`

Enterprise Application Management in Cloud Computing Context

George Feuerlicht[1,2] and Hong Thai Tran[2]

[1] Department of Information Technology, University of Economics, Prague, W. Churchill Sq. 4, Prague, Czech Republic
[2] Faculty of Engineering and Information Technology, University of Technology, Sydney
george.feuerlicht@uts.edu.au, thaitran@ueh.edu.vn

Abstract: Emergence of web services and cloud computing introduced a new form of application module delivery in which software functionality is consumed as a cloud service provided by external provider. As service-based enterprise systems are shifting to using cloud services system administrators are faced with the problem of how to manage cloud services in an efficient manner. In this paper, we propose a service consumer framework to support selection of cloud services and design of service workflows and management of service evolution. This work draws on several research areas including service management and evolution, and cloud service brokerage to present a service-based framework designed to maintain service availability in a constantly changing cloud environment.

Keywords: Service Management, Cloud Computing, Service-Oriented Architecture, Service Evolution, Cloud Service Brokerage.

1 Introduction

Service management traditionally deals with implementation and maintenance of *on-premise* services in an enterprise. Cloud technology has been growing creating a software industry that offers elastic and scalable services to organizations. Cloud technology is associated with a number of benefits including cost savings, elasticity, scalability, and competitiveness [1]. Given the compelling economic and technical advantages the adoption of cloud computing solutions is likely to accelerate. As a result, many organizations today rely on cloud-based services for a significant part of their Information Technology (IT) infrastructure [2] and enterprise information system (EIS). In this context, service consumers are mainly responsible for service selection, integration and management, while the service providers are responsible for providing business functionality as a service.

With diversity of cloud services, the same type of services are available from various cloud providers with different application programming interfaces (APIs) and different quality of service (QoS). An important challenge, in particular in situations where a large number of cloud providers are involved, relates to selecting suitable

`

services and managing evolution and outages. The Programmable Web API directory [3] currently lists more than eleven thousand available APIs for various types of services making selection of suitable services challenging. As the dependence of enterprise applications on cloud services increases the quality of applications will to a large extend depend on the quality of cloud services and the effective management of cloud services will become more important.

Cloud services are subjects of continuous evolution due to the changes of business requirements and processes to support business agility and innovation. As service providers introduce new features and maintain applications they attempt to provide backward compatibility between versions of services, however this may not be always possible. Service consumers are often unaware of the changes to external services until they encounter runtime issues. This may result in service consumers being forced to re-develop their applications resulting in ongoing maintenance costs.

Uncontrolled service changes often create a serious run-time risk for service oriented enterprise systems. Dealing with the service evolution from cloud service provider perspective has become an important system management challenge and many researchers have been working on devising techniques to minimize the impact of service changes on client applications. These techniques include service versioning and conversion, role-based management, and various Quality of service (QoS) driven, approaches that attempt to protect service consumer applications from changes in externally provided services.

In our previous paper [4] we discussed the problem of service evolution management and introduced our proposal for Service Consumer Framework (SCF). In this paper, we present a more detailed description of the framework with focus on support for identification and selection cloud services at design-time. The current version of the SCF framework provides a runtime failover capability that automatically replaces services that become unavailable with equivalent services, as specified at design-time. The aim of SCF framework is to achieve level of failure transparency that approximates that which exists in intra-enterprise environments, and to eventually support service aggregation and integration.

The remainder of the paper is structured as follows. In section 2, we discuss related work dealing with service management, cloud service brokerage and service evolution. In section 3 we describe a motivating example, and in section 4 the proposed SCF framework. The conclusion and directions of future work will be presented in section 5.

