International Journal of Advanced Science and Research
ISSN: 2455-4227, Impact Factor: RJIF 5.12
www.allsciencejournal.com
Volume 2; Issue 4; July 2017; Page No. 101-105

Integration of cloud based services

[1] Manoj Kumar, [2] Dr. Aman Jain

[1] Research Scholar, Singhania University, Pacheri Bari, Jhunjhunu, Rajasthan, India
[2] Professor at Maharishi Arvind Institute of Science and Management, Jaipur, Rajasthan, India

Abstract
Cloud can supply a server infrastructure to achieve e-utilities requirements that can be easily paid on demand to meet specific customer needs and business success. It supports dynamic resource allocation in accordance with service level agreement policies and reuse of the IT infrastructure at the higher utilization level, and handling the management from the network edge to application and data servers. Meanwhile, the real time configuration can reduce time and cost and advance service reuse efficiently. This paper presents an EBS (Enterprise Service Bus) based framework that integrates the XML packaged legacy system resources into the Cloud service environment. The required resources are defined and building the Cloud services based on ESB components. The work on how to implement the integration has been discussed in this paper.

- The ESB(Enterprise Service Bus) based integration architecture is proposed, which combines the advantages of web service techniques including WSDL (Web Services Description Language) based on SOAP (Simple Object Access Protocol) to build the interface defined by WSDL.

Keywords: cloud computing, SOAP, WSDL

1. Introduction

What is Cloud Computing?

Cloud computing is a model for enabling ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources including networks, servers, storage and applications that can be rapidly provisioned and released with few management effort [1]. Currently, three main service models are researched

- Software as a Service (SaaS). The capability provided to the consumer is to use the provider‘s application running on a Cloud infrastructure. The applications are accessible from client devices through a thin client interface, such as a web browser, or a program interface. Few configurations are required for consumers.
- Platform as a Service (PaaS). The capability provided to the consumer is to deploy onto the Cloud infrastructure or acquired applications created using program languages, services, libraries and the tools supported by the provider. The consumer has control over the deployed applications and possibly configuration settings for the applications.
- Infrastructure as a Service (IaaS). The capability provided to the consumer is to provision processing, storage, networks and other fundamental computing resources where the consumer is able to deploy and run software, which can include operating systems and other applications. The consumer still needs to control the operating systems, storage and developed applications, and possibly limited control of selecting networking components, such as firewalls.

While the Cloud service is referred, the service model needs to be pointed out. Currently, SaaS is much preferred for most consumers and small business because most of them do not want more management for the configurations, but like to use the services at any time. Hence, we focus on how to invoke the functionalities from the evolved legacy application through the web service techniques that run on the Cloud.

The Cloud service oriented reengineering could bring great benefits for both legacy systems and Cloud services. It performs by reusing legacy system resources into the Cloud service environment. Compared with designing a new application, the proposed approach is less risky and massively reduces cost and time. After the process of Cloud-oriented service migration, the evolved service is required to integrate and deploy into the Cloud environment, which can interact with the outside world.

This paper focuses on addressing the following issues

- Design a Cloud service enabled integration architecture that allows the functionalities of legacy systems to interact with the existing Cloud service easily.
- Define appropriate middleware to integrate the identified functionalities to a heavily heterogeneous Cloud service environment.

Before the discussion of the above issues, the concepts of web services and EBS (Enterprise Business Bus) need to be expressed first which are involved in the solution process.

2. Web Service

Web service is one of the most active and widely used. It is based on an interoperable protocol called SOAP (Simple Object Access Protocol) [2] that provides an envelope which encapsulates XML data for exchanging through the web architecture. SOAP is an XML based protocol that handles all communications between client to client, server to server, server to client and application to application.

Web services are discoverable by the clients through searching the UDDI (Universal Description Discovery and Integration) directories, which publishes the web services as standard constructs like WSDL (Web Services Description Language). UDDI is a specialist for web services registration, which describes the service provider and publisher details. It

is also built on XML and SOAP. A WSDL is a construct document, which is merely an XML document describing the messaging format of the exposed services. In most cases, WSDL documents are static and do not change very often. Therefore, most programming languages generate proxies for the client from WSDL. If the WSDL document changes from one version to another, the vendors need to keep old bindings and just add new bindings. Hence, the older clients can continue to work [7, 8].

