 35: Using EC2 method

In addition, the resulting SOAP message (list below) is much more involved, which include the following parts: the WS-Addressing header if it is being used (Fig. 36), the BinarySecurity Token (Fig. 37), the SignedInfo and Digest (digest is usually supplied to speed up the processing time) (Fig. 38), the Signature and Key info (Fig. 39) and the Payload itself (Fig. 40). The SOAP body in this example is signed.

```
<?xml version="1.0" encoding="UTF-8"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:ns1="http://ec2.amazonaws.com/doc/2008-08-08/"
xmlns:wsa="http://schemas.xmlsoap.org/ws/2004/08/addressing">
<SOAP-ENV:Header>
<wsa:Action xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" wsu:Id="pfx1746d739-af39-4174-9af3-c4441c0a3eb3">DescribeImages</wsa:Action>

<wsa:To xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-
1.0.xsd" wsu:Id="pfxe71915ed-81cf-6b77-5d98-
b983790eb1ba">https://ec2.amazonaws.com/</wsa:To>

<wsa:MessageID xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="pfx1c734562-6756-9dad-bf7e-36e9e8b8c4f4">uuid:8a15ebb0-9bea-4183-ce0c-
148905ecb180</wsa:MessageID>

<wsa:ReplyTo xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="pfxc5760bb4-4bc9-e40d-4576-f72321410aa6">
<wsa:Address>http://schemas.xmlsoap.org/ws/2004/08/addressing/role/anonymous
</wsa:Address></wsa:ReplyTo>

<wsa:FaultTo xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
utility-1.0.xsd" wsu:Id="pfx2e6b22bb-db89-cbbe-991b-8197f99d3e67">
<wsa:Address>http://stockcamera.com/replyto</wsa:Address></wsa:FaultTo>
```

Figure 36: Signed WS-Addressing headers

```
<wsse:Security xmlns:wsse="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-
secext-1.0.xsd" SOAP-ENV:mustUnderstand="1">
<wsse:BinarySecurityToken xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" EncodingType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
soap-message-security-1.0#Base64Binary" wsu:Id="pfx726e08ed-8626-3c53-368e-bd08b23c2306"
ValueType="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-x509-token-profile-
1.0#X509v3">MIICdjCCAd+gAwIBAgIFHwiWRC8wDQYJKoZIhvcNAQEFBQAwUzELMAkGA1UEBhMCVVMx
EzARBgNVBAoTCkFtYXpvbi5jb20xDDAKBgNVBAsTA0FXUzEhMB8GA1UEAxMYQVdTIExpbWl0ZWQtQXNz
dXJhbmNlIENBMB4XDTA4MTEwNTA2MTQxM1oXDTA5MTEwNTA2MTQxM1owUjELMAkGA1UEBhMCVV
MxEzARBgNVBAoTCkFtYXpvbi5jb20xFzAVBgNVBAsTDkFXUy1EZXZlbG9wZXJzMRUwEwYDVQQDEwxsZ20
0ZGJwbXJ1cXcwgZ8wDQYJKoZIhvcNAQEBBQADgY0AMIGJAoGBAN/IdgOrW54/gQ5/5qIAL4IVmjX4I2xN9
HsKBx75GwAqSMfVNjlE9MVpj4vaZgd3bqU8/49g6UuGID7yid40VHF171PS5Z82HbJjea+crEBLQZOhTe0s2
de7+tD6Pzogy9OAnNEo9xfaBMpLXO9hhU1oz2VWwwwSNDYk01RhzHwNAgMBAAGjVzBVMA4GA1UdD
wEB/wQEAwIFoDAWBgNVHSUBAf8EDDAKBggrBgEFBQcDAjAMBgNVHRMBAf8EAjAAMB0GA1UdDgQW
BBQ5eYPukh3uUDOjX0MxUFiYzov81TANBgkqhkiG9w0BAQUFAAOBgQB+MyEcBJNGpMzAQG8L+dFQkQ9
jb5Whzh6sW+v3POWrLyq8tnmcCGlk0YbME2PFiz3LhWR1Q6B/YrMho8OBVxJaiX5a8IOWquA4Twaz+bYH
AC1CiQ+CAL4k3KvRx1umzzMpWOtGw7vhr8xSb4ZVmvxQzU0Lb4G1qkSOEPIWjkC/fQ==</wsse:BinarySec
urityToken>
```

