บทที่ 13 เว็บเซอร์วิสแบบ SOAP

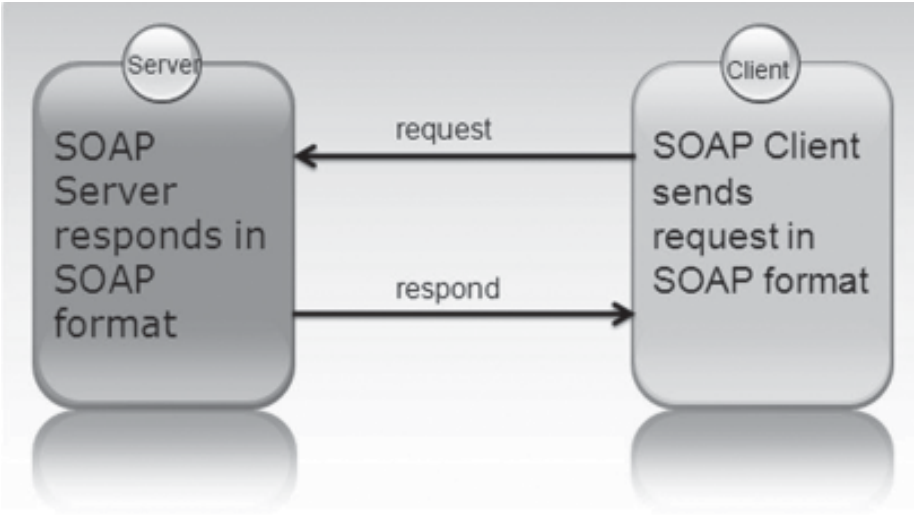
>> ทรงฤทธิ์ ลีมัคเดช

วัตถุประสงค์การเรียนรู้

ผู้เรียนจะศึกษาโครงสร้างของการใช้เว็บเซอร์วิสแบบ SOAP ตลอดจนการพัฒนาโปรแกรมด้วยภาษา Ruby เพื่อเรียกใช้งาน หรือให้บริการข้อมูลแบบ SOAP

ถ้าเปรียบ REST เหมือนกับเป็นตลาดนัดแบบอิสระ เปิดกว้างใครอยากจะตั้งแผงขายอะไรตรงไหนก็ได้ SOAP (Simple Object Access Protocol) ก็เปรียบเสมือนมหาวิหารที่มีรูปแบบวิจิตรที่สลับซับซ้อน แต่อยู่ในกฎเหล็กที่แน่นอนของศาสนจักร (ฟังดูคล้ายๆ บทความเรื่อง Cathedral and the Bazaar ของ Eric Reymond) SOAP ใช้โปรโตคอล HTTP เป็นหลักเช่นเดียวกับ REST แต่เลือกใช้วิธี POST อย่างเดียวเท่านั้น SOAP เน้นในเรื่องของรูปแบบการรับการส่ง การประกาศชนิดของข้อมูล อันเป็นผลมาจากรากฐานของการเรียกวัตถุแบบ COM, RMI, และ RPC กล่าวง่ายๆ ก็คือ SOAP เป็นการรวมเทคโนโลยีเหล่านี้ในรูปแบบ Binary ให้อยู่ในรูปแบบ XML เท่านั้นเอง

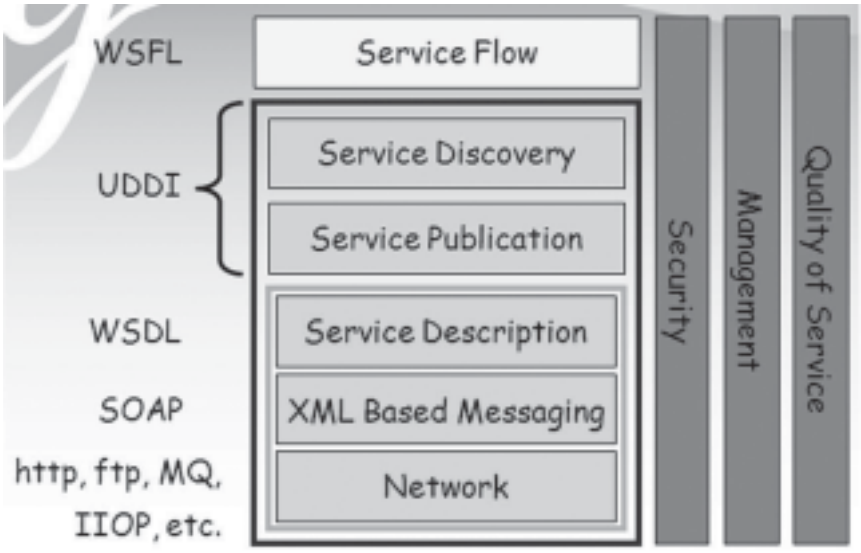
โครงสร้างของ เว็บเซอร์วิสแบบ SOAP คือ มีระบบงานฝั่งผู้ใช้บริการ (SOAP Client) ส่งคำร้องในรูปแบบ XML (SOAP request message) ไปที่ระบบงานผู้ให้บริการ (SOAP Server) ผู้ให้บริการก็ทำการประมวลผล แล้วส่งคำตอบกลับมาในรูปแบบ XML (SOAP response message) ดังแสดงในรูปที่ 13.1



รูปที่ 13.1 การเรียกใช้ SOAP

13.1 The Web Services Stack

SOAP ถือเป็นส่วนหนึ่งของโครงสร้างของมาตรฐานการทำงานร่วมกันของเว็บเซอร์วิส (Web Service Interoperability Organization: WS-I) ซึ่งเป็นการก่อตั้งขึ้นของผู้ประกอบการในอุตสาหกรรมเทคโนโลยีสารสนเทศ เช่น IBM Microsoft Oracle HP โดยโครงสร้างนี้ สามารถแสดงในรูปที่ 13.2