SaaS services are deployed into the Cloud just as web services and can also be implemented using web services. Hence, the web service techniques can still be applied to the integration process of a Cloud service.

3. Enterprise service bus

In the early 2000s, the IT industry introduced a bus-architecture technology, ESB (Enterprise Service Bus) [3]. It improved the central service registry mechanism in SOA and provided a software infrastructure for SOA implementation in enterprise applications. ESB is a message-based bus architecture that consists of a series of service containers. Each service container includes an adapter module, mediation module, message routing module and processing module, management module and security module. Each module is responsible for specific tasks. Figure 6-1 shows the components of ESB.

- **Security module**. It is responsible for unifying security management throughout the service container during handling the message.
- **Adapter module**. It acts as a service connector to connect software services and systems. A service adapter uses the native transaction interfaces to transfer messages between the service container and service.
- **Mediation module**. It transforms the protocol, message format and message content between the requesting service and service provider.
- **Message process and routing module**. It processes incoming and outgoing messages and routes messages from a service requester to the service providers using XML format. SOAP and XML can be used during the process.
- **Management module**. It plays an important role in ESB; it tracks activities happening in a service container and deals with a set of exceptions. A centralised management tool can be built to manage and configure all the service containers in the ESB.

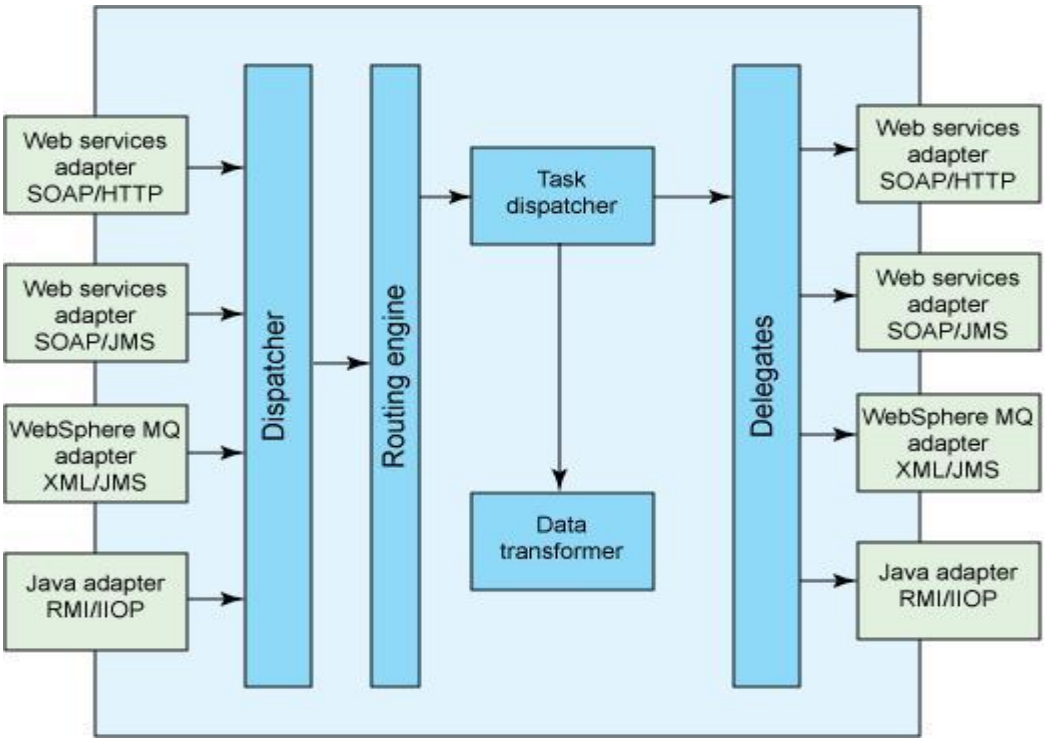

Fig 1: Components of the ESB [4]

In summary, ESB can be regarded as a type of middleware that helps to integrate a legacy system into a Cloud service by existing web service techniques.

