REBUILDING THE SEMANTIC WEB SERVICE ARCHITECTURE

Xuan Shi[1] and Vasudevan Jagannathan[2]
[1]Department of Geology and Geography, West Virginia University; [2]Department of Computer Science and Electrical Engineering, West Virginia University

Abstract: Semantic Web services can be defined as "the augmentation of Web Service descriptions through Semantic Web annotations, to facilitate the higher automation of service discovery, composition, invocation, and monitoring in an open, unregulated, and often chaotic environment" (Payne, et al., 2004). Web service infrastructure (WSDL, UDDI and SOAP) has been criticized in almost all papers on the topic of "Semantic Web Services", typically as "existing technologies for Web services only provide descriptions at the syntactic level, making it difficult for requesters and providers to interpret or represent non-trivial statements such as the meaning of inputs and outputs or applicable constraints" (Cabral, et al., 2004). Adding rich semantics into Web Services are expected to "support greater automation of service selection and invocation, automated translation of message content between heterogeneous interoperating services, automated or semi-automated approaches to service composition, and more comprehensive approaches to service monitoring and recovery from failure" (Martin, D., et al., 2004). Current approaches for Semantic Web Service infrastructures are developed by a variety of research groups, among those efforts, OWL-S (Martin, D., et al., 2004), WSMO (Roman, et al., 2004), etc. were the most recognized and outstanding achievement to date in this field. However, how to derive service semantics automatically has been problematic with the current syntactically oriented Web services technologies.

This paper proposes a fundamentally different approach to rebuild the semantic Web Services architecture. The semantic web service generated by this proposed new approach is:

- Self-describing.
- Document based rather than object based.
- Meaning or semantics oriented rather than name or syntax oriented.
- Service domain and functional purpose independent.
- Human semantics and behavior simulated.
- Standardized Web service with ONE interface with proven efficiency.
- Obviating the need for WSDL for semantic description.
- With simplified implementation for the discovery, matchmaking, composition and invocation of Web service within its current infrastructure (UDDI, WSDL, SOAP).

This proposal is being demonstrated by an initial pilot project. The goal is to support the dynamic and intelligent discovery, matchmaking, composition and invocation of semantic Web services.

Key words: Semantic request and response, self-describing, document-based, Web services, WSDL, service domain, ontology, standardization

1. THE PROBLEM OF SEMANTIC SERVICE DESCRIPTION

According to W3C (2004), WSDL defines service description about the message formats, data types, transport protocols, and transport serialization formats as a machine-processable specification of the Web service's interface. *The semantics of a Web service is the shared expectation about the behavior of the service, and, is the "contract" between the service requester and the service provider regarding the purpose and consequences of the interaction*. "While the service description represents a contract governing the mechanics of interacting with a particular service, the semantics represents a contract governing the meaning and purpose of that interaction". By design, "a service description is a machine-processable description of a service" that contains a machine-processable description of the messages that are exchanged by the service" and "*may include a description of the service's semantics*", while it has been expected that "*a service semantics is about* *the service tasks that constitute the service*" and "*should be identified in a service description*" and "*may be described in a formal, machine-processable language*".

Semantics of web services are tied to names given to functions and arguments in WSDL. They are not derived from any standardized ontology. This fact is a source for confusion, as meaning of the services has to be guessed. Some examples of problems drawn from the domain of geospatial applications are as follows:

1. Service semantics can be ambiguous when data is passed using XML/GML document, one of the common representation, for geospatial features. This is because a GML document can represent many different things from points to polylines to polygons. And, the representations in GML can have other complicating factors. Polygons for instance, can be geo-referenced (tied to a global referencing scheme). WSDL function names typically used in current day practice do not account for such interpretations.
2. Operational semantics are hard to specify using WSDL as the numbers of graphical operations that can be performed are innumerable. A complete listing of such operations will make WSDL very specific with limited use or extremely large or both.
3. Composition of web services is difficult, as the operational semantics are ambiguous.

A typical problem in the communication between Web service requester and provider can be described as follows: ***if you give me the object 1, 2, and 3 in order, I will invoke Function C and return you the new object***. Interpretation of such a contract is tied to the meaning of "object 1, 2 and 3 and Function C". To understand the meaning, the service requestors must decode and understand how the function treats the arguments by examining documentation and information not represented in the WSDL. This essentially means one cannot support automated discovery, composition and invocation.

One approach to the semantics problem relies on developing a completely different infrastructure that captures knowledge about the semantics of the service. OWL-S approach is an example of such an approach (Martin et. al., 2004). In this approach, ontologies are used to capture the computing semantics using object/class hierarchy. For instance, one can define a class of ***Buy*** with subclasses ***BuyTicket*** containing the subclasses ***BuyMovieTicket*** and ***BuyAirlineTicket*** (McIlraith, et. al., 2001).