รูปที่ 13.2 โครงสร้าง WS-I

เริ่มจากชั้นล่างสุดเป็นโปรโตคอลสื่อสารที่ใช้รับส่งข้อความ SOAP สามารถเลือกใช้ได้หลายโปรโตคอล เช่น http, ftp, MQ (Message Queue), IIOP (Internet Inter-Orb Protocol - Orb หมายถึง Object Request Broker) จากนั้นในระดับของข้อความ จึงเป็นข้อกำหนดของ SOAP ซึ่งอยู่ในรูปแบบของ XML ระดับสูงขึ้นมาเป็นการบรรยายหน้าที่และการเรียกใช้บริการ จะมีภาษามาตรฐานเรียกว่า WSDL (Web Service Description Language)

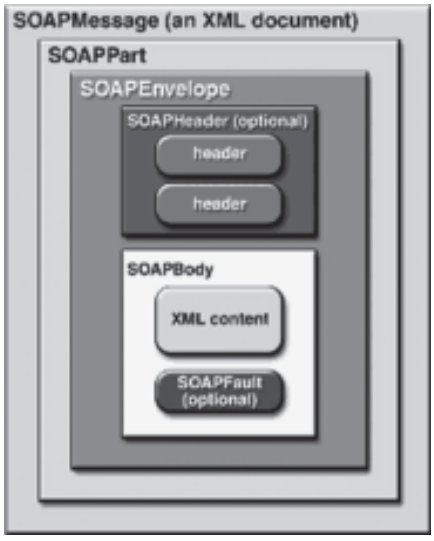
ระดับสูงขึ้นมาเป็นศูนย์เก็บข้อมูลเกี่ยวกับการให้บริการต่างๆ โดยทำหน้าที่ 2 อย่าง คือ Publish (เผยแพร่) และ Discovery (ค้นหา) เรียกว่า UDDI (Universal Description, Discovery and Integration) ส่วนระดับสูงขึ้นมาอีกเป็นการนำเอาบริการต่างๆ มาประกอบเข้าเป็นระบบงาน ซึ่งมีอีกหลายภาษา เช่น WSFL (Web Service Flow Language) และ BPEL (Business Process Execution Language)

Workshop 13.1

1. อภิปรายว่า SOAP ต่างจาก REST อย่างไร
2. แบ่งปันประสบการณ์ในการใช้ SOAP ภายในหน่วยงานของท่าน

13.2 โครงสร้างของ SOAP

โครงสร้างของ SOAP จะคล้ายกับการส่งข้อความในจดหมาย คือมีซอง มีหัวเรื่อง และตัวข้อความ ดังแสดงในรูปที่ 13.3



รูปที่ 13.3 โครงสร้างของ SOAP

13.2.1 Request

ตัวอย่างคำร้อง (SOAP request message) เช่นการขอราคาหุ้นบริษัทไซเบส (สัญลักษณ์ SY) ตามตัวอย่างนี้เป็นการเรียกข้อความชนิด `GetQuote` โดยส่งค่าชนิดตัวอักษรอยู่ในแท็ก `<symbol>`

```
<?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:GetQuote xmlns:n1="urn:ActionWebService"
    env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <symbol xsi:type="xsd:string">sy</symbol>
  </n1:GetQuote>
 </env:Body>
</env:Envelope>
```

13.2.2 Response

ตัวอย่างคำตอบ (SOAP response message) อยู่ในรูปแบบจดหมายเช่นเดียวกัน ในที่นี้เป็นการส่งราคาหุ้นกลับมาในรูปแบบของตัวเลข (double) ราคา $27.99 ชนิดของข้อความที่ตอบกลับมาคือ `GetQuoteResponse` ซึ่งรายละเอียดว่า `GetQuote` เป็นตัวอักษร และ `GetQuoteResponse` เป็นตัวเลขจะต้องถูกกำหนดไว้ในคำบรรยายตามมาตรฐาน WSDL

```
<?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:GetQuoteResponse xmlns:n1="urn:ActionWebService"
    env:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
   <return xsi:type="xsd:double">+27.99</return>
  </n1:GetQuoteResponse>
 </env:Body>
</env:Envelope>
```

13.2.3 Uniform Resource Identifier (URI)

ในการใช้งานเว็บไซต์ทั่วไป หรือเว็บเซอร์วิสแบบ REST เรามักจะคุ้นเคยกับการอ้างอิงโดยใช้ URL (Uniform Resource Locator) แต่ในเว็บเซอร์วิสแบบ SOAP จะรวมหลายบริการเข้าใน URL เดียวกัน ดังนั้นการอ้างอิงจะใช้ URL รวมกับ URN (Uniform Resource Name) โดยที่ URN จะอ้างอิงถึงชื่อของกิจกรรมที่ต้องการ เช่นในเว็บเซอร์วิสของ ปตท. (http://www.pttplc.com/pttinfo.asmx?wsdl) มีบริการ CurrentOilPrice, GetOilPrice, CurrentNews, GetNews รวม 4 บริการ ผลรวมของ URL และ URN คือ URI (Uniform Resource Identifier)

