A Syntactically Enriched Tool Enabling Dynamic Binding to Web Services.

Oyebode Kazeem Oyeyemi* and Esenogho Ebenezer

School of Electrical, Electronic, and Computer Engineering, University of KwaZulu-Natal, Durban, South Africa.

E-mail: 213573292@stu.ukzn.ac.za*
213574155@stu.ukzn.ac.za

ABSTRACT

Software availability and reliability is crucial within the context of critical software systems that depend on web services to fulfill their business processes. As such, there is need to cater for scenarios where there is a failure in a given web service, servicing a particular software system. There should be a fall over mechanism from a failed web service to another, in real time, in order to maintain software availability to service request. Based on this argument, we propose a software tool called D-Web Service. D-Web Service tries to improve the reliability and availability of software systems leveraging web services by providing a platform for automatic switching from a failed web service to an available web service performing the same functionality. To achieve this capability, D-Web Service provides a repository where acquired web services are uploaded, providing an avenue for software systems to select from a pool of available web services at runtime. The capability of D-Web Service is put to test by another critical software system called Global Money Transfer (GMT).

(Keywords: web services, dynamic binding, D-Web Service, GMT)

INTRODUCTION

Because web services can be invoked independently irrespective of their underlying language implementation, thus making them interoperable with other software system vendors, [10] points out that organizations are now using them to reduce the cost of maintaining and developing software systems. However, when web service specification is no longer valid [12] [1], there are failures encountered in the invoked web service as the software system tries to fulfill its business process. Based on the criticality of organizations business processes relying on web services, a failure in a business critical software system may result in a major economic loss as [7] points out. This strongly suggests that a suitable technique to dynamic binding to web services is imperative in order to guarantee system availability and reliability.

This paper introduces a software tool (D-Web Service) capable of selecting a suitable web service to fulfill a business process from a pool of web services performing the same functionality and dynamically bind to the chosen web service at runtime.

The rest of the paper discusses related work with respect to dynamic binding to web services; the concept of D-Web Service; the software system – Global Money Transfer (GMT) and how it puts the capability of D-Web Service to test; the limitations of D-Web Service; future enhancements that can be incorporated into D-Web Service; and lastly, graphical behavior with different plotting tools and conclusion.

RELATED WORK

In [18] the author developed a technique where web crawlers are used to discover web services on the internet and are dynamically invoked. It involves creating a simple HTML page and writing (embedding) Web Service Description Language (WSDL) into web pages so as to enable web crawlers search web pages for key web service information such as their names and operations. However, search engines or web crawlers have restrictions as they may not have the capability to reach out to web pages that need authentication or approval before such pages are viewed and used for dynamic web service linking as [2] argues. Furthermore, this technique or approach does not cater to a

scenario where a switch can be made from a failed web service to an available one.

An approach is describes by [8] using identified concepts in domain ontology to describe the operations of web services in a semantic annotation. The web service user uses such semantic annotation to dynamically search and invoke web services at runtime. However, the process of describing web service functionality using domain ontology is very difficult.

Firstly, [4] argues that service requesters will have to formally describe web service operations with the same concept captured by service providers in order to avoid semantic heterogeneity. Secondly, the requester may find it difficult to describe his request as a result of strict ontological rules, and lastly one need a good understanding of ontology before exploiting its capabilities in the context of web services. Furthermore, [11] described a syntactic approach using content analysis research methods in carrying out an experiment to develop an algorithm that will enable clients to bind to web services in the Universal Description, Discovery and Integration (UDDI) repository, based on web service names and operations.

The content analysis approach was adopted in order to understand the structure of web services in the UDDI. Based on this algorithm, if a match is found, the client dynamically binds to a web service. However, this approach does not allow dynamic web service linking based on web service quality attributes such as response time to request. In order to enhance this approach, [3] and [5] carried out an experiment in which Quality of Service (QoS) broker based architecture was incorporated into the UDDI repository.

The result of the experiment indicated that clients can dynamically bind to web services in the UDDI based on quality attributes. However, as [5] points out, the lack of maintenance of the UDDI repository and the lack of verifying the correctness of UDDI content by regulative bodies has discouraged web service providers from publishing web services in the UDDI. With this development, finding suitable services in the UDDI for dynamic binding becomes a challenge. This development has given rise to service providers publishing their services on the internet (distributed architecture) [2].