This paper proposes an alternative approach to build semantic Web Services based on the following:

1) Web services interfaces are simplified to support XML-document exchange: *getService(String request): String response*.
2) Web service interface also needs to provide a mechanism to learn about the service offering. Service offering description is an ontologically-derived XML-document.
3) Web service invocation happens via the exchange of ontologically-derived documents of service request and response.

This approach is further elaborated below.

2. WEB SERVICE SIMPLIFICATION

Simplified Web services are envisioned as self-describing services. Communication between service requester and provider will actually be document exchange rather than functional invocation. Such process can be described as: ***the service requester sends the service provider one document with explicit and detailed service request and service provider will return another document back to the service requester with the generated result corresponding to the service request***.

For service requester to understand what the service can provide, service provider will ***first*** offer a template XML document that describes explicitly both the input variables to invoke the services and the output results. Service requester ***then*** get the XML document on service description by invoking the function, *getServiceDescription(): String ServiceDescription,* to retrieve the XML document. [Note: the *getServiceDescription()* method is strictly not necessary. This functionality can be subsumed by the method: *getService()* or as suggested below, the description can be made available directly]

In the implementation described here, an XML document titled *ServiceDescription.xml* was co-located with the web services server to improve performance. The document-based invocation of web services was shown to be useful for Web service discovery and matchmaking as illustrated by Table 1 and 2. By the time of writing this paper, 9 Web services were developed with exactly the same two WSDL interfaces except the location of the services. The XML document *ServiceDescription.xml* for each Web service is located at the same Web server directory with the service. To get the service description document for each Web service by invoking the function *getServiceDescription():String ServiceDescription*, it takes 761-771 milliseconds. By directly reading the XML document from its URL, however, it takes only 0-10 milliseconds, i.e. about 1% of the time used for functional invocation.

Table 1. Code segments for retrieving the content of Service Description (ASP.NET in VB)

Code segment for the dynamic invocation of Web service	**Code segment for reading XML file by URL at client side**
Dim strURL As String Dim strNameSpace As String Dim classname As String Dim ***methodname*** As String = "***getServiceDescription***" Dim strInput() As String …… Dim obj As Object = InvokeWebservice(strURL, strNameSpace, classname, methodname, strInput) Dim ***strServiceDescription*** as String = CType(obj, String)	Dim strURL As String Dim xdoc As New XmlDocument() Dim ***strServiceDescription*** As String …… xdoc.Load(strURL) ***strServiceDescription*** = xdoc.OuterXml

To test a matchmaking and classification on all 9 Web services, time used through functional invocation is about 14310-15032 milliseconds while time used by reading the same 9 XML files directly from the URLs is about 60-90 milliseconds, about 0.5% of the former one. The pilot project developed for this demonstration can be accessed at: http://157.182.136.76/AItest/ws/AIDemo1/WebForm3.aspx , which is shown by Figure 1.

Table 2. Time used to retrieve the service description by different methods

Item	**Comparison**	**Time (milliseconds)**	**%**
A	Time used to invoke one function	761-771	***0~1.3***
B	Time used to read one XML document by URL	0-10	**(B/A)**
C	Time used to invoke 9 functions	14310-15032	***0.4~0.6***
D	Time used to read 9 XML documents by URLs	60-90	**(D/C)**

Thus the proposed approach for semantic Web services is composed of two elements:

- One standardized WSDL interface *getService(String request): String response*
- An XML template, *ServiceDescription.xml,* that describes the details of the provided Web service