13.3 WSDL

การใช้บริการต่างๆ ของเว็บเซอร์วิส ผู้พัฒนาจะต้องเรียกใช้และรับส่งข้อมูลให้ตรงกับความต้องการของผู้ให้บริการ ดังนั้น จึงเป็นหน้าที่ของผู้ให้บริการ ที่จะต้องอธิบายว่าตัวเองมีบริการอะไรบ้าง จะเรียกใช้ได้อย่างไร โดยเฉพาะจะต้องส่งข้อความขาเข้าอะไรบ้าง และจะได้รับข้อความอะไรกลับไปบ้าง เช่นกรมการกงศุล ให้บริการทำหนังสือเดินทาง ผู้จะใช้บริการจะต้องเตรียมข้อความขาเข้าคือบัตรประชาชนในรูปแบบของเอกสารต้นฉบับ และค่าธรรมเนียมในรูปแบบของธนบัตรจำนวน 1,000 บาท ส่วนข้อความขาออกคือหนังสือเดินทางในรูปแบบเอกสารราชการ ซึ่งทางกรมฯ จะต้องจัดทำเอกสารหรือวิธีการอย่างหนึ่งอย่างใดเพื่อให้ผู้ที่จะใช้บริการสามารถจัดเตรียมข้อความขาเข้า และขาออกได้อย่างถูกต้อง

กรณีเว็บเซอร์วิสแบบ REST ไม่มีข้อกำหนดว่าจะต้องอธิบายอย่างไร ผู้ให้บริการสามารถจัดทำได้อิสระ โดยมากมักจัดทำเป็นเว็บอธิบายวิธีการใช้งานพร้อมตัวอย่างการพัฒนาโปรแกรมในภาษาต่างๆ ผู้ให้บริการสามารถใช้สื่อผสมเช่นภาพเคลื่อนไหว หรือดาวน์โหลดโปรแกรมตัวอย่าง หรือ template ก็ได้

แต่ในกรณีของ SOAP มีวัตถุประสงค์ที่จะใช้กระบวนการอัตโนมัติในการเชื่อมต่อบริการเข้าด้วยกัน จึงสร้างภาษามาตรฐานในการอธิบายเว็บเซอร์วิสอยู่ในรูปแบบของ XML เช่นเดียวกับ SOAP โดยภาษาที่ใช้ในการอธิบายเว็บเซอร์วิส เรียกว่า WSDL (Web Services Description Language) การอธิบายความหลักๆ ของ WSDL ก็คืออธิบายว่าบริการอยู่ที่ไหน (URL) มีบริการอะไรบ้าง และในการเรียกใช้มีข้อความขาเข้า และขาออกอะไรบ้าง โครงสร้างของ WSDL จะมีแท็ก `<definition>` เป็นแท็กราก (root tag) และมีแท็กหลักๆ ภายในอยู่ 4 แท็ก ดังต่อไปนี้

```
<definitions xmlns="http://schemas.xmlsoap.org/wsdl/">
<message>...</message>
<portType>...</portType>
<binding>...</binding>
<service>...</service>
</definitions>
```

WSDL นั้นถูกออกแบบให้คอมพิวเตอร์เป็นตัวประมวลผล แต่เนื่องจากความล้มเหลวของการเชื่อมต่อบริการแบบอัตโนมัติของระบบขั้นสูงขึ้นไป ทำให้ผู้พัฒนาที่จะเรียกใช้บริการ จำเป็นต้องทำความเข้าใจกับ WSDL ด้วยตัวเอง ปัญหาคือคอมพิวเตอร์ มีความจำที่ดีกว่ามนุษย์ และต้องการคำอธิบายจากรายละเอียดไปหาภาพรวมตามลำดับ จึงต้องเรียง `message`, `portType`, `bining`, และ `service` แต่สำหรับมนุษย์จะมองภาพรวมก่อน แล้วจึงเจาะลงไปหารายละเอียด จึงแนะนำให้ผู้เริ่มต้น อ่าน WSDL จากท้ายไปหน้าจะเข้าใจได้ดีกว่า คือจาก `service` จะอธิบายว่าบริการนี้อยู่ที่ไหน ไปหา `binding` อธิบายว่าจะเชื่อมต่อได้อย่างไร จากนั้น `portType` จะอธิบายว่ามีบริการอะไรบ้าง ต้องการข้อความขาเข้า และขาออกเป็นอย่างไร และบนสุด `message` จะอธิบายรายละเอียดของข้อความต่างๆ ที่ใช้ในบริการทั้งหมด

13.3.1 definitions

แท็กรากของ WSDL ใช้ในการอ้างอิงถึง `namespace` ต่างๆ ที่จะใช้ในระบบ ตัวอย่างที่นำมาแสดงในที่นี้เป็นบริการแจ้งอุณหภูมิตามรหัสไปรษณีย์

```
    <definitions name="Temperature"
targetNamespace="http://example.org/temperature"
xmlns:tns="http://example.org/temperature"
xmlns:electric="http://www.themindelectric.com/"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/" xmlns:wsdl="http://schemas.xmlsoap.org/
wsdl/" xmlns="http://schemas.xmlsoap.org/wsdl/">
```

13.3.2 message

อธิบายชนิดของข้อความต่างๆ ทั้งหมด ที่จะใช้ในเว็บเซอร์วิส ในตัวอย่างนี้ มีข้อความที่จะใช้อยู่ 2 แบบ แบบแรกเรียกว่า `getTempRequest` ชื่อ `Zip` เป็นชนิดตัวอักษร (string) ในส่วนนี้ผู้ใช้บริการจะต้องส่งรหัสไปรษณีย์ของบริเวณที่ต้องการทราบอุณหภูมิไปให้ระบบ

แบบที่ 2 เรียกว่า `getTempResponse` ชื่อ `Result` เป็นชนิดตัวเลขทศนิยม (float) แสดงผลลัพธ์ คืออุณหภูมิในบริเวณดังกล่าวที่ต้องการ

```
<message name='getTempRequest'>
  <part name='Zip' type='xsd:string'/>
</message>
<message name='getTempResponse'>
  <part name='Result' type='xsd:float'/>
</message>
```