Figure 37: X.509 Security Token

<ds:SignedInfo><ds:CanonicalizationMethod Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/>
<ds:SignatureMethod Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1"/>
<ds:Reference URI="#pfxb0abffb7-608e-b790-563c-6f08a4eac005">
<ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transforms><ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>EIEEDlDAmiEexx+pmkN21Bcwi14=</ds:DigestValue></ds:Reference><ds:Reference URI="#pfx1746d739-af39-4174-9af3-c4441c0a3eb3">
<ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/>
<ds:DigestValue>zxQ5Z/91I5ErBULVQnJ72GWBG2Y=</ds:DigestValue></ds:Reference><ds:Reference URI="#pfxe71915ed-81cf-6b77-5d98-b983790eb1ba">
<ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><ds:DigestValue>syP/7EtX/RG4OLIK3fEF2IsaA+E=</ds:DigestValue></ds:Reference>
<ds:Reference URI="#pfx1c734562-6756-9dad-bf7e-36e9e8b8c4f4"><ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transforms>
<ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><ds:DigestValue>SGOeSpE9n81wP0RY6dnQj1D0SRo=</ds:DigestValue></ds:Reference><ds:Reference URI="#pfxc5760bb4-4bc9-e40d-4576-f72321410aa6"><ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transforms><ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><ds:DigestValue>9IOlqEpeko4jYxtWiW7TAri8gq4=</ds:DigestValue></ds:Reference><ds:Reference URI="#pfx2e6b22bb-db89-cbbe-991b-8197f99d3e67"><ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transforms><ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><ds:DigestValue>2AtG966Pl0wwPJaqY3zfqmTCMo8=</ds:DigestValue></ds:Reference><ds:Reference URI="#pfxab5cb5c1-c345-662f-dd00-b928fd45dbfb"><ds:Transforms><ds:Transform Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#"/></ds:Transforms><ds:DigestMethod Algorithm="http://www.w3.org/2000/09/xmldsig#sha1"/><ds:DigestValue>wk0HqGO9Lf1jN/W7k1l8SJ9PCR4=</ds:DigestValue></ds:Reference></ds:SignedInfo>

Figure 38: SignedInfo and Digest

```
<ds:SignatureValue>vTcCLkxk27ifyLQfm79ZhZjmdiNZHY31p0QJr2a5jHZisDXmDJUnr9xCA+3N6kxw5
ObZbah784RWA1iEeNCjvuHVw3PywnmVjI96q9ZEXnm3hfef+h5zwn9/IYqwSyF/apiKXPKspuNC+qken
Mwuu+giApSFCW3ZzoWJcuaelJs=</ds:SignatureValue>

<ds:KeyInfo><wsse:SecurityTokenReference><wsse:Reference URI="#pfx726e08ed-8626-3c53-368e-
bd08b23c2306"/></wsse:SecurityTokenReference></ds:KeyInfo></ds:Signature><wsu:Timestamp
xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-wssecurity-utility-1.0.xsd"
wsu:Id="pfxb0abffb7-608e-b790-563c-6f08a4eac005"><wsu:Created>2008-11-
10T02:46:44Z</wsu:Created><wsu:Expires>2008-11-
10T03:46:44Z</wsu:Expires></wsu:Timestamp></wsse:Security>
```

Figure 39: Signature and Key Info and other Security Headers

```
<SOAP-ENV:Body xmlns:wsu="http://docs.oasis-open.org/wss/2004/01/oasis-200401-wss-
wssecurity-utility-1.0.xsd" wsu:Id="pfxab5cb5c1-c345-662f-dd00-
b928fd45dbfb"><ns1:DescribeImages><ns1:imagesSet><ns1:item><ns1:imageId>ami-
be3cd8d7</ns1:imageId></ns1:item></ns1:imagesSet></ns1:DescribeImages>
</SOAP-ENV:Body>
```

Figure 40: A signed payload

Chapter 5 Applications and Extensions

5.1 Main problems and Solution Details

The main problems of this thesis were that how well PHP SOAP, a weak-typed language, would consume Web services with complex types, and if PHP SOAP could easily support WS-* specifications. As we have described and illustrated by implementing the Amazon Web Service Consumer system using a combination of existing methods and libraries [Richards, Delves], PHP SOAP could be easily extended to support WS-* specifications and to consume complex types from various Web services.