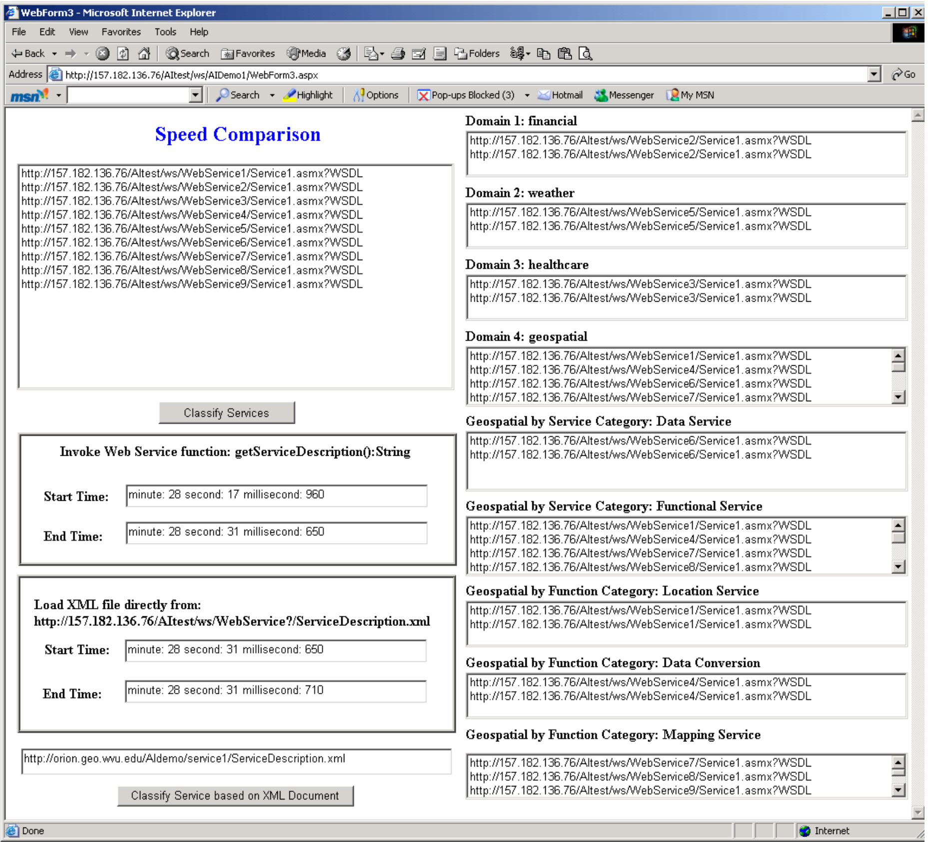

Figure 1. Speed comparison to retrieve the service description by different methods

Most importantly, this suggested approach will enable service provider to transform all composite Web services into atomic ones to remove one of the syntactic obstacles to build semantic Web services (Shi, 2004).

3. MESSAGE-ORIENTED SEMANTIC WEB SERVICES

Clearly with this approach, the web services interface is simply a conduit for exchanging XML-documents. The semantics, hence, has to be in the document exchanged. Ontology tools, such as Protégé and OWL, etc. can be used to specify such semantics.

The semantic service request and response relies on using ontology-derived documents and agent-based web services. The agent, here, uses its knowledge base to determine if it knows how to answer the request, or needs to invoke a series of other agent-based services to provide the answer. The solution framework can be recursive and relies on distributed ontologies and knowledge that is represented in each agent. As a result, no supplementary and separate ontology knowledge base and mechanism are needed to process the request and response to interpret the meaning of the exchanged message.

Interoperability issues have been discussed in GIS (Geographic Information System) community for decades. One big ***obstacle*** is who has the power to define the semantics and how to define service domains and functional categories, such as the service domain named as “Travel” or “Transportation”. In another example, “city” can have different meaning defined by different people and organizations. In the spatial domain, a city can be represented as a point feature in a map drawn at country or state level. The same city shows us as a polygon, in a large-scale map at the city level.

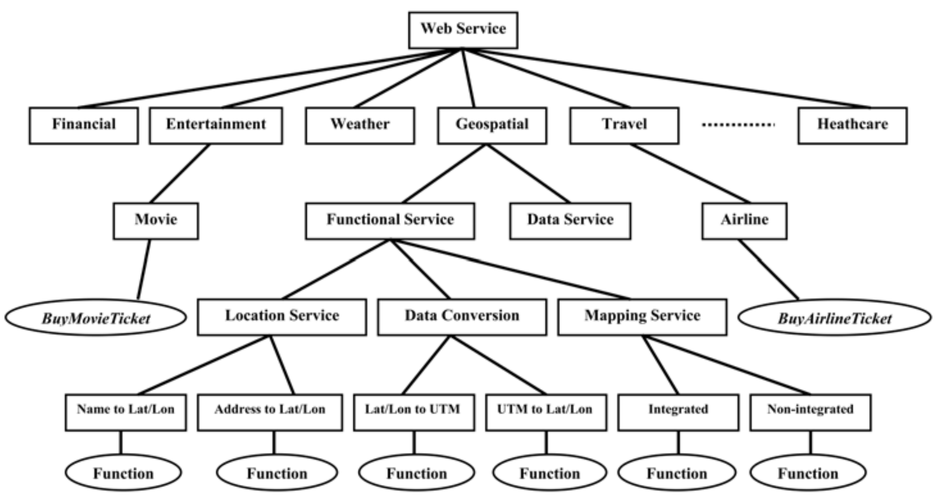

Figure 2. Hierarchy of service domain description

This paper only provides a simple simulation and classification on the service domains and will focus on the geospatial service as described in Figure 2. To accomplish such a huge task, computer scientists have to cooperate with specialists in all other domains who can contribute to build the domain specific service ontologies in order to integrate each section into a whole system.