13.3.3 portType

อธิบายว่าเว็บเซอร์วิสนี้มีบริการอะไรบ้าง มีข้อความขาเข้า และขาออกเป็นอะไร `portType` เป็นแท็กที่จะอธิบายภาพรวมได้ค่อนข้างชัดเจนที่สุด ในตัวอย่างนี้แสดงว่าบริการทั้งหมดมีอยู่ 1 บริการ ชื่อ `getTemp` โดยต้องการข้อความขาเข้าคือ `getTempRequest` และจะส่งข้อความตอบกลับในรูปแบบ `getTempResponse` รหัสขึ้นต้น `tns` ถูกกำหนดไว้ให้หมายถึงเป็นข้อความที่กำหนดในไฟล์เดียวกันนี้ (target namespace) ตามที่ระบุไว้ในแท็ก `definitions` ส่วนรายละเอียดของ ข้อความ `getTempRequest` และ `getTempResponse` จะถูกอธิบายไว้ในแท็ก `<message>` ก่อนหน้านี้

```
<portType name='TempPortType'>
  <operation name='getTemp'>
    <input message='tns:getTempRequest'/>
    <output message='tns:getTempResponse'/>
  </operation>
</portType>
```

13.3.4 binding

ส่วนของ binding แสดงการเชื่อมต่อระหว่างบริการต่างๆ เข้ากับเครือข่าย เช่นการเรียกใช้บริการ ใช้โปรโตคอลอะไร เนื่องจาก SOAP ไม่จำเป็นจะต้องใช้ HTTP เท่านั้น นอกจากนั้นในส่วนของการอธิบายข้อความขาเข้า และขาออก (input & output) ก็เปิดให้สามารถที่จะกำหนด encoding อื่นได้

```
<binding name='TempBinding' type='tns:TempPortType'>
  <soap:binding style='rpc'
   transport='http://schemas.xmlsoap.org/soap/http'/>
  <operation name='getTemp'>
      <soap:operation soapAction='urn:localhost-
temperature#getTemp'/>
    <input>
     <soap:body use='encoded' namespace='urn:localhost-
temperature'
       encodingStyle='http://schemas.xmlsoap.org/soap/
encoding/'/>
    </input>
    <output>
     <soap:body use='encoded' namespace='urn:localhost-
temperature'
       encodingStyle='http://schemas.xmlsoap.org/soap/
encoding/'/>
    </output>
  </operation>
</binding>
```

13.3.5 service

แท็กนี้จะเป็นตัวบอกว่าเครื่องแม่ข่ายที่ให้บริการนี้อยู่ที่ไหน (URL) เช่น ในตัวอย่างนี้เราเขียนโปรแกรมเป็น php ไว้ที่ http://192.168.0.20/gdi/soapServer.php ก็ระบุดังนี้

```
    <service name='TemperatureService'>
      <port name='TempPort' binding='TempBinding'>
        <soap:address location='http://192.168.0.20/
gdi/soapServer.php'/>
      </port>
    </service>
```

Workshop 13.2

1. พิจารณา WSDL จาก http://api.google.com/GoogleSearch.wsdl ซึ่งเป็น WSDL ระบบค้นหาของ Google แล้วตอบคำถามต่อไปนี้

   1.1. บริการนี้อยู่ที่ไหน (URL)

   1.2. มีกี่บริการ อะไรบ้าง

   1.3. คำที่ต้องการค้น อยู่ในตัวแปรนำเข้าชื่ออะไร

อธิบาย WSDL ของ ปตท. จาก http://www.pttplc.com/pttinfo.asmx?wsdl

13.4 SOAP Consumer

ในหัวข้อนี้ เราจะมาพัฒนาโปรแกรมที่มีการเรียกใช้เว็บเซอร์วิสในชีวิตจริงกันบ้าง ตัวอย่างนี้เป็นการเรียกใช้เว็บเซอร์วิสของ Google ในการค้นหาเว็บไซต์ที่มีข้อความที่เราต้องการ รายละเอียดของบริการค้นหาของ Google สามารถหาได้จากเว็บไซต์

```
http://code.google.com/apis/soapsearch/
```

สำหรับคำอธิบายบริการในรูปแบบ WSDL จะอยู่ที่เว็บไซต์

```
http://api.google.com/GoogleSearch.wsdl
```