4. An ESB Based Architecture for Cloud Services Integration

Cloud computing is evolving as an important IT services platform with its benefits of cost effectiveness and global access. In order to become a wider utilisation, Cloud computing has to be integrated with other existing services. Currently, there are few contributions of Cloud service integration proposed. Built on ESB as an integration direction, this section attempts to give integration architecture that combines the existing web service techniques to integrate a legacy system or other services without software redevelopment. Figure 2 gives the proposed integration architecture.

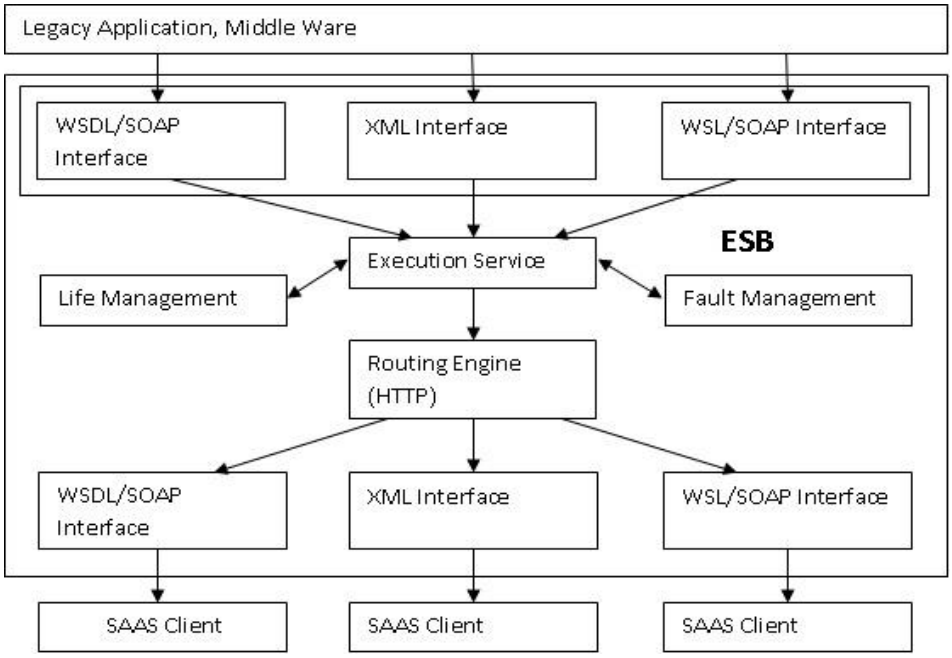

Fig 2: An ESB-based Architecture for Integrating Cloud Service

The integration architecture represents simple point to point service integration using wrapper or adaptor technology; such technology might be enabled through WSDL-defined SOAP access. XMI data and message format can be used to facilitate interoperability and platform independence in the web services standards. The fault management and life management are involved in the execution service to ensure service security. A routing engine will transfer the execution service to the client in a SaaS Cloud. With the execution description obtained from XML information, the client in the SaaS Cloud will invoke and use the legacy functionalities through the WSD-defined SOAP access.

5. Cloud Specific Service

Cloud services have emerged by combining Cloud computing and Web services to achieve faultless information processing system across heterogeneous, distributed, dynamic virtual organisations that the user can access from any location. To construct Cloud services with legacy components is tedious work, it may include a switch between many tools, such as command shells, file managers, editors, application containers, build tools, etc.

Being dynamic and required is the most difference between Cloud services and Web services. With the support of required resources, Cloud can remember what has been done from one invocation to another. The term state may be unclear and can cover many different aspects of a system, from the value stored in a specific database record to the temperature of the disk drive during the seeking process. A Cloud required resource is defined as resources which have the following features: a well-defined lifecycle known by one or more service providers and a specific set of state data, expressible like an XML document format. Required resources can be managed via Cloud service regulation. Once a system possesses the primary mechanisms of system adaptation and hiding complexity, it will show a series of attributes, such as independence in the control and management of the resources inside the system and the service provisions outside the system.

Following the analysis and representing legacy system components by extracting and transforming useful legacy assets into a Cloud oriented XML components, the extracted useful resources are able to be deployed in the Cloud environment as required resources to create Cloud services. These required resources can be isolated and fashioned into components that can be integrated in Cloud service oriented architectures. Previous sections show the process of legacy resource extraction and Cloud-oriented migration. To perform a cooperate Cloud service in this section, the components identification, and migration approach have been applied on a HRM system. The system is decomposed, and interested components are migrated. As a result, the concerned addition and subtraction Cloud components are retrieved. The following sections will discuss the generation of Cloud required services.