Figure 3 is an example of the template *ServiceDescription.xml* file used in a geospatial data conversion web service based on the hierarchy described in Figure 2. Each template has 5 building blocks:

1. *Service domain and function category description*: defines both the service domain/subdomain and function category/subcategory within the tags **<*Service*>** and **</*Service*>**.
2. *Format of the service request input XML document*: defines the format and style of the input XML document within the tags **<*ServiceRequestXMLDocFormat*>** and **</*ServiceRequestXMLDocFormat*>.** It is recommended that service request be sent as a string data type.
3. *Format of the service response output XML document*: defines the format and style of the output XML document within the tags **<*ServiceResponseXMLDocFormat*>** and **</*ServiceResponseXMLDocFormat*>**. That is to say, service response returned to the service requester can be either a string of the service response or an URL refers to the XML document of the service response can be accepted. However, it is recommended that service response be sent back as a string data type to keep consistency in the development and communication.
4. *Service request input requirements*: defines the template for service request. The service request will consist of 2 elements to formulate a XML document: the XML declaration header, i.e. ***<?xml version="1.0" encoding="utf-8">*** *,* and the elements within the tags **<*ServiceRequest*>** and **</*ServiceRequest*>** as described in the **<*RequestTemplate*>** section of *ServiceDescription.xml* file.
5. *Service response output prototype*: defines the template for service response. The service response will consist of 2 elements to formulate a XML document: the XML declaration header, i.e. ***<?xml version="1.0" encoding="utf-8">*** *,* and the elements within the tags **<*ServiceResponse*>** and **</*ServiceResponse*>** as described in the **<*ResponseTemplate*>** section of *ServiceDescription.xml* file. The output result can be either a returned outcome of functional invocation or an error message.

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<WebService>
  <Service>
    <Domain>geospatial</Domain>
    <Category>functional service</Category>
    <Provider>minos</Provider>
```

```
        <Name>DataConversion</Name>
        <Description>
            This service provides function to convert geospatial data from one format to another.
        </Description>
        <Function>
          <Name>LL2UTM</Name>
          <Category>Data Conversion</Category>
          <Description>
            This function will convert longitude/latitude pair into UTM meters.
          </Description>
          <namespace>LL2UTM.DataConversion.minos.geo.wvu.edu</namespace>
        </Function>
    </Service>
    <ServiceRequestXMLDocFormat>
     <description>Required elements in service request</description>
     <Elements>
         <element>XML declaration</element>
         <element>ServiceRequest in RequestTemplate</element>
     </Elements>
     <ServiceRequestXMLDocType>XML document string</ServiceRequestXMLDocType>
    </ServiceRequestXMLDocFormat>
    <ServiceResponseXMLDocFormat>
     <description>Returned elements in service response</description>
     <Elements>
         <element>XML declaration</element>
         <element>ServiceResponse in ResponseTemplate</element>
     </Elements>
     <ServiceResponseXMLDocType>XML document string</ServiceResponseXMLDocType>
    </ServiceResponseXMLDocFormat>
    <RequestTemplate>
     <ServiceRequest>
         <Service Name="DataConversion">
             <Function Name="LL2UTM">
                 <InputVariables>
                     <FeatureType>Point</FeatureType>
                     <SourceData>
                         <SRS>EPSG:4326</SRS>
                         <coords multiElements = "false">
                             <longitude>longitude</longitude>
                             <latitude>latitude</latitude>
                         </coords>
                     </SourceData>
```