PROPOSED D-WEB SERVICE CONCEPT EXPLAINED

D-Web Service is a software system that functions as a repository where acquired web services are uploaded (Figure 1). Before a web service is uploaded into D-Web Service, a web service category and a web service subcategory must be created. For example, if a user wants to upload a web service that verifies credit card numbers with name “Verify Credit Card” into D-Web Service, then such a web service needs to belong to a given subcategory say “Get Credit Card Verification” and lastly, a category say “Credit Card Verification”. The reason for making this arrangement as indicated in [6] is to create a super-class (category) sub-class (subcategory) relationship that enhances proper classification of entities (web services), thus resembling an ontological web service modelling, holding concepts and knowledge in a particular domain, as such, enhancing an organised way to search for web services on the fly for dynamic web service invocation.

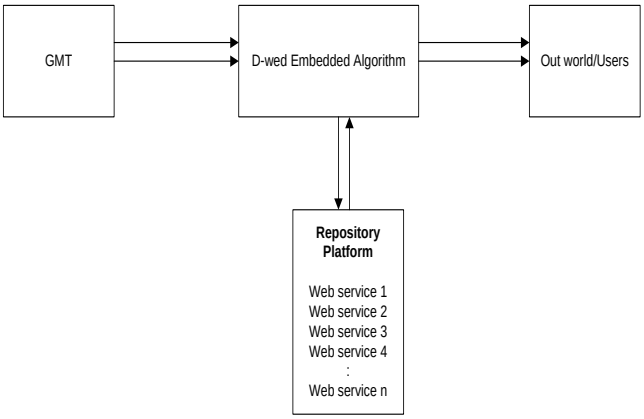


Figure1: System/Network Architecture of the Proposed D-web concept.

This way of categorizing and grouping acquired web services in D-Web Service software as seen in Figure 2 is similar to the lexicon approach of building ontology in a given domain as described in [13]. In their approach, important concepts in a given domain are identified by the Language Extended Lexicon (LEL) and stated precisely in a software tool called OilEd tool. The OilEd tool enables the creation of classes, subclasses, also the creation of relationships and restrictions among identified concepts. The constructed ontology using their approach can be processed by a computer software agent. D-Web Service is

also trying to achieve a similar objective, that is clearly categorizing and precisely stating a given web service properties such as its operations, names, number of input parameters, number of output parameters, type of input and output parameters and also the category and subcategory it belongs to. However, unlike the ontology developed using approach in [13] which can be processed by a computer agent, the precise stating of web service properties in D-Web Service enables D-Web Service to intelligently decide (syntactic selection) what sort of web service will be suitable to call upon dynamically at run time on behalf of a client.

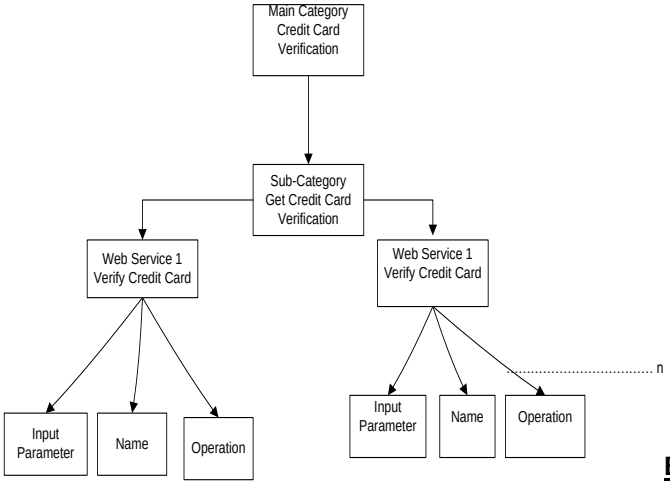


Figure 2: Ontological Arrangement of Web Services.

ABOUT GLOBAL MONEY TRANSFER (GMT) SOFTWARE (CASE-STUDY)

GMT is developed in order to test the dynamic web service capability of D-Web Service. GMT leverages on uploaded web services in D-Web Service in order to fulfil a given business process in real time. GMT enables users transfer money across the globe to another person in their home currency (see Figure 6 for GMT user interface).

A. GMT's Business Process

GMT is easy to use, before the user sends money, he/she needs to get the exchange rate of the currency to send, for example if a user wants to send 100 British pounds to Nigerian Naira, then the user needs to get the Naira equivalent of 100 British pounds in Nigerian Naira in real time. Next the user enters the credit card he/she wishes to use for the transaction followed by the sender's information and finally enters the beneficiary's name and address.

