www.ijecs.in
International Journal Of Engineering And Computer Science ISSN:2319-7242
Volume 2 Issue 4 April, 2013 Page No. 972 -975

An Adaptive Architecture for Autonomic Orchestration of Web Services

Mrs.C.Sathya, Mrs.T.Hemalatha

Lecturer, Department of CSE
PSNA College of Engineering and Technology Dindigul, India
sathi.saras@gmail.com
Asso. Prof , Department of CSE
PSNA College of Engineering and Technology Dindigul, India
hemashek@yahoo.com

Abstract

Web service is an emerging paradigm in which loosely coupled software components are published, located and invoked on the web as a part of distributed applications. The advantage of employing web service composition is to create and consume a value added service by composing simple and complex software components which are deployed at different locations in an autonomic manner. The centralized web service composition approaches suffer from performance bottleneck and a single point failure. The web services are also distributed across geographical boundaries and they may be constantly removed or upgraded. The solution for the above problem is dynamic composition of web services. The proposed framework can handle any kind of users irrespective of their role in order to utilize the proposed architecture either as a provider or as a consumer. The proposed system performs autonomic composition of web services on-the-fly in order to perform dynamic composition of the services related to each other.

General Terms

Web services, SOAP, XML, UDDI

Keywords

Autonomic Composition, Centralized Service composition, Dynamic Service Composition, Service Invocation.

1. INTRODUCTION

Web service is a method of communication between two electronic devices over the World Wide Web. The W3C defines a "Web service" as "a software system designed to support interoperable machine-to-machine interaction over a network". Web Services are self-contained, modular, distributed, dynamic applications that can be described, published, located, or invoked over the network to create products, processes, and supply chains. These applications can be local, distributed, or Web-based. Web services are built on top of open standards such as TCP/IP, HTTP, Java, HTML, and XML. It has an interface described in a machine-process able format (specifically WSDL-Web Service Description Language). Other systems interact with the Web service in a manner prescribed by its description using SOAP(Simple Object Access Protocol) messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards. Web services are elements of distributed applications. The applications use the services by composing or putting them together. Architecture for service-based applications has three main parts:

- Service provider
- Service requestor
- Service registry

Providers publish or announce their services on registries, where the requestors find them and invoke them.

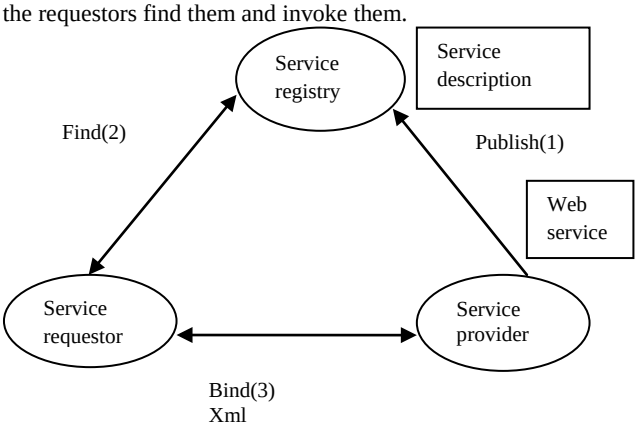

Fig 1: Web Service Architecture

The service providers will register their services in service registry. Service requestor finds required services in registry using a service broker and service requestors bind to them.

2. WEB SERVICE STANDARDS

2.1 XML (eXtensible Markup Language)

XML is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and

machine-readable. The design goals of XML emphasize simplicity, generality, and usability over the Internet.

2.2 SOAP

SOAP is a simple, lightweight XML-based protocol for exchanging structured and type information on the web. The overall design goal of SOAP is to provide a minimum of functionality. The protocol defines a messaging framework that contains no application or transport semantics. It uses HTTP/FTP/SMTP as a transport protocol. As a result, the protocol is modular and very extensible. SOAP message structure consists of three parts, envelope, header and body.

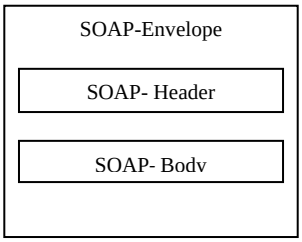

Fig 2: SOAP Structure

2.2.1 SOAP-Envelope

The SOAP Envelope element is the root element of a SOAP message. This element defines the XML document as a SOAP message. A SOAP protocol message contains an envelope which contains a header and a body.