2 Related work

In this section we review two research topics that are closely related to our proposal for Consumer Service Framework: IT Service management (Section 2.1), cloud service brokerage (section 2.2), service evolution management (section 2.3)

`

2.1 IT Service management (ITSM)

The objective of IT Service Management (ITSM) is the coordination of organizational capabilities to deliver IT services effectively and efficiently. The promise of ITSM is to provide better IT alignment for enterprise system regarding, resulting in cost savings and risk reduction [5-7]. Among service management frameworks, IT infrastructure library (ITIL) has become worldwide de-facto-standard that is widely used by organizations as a service management framework. The ITIL lifecycle contains five different process phases: service strategy, service design, service transition, service operation and continual service improvement [8]. Each phase includes instructions and principles to manage on premise services. An alternative service management framework for enterprise systems based on a model-driven architecture that includes major service processes and activities of service management was proposed by Huang, et al. [9]. In a recent paper, Gama, et al. [5] proposed an integration model between ITIL and Enterprise Architecture principles to manage and align IT with business. They used services as key integration points to map ITIL concepts and enterprise architecture concepts. The focus of ITSM related research is on the management of on premise services [5-7, 10], and this makes it difficult to apply to situations that involve extensive use of cloud services management of cloud services.

2.2 Cloud service brokerage

Cloud service broker (CSB) as an intermediate layer between service provider and service consumer is becoming a key concern topic of future research and development on service oriented computing and cloud computing. In particular purpose, service brokerage is a layer playing the important role in quality assurance and normalization. According to a Gartner research article on the topic of cloud service brokerage, Pettey and Meulen [11] categorized CSB into following types:

a. Intermediation: An intermediation broker is a cloud broker based platform to enhance a particular service by providing adding value and extending capability of service to consumers. This type of CSB is intermediated layer for utilizing cloud services that improve management of cloud services such as simple access to services, identity management, performance reporting, and security enhancement. Intermediation brokers are able to assistant cloud service customer in cost saving and billing issue. Regarding to location of deployment, cloud brokerage can be located in three different places: (1) in provider side to deliver a level of governance in original cloud service; (2) in consumer side to supervise consumer in management and administration of cloud services; or (3) as an independent third party between providers and consumers

b. Aggregation: Although some consumer organizations are able to afford the data integration, process integrity, aggregation brokerage can to bring a simple solution to implement enterprise system integration. This type of broker aims to compose multiple services into one service. These new services can provide business services that closely meet consumer functionality requirement regardless to limitation of

providers. It will ensure security of data between consumer and multiple providers which involved in deliver all component services integrated in new service. Despite of this, aggregation brokers is normally a layer of service provisions that can be placed as a man-in-the-middle between service consumers and service providers. They are primarily considered cloud service providers and their services brokered are generally stable and seldom change.

c. Arbitrage: Similar to aggregated cloud service broker, arbitrage brokers also provide functionality of service composition. However, the cloud services in arbitrage brokers are not fixed and can agilely choose from multiple services in same type.

In addition, Flowley et al [12] and ComputeNext,[13], one of the commercial cloud service brokerage platform, stated that a cloud service brokerage platform enables one more important category is Integration. Integration CSB is a type of cloud service brokerage that helps to maintain the data dependability for organizations.

Programmable Web [3], as a service repository was found in 2005, is a world class commercial project to provide information about service application programming interfaces (APIs). Programmable Web has been daily kept track of evolution of the service API and it has provided functionalities of browsing service repository, searching and identifying cloud services. Regarding to service information, Programmable Web project provides specific details of (cloud) services including user guide from provider, best practices and service notification of evolution.

Usha et al [14] proposed a cloud broker service framework to decide on an appropriate service offered by different CSPs depending on the QoS requirements from the users. They discussed that same type of services is provided by different Cloud service providers (CSPs) with varying quality. Therefore it is important to provide the best service among the services offered by the CSPs. The objective of this framework is to provide the requested services with minimum searching time, with cheapest cost and access mechanisms. The QoS parameters considered are based on the user preferences and these constraints are used to select the best provider from all the available service providers. Their proposed work focuses on the QoS parameters throughput and response time. The primary functionality of this framework is as an automate framework to select the appropriate service for users based on QoS parameters (throughput, response time) stored in user references.

Flowley et al. [12] propose a comparison framework for service brokerage solutions, and Usha et al. [14] use QoS (Quality of Service) attributes to identify suitable services from a range of services of the same type with the objective of minimizing response time and cost. QoS attributes used to select service providers are based on user preferences, and the proposed framework automates the process of matching these preferences against QoS attributes of available services.