5.2 Using the Amazon Web Service Consumer System

In order to use Amazon Web service, a free Web service Id (AWS Access Key) is required. That key will allow user to access to Amazon Associates Web service where users could perform a wide range of functionalities, from searching for Items listed in Amazon, getting detail on certain item, or even using the Remote Shopping Cart. However, in order to earn commissions for the referrals, user will need to sign up with Amazon Affiliates Central (https://affiliate-program.amazon.com) to get the Associate Tag Id, which end in "-20" for North America customers. This Associate Tag must be included in order to earn commissions.

In addition, while creating the AWS Access Key, it is recommended to create the Private Key and the X.509 certificate. They will be needed in order to use Amazon EC2 and Amazon SimpleDB service.

In the shared hosting account, create the "phpinclude" folder in the Root directory, not in any domain folder (Fig. 41).

Figure 41: Create the phpinclude directory

Next, create the “ws” and the “templates” folders and upload the handler.inc.php and templates.inc.php file to the “phpinclude” directory (Fig. 42).

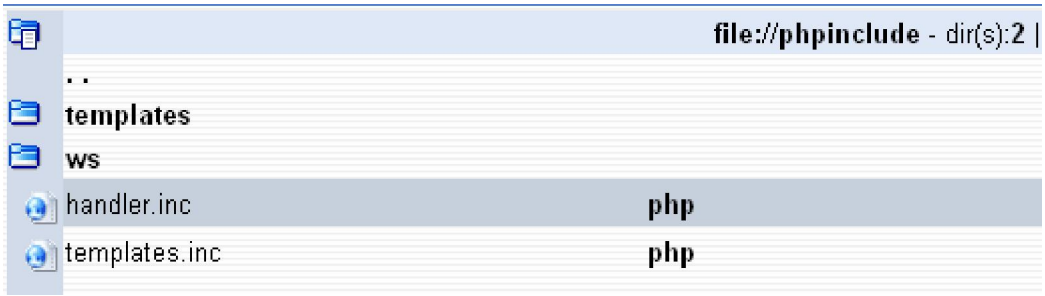

Figure 42: uploading hander and templates

After that, the core classes will need to be uploaded to the “ws” directory (Fig. 43).

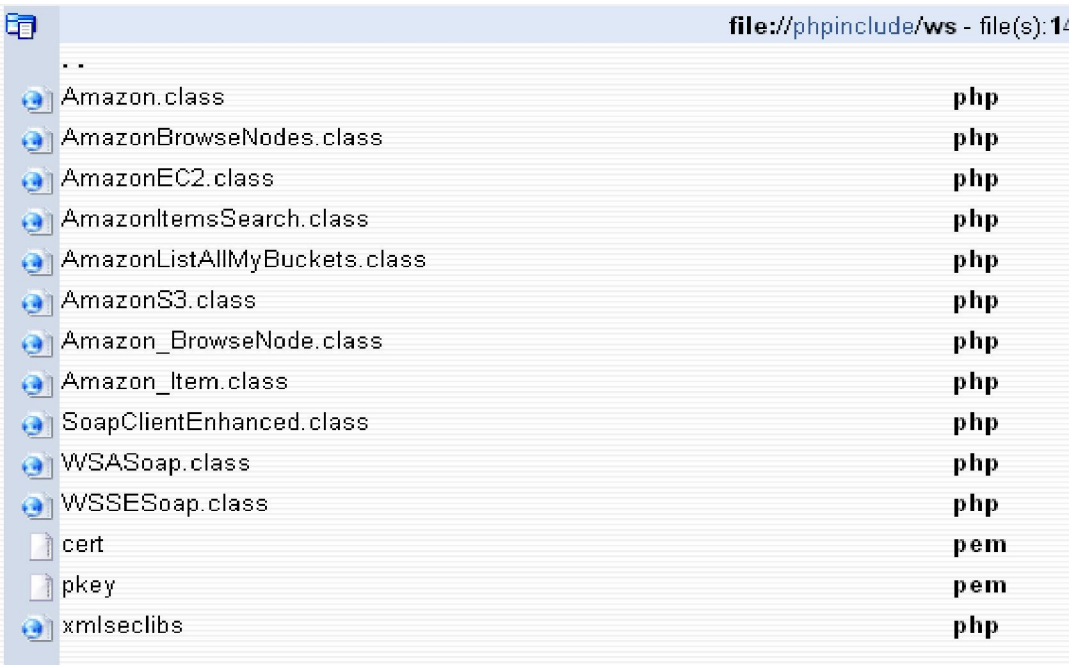

Figure 43: the content of ws directory

I create a directory called "index.php" inside the "templates" directory and upload the HTML templates for this project (Fig. 44). In addition, make sure to upload any required image for these HTML templates to the appropriate directory.

