IJREAT International Journal of Research in Engineering & Advanced Technology, Volume 2, Issue 2, Apr-May, 2014
ISSN: 2320 – 8791 (Impact Factor : 1.479)
www.ijreat.org

Stubbles Invocation of Dynamically Composed Web Services

Mrs. Ramya Ram Prasad[1], Rashmi Thapa[2], Rashmita Barua[3]

[1, 2,3]Information Technology, Anand Institute of Higher Technology, Chennai, Tamil Nadu, India

Abstract

A very vital capability of service oriented systems is the execution of composite web services. In existing systems, the invocation of the services is performed by the client components i.e stubs, which is generated at the design time from the service description. The efficiency of dynamically composed services can only be utilized if the service invocation is done at the run-time. We are addressing this problem of run time invocation by a method which is message based and is stub less. This is achieved by exploiting the structural properties of service description for the run time generation of messages

Keywords: *dynamic composition, stubless, dynamic invocation, ontologies.*

1. INTRODUCTION

A **web service** is a method of communication between two electronic devices over the World Wide Web. A **web service** is a software function provided at a network address over the web or the cloud.Web Services can convert your applications into Web-applications. Web Services are published, found, and used through the Web.These are application components that communicate open protocols and are self contained and self described. Web services can be discovered using UDDI, these services can be used by other applications.

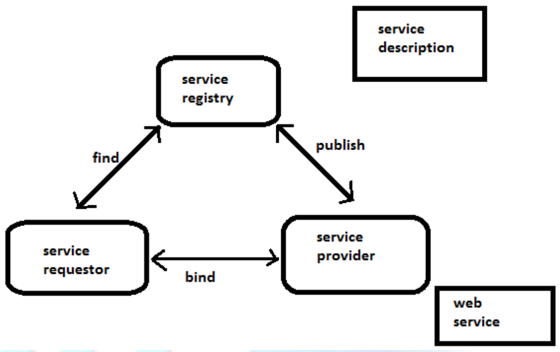


Fig:1 Web service frame work

The basic Web services platform is XML + HTTP. XML provides a language which can be used between different platforms and programming languages and still express complex messages and functions. Web services platform elements:

1.SOAP (Simple Object Access Protocol

3.UDDI (Universal Description, Discovery and Integration)

2.WSDL (Web Services Description Language)

2. WEB SERVICE COMPOSITITON

Web service composition is a distributed model to construct new web service on top of the existing primitive or other composite web services. It provides an open, standards-based approach for connecting web services together to create higher-level business processes. Standards are designed to reduce the complexity required to compose web services, hence reducing time and costs, and increase overall efficiency in businesses

Service oriented architecture is widely used for the creation of web services as it eases the design and development of distributed web services. The main principle of SOA which is followed here is loose coupling between the interacting services. A way of achieving loose coupling is to rely on the intermediation capabilities provided by a third-party component, usually called a Service Broker, especially for service discovery and composition. Given a service consumer request, the Service Broker specifies the workflow of service invocations that have to be executed to address the request. Here in this work we enable the concept of loose coupling by using composite services which are invoked in run time.

Presently various frameworks provide web services which are invoked both dynamically and statically, e.g.,Apache WSIF , Apache Axis 2, Codehaus XFire , Apache CXF. These framework invoke the web services by using static components called stubs(client component). A weakness of these solutions is that the client code is highly dependent upon specific toolkit APIs .

Stubs i.e the client component which is generated by machine becomes a burden for invocation of composite services. Hence in our work we follow a message driven method for invoking the composite web services.

3. MESSAGE DRIVEN INVOCATION METHOD

It is a stubless approach for invoking the composite web services in the run time. Our service invocation approach addresses this issue by removing the need for generating an internal representation of XML data types at design time. Semantic and structural message properties encoded in service description files are exploited to implement a mechanism that allows the run-time specification of composite service plans, and their execution through the stubless invocation of constituent services.