In another project work paper, Pawlur et al [15] introduce an initial version of broker service layer named STRATOS which represents an initial step toward the automated cross-cloud resource provisioning and inter-cloud platform. While abstraction layers to IaaS already exist, their focus is on delaying choosing a cloud provider from design/develop time to deployment time, with the ability to change the decision with limited development effort. STRATOS automate the decision entirely, and move the decision point from deployment time to runtime. STRATOS allows the

`

application deployer to specify what is important to them in terms of KPIs so that when a request for resource acquisition is made it is able to consider the request against all providers' offerings and select the acquisition that is best aligned with the objectives. Some experiments are presented in this paper that demonstrate how cross-cloud distribution of an application can decrease the cost of the topology and create one that is better fitted to the deployer's objectives. They also discussed on exploring the design issues and challenges.

2.3 Service evolution management

Most service systems manage service evolution via controlled releases of service versions that attempt to maintain backward compatibility with older versions of the services. In general, changes that involve addition of new data types and operations are backward compatible and do not *break* existing client applications. However, in practice it is difficult to avoid changes that do not preserve version compatibility. Such changes include removal of elements that form the service interface (e.g. operations, complex data types, attributes, etc.). While service versioning allows service consumers to control the timing of upgrades of their applications, it imposes additional complexity and overheads on service providers resulting from having to maintain multiple service versions.

In general, service changes can be classified into changes to functional characteristics (i.e. changes that affect structure of service interfaces, business protocols, policy assertions, and operational behavior) and changes to non-functional characteristics (i.e. quality of service attributes, e.g. security, availability, accessibility, etc.) [16]. Papazoglou et al. [17, 18] further classify service changes into shallow (changes that are localized to a single service) and deep (changes that cascade across a number of different services).

A number of approaches for the management of service evolution have been proposed in the literature, including methods that identify and classify changes to service interfaces as services evolve from version to version [19-21], and life-cycle methods that attempt to address changes that affect multiple services [17]. Papazoglou et al. proposed a change-oriented service life-cycle to address the issues that arise with deep changes. The life-cycle starts with the identification of the need for service change and scoping its extent, and then progresses to a service analysis phase that uses the model of the current state of the services (*as-is model*) and the *to-be* service model to perform gap analysis. Following the analysis of the impact of the required changes, decisions are made about how to deal with overlapping and conflicting service functionality. During the final change life-cycle phase new services are aligned, integrated, tested and released into production. Borovskiy and Zeier [22] focus on evolution of Web Services and identify two main types of drivers that cause service changes: intrinsic and extrinsic. Intrinsic change drivers are poor design and poor implementation, and extrinsic change drivers include market and business requirements drivers, operational process drivers, legislative and regulatory drivers, and other types of external drivers. Given this classification, the authors discuss versioning and message conversion techniques designed to address changes to Web

`

Service interfaces that involve removing operations and parameters, and cardinality mismatches. Fokaefs, Mikhaiel [20] present an empirical study of WSDL change analysis of Amazon EC2 service, PayPal service and FedEx services. The paper provides a detail analysis of Amazon EC2 services (18 versions), FedEx Rate services (9 versions), FedEx Package Movement Information Services (3 versions), PayPal SOAP API services (4 versions), and Bing Search services (2 versions). The authors developed a tool (VTracker) based on a tree-alignment algorithm to compare complex WSDL specifications. VTracker calculates the *tree distances* between a pair of operations for two service versions. Based on their empirical analysis of Amazon EC2, PayPal and FedEx Web Services the authors conclude that removal of existing elements is relatively rare and that the evolution of services involves mostly adding new elements that do not *break* existing client applications. Romano and Pinzger [19] describe the WSDLDiff tool that is used to identify fine-grained changes between versions of a Web Services by comparing WSDL interfaces. WSDLDiff is based on the UMLDiff algorithm of Xing and Stroulia [23] and identifies most of the frequently occurring types of changes, including changes in XSD elements, attributes, references and enumerations. To allow comparison with the results of Fokaefs and Mikhaiel [20], Romano and Pinzger [19] computed metrics for Amazon EC2, PayPal and FedEx Web Services using the WSDLDiff tool. The resulting analysis shows the number of added, deleted, and changed elements for each type of Web Service. The authors identify differences in the evolution of Web Services and suggest that this information can be used to estimate the risk associated with the use of Web Services from a particular provider.

