ON THE USE OF WEB SERVICES TECHNOLOGY IN E-HEALTH APPLICATIONS

[#]JOHN B. OLADOSU, FUNMILOLA A. AJALA, AND OLUKUNLE O. POPOOLA

Computer Science and Engineering Department, Ladoke Akintola University of Technology, Ogbomoso, Nigeria

[#]*Correspondence Author* (johnoladosu@gmail.com **)**

ABSTRACT

This work takes a study on the Web Services—a technology that has come to change the future of Computing and e-commerce. Web Services is a distributed computing technology that offers interaction and collaboration among vendors and customers, with the vision of providing ubiquitous computing.

When you plug an appliance into the electricity socket, you don't worry about how the electricity generation and distribution takes place. All that is expected is uninterrupted power and the utility bill that you get at the end of the month! Similarly, Web Services will make computing resources, both hardware and software, accessible through the Internet just like electricity is made available. Web Services will do for computing what the Internet did for data. They would encourage a pay-per-usage model and make dynamic collaborations possible. One of the key definitions of Web Services is: *"Web Services are loosely coupled software components delivered over Internet-standard technologies."*

This paper considers the importance of web service as well as the success and the advantages of web service over the early technologies like: EDI, CORBA, and COM. Distinction is made between Web Service and Service. The components technologies of Web Service such as the WSDL, XML, UDDI, and, SOAP and how they apply to e-health applications, were examined. The creation of Web Service using the ASP.NET, Apache axis and Java2 Platform among other issues were also considered. We present a model for web services enabled infrastructure which is a framework for our e-health project at our research centre. In effect this paper will create a good knowledge for those new to web services as well as make more enlightenment to those already acquainted with the technology especially those developing applications for e-health services.

Keywords: *E-health, eXtensible Markup Language (XML), Simple Object Access Protocol (SOAP), Universal Description, Discovery, and Integration (UDDI),* Web services, *Web Service Definition Language (WSDL).*

I INTRODUCTION

Consider a scenario in which one needs to locate a particular pharmaceutical store in an area. It will be out of place to go out on the road and ask every person you met the way to the store. You might, instead, refer the Web site of the pharmacy on the Internet. If you knew the pharmacy's Web site, you would look it up directly and find the location through the store locator link. If not, you would go to a search engine and type out the name of the pharmacy in the language that the search engine was meant to recognize. After getting the location, you would find the directions to the store, and then go to the store[15].

The structure of Web Services is also very similar .Web Services provide for each of these previously described activities. Web service makes software application resources available over the networks in a standardized fashion. Other technologies have done the same thing, such as Internet browsers, which make web pages available using standard Internet technologies such as HTTP and HTML. However, these technologies are generally used

as a way for human users to view data on a web server and, on their own, are not well suited to enabling application-to-application communication and integration[1].

II HISTORY OF WEB SERVICE

In today's world of extreme competition on the business front, information exchange and efficient communication is the need of the day. Information Technology has grown by leaps and bounds, and sustained, not because it seems savvy, but because businesses can function more efficiently. This need for information exchange brings in another need to make this information selectively visible, and its possibility to be changed on the fly.

For example, with the introduction of the telephone came the need to have a directory service. This gave rise to the ever-popular "Yellow Pages," which brought the consumer and the provider closer to each other. The revolution of computerizing services of companies gave rise to isolated computer systems. Each company had software developed and customized to its specific needs. However, mergers, acquisitions, and business growths saw the need to share information stored in these isolated computer systems. The Internet did solve this problem to some extent. However, the Internet also opened many loopholes in security, making the owners of this information uneasy about the scope of their information's availability.

Hence, it became imperative that, for better B2B (Business-to-Business) communication, these systems must have the ability to link up to each other, grant permissions through a system other than the Internet, and which would make all the systems network with each other like an Intranet[1].

Today, companies rely on thousands of different software applications each with their own role to play in running a business. To name just a few, database applications store information about customers and inventories, web applications allow customers to browse and purchase products online and sales tracking applications help business identify trends and make decisions for the future. These different software applications run on a wide range of different platforms and operating systems, and they are implemented in different programming languages. As a result, it is very difficult for different applications to communicate with one another and share their resources in a coordinated way. Take, for example, a company that has its customer data stored in one application, its inventory data stored in another application, and its purchasing orders from customers in a third. Until now, if this company wanted to integrate these different systems, it had to employ developers to create custom bridging software to allow the different applications to communicate with one another. However, these sorts of solutions are often piecemeal and time consuming. As soon as a change is made to one application, corresponding changes have to be made to the other applications linked to it and to the bridges that link the applications together. [2].

To solve the problem of application-to-application communication, businesses need a standardized way for applications to communicate with one another over networks, no matter how those applications were originally implemented, web Services provide exactly this solution by providing a standardized method of communication between software applications. With a standardized method of communication in place, different applications can be integrated together in ways not possible before. Different applications can be made to call on each other's resources easily and reliably, and the different resources that applications already provide can be linked together to provide new sorts of resources and functionalities. Moreover, applications integration becomes much more flexible because Web services provide a form of communication that is not tied to any particular platform or programming language. The interior implementation of one application can change without changing the communication channels between it and the other applications with which it is coordinated. In short, Web services provide a standard way to expose an application's resources to the outside world so that any user can draw on the resources of the application.[12].