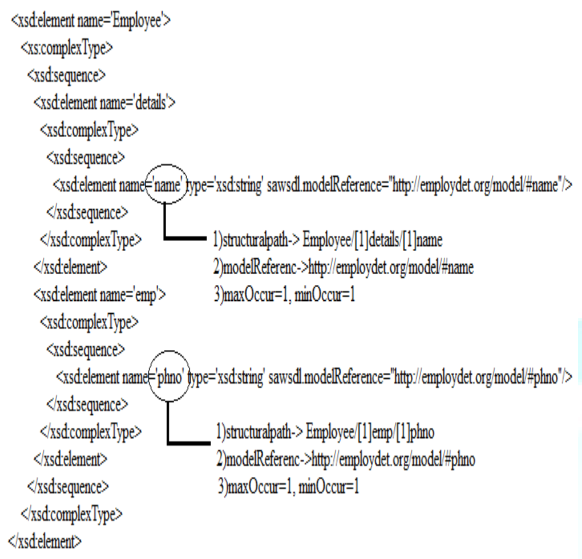
Our approach is based on two main principles.

First, we design a service as a message processor, which converts the interface into a message it can transmit and receive. Second, we rely on syntactic, structural, and semantic information that can be encoded in service descriptions for achieving dynamic service discovery, composition, and invocation. The main folds of the approach are stated in the following three levels. 1.**physical level** data have to be serialized in properly formatted messages to be sent on the wire. Mostly an XML document, and so the invocation is achieved by sending the message over proper protocol. 2. **logical level** extraction of atom values form the document and embedding of those atoms in an XML document request message using the syntactic and structural properties.3.**conceptual level** we explicitly represent concepts and relations that are embedded in the exchanged messages.

3.1 STRUCTURAL PROPERTIES

The structural properties generally helps to retrieve atom values they are specified using qualified name and structural properties that specify how the atom is embedded in a message. These structural properties provide i)the type attribute ii) (ii) the structuralPath attribute. The type attribute specifies the atom data type and structuralpath describes position of the atom in the XML document tree. To retrieve these atom values we use XPath, XPath helps to select nodes in an XML document and specifies the path through which the atom values can be retrieved from XML document tree.

Structural property helps to perform the following two operation using XPath (i) extracting atoms from leaf node from an XML document; and (ii) assembling the available atoms in an XML document that has been provided in XML document tree. These two operation help in



```
<xsd:element name='Employee'>
  <xs:complexType>
   <xsd:sequence>
    <xsd:element name='details'>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name='name' type='xsd:string' sawsdl.modelReference="http://employdet.org/model/#name"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name='emp'>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name='phno' type='xsd:string' sawsdl.modelReference="http://employdet.org/model/#phno"/>
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

1)structuralpath-> Employee/[1]details/[1]name
2)modelReferenc->http://employdet.org/model/#name
3)maxOccur=1, minOccur=1
1)structuralpath-> Employee/[1]emp/[1]phno
2)modelReferenc->http://employdet.org/model/#phno
3)maxOccur=1, minOccur=1

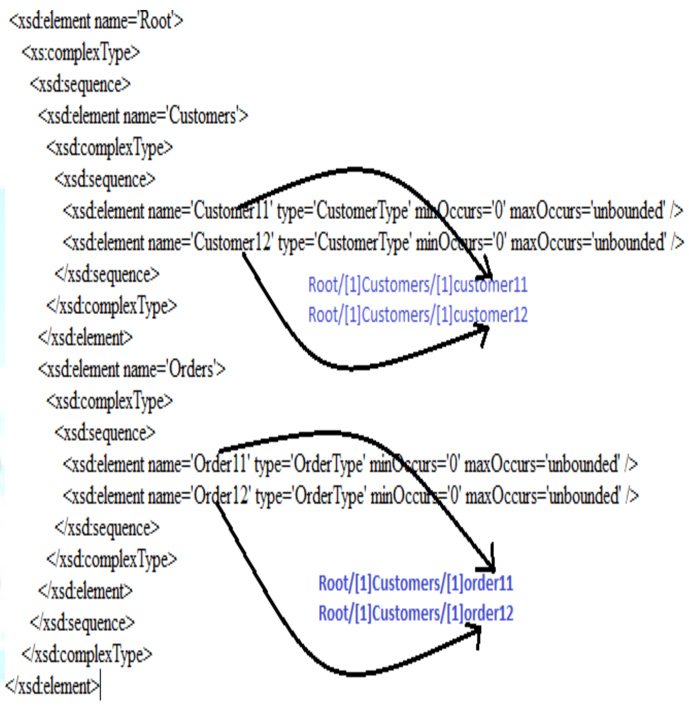
Fig:2example showing the way atom values are extracted from xml document.

composition of services over the web that will help in automated execution of operation and thus providing stubless invocation.

The structural information provided by these properties will enable user to traverse node to node in an XML document tree in order to retrieve the target atom value that can be further used to create an XML request and invoke the target service. These atom values can be retrieved either from the user XML request or previously invoked services. These properties are clearly based on structural paths so every time when user needs to invoke service the user will check the atom values and traverse following specified path from root to node.

3.2 SEMANTIC PROPERTIES

Semantic properties provide the meaning to the structural atom values. These helps in providing relation among the atom values through some semantic reasoning. It also helps to define the composite services for dynamic service invocation, as it helps to decide whether the atom data retrieved will be helpful to build a valid request message for the invocation of the next service in a series of composite service.