2.2.2 SOAP header

SOAP header is optional, and contains information that might be subject to processing by intermediate SOAP nodes. When designing applications that use SOAP, information that might be useful to intermediaries should be placed in the header.

2.2.3 SOAP body

The SOAP body is mandatory, and contains the payload of the message, which is intended for the final SOAP receiver, and is not intended for processing by intermediate nodes.

2.3 UDDI

Universal Description, Discovery and Integration is a platform-independent, XML based registry by which businesses can list themselves on the Internet, and a mechanism to register and locate web service application. UDDI has led to automated discovery and the resulting execution of e-commerce transactions would result in an exceedingly liquid and frictionless environment for business.

2.4 WSDL

The Web Services Description Language is an XML-based interface description language which is used for describing the functionality offered by a web service. A WSDL description of a web service (also referred to as a WSDL file) provides a machine-readable description of how the service can be called, what parameters it expects, and what data structures it returns. It thus serves a roughly similar purpose as a method signature in a programming language. WSDL describes services as collections of network endpoints, or ports. The WSDL specification provides an XML format for documents for this purpose. WSDL is often used in combination with SOAP and an XML Schema to provide Web services over the Internet.

3. RELATED WORK

Place Agent based autonomic service composition was described by Hongxia Tong et al [1] in which a distributed algorithm for web service composition (DPAWSC) is presented. DPAWSC is based on the distributed decision making of autonomous service agents and addresses the distributed nature of web service composition. Since it involves multiple agents, it is difficult to achieve coordination. One more drawback is prior knowledge about the service is needed. Dynamic Invocation of web services proposed by Tere G.M et al [2] explains a framework for client to dynamically invoke web services. This framework can increase the use and reliability of web services invocation in a dynamic, heterogeneous environment. But service registry accepts service information passively and it contains too much classifications and information. Ontology bootstrapping process for web services is described by Aviv Segev et. al. [3]. Ontological bootstrapping aims at automatically generating concepts and their relations in the given domain and is a promising technique for ontology construction. The WSDL descriptor is evaluated using two methods, namely Term Frequency/Inverse Document Frequency (TF/IDF) and web context generation. The result of two methods is integrated and evaluated using third method called concept evocation. The drawback here is designing and maintaining ontologies. Christof Ebert et al [4] explains guidelines for orchestrating web services with BPEL. The business process execution language supports modeling and executing business processes from both the user and system perspectives. Web service application developers can use BPEL to orchestrate service interactions in a global system view to manage individual interactions based on outside events. But proficient use of any BPEL framework requires knowledge of underlying technologies. After programmers write a BPEL program they must package it and deploy it in the selected container. The steps differ among products and require considerable effort to get right the first time. Philipp Leitner et al [5] presents a message based service framework that supports implementation of SOAs, enabling dynamic invocation of web services. This framework enables the application developers to create service clients which are not coupled to any service provider. The disadvantage here is clients have to find a service that they want to invoke. Abdaladhem Albreshne et al [6] explain different technologies used for web service orchestration and composition. Frances M.T. Brazier et al [7] provides information regarding agents and service oriented computing for autonomic computing. Autonomic computing systems are expected to free system administrators to focus on higher-level goals.

4. PROBLEM FORMULATION

4.1 Requirements of service composition

The Service composition mechanisms are being employed to deliver support to complex user tasks within service-oriented environments. The mechanism of combining two or more services together to form a complex service is known as service composition. Service composition mechanisms are classically treated as extensions to service discovery techniques. Typically, a service composition system accepts a complex user task as an input and attempts to meet the needs of the task at hand by appropriately matching the task requirements with the available services.

Web service composition lets developers create application on top of service-oriented computing native description, discovery and communication capabilities. Such applications are rapidly deployable and offer developers with reuse possibilities and users can seamlessly access to a variety of simple and complex services. Composition of

Web services has received much interest to support business-to-business or enterprise application integration.

4.2 Service Invocation

To invoke a service we need following information,

- Service Address- Where to contact the service. For example, the endpoint of a web service.
- Service Contract-What you are supposed to send to the server and what. If anything, it is supposed to sent to you.
- Service Semantics- What the service actually does.