6. Cloud Services Description and Implementation

A resource properties document collects resource property elements, associated with a web service‘s WSDL 2.0 port Type definition to provide the declaration of the exposed resource properties of the Web Services Resource Framework (WS-Resource). It may be used by a service requestor to form an XML based query on the WS Resource. The resource property elements are almost identical to service data elements.

In WS Resource properties specification, it defines the type and values of those components of a WS Resource‘s state that can be viewed and modified by service requestors through a web service interface. This specification does not dictate the means by which a service implements a resource properties document. A given service implementation may be selected to realise its implementation of the resource properties document as an XMI instance document that is stored in some XMI repository.

Being required is necessary in Cloud service and Cloud can remember what has been done from one invocation to another

by the required resources support. The extracted legacy resources may be deployed as required resources to create the Cloud services. The defined required resources will have a specific set of state data expressible as an XML document. A few rules are defined to create the required resources

1. The complex type defining the resource properties document must define child elements only. Using xsd: sequence or xsd: all is applied to the child aggregation.
2. The complex type defining the resource properties document must define a sequence of one or more child elements, called resource property elements. Using XML schema element reference defines the child elements.
3. The complex type defining the resource properties document allows open element content – xsd: any.

7. Cloud Service Configuration and Deployment

For the service configuration and deployment, the following steps will aggregate all the loose pieces which have been written by the service container. The publishing description file with Web Service Deployment Descriptor (WSDD) format contains the details about the back-end components that implement a service. The SOAP message handlers intercept SOAP messages. WSDD file is an XML file.

In a Cloud environment, users should be able to access the created Cloud services through a high- level friendly Cloud portal. Currently, there are four Cloud deployment models [5]. They are Private Cloud, Public Cloud, Hybrid Cloud and Community Cloud. The deployment methods can be found on those service companies'web. A brief introduction for each deployment model is given as follows.

- **Private Cloud** The cloud infrastructure is provisioned for exclusive use by a single organisation comprising multiple consumers (e.g., business units). It may be owned, managed, and operated by the organisation or a third party.
- **Public Cloud** The cloud infrastructure is provisioned for open use by the general public. It may be owned, managed, and operated by business, academic or government organisations.
- **Hybrid Cloud** The cloud infrastructure is a composition of two or more distinct cloud infrastructures (private, public, or community) that keep unique entities, but are bound together by standardised techniques that enable data and application portability.
- **Community Cloud** the Cloud infrastructure is provisioned for exclusive use by a specific community of consumers, from organisations that have shared concerns including mission, security requirements, policy, and compliance considerations. It may be owned, managed, and operated by one or more organisations or a third party, or a combination of them.

No matter what Cloud service model is selected, some modules should be added to extend the original framework

- Dictionary Services. It can supply an index of all available components and data sources for system utilisation.
- Monitoring and Discovery Service. It can provide information about the available resources within the Cloud and their status.
- Resource Management. It provides the services to actually launch a job on particular resources.

From the user's aspect, they will be supplied with visualisation Cloud services which enable resource access across multiple heterogeneous platforms without regard for how these services are implemented. Each Cloud service model has its own deployment mechanism, e.g, Google App engine, IBM, Sun, Microsoft, and Eucalyptus. These Cloud service providers have detailed deployment for the developers. The information for deploying the service in these Cloud environments can be found on their websites.