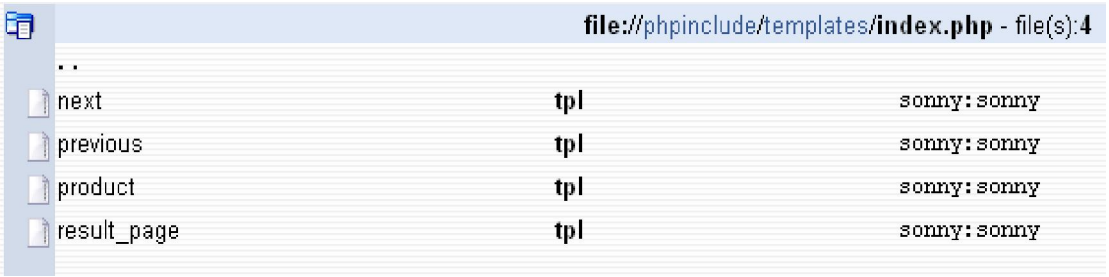

Figure 44: HTML templates

Then, I made sure to set the AWS Access Key and Associate Tag in the Amazon.class.php class. Go to the one of the domain folder and edit the index.php file to use the new Web application. The codes fragment is listed below (Fig. 45):

```
<?php
/**
 * $Id: $
 * @name index.php
 * Handler request
 *
 */
require_once('/hsphere/local/home/sonny/phpinclude/handler.inc.php');
$defaultSearch = 'Digital Cameras';
$defaultSearchIndex = 'Photo';
$defaultResponseGroup = 'Medium';
$defaultNodeId = 51547011;//This is for the nav bar and should not be changed
$domainTitle = 'Stock Cameras';
$specialBanner = "<iframe src='http://rcm.amazon.com/e/cm?t=stockcacom-
20&o=1&p=8&l=as1&asins=B000FI73MA&fc1=000000&IS2=1&lt1=_blank&m=amaz
on&lc1=0000FF&bc1=000000&bg1=FFFFFF&f=ifr' style='width:120px;height:240px;'
scrolling='no' marginwidth='0' marginheight='0' frameborder='0'></iframe>";

handler($defaultSearch, $defaultSearchIndex, $defaultResponseGroup,
$defaultNodeId, $domainTitle, $specialBanner);

?>
```

Figure 45: index.php

The handler.inc.php's handler function requires the default search, search index, the response group, the node id, the domain title and the optional special banner as its arguments.

By repeating the same process, I could use other websites as well. To display different type of items, the only required changes are the default search, default search index and the node id. The optional special banner is a link created within the Amazon's Affiliates Central dashboard.

The Items from Amazon are listed on the websites (Fig. 46 and Fig. 47).