ซึ่งมีรายละเอียดดังนี้

```
<!— WSDL description of the Google Web APIs.
       The Google Web APIs are in beta release. All
interfaces are subject to change as we refine and extend
our APIs. Please see the terms of use for more information.
—>

<!— Revision 2002-08-16 —>

<definitions name="GoogleSearch"
             targetNamespace="urn:GoogleSearch"
             xmlns:typens="urn:GoogleSearch"
                   xmlns:xsd="http://www.w3.org/2001/
XMLSchema"
               xmlns:soap="http://schemas.xmlsoap.org/
wsdl/soap/"
            xmlns:soapenc="http://schemas.xmlsoap.org/
soap/encoding/"
               xmlns:wsdl="http://schemas.xmlsoap.org/
wsdl/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

   <!— Types for search - result elements, directory
categories —>

  <types>
   <xsd:schema xmlns="http://www.w3.org/2001/XMLSchema"
               targetNamespace="urn:GoogleSearch">

      <xsd:complexType name="GoogleSearchResult">
        <xsd:all>
              <xsd:element  name="documentFiltering"
type="xsd:boolean"/>
```

```
                <xsd:element  name="searchComments"
type="xsd:string"/>
         <xsd:element name="estimatedTotalResultsCount"
type="xsd:int"/>
                <xsd:element  name="estimateIsExact"
type="xsd:boolean"/>
                <xsd:element  name="resultElements"
type="typens:ResultElementArray"/>
                 <xsd:element  name="searchQuery"
type="xsd:string"/>
                 <xsd:element  name="startIndex"
type="xsd:int"/>
                   <xsd:element  name="endIndex"
type="xsd:int"/>
                 <xsd:element  name="searchTips"
type="xsd:string"/>
                <xsd:element  name="directoryCategories"
type="typens:DirectoryCategoryArray"/>
                 <xsd:element  name="searchTime"
type="xsd:double"/>
         </xsd:all>
      </xsd:complexType>

      <xsd:complexType name="ResultElement">
        <xsd:all>
         <xsd:element name="summary" type="xsd:string"/
>
          <xsd:element name="URL" type="xsd:string"/>
         <xsd:element name="snippet" type="xsd:string"/
>
          <xsd:element name="title" type="xsd:string"/>
                 <xsd:element  name="cachedSize"
type="xsd:string"/>
```

```
          <xsd:element name="relatedInformationPresent"
type="xsd:boolean"/>
        <xsd:element name="hostName" type="xsd:string"/
>
                <xsd:element  name="directoryCategory"
type="typens:DirectoryCategory"/>
                 <xsd:element  name="directoryTitle"
type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>

      <xsd:complexType name="ResultElementArray">
        <xsd:complexContent>
          <xsd:restriction base="soapenc:Array">
                 <xsd:attribute  ref="soapenc:arrayType"
wsdl:arrayType="typens:ResultElement[]"/>
          </xsd:restriction>
        </xsd:complexContent>
      </xsd:complexType>

      <xsd:complexType name="DirectoryCategoryArray">
        <xsd:complexContent>
          <xsd:restriction base="soapenc:Array">
                 <xsd:attribute  ref="soapenc:arrayType"
wsdl:arrayType="typens:DirectoryCategory[]"/>
          </xsd:restriction>
        </xsd:complexContent>
      </xsd:complexType>

      <xsd:complexType name="DirectoryCategory">
        <xsd:all>
                <xsd:element  name="fullViewableName"
type="xsd:string"/>
```

```
                <xsd:element  name="specialEncoding"
type="xsd:string"/>
        </xsd:all>
      </xsd:complexType>

    </xsd:schema>
  </types>

  <!— Messages for Google Web APIs - cached page, search,
spelling. —>

  <message name="doGetCachedPage">
    <part name="key"             type="xsd:string"/>
    <part name="url"             type="xsd:string"/>
  </message>

  <message name="doGetCachedPageResponse">
    <part name="return"        type="xsd:base64Binary"/
>
  </message>

  <message name="doSpellingSuggestion">
    <part name="key"             type="xsd:string"/>
    <part name="phrase"          type="xsd:string"/>
  </message>

  <message name="doSpellingSuggestionResponse">
    <part name="return"          type="xsd:string"/>
  </message>

  <!— note, ie and oe are ignored by server; all traffic
is UTF-8. —>
```

```
  <message name="doGoogleSearch">
    <part name="key"            type="xsd:string"/>
    <part name="q"              type="xsd:string"/>
    <part name="start"          type="xsd:int"/>
    <part name="maxResults"     type="xsd:int"/>
    <part name="filter"         type="xsd:boolean"/>
    <part name="restrict"       type="xsd:string"/>
    <part name="safeSearch"     type="xsd:boolean"/>
    <part name="lr"             type="xsd:string"/>
    <part name="ie"             type="xsd:string"/>
    <part name="oe"             type="xsd:string"/>
  </message>

  <message name="doGoogleSearchResponse">
                          <part      name="return"
type="typens:GoogleSearchResult"/>
  </message>

  <!— Port for Google Web APIs, "GoogleSearch" —>

  <portType name="GoogleSearchPort">

    <operation name="doGetCachedPage">
      <input message="typens:doGetCachedPage"/>
     <output message="typens:doGetCachedPageResponse"/
>
    </operation>

    <operation name="doSpellingSuggestion">
      <input message="typens:doSpellingSuggestion"/>
                                          <output
message="typens:doSpellingSuggestionResponse"/>
    </operation>
```

```
    <operation name="doGoogleSearch">
      <input message="typens:doGoogleSearch"/>
      <output message="typens:doGoogleSearchResponse"/
>
    </operation>

  </portType>


  <!— Binding for Google Web APIs - RPC, SOAP over HTTP
—>

       <binding    name="GoogleSearchBinding"
type="typens:GoogleSearchPort">
    <soap:binding style="rpc"
                                transport="http://
schemas.xmlsoap.org/soap/http"/>

    <operation name="doGetCachedPage">
                                      <soap:operation
soapAction="urn:GoogleSearchAction"/>
      <input>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                              encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                              encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/"/>
      </output>
```

```
    </operation>

    <operation name="doSpellingSuggestion">
                                        <soap:operation
soapAction="urn:GoogleSearchAction"/>
      <input>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                                encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                                encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>

    <operation name="doGoogleSearch">
                                        <soap:operation
soapAction="urn:GoogleSearchAction"/>
      <input>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                                encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
                   namespace="urn:GoogleSearch"
                                encodingStyle="http://
schemas.xmlsoap.org/soap/encoding/"/>
```

```
      </output>
    </operation>
  </binding>

  <!– Endpoint for Google Web APIs –>
  <service name="GoogleSearchService">
              <port    name="GoogleSearchPort"
binding="typens:GoogleSearchBinding">
       <soap:address location="http://api.google.com/
search/beta2"/>
    </port>
  </service>

</definitions>
```

ในการเรียกใช้บริการนี้จะต้องขึ้นทะเบียนกับ Google ก่อน ซึ่งจะได้รหัสอนุญาตมาสำหรับตัวอย่างซึ่งทำงานอยู่ที่ http://localhost:3000 สามารถใช้รหัสอนุญาตได้ดังนี้

```
ecDc6bFQFHLXPKTEO0McWIoE3LVRSEw2
```