3 Motivating example

In this section we briefly describe the motivation for the SCF framework using an example scenario. The Conference System (CMS) is a simplified scenario based on a *real-world* application system that consumes both internal (on-premise) and external (cloud) services. On-premise applications include:

- Contributions: application that manages member donations and corporate contributions
- Subscriptions: application that allows customers to subscribe and access publications online as well as deliver the hard copy of the publications
- Conference Management: application that supports a range of conference management functions, including enrolment of participants, online payments, booking of accommodation and transportation, management of conference publications and related documents

Externally provided cloud services include:

- Payment services: Paypal Payment Gateway (www.paypal.com), SecurePay (www.securepay.com)

`

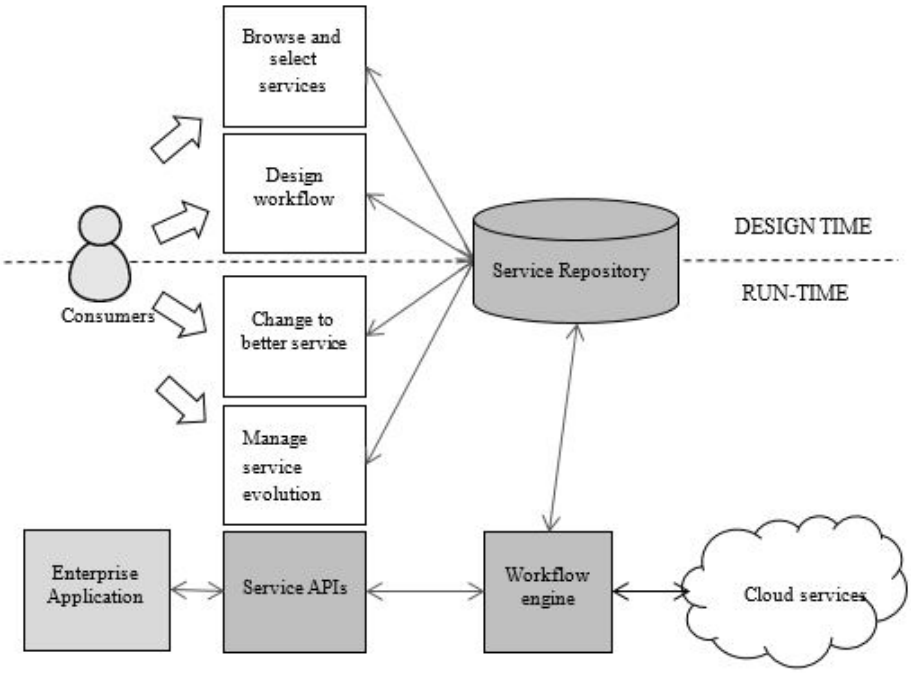
- Flight tracking services: FlightAware (flightaware.com), Flight Explorer (www.flightexplorer.com)
- Address validation services: Google Geocoding API (developers.google.com/maps/documentation), QAS Pro Web (www.qas.com)
- Shipment tracking services: FedEx Express (http://www.fedex.com/us/), UPS (http://www.ups.com/)

The CMS system operates in an environment where external services are continually evolved with providers upgrading their services by adding new features, fixing programming and design errors, and decommissioning old versions of services. Services are subject to outages and may become temporarily unavailable. A key requirement for the CMS system is to maintain continuous operation in this challenging environment.

At design time, application designers identify and verify the cloud services that meet both functionality and QoS requirements. In practice this task has to be done separately for each project and can be quite complex as there are a large number similar types of potential services (e.g. payment services). If the same service (e.g. payment service) is used across multiple applications, and it becomes unavailable, then the applications will affected, unless an alternative service can be deployed at runtime, without excessive delay.

4 Proposed SCF framework

In this section we introduce the main modules of the SCF framework and discuss how the framework is used to support management of cloud services. Availability of client applications in cloud environment depends on the availability of their constituent services. The SCF framework is a platform that supports consumers in utilization of cloud services helps to reduce the dependency of client systems on cloud services. The framework is designed to manage service maintenance and evolution issues.



`

There are a number of key characteristics of the SCF framework:

Service Intermediation: SCF as an intermediate layer that provides extended capabilities for using cloud services. It helps to manage cloud service usage as well as support *hot swapping* of services, i.e. change between services of the same type without any development effort. Clients can deploy new services with superior QoS characteristics as these become available.