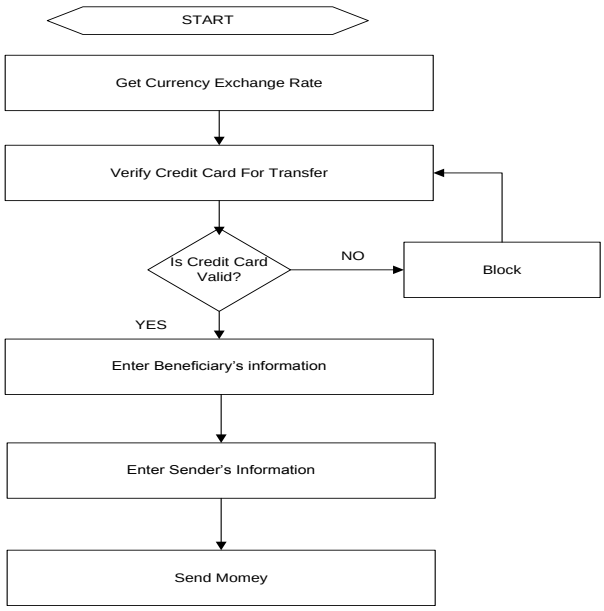


Figure 3: Work Flow Diagram of GMT Business Process.

B. How GMT Leverages D-Web Service

In order for GMT to fulfil its business processes, the researcher went in search for free web services on the internet, the researcher discovered:

1) Two web services that provides currency exchange rate in real time, the WDSL of these web services are:
   a) http://www.webservicex.net/CurrencyConvertor.asmx
   b) http://www.currencyserver.de/webservice/CurrencyServerWebService.asmx

2) One web service that verifies credit/debit cards online, the WDSL of the web service is
   a) http://www.ezzylearning.com/services/CreditCardValidationService.asmx

We then uploaded these web services into D-Web Service; however, it is interesting to note that a password was given by the provider of web service 1b in order to be successfully invoked. In the case of web service 1b (Figure 7), one cannot dynamically bind to it successfully without knowing its invocation password. After uploading

discovered web services on the D-Web Service (Figure 7), GMT was setup on the Local Area Network (LAN) to discover the uploaded web services on D-Web Service.

C. Before GMT Calls On D-Web Service

Before GMT invokes uploaded web services, it is important to point at some key facts about Figure 7.

1) Cost per request for web service 1a as shown in Figure 7 is £0, while for 1b is £0.2.

2) Both services have never been used before.

3) Even though they are enabled to respond to clients call, D-Web Service defaulted their availability status to false, this is because D-Web Service has not used any of these services before and as such, it defaulted their availability statuses to false until they are used.

4) The "last used" status of web services as shown in Figure 7 is "Never Used", meaning that these services have never been invoked before.

D. GMT Calling On D-Web Service

As GMT is about to call upon D-Web Service to get the latest currency rate information in real time, GMT does not know any information about any web service (information such as web service address is unknown) to fulfil this business process, all it knows is the subcategory – Get Currency Rate, as shown in Figure 8. Figure 9 shows the result as GMT prepares to call upon D-Web Service. The user intends to send £100 to Nigeria, in Nigerian currency (Naira), and the result is 24930.1277 Naira.

E. D-Web Service, as GMT Calls It

Though Figure 8 seems not to contain tangible data, however, a closer look at it reveals a great deal of information. Figure 8 shows two web services that D-Web Service can call on behave of GMT. Before D-Web Service calls any of these web services, it needs to apply the web service selection algorithm as seen in Figure 3. The selection algorithm checks web services if any is enabled for usage (availability), followed by cost and lastly the response time to request.

Based on the web service selection algorithm, D-Web Service further selects the web service with the least cost, which is web service 1a, however web service 1a generated an internal error while invoking it as seen in Figure 8, as such D-Web Service tries the next web service which is 1b. A closer look at the 'Last Used' column in Figure 8 reveals that D-Web Service dynamically invoked web service 1b, 2 seconds after web service 1a failed. The next step is for GMT users to enter their credit card information and the beneficiary's information and the amount will be sent to the beneficiary.