```
<xsd:element name='Root'>
  <xs:complexType>
   <xsd:sequence>
    <xsd:element name='Customers'>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name='Customer11' type='CustomerType' minOccurs='0' maxOccurs='unbounded' />
       <xsd:element name='Customer12' type='CustomerType' minOccurs='0' maxOccurs='unbounded' />
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
    <xsd:element name='Orders'>
     <xsd:complexType>
      <xsd:sequence>
       <xsd:element name='Order11' type='OrderType' minOccurs='0' maxOccurs='unbounded' />
       <xsd:element name='Order12' type='OrderType' minOccurs='0' maxOccurs='unbounded' />
      </xsd:sequence>
     </xsd:complexType>
    </xsd:element>
   </xsd:sequence>
  </xsd:complexType>
</xsd:element>
```

Root/[1]Customers/[1]customer11
Root/[1]Customers/[1]customer12
Root/[1]Customers/[1]order11
Root/[1]Customers/[1]order12

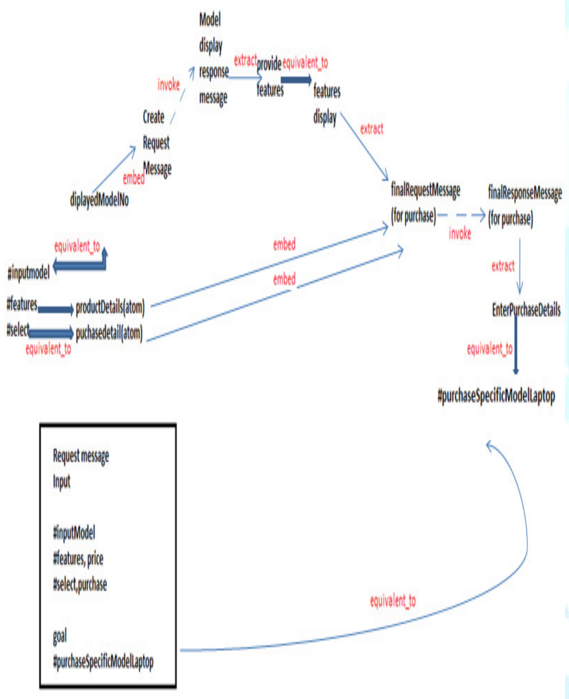
Fig.3 Example of the way structural path is expressed for XML document.

Here using the semantic properties the atoms are checked for equivalence using equivalent_to operator. When the atom values matches using this operator then the request message can be prepared and thus next service can be invoked.

4. SEQUENCE SERVICE COMPOSITION

This is the main process through which the stubless invocation of service can be done. It is generally threefold process; 1)analyzing client request message so that input and expected goals can be extracted here the semantic properties are used to extract the values that are embedded in the request message ;2)then using specific model the model is viewed ;3)if there is any solution then the invocation is performed else the process terminates after finite number of steps stating no solution exist.

The following fig:5 specifies the way service composition is performed. Here first the input parameters and expected goals are specified in the XML request message. The solution is specified in the following diagram. The inputModel is onto mapped to an atom



displayModelNo via operator used is health care. Clouds are being used to Fig:4 example of service composition(online laptop purchase).equivalent_to. Then this atom is further embedded in an XML message(createRequestMessage) to invoke service service that display model of Laptop. Similarly features and select is used to invoke service that invokes service to purchase the laptop.

5. DOCUMENT ANALYZER

Here we focus on structural model of service that provides dynamic invocation of services. The service profile model defines automated process run-time message creation and analysis, the mechanisms involved for dynamic invocation are 1) Dynamically creating an instance of XML request message 2) Sending the message to the service end-point address. To retrieve the dynamic stubless invocation of web services we use two models i)document analyzer ii) document builder.

Document analyzer is first module For dynamic and stubless invocation of services we use the invocation flow of composite services.