Service runtime substitution: protects client applications from unexpected changes initiated by service providers. When the primary service (i.e. preferred service selected at design-time) service is modified by the provider, causing a failure of the client application, an alternative service is automatically deployed, improving availability of client applications.

The SCF framework helps to centralize service programming code within enterprise applications and support system management activities. Availability requirements can be met by using multiple services redundantly. The use of service adaptors reduces the programming effort by avoiding the need to modify application code in different sub-systems in response to changes by services providers.

Cost savings can be achieved by using identical services across all enterprise applications. In our example example, if the various sub-systems (i.e. Contributions, Conference Management, etc.) use the same payment or storage service, this could result in cost reduction based on the volume of consumed services. Applications also have the ability to switch to another less expensive service when it becomes available.

In order to improve throughput, the SCF framework can also be used to deploy

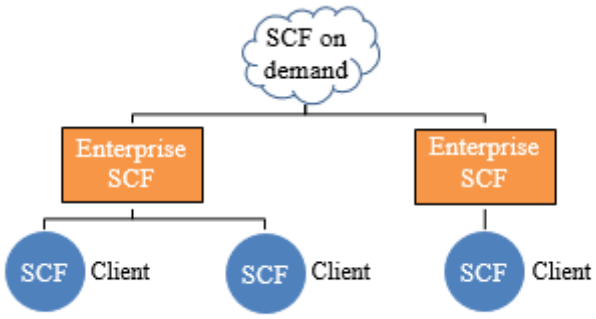


Figure 2. Distributed model of service consumer framework

multiple services using a distributed model as shown in Figure 2. Another planned function for the SCF framework is to provide statistical analysis of service utilization and to accurately predict QoS attributes (availability, response time, and throughput) for individual cloud services. Subscriptions to cloud services are not automatic, but selected services are verified by enterprise system administrators, and confidential data such as sensitive account information is stored inside the SCF framework.

Given the large number of available cloud services, the selection of suitable services presents a challenge to system administrations and service designers. It can

be very time-consuming for service consumers to browse service information, verify, and analyse services, in particular if this task is conducted multiple times by different projects that require similar services. [24]. Using the SCF service repository to store approved cloud services ensures that services are shared among different projects and that the service selection and approval process is not unnecessarily repeated.

The function of the service repository is to maintain information about available services and adaptors. The information in the service repository is used at design-time to identify suitable services and to define the sequence of service execution. Quality of Service (QoS) attributes stored in the repository can be used to identify cloud services based on their anticipated availability, response time, cost, or some other QoS attribute, and to define the processing rules that determine the sequence of service execution at run-time. The QoS data will in future be derived from run-time statistics collected on previous projects.

Service repository is the central component of the SCF framework that contains information about services, including functional information (WSDL, technical information, etc.) and non-functional information (response time, availability, cost, etc.). Service designers access the repository via a web interface that supports selection of services. Following registration of an application, developers can build workflows from services available in the repository.

4.1 Managing service evolution at run-time

Our framework provides ability of handling service evolution using workflow engine. Workflow engine uses workflows and routing rules stored in the service repository to route service requests to alternative services when the primary service becomes unavailable. The main purpose of the workflow engine is to provide redundancy and failover transparency for important services and to improve consumer application availability, as well as overcome service evolution issues. For example, based on routing rules stored in the service repository, workflow engine will select the first available service adaptor to process incoming payment requests. If no there are no available payment services, the workflow engine will add the payment request to a queue and process requests asynchronously when the service becomes available.

Another function of the workflow engine is to control the routing of requests to a provider according to priority rules defined in the service repository. For example, the Conference Management application has a payment workflow containing PayPal, SecurePay and OnePay services. During runtime, a payment request can be routed to an alternative payment gateway (e.g. SecurePay and then OnePay) via corresponding service adaptors if the PayPal service becomes unavailable. The service router uses information in the service repository to execute a service invocation sequence. The Conference Management application passes a payment request to the internal payment service that forwards this request to the external PayPal service via the PayPal adaptor. The external PayPal service response is sent back to the internal payment service via the PayPal adaptor, and then to the Conference Management application. If the PayPal service fails to respond within a specified time period the request is routed to the next external payment service (SecurePay in our example) via

`

the corresponding adaptor (i.e. SecurePay adaptor). If the SecurePay service fails to respond the request may be re-directed to another payment service (OnePay), or return an error status to the application. The order of service execution is defined at design-time based on designer preferences. For example, the designer may choose to call the least expensive payment service first, and execute a more expensive service only in the event of failure of the first service. Alternatively, the designer may decide to call the service that gives the best response time first, and only execute alternative services in the event of failure.