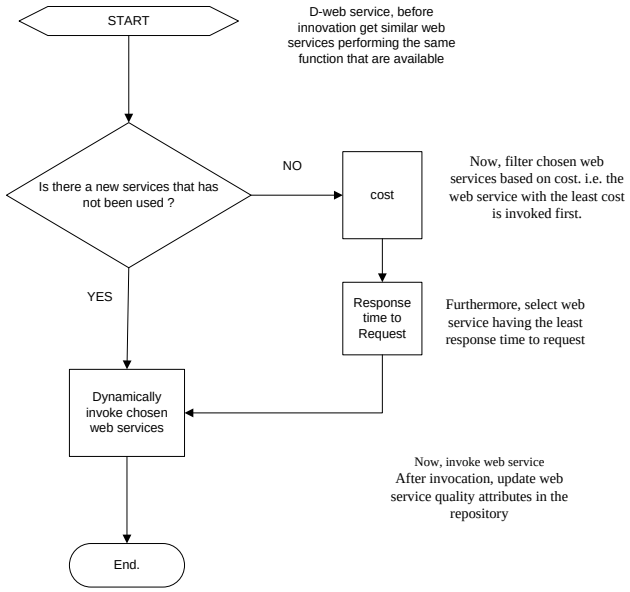


Figure 4: The Web Service Selection Algorithm.

RESULTS OF THE BEHAVIOUR OF THE SYSTEM

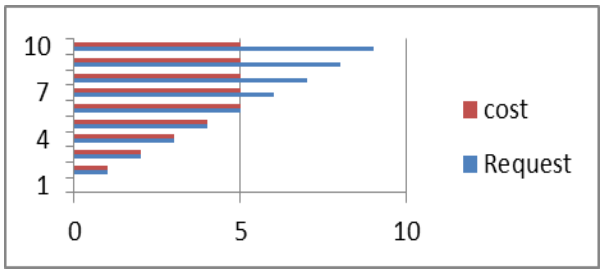
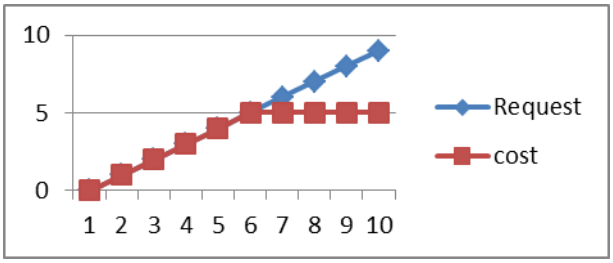


Figure 5a: Cost vs Request Represented in Bars.



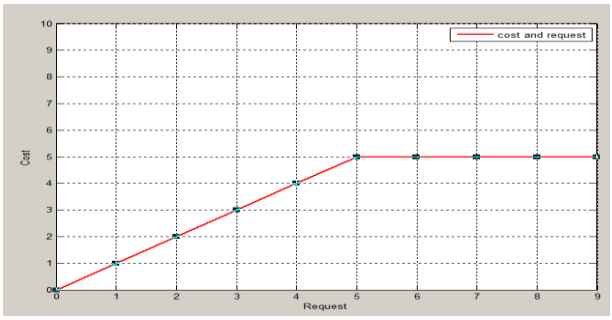


Figure 5b: Cost vs Request Represented in Graph Plots.

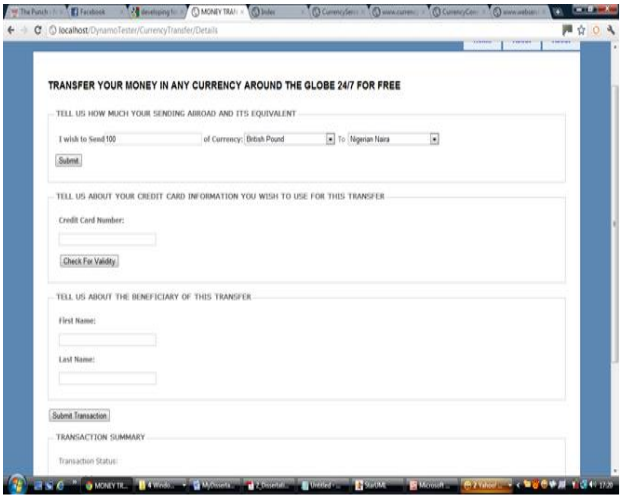


Figure 6: GMT User Interface.