```
                <TargetData>
                    <SRS>EPSG:26917</SRS>
                </TargetData>
            </InputVariables>
        </Function>
    </Service>
</ServiceRequest>
</RequestTemplate>
<ResponseTemplate>
<ServiceResponse>
    <Service Name="DataConversion">
        <Function Name="LL2UTM">
            <InputVariables>
                <FeatureType>Point</FeatureType>
                <SourceData>
                    <SRS>EPSG:4326</SRS>
                    <coords multiElements = "false">
                        <longitude>longitude</longitude>
                        <latitude>latitude</latitude>
                    </coords>
                </SourceData>
                <TargetData>
                    <SRS>EPSG:26917</SRS>
                </TargetData>
            </InputVariables>
            <OutputResult>
                <UTMCoords>
                    <SRS>EPSG:26917</SRS>
                    <X>UTM X</X>
                    <Y>UTM Y</Y>
                </UTMCoords>
                <OutputErrorMessage>error message</OutputErrorMessage>
            </OutputResult>
        </Function>
    </Service>
</ServiceResponse>
</ResponseTemplate>
</WebService>
```

Figure 3. Template ServiceDescription.xml file for a geospatial data conversion web service

Ideally, each serviced can be identified within one independent service domain or more specific subdomain with multiple functions. Each service request template can then contain multiple requests identified by their unique namespace. Correspondingly the service response template can have multiple responses. In this case, to send a request to the service provider of geospatial data conversion web service, the service request is shown in Figure 4:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ServiceRequest>
  <Service Name="DataConversion">
    <Function Name="LL2UTM">
      <InputVariables>
        <FeatureType>Point</FeatureType>
        <SourceData>
          <SRS>EPSG:4326</SRS>
          <coords multiElements = "false">
            <longitude>longitude</longitude>
            <latitude>latitude</latitude>
          </coords>
        </SourceData>
        <TargetData>
          <SRS>EPSG:26917</SRS>
        </TargetData>
      </InputVariables>
    </Function>
  </Service>
</ServiceRequest>
```

Figure 4. Service request to invoke the geospatial data conversion web service

Once the service requester send this request as a string data type to the service provider, the later will then send back an XML document as the response with the prototype described in Figure 5:

```
<?xml version="1.0" encoding="utf-8" standalone="no"?>
<ServiceResponse>
  <Service Name="DataConversion">
    <Function Name="LL2UTM">
      <InputVariables>
        <FeatureType>Point</FeatureType>
        <SourceData>
          <SRS>EPSG:4326</SRS>
          <coords multiElements = "false">
            <longitude>longitude</longitude>
            <latitude>latitude</latitude>
```

```
            </coords>
        </SourceData>
        <TargetData>
            <SRS>EPSG:26917</SRS>
        </TargetData>
    </InputVariables>
    <OutputResult>
        <UTMCoords>
            <SRS>EPSG:26917</SRS>
            <X>UTM X</X>
            <Y>UTM Y</Y>
        </UTMCoords>
        <OutputErrorMessage>error message</OutputErrorMessage>
    </OutputResult>
  </Function>
 </Service>
</ServiceResponse>
```

Figure 5. Service response prototype of the geospatial data conversion web service

Web service composition thus can be simplified to the following three steps:

1. Retrieve the element ***<ServiceRequest>*** from the template *ServiceDescription.xml* file.
2. Add the XML declaration header, i.e. ***<?xml version="1.0" encoding="utf-8">*** with the element ***<ServiceRequest>*** to formulate a service request XML document.
3. Change the necessary values within the tags ***<InputVariables>*** and ***</InputVariables>***.

4. REBUILDING THE SEMANTIC WEB SERVICE ARCHITECTURE

Semantic Web service architecture can be reorganized in the following ways:

- ***Registration*** through UDDI
- ***Discovery***, ***matchmaking*** through *ServiceDescription.xml*
- ***Composition*** based on *ServiceDescription.xml*
- ***Invocation*** through WSDL interface *getService(String request): String response*
- ***Transport*** through SOAP

Currently, UDDI is expected to be a registry for Web service publication so that people can find necessary services through such yellow/white book service. While semantic Web service is under construction, current UDDI registry nodes provided by Microsoft and IBM are not very practical. The naming systems on the Web service as well as its functions are based on ***common sense*** but not semantically and explicitly well defined with ontology-based independent service domain and functional category. Searching Web services by their names through UDDI is akin to searching Google, useful possibly for people, but not for the purpose of automated discovery and invocation. With the approach suggested here, UDDI augmented with *ServiceDescription.xml* can aid in the automated discovery and invocation of web services.

This research provides an initial implementation to demonstrate such a new infrastructure. The pilot project can be accessed at http://157.182.136.76/AItest/ws/AIDemo1/WebForm1.aspx Currently 9 Web services have been developed all with exactly the same interface *getService(String request): String response* and a service template document *ServiceDescription.xml*. Here, WebService1 provides location service that converts a location name into its geographic coordinates (longitude and latitude). WebService4 provides data conversion service that converts a longitude and latitude pair into the corresponding (x, y) coordinates in UTM meters.