13.5 การพัฒนาโปรแกรม Ruby on Rails เพื่อใช้งานเว็บเซอร์วิสแบบ SOAP

ในส่วนนี้ เราจะทำการพัฒนาโปรแกรมเพื่อใช้งานบริการเว็บเซอร์วิสของผู้อื่นที่มีอยู่แล้ว โดยเราจะแบ่งเป็น 2 ส่วนคือส่วนการทำงานด้วย Controller และส่วนการแสดงผลใน View ในส่วนของ Controller เราจะทำเป็น 2 รูปแบบ โดยรูปแบบแรกกรณีที่เราทราบว่าเว็บเซอร์วิสอยู่ที่ไหน (URL) และเราต้องการใช้บริการอะไร (URN) ส่วนในรูปแบบที่ 2 เราจะอ้างอิงโดยใช้ WSDL

13.5.1 Controller

การเขียนโปรแกรมเว็บเซอร์วิสโดยใช้ SOAP ในทางทฤษฎีแล้วก็คือการสร้างข้อความในรูปแบบ XML ส่งไปให้เครื่องผู้ให้บริการ เมื่อได้รับข้อความตอบกลับมาก็จะอยู่ในรูป XML เช่นเดียวกัน เมื่อเราถอดรหัสตามรูปแบบที่กำหนดใน WSDL ก็จะได้ข้อความที่ตอบกลับมา แต่การเขียน XML ในรูปแบบของ SOAP โดยตรงนั้นค่อนข้างซับซ้อนมากกว่า REST ภาษา Ruby จึงมีการสร้างโมดูล SOAP ขึ้นมาช่วยในการแปลงรูปแบบทั้งขาไปและขากลับ

สำหรับ Controller ในรูปแบบของ URI เราจะใช้คลาส Driver ในโมดูล SOAP::RPC เพื่อสร้าง Instance สำหรับ URI ซึ่งประกอบไปด้วย URL คือ http://api.google.com/search/beta2 และ URN คือ GoogleSearch จากนั้นเพิ่มเมธอดที่เราต้องการเข้าไปด้วยเมธอด `add_method` สำหรับผู้ที่พิมพ์ตามตัวอย่างนี้ อย่าลืมแก้ไขตัวแปร `yourkey` และ `@yourquery` ให้เป็นค่าที่ถูกต้องก่อน

```
    class CodeController < ApplicationController
      def googletest
        yourkey = 'YOUR GOOGLE DEVELOPER KEY'
        @yourquery = 'SEARCH TEXT'
        XSD::Charset.encoding = 'UTF8'
         googleurl = "http://api.google.com/search/
beta2"
        urn = "urn:GoogleSearch"
       driver = SOAP::RPC::Driver.new(googleurl, urn)
         driver.add_method('doGoogleSearch', 'key',
'q', 'start',
              'maxResults', 'filter', 'restrict',
'safeSearch', 'lr', 'ie', 'oe')
          @result = driver.doGoogleSearch(yourkey,
@yourquery, 0, 3, false, '', false, '', '', '')
      end
    end
```

13.5.2 View

เมื่อทำงานในฝั่ง Controller เสร็จแล้ว เราก็จะได้ผลลัพธ์ของการค้นหามาเก็บไว้ที่ตัวแปร `@result` ในรูปแบบของ `doGoogleSearchResponse`

```
    Query for: <%= @yourquery %><br>
    Found: <%= @result.estimatedTotalResultsCount
%><br>
    Query took about <%= @result.searchTime %>
seconds<br>
    <% @result.resultElements.each do |rec| %>
      <b>Title:</b> <%= rec["title"] %><br>
      <b>Summary:</b> <%= rec.snippet %><br>
      <b>Link:</b> <a href="<%= rec["URL"] %>"><%=
rec["URL"] %></a> <br><br>
    <% end %>
```

13.5.3 สร้างวัตถุโดยใช้ WSDL

อีกรูปแบบหนึ่ง เราสามารถสร้างวัตถุ WSDLDriverFactory ได้โดยตรงจาก WSDL ในตัวอย่างนี้ผู้ใช้เปลี่ยนแปลงเฉพาะการทำงานในฝั่ง Controller ส่วนในฝั่ง View นั้นยังสามารถใช้ของเดิมได้

```
    require 'soap/wsdlDriver'
    class CodeController < ApplicationController
      def googletest
        yourkey = 'YOUR GOOGLE DEVELOPER KEY'
        @yourquery = 'SEARCH TEXT'
        XSD::Charset.encoding = 'UTF8'
            wsdl = "http://api.google.com/
GoogleSearch.wsdl"
        driver =
         SOAP::WSDLDriverFactory.new(wsdl).create_rpc_driver
          @result = driver.doGoogleSearch(yourkey,
@yourquery,
            0, 3, false, '', false, '', '')
      end
    end
```

Workshop 13.3

1. เขียนโปรแกรมเพื่ออ่านข้อมูลราคาน้ำมันจากเว็บเซอร์วิสของ ปตท. ตาม WSDL ดังต่อไปนี้