Each service adaptor is associated with a record in the service repository. During execution, the service adaptor invokes an external service, logs the result of service execution, and notifies application administrators if a failure occurs. The primary function of a service adaptor is to transform outgoing requests into the format supported by the current version of the corresponding external service, and to ensure that incoming responses maintain compatibility with internal applications. For example, the PayPal adaptor accepts payment requests from the CMS application with the interface containing attributes <Membership_ID, Name, Address, Payment_Type, Card_Type, Card_Holder_Name, Credit_Card_Number, Expiration_Date, Amount, CCV_Number, Note>. The payment request is logged and transformed to a PayPal payment request that has the interface containing attributes <Acct, Expdate, Amt, Comment1, Comment2, Cvv2, Firstname, Lastname, Street, Swipe, Tender, Trxtype, Zip>. Following a successful request execution the PayPal service response is transformed into a message compatible with the CMS application. The response message from the external service indicates success or failure of the request. For example, if the response indicates a communication failure, the service adaptor will mark the transaction as failed and the service router will route the payment request to another adaptor. Alternatively, the response message may indicate that the transaction was declined due to invalid credit card information (e.g. card number or expiration date) and the router may request for the information to be resubmitted

5 Conclusions

With the increasing complexity of SOA applications and the rapid adoption of cloud services, management of cloud services in the context of enterprise applications has become an important research topic. Service providers continuously evolve their services to improve their quality and to introduce new functionality, creating a challenging environment for service consumers. In this paper we have proposed an approach for the management of cloud services in enterprise systems. We have argued that there is a need to consider the problem of management of cloud service evolution from the perspective of service consumers and develop effective methods to protect service consumer applications from changes in external services. While most service broker proposals introduce a third party platform that is external to the service consumer organization, the proposed SCF framework is designed to support the management of cloud services within the consumer organization, eliminating additional level of dependence on a third party (the service broker). The SCF framework provides resilience for consumer applications to changes from external

`

services when these services are evolved or become temporarily unavailable. The basic idea of the framework involves the use of service repository to support in selecting, identifying cloud services and the use of service adaptors in combination with a workflow engine that re-directs requests to different service providers based on the availability of services at runtime. The use of service adaptors ensures that consumer applications can take advantage of new superior cloud services as they become available, and at the time determined by the service consumer. This avoids the need to immediately react to changes in external services as dictated by the service provider. Our current efforts focus on developing a proof-of-concept prototype of the SCF framework, extending its functionality and providing design-time interfaces for selection of services and definition of routing rules. At the same time we are working on developing a methodology for the management of cloud services for enterprise applications.

Acknowledgements

References

[1] M. Armbrust, A. Fox, R. Grifth, A. Joseph, R. Katz, A. Konwinski*, et al.*, "Above the clouds: A berkeley view of cloud computing," *Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, Berkeley, CA,* 2009.

[2] G. Horn, "Cloud brokerage: a key enabler for widespread cloud usage," presented at the Proceedings of the Second Nordic Symposium on Cloud Computing & Internet Technologies, Oslo, Norway, 2013.

[3] ProgrammableWeb. (2014). *ProgrammableWeb: the world's largest API repository, growing daily*. Available: http://www.programmableweb.com/apis/directory

[4] G. Feuerlicht and H. T. Tran, "Service consumer framework: Managing Service Evolution from a Consumer Perspective," in *ICEIS-2014. 16th International Conference on Enterprise Information Systems*, Portugal, 2014.

[5] N. Gama, P. Sousa, and M. da Silva, "Integrating Enterprise Architecture and IT Service Management," in *Building Sustainable Information Systems*, H. Linger, J. Fisher, A. Barnden, C. Barry, M. Lang, and C. Schneider, Eds., ed: Springer US, 2013, pp. 153-165.