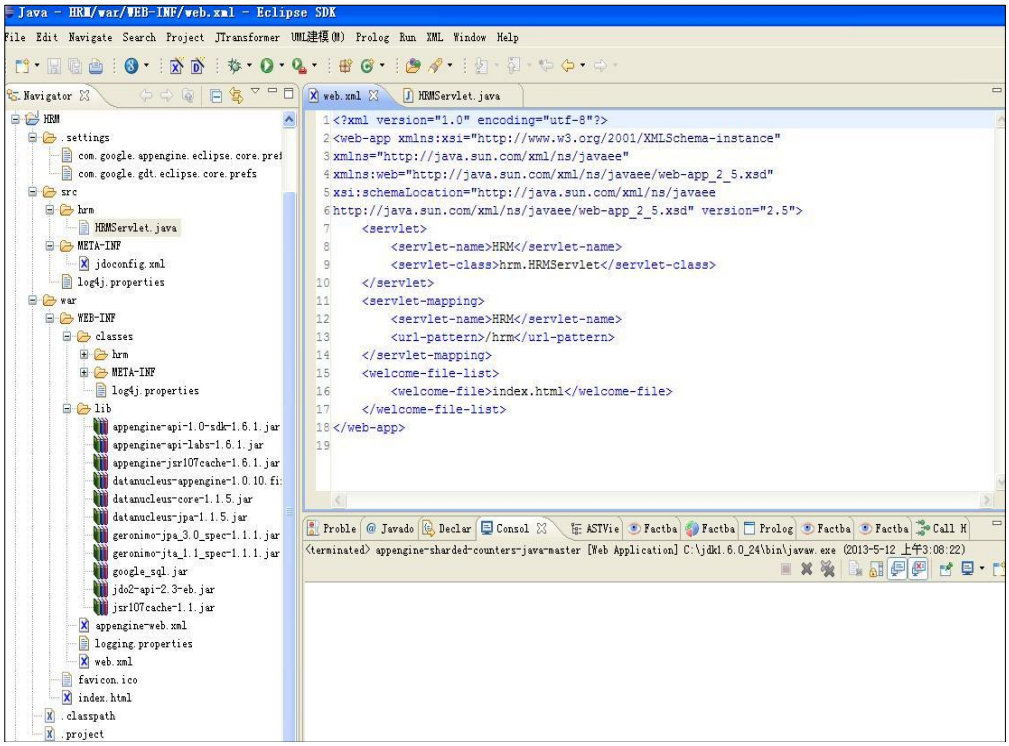

Fig 3: A Simple of Example of Service Deployment in Google Cloud

Take Google App engine as for example, it is a SaaS Cloud that can make the application run in their infrastructure. The eclipse plug- in for a Google App is easy to get and utilise. A simple example of service deployment in Google Cloud is shown in Figure 3.

8. Fault and Lifetime Management

The fault management takes the responsibility of reporting the faults when something goes wrong during the execution of a WS Service. The WS based Faults specification [6] defines an XML schema type for a base fault, along with the rules for how this fault type is conducted by the Web services. To address the issue, an XML schema definition must be defined and associated with the semantics for fault information. The definition is set by having a common base set of information that all the fault contents include. It is noted that the approach simply defines the base format for the faults without changing the WSDL message model.

Life management can trace each WS Resource Property and updates their state following a series of resource requests. The WS Resource Lifetime management defines the equal message exchange. If a service requestor wants to destroy a WS Resource, it must utilise the WS Resource qualified end state reference to send the destroy message to the web service provider. There are two ways to destroy a service request, namely an immediate way and a scheduled way. They both provide the flexibility to design how their service applications can clean up requests that are no longer required.

9. References

1. NIST, Cloud Definition, http://csrc.nist.gov/groups/SNS/cloud-computing/, 2013.
2. W3schools, SOAP, http://www.w3schools.com/soap/, 2013.
3. Chen LQ. Integrating Cloud Computing Services Using Enterprise Service Bus (ESB), *Business and Management Research*, Sciedu Press, 2012; 1(1).
4. MSDN ESB SOA Frameworks, http://www.ibm.com/developerworks/web/library/wa-soaesb/, 2013.
5. Armbrust M, Fox A, Griffith R, Joseph AD, Katz RH, Konwinski A, *et al.* Above the Cloude: A Berkeley View of Cloud Computing, *TechnicalReport*: *UCB/EECS*-EECS Department, University of California Berkeley, 2009.
6. Hanna S, Munro M. An Approach for Specification-Based Test Case Generation for Web Services", *IEEE/ACS International Conference on Computer Systems and Applications*, Amiman, 2007, 16-23.
7. Gimnich G. SOA Migration-Approaches and Experience", *TechnicalReport*: *30-34*, IBM Software Group, SOA Advanced Technology / Enterprise Integration Solutions, 2013.
8. Service Architecture, http://www.service-architecture.com/articles/web-services/soap.html, 2017.