http://www.pttplc.com/pttinfo.asmx?wsdl

2. วิเคราะห์ปัญหาจากการใช้ SOAP

13.6 การให้บริการเว็บเซอร์วิสแบบ SOAP

ในส่วนของผู้ให้บริการ นอกเหนือจากการรับส่งข้อความ XML ในรูปแบบของ SOAP แล้ว จะต้องสร้าง WSDL เพื่อให้ผู้ใช้บริการเข้าใจและรับส่งข้อมูลตามที่ระบบต้องการได้ถูกต้อง ใน Ruby on Rails (RoR) เคยมี gem `ActionWebService` ในรุ่น 1 แต่หลังจากรุ่น 2 มา RoR ไม่สนับสนุนการใช้งานในแบบ SOAP ก็เลยแยกออกมาเป็น gem อิสระ และยุติการพัฒนาไป แต่หลังจาก RoR 2.2.2 มีบริษัท datanoise ได้นำมาพัฒนาต่อให้สามารถใช้กับ RoR รุ่นใหม่ๆ ได้ วิธีการติดตั้ง gem `ActionWebService` ขั้นที่ 1 คล้ายกับ gem ทั่วไป คือ ใช้คำสั่ง `gem install` แต่ขั้นที่ 2 ต้องสั่ง `ruby setup.rb` เพื่อให้ `ActionWebService` ติดตั้งองค์ประกอบต่างๆ ของ Ruby ในเครื่องเราให้พร้อมก่อนใช้งาน

```
sudo gem install datanoise-actionwebservice —source http://gems.github.com
cd ไปทีติดตั้ง AWS gem, มองหาไฟล์ setup.rb
ruby setup.rb
```

13.6.1 Generate Web Service

ขั้นแรกทำการ `generate web_service` ขึ้นมาก่อน คล้ายกับการสร้าง controller แต่การสร้าง `web_service` จะเป็นการสร้างไฟล์หลักขึ้นมา 2 ไฟล์ เช่น ตามตัวอย่างเมื่อสร้างเว็บเซอร์วิสชื่อ test_ws ก็จะได้ไฟล์ `test_ws_api.rb` และ `test_ws_controller.rb`

```
script/generate web_service test_ws
```

test_ws_api.rb

ในส่วนของไฟล์ api เป็นการสร้างคลาสซึ่งเกิดมาจากคลาส `ActionWebService::API::Base` เป็นส่วนสำคัญที่จะช่วยในการสร้าง API เนื่องจากภาษา Ruby เป็นภาษาแบบพลวัตร (Dyamic) คือ การใช้งานตัวแปรต่างๆสามารถใช้ได้ทันที แตกต่างจากการรับส่งข้อมูลแบบ SOAP ซึ่งจะต้องมีการกำหนดชนิดของตัวแปรที่รับส่งอย่างแน่ชัด และใช้งานได้ตามที่กำหนดเท่านั้น การสร้างไฟล์ api ขึ้นมาก็เพื่อจำลองให้เมธอดที่เราสร้างขึ้นมามีการระบุชนิดของตัวแปรให้เป็นที่แน่ชัดตามที่ SOAP ต้องการนั่นเอง ดังนั้น การโปรแกรมในส่วนของ api ก็คือการประกาศรูปแบบของเมธอดที่เราจะสร้างขึ้นเท่านั้น รายละเอียดการทำงานจะไปอยู่ในส่วนของ controller เหมือนกับการสร้างเมธอดปรกติ

```
class TestWsApi < ActionWebService::API::Base
  api_method :dogreeting,
    :expects => [{:username => :string}],
    :returns => [{:greeting => :string}]
end
```

ในตัวอย่างนี้ เราจะสร้างเมธอดชื่อ `dogreeting` เราก็มาประกาศว่าเมธอดนี้ ซึ่งจะกลายไปเป็น operation หนึ่งในเว็บเซอร์วิสของเรา ต้องการข้อความขาเข้าเป็นตัวแปรหนึ่ง ชื่อ `username` ซึ่งเป็นชนิดตัวอักษร และจะให้คำตอบเป็นตัวแปรชื่อ `greeting` ชนิดตัวอักษรเช่นเดียวกัน

```
test_ws_controller.rb
```

ในส่วนของ controller เราก็ทำการสร้างเมธอดตามปรกติ ไม่ต้องคำนึงถึงว่าเมธอดนี้จะเป็นเว็บเซอร์วิสหรือไม่ ในการสร้าง WSDL เพียงใช้เมธอด wsdl_service_name ระบบก็จะสร้าง WSDL ขึ้นมาให้โดยอัตโนมัติอยู่ที่ http://localhost:3000/test_ws/wsdl

```
class TestWsController < ApplicationController
  wsdl_service_name 'TestWs'

  def dogreeting(username)
    "Hello #{username}"
  end
end
```

WSDL ที่ระบบสร้างขึ้นมีลักษณะเป็นดังนี้

```
    <definitions name="TestWs"
xmlns:typens="urn:ActionWebService"
xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
targetNamespace="urn:ActionWebService"
xmlns:xsd="http://www.w3.org/2001/XMLSchema"
xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
xmlns:soapenc="http://schemas.xmlsoap.org/soap/
encoding/" xmlns="http://schemas.xmlsoap.org/wsdl/">
      <message name="Dogreeting">
        <part name="username" type="xsd:string"/>
      </message>
      <message name="DogreetingResponse">
        <part name="return" type="xsd:string"/>
      </message>
      <portType name="TestWsTestWsPort">
        <operation name="Dogreeting">
          <input message="typens:Dogreeting"/>
          <output
message="typens:DogreetingResponse"/>
        </operation>
```

```
      </portType>
      <binding name="TestWsTestWsBinding"
type="typens:TestWsTestWsPort">
        <soap:binding transport="http://
schemas.xmlsoap.org/soap/http" style="rpc"/>
        <operation name="Dogreeting">
          <soap:operation soapAction="/test_ws/api/
Dogreeting"/>
          <input>
            <soap:body
namespace="urn:ActionWebService" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
          </input>
          <output>
            <soap:body
namespace="urn:ActionWebService" use="encoded"
encodingStyle="http://schemas.xmlsoap.org/soap/
encoding/"/>
          </output>
        </operation>
      </binding>
      <service name="TestWsService">
        <port name="TestWsTestWsPort"
binding="typens:TestWsTestWsBinding">
          <soap:address location="http://
localhost:3000/test_ws/api"/>
        </port>
      </service>
    </definitions>
```