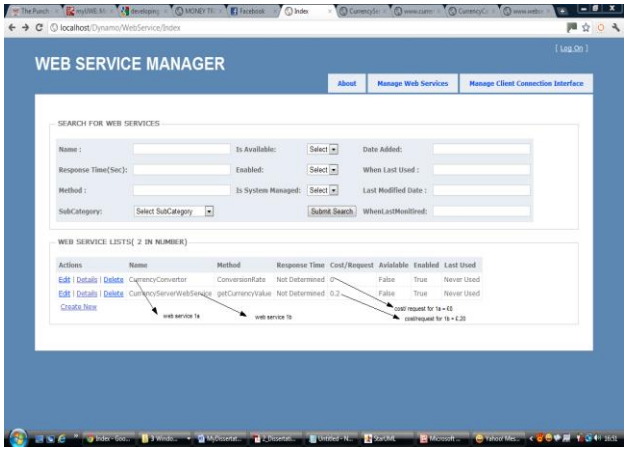


Figure 7: Web Service 1a and 1b Uploaded into D-Web Service, Before Invocation by GMT.

Figure 8: GMT Source Code Calling D-Web Service.

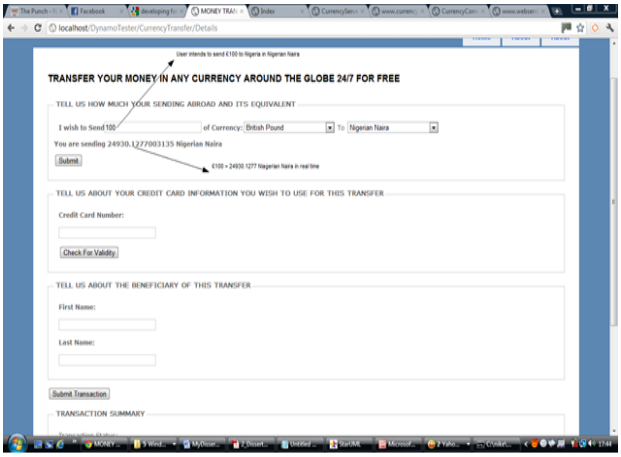


Figure 9: GMT Calling on D-Web Service on a LAN.

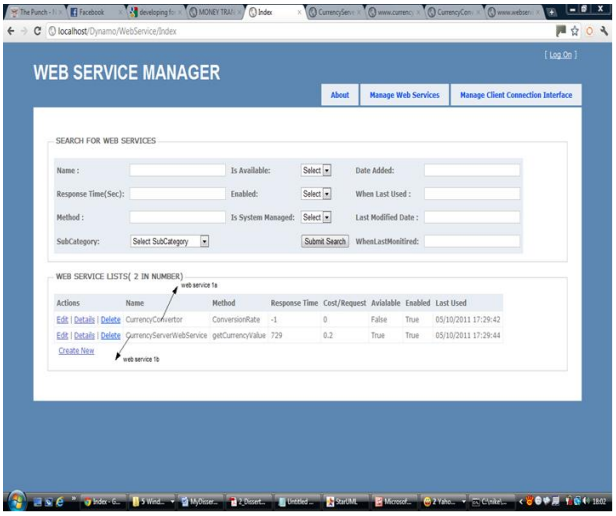


Figure 10: D-Web Service after Being Called by GMT.

D-WEB SERVICE CALLING ALGORITHM

1) Collect all web services performing same function into a collection list.

2) Rearrange or prioritize web services, those services that have not been used will be at the top of the collection list followed by web services with low calling cost.

3) Call the web service at the top of the collection list.

4) If there are errors, record the failure against this web service. Remove this web service that generated error from the collection list. Repeat step 3.

5) Send result to the client.

6) End.

D-WEB SERVICE LIMITATIONS

Though D-Web Service can dynamically bind to web service at runtime, this is not the case for all web services, for example D-Web Service cannot dynamically bind/communicate to RESTful web services. Furthermore, because of the complexity attributed to some web service WDSL (Web Service Description Language) design, D-Web Service may find it difficult to read and dynamically invoke such web services having complex and complicated WSDL files. In addition, though D-Web Service monitors web service quality attributes such as availability by periodically updating this property against each web service in the database, this is still not enough in improving the reliability of a software system leveraging D-Web Service. For example, in a case D-Web Service notices the unavailability of a service, it does not alert a system administrator via test messages or email for immediately action to be taken. It would have made sense for D-Web Service to contact the system administrator at the point a web service fails; this will enable immediate action to be taken in order to keep the failed web service running again or contact the web service provider in time.

FUTURE DEVELOPMENT TO ENHANCE D-WEB SERVICE

D-Web Service functionalities can be improved as follows:

1) D-Web Service can implement a function that alerts users (via mobile text) in situations where a web service fails; this will enable web service users contact web service providers for immediate rectification.

2) D-Web Service can implement a functionally that enables auto generation of monthly report, giving information about the performance of a given web service. This kind of performance report will enable its users decide if a web service performance is above expectation.