Access control is also gaining importance in online social net-working where users (members) store their personal information, pictures, videos and share them with selected groups of users or communities they belong to. Access control in online social networking Such data are being stored in clouds. It is very important that only the authorized users are given access to those information. A similar situation arises when data is stored in clouds, for example in Dropbox, and shared with certain groups of people.

It is just not enough to store the contents securely in the cloud but it might also be necessary to ensure anonymity of the user. For example, a user would like to store some As specified below this sequence provides the extraction of atom values from response messages of invoked services, this step is implemented by document analyzer algorithm.

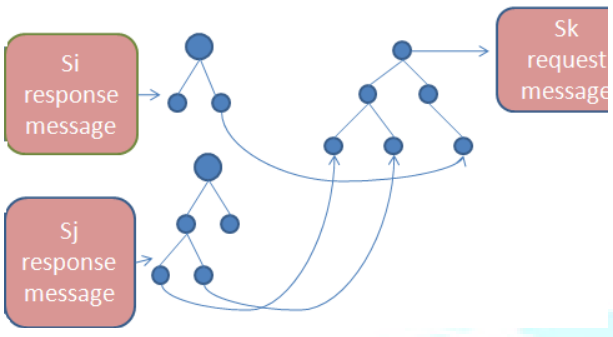


Fig:5 atom extraction and embedding.

The document analyzer takes an XML document as input and returns a set of pairs of value and structural path. So this pair helps to invoke the final aimed services dynamically without referring to any stub components. . For each structural path, the algorithm implements a recursive search in the XML document from the root to the leaf node that embeds a target value. The given value is exracted and using structural path it is associated. The document analyzer uses XPath to perform traversal of XML document.

6. DOCUMENT BUILDER

This is second module that performs embedding of atom values in XML document. The extracted atom value retrieved from the response messages of the invoked services are provided by document analyzer, these extracted values are then embedded into valid XML message for subsequent invocation. The document builder takes input of input data values and atom structural paths and gives the output whose structure complies with the document builder input.

The following steps are needed for document builder algorithm. 1)Build instanceAtomKeys: textual representation of document containing atom. 2)Create Root Node: to create tree representation of the document containing atom. 3) Build sub-tree: the algorithm call a createNode operation, which creates the node and adds it to the parent node.

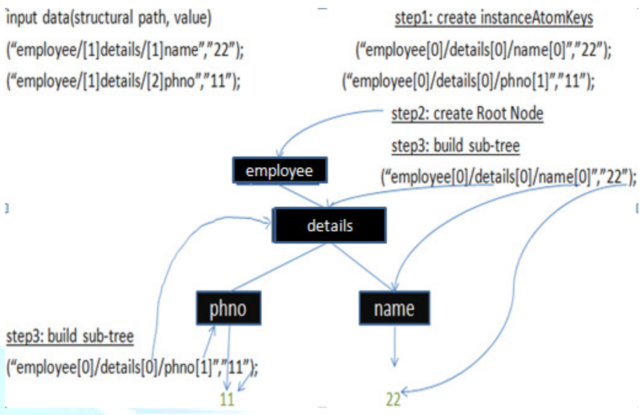

input data(structural path, value)
("employee/[1]details/[1]name","22");
("employee/[1]details/[2]phno","11");
step1: create instanceAtomKeys
("employee[0]/details[0]/name[0]","22");
("employee[0]/details[0]/phno[1]","11");
step2: create Root Node
step3: build sub-tree
("employee[0]/details[0]/name[0]","22");
employee
details
phno
name
step3: build sub-tree
("employee[0]/details[0]/phno[1]","11");
11
22

Fig:6 X-Path implementation

These operations are invoked for each extracted node name up to the leaf node. Which finally embeds the target value. During this step, two constraints are checked: the multiplicity of atom instances and the atom data type. This is another important module that helps to perform composition that will enable invocation of services without stub components

7. OTHER RELATED WORK

Here we specify the field related to dynamic service composition, invocation by service selection.

7.1 SERVICE INVOCATION