[6] M. Brenner, "Classifying ITIL Processes; A Taxonomy under Tool Support Aspects," in *Business-Driven IT Management, 2006. BDIM '06. The First IEEE/IFIP International Workshop on*, 2006, pp. 19-28.

[7] A. Hochstein, R. Zarnekow, and W. Brenner, "ITIL as common practice reference model for IT service management: formal assessment and implications for practice," in *e-Technology, e-Commerce and e-Service, 2005. EEE '05. Proceedings. The 2005 IEEE International Conference on*, 2005, pp. 704-710.

`

[8] ITIL. (2014). *What is ITIL*. Available: http://www.itil.org/en/vomkennen/itil/ueberblick/index.php

[9] Y. Huang, S. Kumaran, and J.-Y. Chung, "A service management framework for service-oriented enterprises," in *e-Commerce Technology, 2004. CEC 2004. Proceedings. IEEE International Conference on*, 2004, pp. 181-186.

[10] B. McNaughton, P. Ray, and L. Lewis, "Designing an evaluation framework for IT service management," *Information & Management,* vol. 47, pp. 219-225, 5// 2010.

[11] C. Pettey and R. v. d. Meulen. (2009, Gartner Says Cloud Consumers Need Brokerages to Unlock the Potential of Cloud Services.

[12] F. Fowley, C. Pahl, and L. Zhang, "A comparison framework and review of service brokerage solutions for cloud architectures," in *The 1st workshop on Cloud Service Brokerage (CSB-2013), 11th International Conference on Service Oriented Computing (ICSOC 2013)*, Berlin, Germany, 2013.

[13] ComputeNext. (2014). *What is Cloud Service Brokerage?* Available: https://http://www.computenext.com/cloud-service-brokerage/

[14] M. Usha, J. Akilandeswari, and A. S. S. Fiaz, "An Efficient QoS Framework for Cloud Brokerage Services," in *Cloud and Services Computing (ISCOS), 2012 International Symposium on*, 2012, pp. 76-79.

[15] P. Pawluk, B. Simmons, M. Smit, M. Litoiu, and S. Mankovski, "Introducing STRATOS: A Cloud Broker Service," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 891-898.

[16] V. Andrikopoulos, S. Benbernou, and M. P. Papazoglou, "On the Evolution of Services," *IEEE Transactions on Software Engineering,* vol. 38, pp. 609-628, 2012.

[17] M. P. Papazoglou, "The Challenges of Service Evolution," in *CAISE'08. 20th International Conference on Advanced Information Systems Engineering*, 2008, pp. 1-15.

[18] M. P. Papazoglou, V. Andrikopoulos, and S. Benbernou, "Managing Evolving Services," *IEEE Software,* vol. 28, pp. 49-55, 2011.

[19] D. Romano and M. Pinzger, "Analyzing the Evolution of Web Services Using Fine-Grained Changes," in *ICWS-2012. IEEE 19th International Conference on Web Services*, 2012, pp. 392-399.

[20] M. Fokaefs, R. Mikhaiel, N. Tsantalis, E. Stroulia, and A. Lau, "An Empirical Study on Web Service Evolution," in *Web Services (ICWS), 2011 IEEE International Conference on*, 2011, pp. 49-56.

[21] A. Eisfeld, D. A. McMeekin, and A. P. Karduck, "Complex environment evolution: Challenges with semantic service infrastructures," in *DEST-2012. 6th IEEE International Conference on Digital Ecosystems Technologies*, 2012, pp. 1-6.

[22] V. Borovskiy and A. Zeier, "Evolution Management of Enterprise Web Services," in *Advanced Management of Information for Globalized Enterprises, 2008. AMIGE 2008. IEEE Symposium on*, 2008, pp. 1-5.

[23] Z. Xing and E. Stroulia, "UMLDiff: an algorithm for object-oriented design differencing," presented at the Proceedings of the 20th IEEE/ACM

`

international Conference on Automated software engineering, Long Beach, CA, USA, 2005.

[24] S. Sundareswaran, A. Squicciarini, and D. Lin, "A Brokerage-Based Approach for Cloud Service Selection," in *Cloud Computing (CLOUD), 2012 IEEE 5th International Conference on*, 2012, pp. 558-565.