3) D-Web Service can be improved by incorporating ontology into its selection process. This will enable D-Web Service to be more intelligent in its web service selection capability.

4) D-Web Service can be improved not only to work with SOAP compliant web services, but also with web services that are not SOAP compliant.

5) D-Web Service can be improved so as to read and process security information in SOAP headers; this will enable D-Web Service to be compliant or dynamically invoke web services from providers like Amazon.

CONCLUSION

D-Web Service provides the capability for web service clients to dynamically call a web service at run time. Furthermore, D-Web Service provides the capability for software systems to dynamically switch over from a failed web service to an available one in real time. Respective of the limitations of D-Web Service as discussed, it has been shown to have increased the availability and reliability of critical software systems leveraging web services. Finally, from the graphical behaviour, users with more requests enjoy flat costs just to encourage users to couple with its advantages.

REFERENCES

1. Alonso, G., F. Casati, H. Kuno, and V. Machiraju. 2004. *Web Services: Concepts, Architectures and Applications*. Springer: New York, NY.

2. Al-Masri, E. and Q.H. Mahmoud. 2008b. "Toward Quality-Driven Web Service Discovery". *IT Professional*. 10(3):24-28.

3. D'Mello, D.A., V.S. Ananthanarayana, and T. Santhi. 2008. "A QoS Broker Based Architecture for Dynamic Web Service Selection". Second Asia International Conference on Modeling & Simulation.101-106

4. D'Mello, D.A. and V.S. Ananthanarayana. 2010. "A Review of Dynamic Web Service Description and Discovery Techniques". 2010 First International Conference on Integrated Intelligent Computing (ICIIC). 246-251.

5. Dong, W. 2007. "QoS Driven Service Discovery Method Based on Extended UDDI".. ICNC 2007. Third International Conference on Natural Computation. 5:317-324.

6. Elmasri and Navathe 2007. *Fundamentals of Database Systems. 5th Edition*. Pearson Education: New York, NY.

7. Kontonya, G. and I. Sommerville. 1998. *Requirement Engineering*. Wiley: West Sussex, UK.

8. M'Bareck, N.O.A. and S. Tata. 2007. "How to Consider Requester's Preferences to Enhance Web Service Discovery?".. Second International Conference on Internet and Web Applications and Services. 59.

9. Song, H. C. Doreen, A. Messer, and S. Kalasapur. 2007. "Web Service Discovery Using General-Purpose Search Engines". ICWS 2007. IEEE International Conference on Web Services. 265-271.

10. Sommerville, I. 2010. *Software Engineering 9th Edition*. Addison-Wesley, New York, NY. ISBN 13-978-0-13-705346-9.

11. Wang, Y. and E. Stroulia. 2003. "Flexible Interface Matching for Web-Service Discovery". *Proceedings of the Fourth International Conference on Web Information Systems Engineering*. 147- 156.

12. Weske, M. 2007. *Business Process Management*. Springer: New York, NY.

13. Karin Koogan Breitman and Julio Cesar Sampaio do Prado Leite. 2003. "Ontology as a Requirements Engineering Product". Proceedings of the 11th IEEE International Requirements Engineering Conference. 309-319 2003.

ABOUT THE AUTHORS

Oyebode Kazeem Oyeyemi holds a B.Eng in Electrical / Electronic Engineering from the University of Ado-Ekiti, Nigeria in 2009 with a second class upper division. He also holds a master's degree in Software Engineering from the University of the West of England in 2012. Presently, he is pursuing his Ph.D. research at the University of KwaZulu-Natal in South Africa, his research interest include; cloud computing, web services and

computer vision and cognitive radio network (SDR).

Esenogho Ebenezer received his Diploma, B.Eng. in Computer Engineering and M.Eng. Degrees in Electronic / Telecommunication Engineering from University of Benin in 2003, 2008, and 2012 respectively. Presently, he is pursuing his Ph.D. Program at the University of KwaZulu-Natal in South Africa, his research interest include; cognitive radio network (SDR), wireless sensor network, mobile computing and Artificial intelligence/Machine learning. He is a lecturer with the University of Benin, Benin City Nigeria. From 2011- Date.

SUGGESTED CITATION

Oyeyemi, O.K. and E. Ebenezer. 2013. "A Syntactically Enriched Tool Enabling Dynamic Binding to Web Services". *Pacific Journal of Science and Technology*. 15(2):124-131.

Pacific Journal of Science and Technology