Dynamic service invocation is the process where the requesting system dynamically binds to a service and invoke the operations specified by these services. Generally service invocation is performed using client components(stubs) eg. Apache Axis2. This type of web services make use of marshalling(stub) process where the request sent and response received is converted into SOAP request and soap response which is intermediate process and then the desired operation is done. So every time the when invocation is done the stub components are created.

Our approach is truly message based where the service request is sent dynamically without using stubs through composition of services. Our approach implements a generic, extensible, and truly stubless invocation mechanism that is steered at run-time by structural and semantic properties of the target services. Some framework like Daios and RESTful style provide the slub-less approach of invoking web services. Daios chooses to invoke the service interface whose input message has the lowest structural distance metric to the provided data, the framework then converts the client request data into the encoding expected by the service. Rest style uses different architectural style. Representational State Transfer (REST) has gained widespread acceptance across the Web as a simpler alternative to SOAP- and Web Services Description Language (WSDL)-based Web services. REST defines a set of architectural principles by which you can design Web services that focus on a system's resources, including how resource states are addressed and transferred over HTTP by a wide range of clients written in different languages.

8. CONCLUSION

In this work we provide web service invocation method that follows SOA principles and uses data-centric approach. Here using dynamic web service composition and dynamic binding of services eliminates the need for stubs. Here we use 2 modules that will enable extracting and embedding of atoms that will facilitate the dynamic invocation. In this model, leaf nodes in XML documents are considered first-class entities, as they contain data values in instance messages. We exploit semantic and structural properties that provide embedding at runtime from XML document. Through service description from WSDL file embedding can be done, thus providing process of dynamic service composition and stubless invocation.

REFERENCES

[1] T. Erl, SOA, Principles of Service Design. Prentice Hall, 2008.

[2] OASIS, Web Services Business Process Execution Language Version 2.0, OASIS Standard 11, Apr. 2007. Available: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[3] P. Leitner, F. Rosenberg, and S. Dustdar, "Daios: efficient dynamic web service invocation," IEEE Internet Comput., vol. 13, no. 3, pp. 72–80,May 2009.

[4] Apache Axis 2. Available: http://ws.apache.org/axis2

[5] J. Kramer, J. Magee, and M. Sloman, "A software architecture for distributed computer control systems," Automatica, vol. 20, no. 1, pp. 93–102, 1984.

[6] E. Wilde and R. J. Glushko, "Document design matters," Commun. ACM, vol. 51, no. 10, pp. 43–49, 2008.

[7]DRWSC — To Simplify Dynamic Invocation for RESTful Web services Yanguang Chen, Jiehui Li, Yi Lv, Haihuan Qin and Liang Zhang∗ School of Computer Science

[8]A Dynamic Composition and Stubless Invocation Approach for Information-Providing Services, Fedrica Paganelli and David Parlanti