Service discovery and matchmaking can happen if ontologically-derived descriptions are available for web services. In the demonstration project, after the user types in a place (such as Morgantown, WV) and selects the service "GetLatitude/Longitude" or "Convert to UTM X, Y", the system will loop through the service description documents of each available web service in the list box. By reading through *ServiceDescription.xml*, the system automatically selects the appropriate service by matching the semantic service description. *Once the appropriate service is identified, the system can then compose the service request document, invoke the service, get the service response document and retrieve the necessary values from the service response*. The result of this pilot project can be shown in Figure 6.

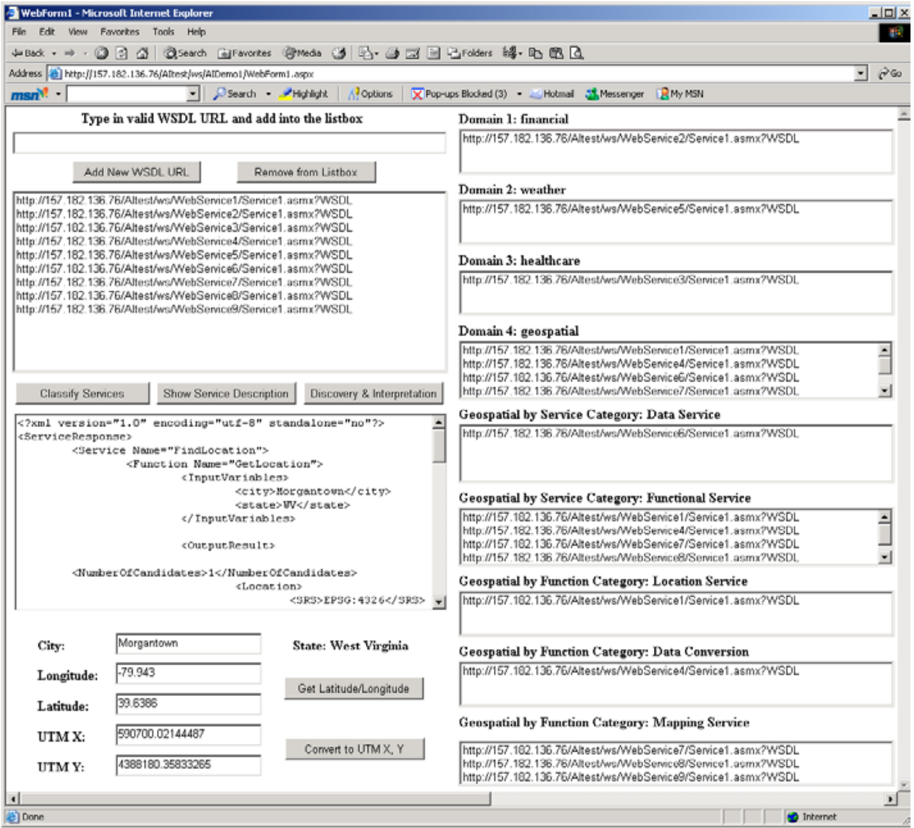

Figure 6. Dynamic Web service invocation by automatic service discovery, matchmaking, composition and interpretation

With the specification of domain-specific ontologies, data and service semantics can be well defined in the service description document *ServiceDescription.xml*. Problems as mentioned in those case studies can be resolved. Given the example to define a geospatial polygon feature layer as the input variable, such polygon feature can be defined as follows in Figure 7 using OWL/RDF:

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
   xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
   xmlns:owl="http://www.w3.org/2002/07/owl#"
   ……>

<owl:Class rdf:ID="Geometry"/>
<owl:Class rdf:ID="SpatialReferenceSystem"/>
……
```

```
    <owl:Class rdf:ID="2Dshape">
    <rdfs:subClassOf rdf:resource="#Geometry"/>
    </owl:Class>

    <owl:Class rdf:ID="Polygon">
    <rdfs:subClassOf rdf:resource="#2Dshape"/>
    </owl:Class>

    <owl:Class rdf:ID="Projected">
    <rdfs:subClassOf rdf:resource="#SpatialReferenceSystem"/>
    </owl:Class>
    ……

    <owl:Class rdf:ID="inputPolygon">
    <rdfs:subClassOf rdf:resource="#Polygon" />
    <owl:ObjectProperty rdf:ID="SRS">
    <rdfs:domain rdf:resource="#SpatialReferenceSystem" />
    <rdfs:range rdf:resource="#Projected"/>
    <owl:hasValue>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
        <rdf:value="EPSG:26917"/>
    </owl:hasValue>
    </owl:ObjectProperty>
    </owl:Class>