13.6.2 Consumption Test (on different port)

ในขั้นตอนนี้ เราจะเขียนโปรแกรมขึ้นมาทดสอบเว็บเซอร์วิสที่เราสร้างขึ้น ในการทดสอบ เราจะต้องสร้างระบบงานขึ้นมาใหม่ทำงานอยู่บนพอร์ตอื่น เนื่องจากถ้าเป็นการเรียกในพอร์ตเดียวกันก็จะเป็นการเรียกใช้เว็บเซอร์เวอร์ตัวเดียวกัน เมื่อผู้ใช้ส่งคำร้องไปยังผู้ให้บริการ ถือเป็นคำสั่งที่ 1 ส่วนกระบวนการที่ผู้ให้บริการทำงานแล้วส่งกลับ ถือเป็นคำสั่งที่ 2 เมื่อเครื่องแม่ข่ายเห็นคำสั่งที่ 1 ค้างอยู่ก็จะรอจนกว่าจะเสร็จ แต่คำสั่งที่ 1 ก็ต้องรอให้คำสั่งที่ 2 ตอบกลับมา หากเครื่องแม่ข่ายไม่ยอมเปิดให้คำสั่งที่ 2 เริ่มทำงาน คำสั่งที่ 1 ก็จะรอไปเรื่อยๆ จนหมดเวลา (Time out)

```
def test_soap
  require 'soap/wsdlDriver'
  XSD::Charset.encoding = 'UTF8'
  wsdl = "http://localhost:3999/test_ws/wsdl"
  driver = SOAP::WSDLDriverFactory.
    new(wsdl).create_rpc_driver
  @result = driver.dogreeting('songrit')
  render :text => @result
end
```

13.6.3 More Parameters

ในกรณีที่เรามีเมธอดมากขึ้น หรือมีตัวแปรมากขึ้น เราก็สามารถเพิ่มเติมเข้าไปที่ api ได้ในลักษณะเดียวกัน

```
class PersonAPI < ActionWebService::API::Base
  api_method :add,
    :expects => [:string, :string, :bool],
    :returns => [:int]
  api_method :remove, :expects => [:int],
:returns => [:bool]
end
```

จากตัวอย่างมี 2 เมธอด คือ `add` และ `remove` ซึ่งจะถูกกำหนดไว้ใน `person_controller.rb` เราสามารถที่จะกำหนดเมธอดอื่นๆใน `person_controller.rb` ก็ได้ ในส่วนของ api เรานำมาประกาศเฉพาะเมธอดที่ต้องการเปิดให้ระบบภายนอกมาใช้เท่านั้น จากตัวอย่างนี้ เราไม่ได้ตั้งชื่อตัวแปรในการรับส่ง แต่เราจะต้องระบุชนิดของตัวแปร

Workshop 13.4

ให้สร้างเว็บเซอร์วิสแบบ SOAP เพื่อแจ้งราคาหลักทรัพย์โดยใช้ข้อมูลจากเว็บไซต์

```
http://marketdata.set.or.th/mkt/stockquotation.do?
symbol=pttep
```

13.6.4 Complex Structure & Array

ในกรณีที่ข้อความการรับส่งของบริการอยู่ในรูปแบบที่ซับซ้อนมากขึ้น เช่นเป็น Array หรือ Hash เราสามารถใช้คลาส `ActionWebService::Struct` เข้ามาช่วยในการประกาศโครงสร้างของ Hash ส่วนการประกาศ Array เพียงแค่ใช้เครื่องหมาย `[]` ครอบบริเวณที่ต้องการก็เป็นอันเรียบร้อย

ตัวอย่างนี้เป็นการประกาศโครงสร้างของบริการค้นหาของ Google ในไฟล์ api ลองพิจารณาเปรียบเทียบกับ WSDL ที่เป็นผลลัพธ์จาก http://api.google.com/GoogleSearch.wsdl

```
class DirectoryCategory < ActionWebService::Struct
  member :fullViewableName, :string
  member :specialEncoding, :string
end

class ResultElement < ActionWebService::Struct
  member :summary, :string
  member :URL, :string
  member :snippet, :string
```

```
  member :title, :string
  member :cachedSize, :string
  member :relatedInformationPresent, :bool
  member :hostName, :string
  member :directoryCategory, DirectoryCategory
  member :directoryTitle, :string
end

class GoogleSearchResult < ActionWebService::Struct
  member :documentFiltering, :bool
  member :searchComments, :string
  member :estimatedTotalResultsCount, :int
  member :estimateIsExact, :bool
  member :resultElements, [ResultElement]
  member :searchQuery, :string
  member :startIndex, :int
  member :endIndex, :int
  member :searchTips, :string
  member :directoryCategories, [DirectoryCategory]
  member :searchTime, :float
end

class GoogleSearchAPI < ActionWebService::API::Base
  inflect_names false

  api_method :doGetCachedPage, :returns => [:string],
:expects => [{:key=>:string}, {:url=>:string}]
  api_method :doGetSpellingSuggestion, :returns =>
[:string], :expects => [{:key=>:string},
{:phrase=>:string}]
```

```
  api_method :doGoogleSearch, :returns =>
[GoogleSearchResult], :expects => [
    {:key=>:string},
    {:q=>:string},
    {:start=>:int},
    {:maxResults=>:int},
    {:filter=>:bool},
    {:restrict=>:string},
    {:safeSearch=>:bool},
    {:lr=>:string},
    {:ie=>:string},
    {:oe=>:string}
  ]
end
```

เมื่อศึกษามาถึงบทนี้ ผู้เรียนจะสามารถพัฒนาโปรแกรมเพื่อให้บริการเว็บเซอร์วิสแบบ REST และ SOAP ได้แล้ว การเลือกใช้วิธีใดควรคำนึงถึงความสะดวกของผู้รับบริการด้วย