Web service invocation modes specify how the clients consume web services.

Typically the clients consume the web services in any one of the two ways.

4.2.1 Static Invocation:

Static Invocation uses pre-generated stub. This stub corresponds to a WSDL specification agreed in advance. The client need not parse the WSDL file when the method is invoked, but it suffers from tight coupling and versioning problems.

4.2.2 Dynamic Invocation:

Dynamic invocation generates the stub at the time of the method invocation. This stub is generated by parsing the WSDL document that describes the web service. Client does not need proxy stub and the changes to WSDL file will not affect the web service invocation but it is complex and slow compared to static invocation.

-

5. PROPOSED SYSTEM

5.1 Objective

The objective of our proposed system is to design an autonomic model for dynamic invocation and dynamic composition of web services. Autonomic computing is the solution proposed to cope with the complexity of today's computing environments. Autonomic computing systems are expected to free system administrators to focus on higher-level goals. Self-management is an important element of autonomic computing. It can perform following functions without human intervention. They are,

Self-configuration- configuring themselves automatically when computing resources are added or removed.

Self-healing- discovering when, where and why they are ailing and performing the appropriate self-repair and fault-correction operation

Self-optimization- monitoring and controlling resources to ensure optimal functioning with respect to defined requirements, as well as optimizing performance and efficiency by retuning and reconfiguring themselves.

Self-protection- proactively identifying and protecting themselves from arbitrary or malicious attacks or cascading failures.

Among all the functions of autonomic computing, we are focusing on self-configuration in our proposed system.

5.2 System Design

The proposed unified architecture provides a federated environment between the service provider and service consumer. In such procedure, the provider's web method is hosted in the container of federated environment and its corresponding information in the WSDL contracts are stored in both public and private UDDI registry for further use.

5.2.1 System Description

The client requests for a service in the portal. The portal gets the service requirements from the user that will be forwarded to the requirement block. This block in turn, searches for the method in WSDL contracts stored in public UDDI registry and returns the service name to the block. Then this block requests for WSDL contract for the respective service from UDDI registry. The UDDI registry sends all the WSDL contracts that contain the specific method. Then all the contracts are placed in a registry called WSDL contract registry. All the information will be passed to the orchestrator which is responsible for dynamic invocation of web services. If the web service is not found then the tracker will display the error message.

4.3 Challenges in Service Composition

There are two methods of web service composition. They are,
Centralized composition and Autonomic composition

4.3.1 Centralized service Composition

In centralized composition there is a centralized administrator who has a control over all other web services and this controller is responsible for invoking the services. It has its own advantages such as efficient monitoring, fault handing, easy maintenance, and less complexity. But it suffers from
performance bottleneck, Single Point failure and lack of trust.

4.3.2 Autonomic Composition

Here everything is decentralized. The service composition will be done at runtime based on user requirements in an autonomic manner.
Advantages Highly scalable

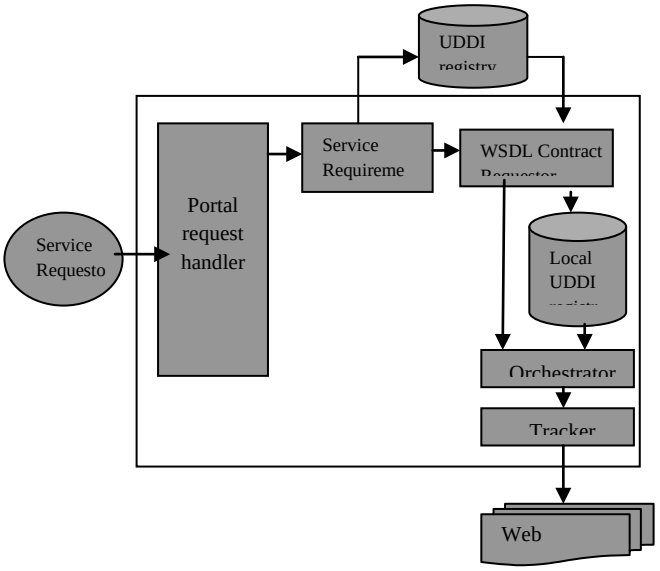

Figure 3: System design

5.2.2 Module Description

The modules in proposed system are,

1. user interface
2. global Registry
3. contract requestor
4. Orchestrator

5. Tracker
6. Web services

5.2.2.1 User Interface

This module processes the requirements of the user and interacts with the federated environment to invoke the web services in an autonomic manner.