    <owl:Class rdf:ID="bufferPolygon">
    <rdfs:subClassOf rdf:resource="#Polygon" />
    <owl:ObjectProperty rdf:ID="SRS">
    <rdfs:domain rdf:resource="#SpatialReferenceSystem" />
    <rdfs:range rdf:resource="#Projected"/>
    <owl:hasValue>
        <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string" />
        <rdf:value="EPSG:26917"/>
    </owl:hasValue>
    </owl:ObjectProperty>
    </owl:Class>
    ……

</rdf:RDF>
```

Figure 7. Semantic data description on georeferenced polygons

The order and relationship of input variables as well as the meaning of the service can also be defined with the aid of XLink as follows in Figure 8:

```
<?xml version="1.0"?>
……
<RequestTemplate>
      <ServiceRequest>
   <Service Name="SpatialSelection">
       <Function Name="selectPolygons">
       <SpatialSelection>
       <Criterion>IsContainedBy</Criterion>
       <LayerRelationship xlink:type="arc" xlink:from="source" xlink:to="overlay" />
       </SpatialSelection>
       <InputVariables>
           <PolygonLayer xlink:type="resource" xlink:label="source">
                   <FeatureType>Polygon</FeatureType>
                       ……
           </PolygonLayer>
           <PolygonLayer xlink:type="resource" xlink:label="overlay">
               <FeatureType>Polygon</FeatureType>
                       ……
           </PolygonLayer>
       </InputVariables>
       </Function>
   </Service>
       </ServiceRequest>
  </RequestTemplate>
……
```

Figure 8. Service semantics for feature selection function

The purpose and behavior of Web service can be defined meaningfully as in the following example:

```
<?xml version="1.0"?>
<WebService xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended"
xlink:title="SpatialDataProcessing">
  <Service>
    <Domain>geospatial</Domain>
    <SubDomain>functional service</SubDomain>
    <Provider>WVGISTC</Provider>
    <Name>SpatialDataProcessing</Name>
```

```
    <Description>This service provides function for spatial data processing.</Description>
    <Function>
      <Name>intersect</Name>
      <Category>Geospatial Data Processing</Category>
      <Description>This function will cut an input layer with the features from an overlay
layer to produce an output layer with features that have attribute data from both layers.
</Description>
      <Namespace>
          intersect.SpatialDataProcessing.services.wvgistc.wvu.edu
      </Namespace>
    </Function>
  </Service>
……

  <RequestTemplate>
     <ServiceRequest>
  <Service Name="DataProcessing">
      <Function Name="createNewPolygons">
      <SpatialDataProcessing>
      <Criterion>Intersect</Criterion>
      <LayerRelationship xlink:type="arc" xlink:from="source"  xlink:to="overlay" />
      </SpatialDataProcessing>
      <InputVariables>
         <PolygonLayer xlink:type="resource" xlink:label="source">
                  <FeatureType>Polygon</FeatureType>
                                  ……

         </PolygonLayer>
         <PolygonLayer xlink:type="resource" xlink:label="overlay">
            <FeatureType>Polygon</FeatureType>
                                  ……

         </PolygonLayer>
      </InputVariables>
      </Function>
  </Service>
      </ServiceRequest>
  </RequestTemplate>
……
```

Figure 9. Service semantics for “intersect” function for geospatial data processing