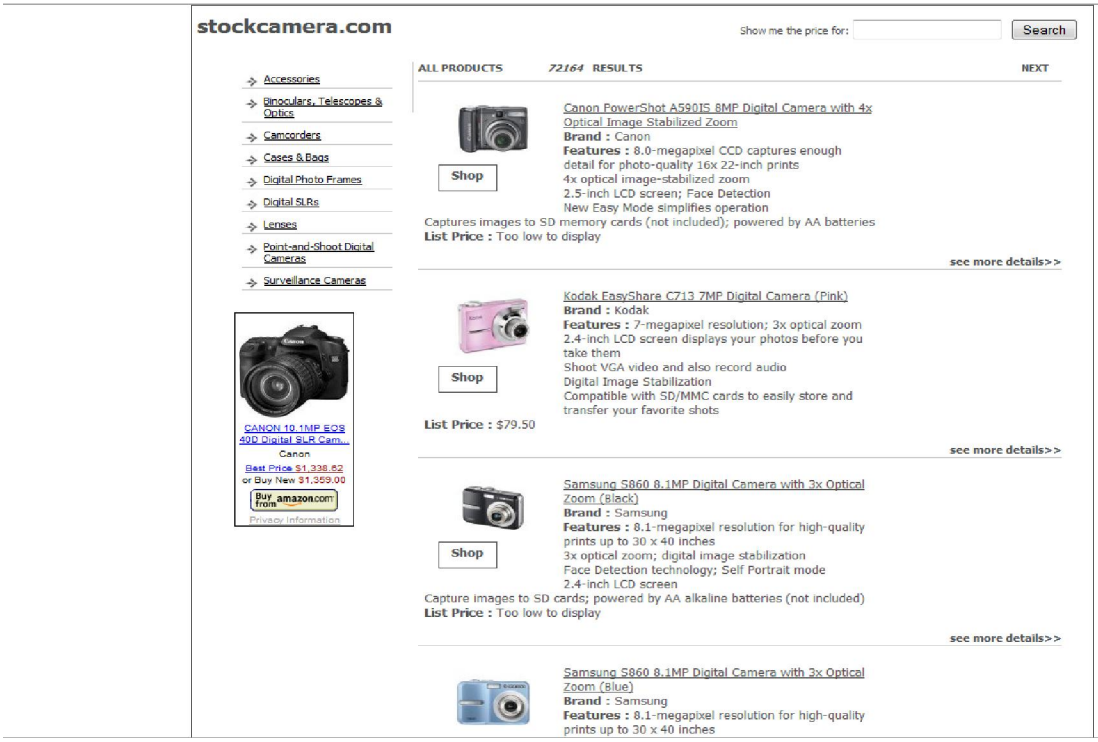

Figure 46: StockCamera.com

Figure 47: TimeclockSolution.com

These sites have the same look since they use the same HTML templates, images and style sheets. However, they could be easily changed since they are independent from the Web application. On the contrary, the contents of each site are not the same since each site has different search keyword and different search index.

5.3 Potential for further development

This Web application could be potentially expended to a fully functional domains contents generation system by adding extra functionalities such as automatically associate a domain name to a keyword and the corresponding search index. Moreover, by simply adding a SOAP server component to it, the same application could potentially serve domains hosted elsewhere as well. For example, a domain hosted elsewhere could request the generated contents by giving the domain name or keyword (Fig. 48).