5.2.2.2 Global registry

The global UDDI registry stores the WSDL contracts of all the services

5.2.2.3 Contract Requestor

Contract requestor module request for WSDL contracts from the UDDI registry.

5.2.2.4 Orchestrator

Orchestrator does the work of service composition and determines the Order in which services will be invoked.

5.2.2.5 Tracker

When there is any problem in invocation of services or if the service is not found then tracker helps in indicating the situation to the orchestrator

5.2.2.6 Web Services

This module is collection of web services designed for the federated environment. List of web services designed for our framework are,

1. Civil Supply Web Service
2. RTO Web Service
3. Passport web service
4. Institutional web service
5. Survey web service
6. Police web service
7. Voters web service

5.3 Experiment/Implementation

The proposed unified architecture is implemented in java. Apache Axis framework is used in order to handle the SOAP-HTTP request and response messages which is in XML constructs.

For experimentation the following web service are being under development for testing the autonomic orchestration of simple web services in order to implement e-governance using complex scenario. When any client(Service Requestor) places the request in the web portal, it is forwarded to the service requirement module. The request submitted by the client is analyzed and is searches the service descriptions that matches both the public registry and in private registry. On fetching the contracts by WSDL contract requestor it is stored in an ordered form in the local registry of the federated environment. Orchestrator does the invocation of independent web services provided by the different providers as per an order. On invoking the web methods, the remote web methods proxy is generated and it executes the service method and returns the response in the form of SOAP messages. The response is stored and forwarded by the tracker for the invocation of subsequent web services, there by orchestrating the simple web service to complex web service composition.

6. CONCLUSION

Web services technologies are becoming a de facto standard to integrate distributed applications and systems using XML-based standards. When a single service cannot satisfy the user needs then many services will be combined to provide a service. The biggest challenge in dynamic orchestration and dynamic invocation is that all the clients cannot have sufficient knowledge about interface description and invocation semantic of web methods. Our proposed architecture aids in creation of highly decentralized, coordinated and federated environment in which any type of clients can consume any services which is registered in our federated system without having any sufficient knowledge about the service description, service interface detail and corresponding details about web method. These can suite variety of e-governance scenarios. The proposed system will be tested for various applications under e-governance scenarios which will be highly significant for satisfying the future demands.

5. REFERENCE

[1] Hongxia Tong, Jian Cao, Shensheng Zhang, and Minglu Li, “A Distributed Algorithm for Web Service Composition Based on Service Agent Model”, IEEE Transactions on parallel and Distributed Systems”, vol.22,No.12, December 2011

[2] Tere G.M, Jadhav B.T and Mudholkar R.R, “Dynamic Invocation of Web Services”, Advances in Computational Research, ISSN:0975-3273 and E-ISSN:0975-9085, volume 4, Issue 1,2012,pp78-82

[3] Aviv Segev,Quan Z.Sheng, “Bootstrapping Ontologies for web services”, IEEE transactions on services computing, vol.5. no.1, January-March 2012

[4] Christof Ebert, “Orchestrating Web Services with BPEL”, March/April 2008, IEEE Software

[5] Philipp Leither, Florian Rosenberg and Schahram Dustdar, “DAIOS- Efficient Dynamic Web Service Invocation”, Distributed Systems Group, Technical University of Vienna, 2007

[6] Abdaladhem Albreshne, Patrik Furer, Jacques Pasquier, “Web Services Orchestration and composition”, September 2009

[7] Frances M.T. Brazier, Jeffrey O.Kephart, H.Van Dyke Parunak and Michael N.Huhns, “Agents and Service-Oriented Computing for Autonomic Computing”, IEEE Internet Computing 2009.

[8] Brahim, Medjahed, Rezgui, Athman, Mourad, “Infrastructure for E-Government web Services”,IEEE Internet Computing 2003

[9] Swaroop Kalasapur, Behrooz A. Shirazi, “Dynamic Service Composition in Pervasive Computing”, IEEE Transactions on Parallel and Distributed Systems, vol. 18, No. 7, July 2007