```xml
<?xml version="1.0"?>
<WebService xmlns:xlink="http://www.w3.org/1999/xlink" xlink:type="extended"
xlink:title="SpatialSelection">
  <Service>
    <Domain>geospatial</Domain>
    <SubDomain>functional service</SubDomain>
    <Provider>WVGISTC</Provider>
    <Name>SpatialSelection</Name>
    <Description>This service provides function for spatial query and
analysis.</Description>
    <Function>
      <Name>selectPolygons</Name>
      <Category>Spatial Query and Analysis</Category>
      <Description>This function will select polygon features from source layer that is
spatially correlated by the overlay layer.</Description>
      <Namespace>
          selectPolygons.SpatialFeatureSelection.services.wvgistc.wvu.edu
      </Namespace>
    </Function>
  </Service>
……
<RequestTemplate>
      <ServiceRequest>
   <Service Name="SpatialSelection">
      <Function Name="selectPolygons">
      <SpatialSelection>
      <Criterion>Intersect</Criterion>
      <LayerRelationship xlink:type="arc" xlink:from="source"  xlink:to="overlay" />
      </SpatialSelection>
      <InputVariables>
          <PolygonLayer xlink:type="resource" xlink:label="source">
                    <FeatureType>Polygon</FeatureType>
                                        ……
          </PolygonLayer>
          <PolygonLayer xlink:type="resource" xlink:label="overlay">
              <FeatureType>Polygon</FeatureType>
                                        ……
          </PolygonLayer>
      </InputVariables>
      </Function>
   </Service>
      </ServiceRequest>
```

</***RequestTemplate***>
……

Figure 10. Service semantics for "intersect" function for spatial query and analysis

The differences between the two "intersect" functions for spatial query and spatial data processing are described in Figure 9 and 10. In this way, the purpose of the function can be described and defined meaningfully. OWL/RDF can also be used to define the content of service description and the relationship of service domain/subdomain or function category/subcategory within the tag <Service> and </Service>.

5. CONCLUSION

WSDL provides a standard interface for the exchange of objects and functions in distributed computing systems. As a description language, WSDL factually describes how the service architecture (the hierarchy and relation of data type, class, object, operation, etc.) is created without the capability to define the content and meaning of the message exchanged in the communication. When developing Web service, semantics are imposed into programming languages to give names to different objects, classes, data types, operations, etc. Such a naming system is based on common sense. For example, one can define a function name such as *GetAirlineTicket* or *BuyAirlineTicket* or any other names to perform the same function. Under such situation, semantic Web service description tightly coupled with WSDL cannot be dependable and believable for service requesters to find what they really want. Although one can purposefully define the name of certain functions and data objects, such as "token" as a string, "description1" or "description2" as string, or "dataSource" as string, etc. (Shi, 2004), such common sense based naming system is difficult to rely on for the purpose of automated discovery and execution of services dynamically.

This research proposes and implements a completely different approach to develop semantic Web services. Other schools of thought rely on external semantics that augment WSDL structures such as OWL, WSMO. The new approach prototyped here, directly encodes the semantics into an XML document as an integral part of Web service itself to transfer, understand and execute the service. The structure of this document enables semantic processing. The service provider here controls what is offered as a service by providing templates that are semantically constructed. There is no third party broker needed in this scheme. The service provider in this architecture becomes an intelligent agent and deals with semantically, human understandable queries and responses back in a human understandable form. Under the proposed new semantic Web service architecture, Web service can be dynamically invoked through automatic service discovery, matchmaking, composition and interpretation. There is no reliance on WSDL file and any other supplementary documents and diagrams. This approach does require collaboration among many groups to develop a common ontological framework.

One final comment on the proposed approach is that it is extensible. Web services can augment or change their service offering without changing their WSDL interface. This makes the infrastructure simpler to manage but does require more intelligence in the processing of messages. In this aspect this approach is similar to Web and email processing. Both Web and email use standardized and simple Internet protocols (HTTP/SMTP) to transfer the messages/content among the users. Such content are self-describing – albeit only to human readers. While WSDL is the protocol to transfer the service request and response messages, self-describing content here relies on ontological derivation and require intelligent-agent as processing engines.

6. REFERENCE

1. Aggarwal, R. 2004. Semantic Web Services Languages and Technologies: Comparison and Discussion. http://lsdis.cs.uga.edu/lib/download/aggarwal_essay_2004.pdf and
2. http://citeseer.ist.psu.edu/cache/papers/cs/32917/http:zSzzSzlsdis.cs.uga.eduzSzlibzSzdownloadzSzaggarwal_essay_2004.pdf/semantic-web-services-languages.pdf
3. Cabral, L., Domingue, J., Motta, E., Payne, T., and F. Hakimpour. 2004. Approaches to Semantic Web Services: An Overview and Comparisons. http://eprints.aktors.org/326/ and http://eprints.aktors.org/326/01/cabralESWS04.pdf
4. Martin, D., Paolucci, M., McIlraith, S., Burstein, M., McDermott, D., McGuinness, D., Parsia, B., Payne, T., Sabou, M., Solanki, M., Srinivasan, N., and K. Sycara. 2004. Bringing Semantics to Web Services: The OWL-S Approach. In: Proceedings of First International Workshop on Semantic Web Services and Web Process Composition (SWSWPC 2004), July 6-9, 2004, San Diego, California, USA.
5. McIlraith, S. A., Son, T. C., and H.Zeng. 2001. Semantic Web Services. IEEE Intelligent Systems. March/April 2001
6. Payne, T. and O. Lassila. 2004. Semantic Web Services. IEEE Intelligent Systems. July/August 2004

7. Roman, D., Keller, U. and H. Lausen (eds.). 2004. Web Service Modeling Ontology - Standard (WSMO - Standard), version 0.1 http://www.wsmo.org/2004/d2/v01/
8. Shi, X. 2004. Semantic request and response for standardized Web services. http://www-106.ibm.com/developerworks/webservices/library/ws-semantic/
9. W3C. 2004. Web Services Architecture. http://www.w3.org/TR/ws-arch/