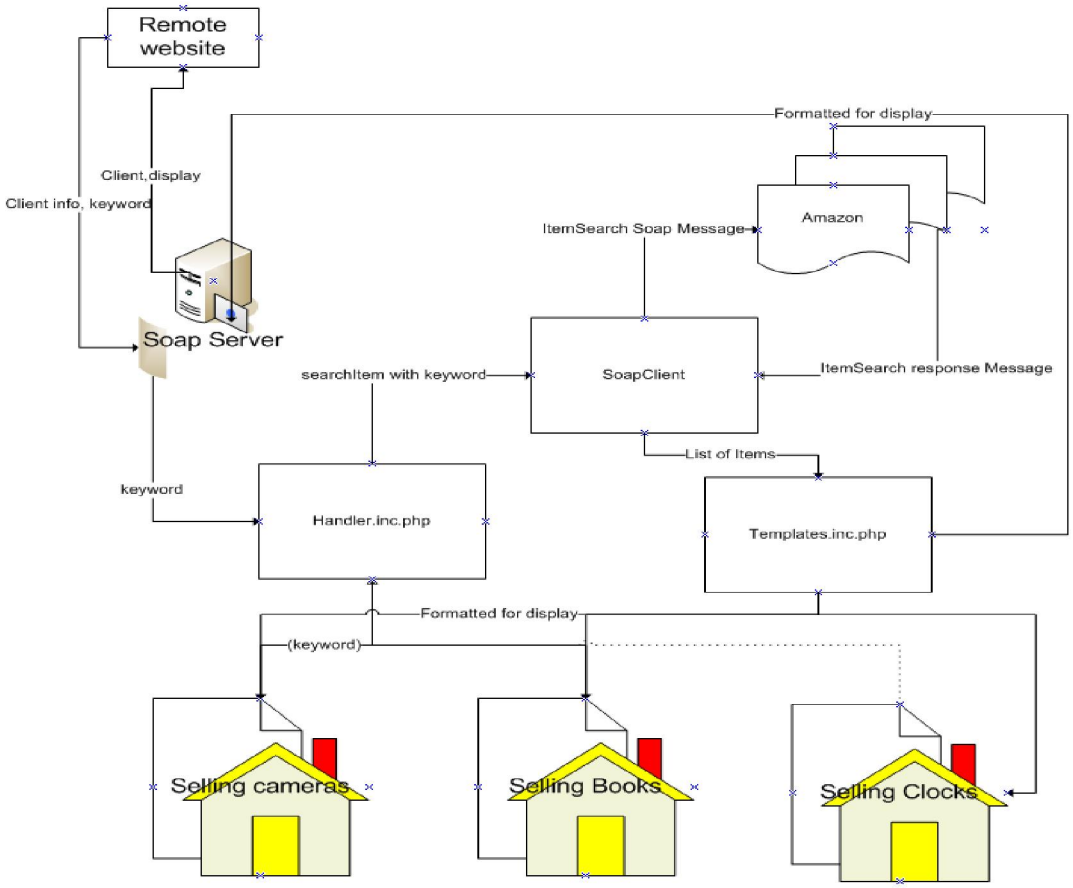

Figure 48: Serving Remote Websites

Furthermore, instead of redirecting visitors back to Amazon website, the application could be extended to utilize Amazon's Remote Shopping Cart feature. Moreover, since the Web application is ready to consume other Amazon web services as well, it could be implemented so that it could backup itself nightly and upload its backup images to Amazon S3.

In another word, an application could go a long way in the SOA world as long as it could efficiently and securely consume Web services.

Chapter 6 Conclusion

With this thesis, I stated the problems and limitations of PHP SOAP in consuming Web services with complex types and require WS-* specifications. I also stated other potential solutions to the problems. I consequently showed the solutions to those problems by implementing the Amazon Web Service Consumer system using PHP SOAP.

In addition, this thesis helped me a great deal on understanding certain WS-* specification such as WS-Addressing and WS-Security. I also had researched intensively on SOA and also the capabilities of the PHP SOAP extension.

The application is very practical and could potentially help users with large number of undeveloped domain names to monetize and to gain traffic with little to none human intervention. In addition, it is much cost effective since it targets shared hosting environments instead of requiring dedicate hosting environments. Moreover, the functionalities are already implemented to consume other Amazon Web services.

References

[AmazonAssociates] Amazon Associates Compensation Overview https://affiliate-program.amazon.com/gp/associates/join/compensation.html

[Delves] Adam Delves, "Using XML, Part 5: SOAP and WSDL", http://www.phpbuilder.com/columns/adam_delves20060606.php3?print_mode=1

[EC2] Amazon Elastic Compute Cloud http://aws.amazon.com/ec2/

[NetCraft] NetCraft, October 2008 Web Server Survey, http://news.netcraft.com/archives/2008/10/29/october_2008_web_server_survey.html

[NuSOAP] NuSOAP http://sourceforge.net/projects/nusoap/

[O'Niell] Mark O'Niell, et al., *Web Services Security*, McGraw-Hill Companies, 2003

[Pear:SOAP] Pear:SOAP http://pear.php.net/package/SOAP

[PHP] PHP: Hypertext Preprocessor http://php.net/

[Richards] Robert Richards, CDATA Zone http://www.cdatazone.org/index.php?/pages/source.html

[S3] Amazon Simple Storage Service http://aws.amazon.com/s3/

[SAML] OASIS Standard, "Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) v2.0", Mar 15, 2005. http://docs.oasis-open.org/security/saml/v2.0/saml-core-2.0-os.pdf

[SimpleDB] Amazon SimpleDB http://aws.amazon.com/simpledb/

[SOAP] W3C Recommendation, "SOAP version 1.2", Apr 27, 2007. http://www.w3.org/TR/soap12-part1/

[TIOBE] TIOBE Software http://www.tiobe.com/content/paperinfo/tpci/index.html

[UDDI] OASIS' Universal Description, Discovery and Integration, http://uddi.xml.org/

[W3C] The World Wide Web Consortium http://www.w3.org/

[WS-Addressing] W3C Recommendation, “WS-Addressing”, May 9, 2006. http://www.w3.org/TR/ws-addr-core/

[WSDL] W3C Note, “Web Service Description Language”, Mar 15, 2001. http://www.w3.org/TR/wsdl

[WSF/PHP] WS02 OxygenTank’s Web Service Framework for PHP, http://wso2.org/projects/wsf/php

[WS-Security] OASIS Standard Specification, “Web Services Security: SOAP Message Security 1.1 (WS-Security 2004)”, Feb 01, 2006. http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf

[Xpath] W3C Recommendation, “XML Path Language”, Nov 16, 1999. http://www.w3.org/TR/xpath

Vita

Sonny Vo was born in Vietnam. He received a degree of Bachelor of Science in Computer Science from the University of New Orleans, New Orleans, Louisiana in 2000. In the following years, he entered the graduate study program at the University Of New Orleans. His research interests include distributed computing, web-based applications and event-driven systems.