A. Why Web Service?

Web Services is probably not the first solution to such a problem. RMI, COM, CORBA, EDI, and ebXML also address the same problem space. So, what would make Web Services so special and different from the rest?

Web Services is based on the already existing and well-known HTTP protocol, and uses XML as the base language. This makes it a very developer-friendly service system. However, most of the above-mentioned technologies such

as RMI, COM, and CORBA involve a whole learning curve. New technologies and languages have to be learnt to implement these services. Also, Web Services is based on a set of standardized rules and specifications, making it more portable. This was not the case with the technologies mentioned earlier[3]. Since Web Services are the basis for Grid Services, understanding the Web Services architecture is fundamental to using GT3 and programming Grid Services.

Lately, there has been a lot of buzz about "Web Services", and many companies have begun to rely on them for their enterprise applications. So, what exactly are Web Services? To put it quite simply, they are *yet another* distributed computing technology (like CORBA, RMI, EJB, etc.) They allow us to create client/server applications. For example, suppose there is need to develop an application for a chain of stores. These stores are all around the country, but the master catalog of products is only available in a database at one central office, yet the software at the stores must be able to access that catalog. One could *publish* the catalog through a Web Service.

Information on a website is intended for humans. Information which is available through a Web Service will *always* be accessed by software, *never* directly by a human (despite the fact that there might be a human using that software). Even though Web Services rely heavily on existing Web technologies, they have no relation to web browsers and HTML[10].

B. Success of Web Service

Looking back over the years, it is hard to imagine networked computing without the Web. The reason why the Web succeeded where earlier hypertext schemes failed can be traced to a couple of basic factors: *simplicity* and *ubiquity*. From a service provider's (e.g. an e-shop) point of view, if they can set up a web site they can join the global community. From a client's point of view, if you can type, you can access services. From a service API point of view, the majority of the web's work is done by 3 methods (`GET`, `POST`, and `PUT`) and a simple markup language. The web services movement is about the fact that the advantages of the Web as a platform apply not only to information but to *services*[4].

Web service would have been too inefficient to be interesting a few years ago. But the trends like cheaper bandwidth and storage, more dynamic content, the pervasiveness and diversity of computing devices with different access platforms make the need for glue more important, while at the same time making the costs (bandwidth and storage) less objectionable[6].

III SERVICE AND WEB SERVICE

The Internet and the World Wide Web (WWW) are tremendous success stories and have changed the way we publish and communicate information in our modern society. Web services now start to add a new level of functionality on top of the existing Web and transform them from a place where we share and find data to a place where we find and share dedicated services and functionalities too. Finding suitable Web services that help to achieve a certain goal is widely considered as a key task in a shared global marketplace of services. Semantic description of what is provided by a Web service allows the automation of the process (or parts of it) and therefore dynamically adapting software systems using a service-oriented architecture. An efficient solution of the discovery problem allows for a cost-effective construction of software systems from pre-existing components whereby single elements in the system's architecture can be dynamically exchanged[7].

These often regard the terms *service* and *Web service* as synonymous. There is belief that these two terms are not equivalent and, furthermore, it is relevant to distinguish them and to explore their relation. This fundamental distinction is necessary in order to achieve scalable and realistic Semantic Web service Discovery.

Web services are technological means for accessing or specifying services offered by some specific provider. In a sense a Web service is an access point to services of a particular provider. Users are not specifically interested in Web services but rather in the services that can be delivered by a specific provider. Clients think in terms of services whereas providers advertise sets of services they are able to offer to their clients (using a technical entity Web service). Hence, the description of a Web service is the smallest unit of advertisement for providers. The notion of *service* is used in different communities and even within the same communities in various ways[7]. Many people have a different understanding of the term *service* which causes a lot of confusion and makes it hard to compare different work directly.

For example, in the business community a *service* is seen as *a* business activity that often results in intangible outcomes or benefits, while in Computer science the terms service and Web service are often regarded as interchangeable to name *a* software entity accessible over the Internet[9].

A. Service

Service is *a provision of value to a client in some domain*. A service represents the kind of entity, a user is interested in and that he wants to have discovered. As an example, let us consider a user who wants to book a train ticket from Lagos to Kaduna on a given date. The service he is looking for is the provision of a train ticket with the specified constraints. Such provision is independent on how the supplier and the provider interact, i.e., it does not matter whether the requester goes to a train tickets office or uses the train Web site to book his trip[7,8]. Take Health service for example. It deals with provision of healthcare by health caregivers such as medical practitioner to a patient.

B. Web Service

"Web service is a computational entity accessible over the Internet (using Web service standards and protocols)"[7]. For example, a railway company might provide a software component accessible via Web service standards, i.e., a Web service to request the booking of a trip. Thus, the Web service is an electronic means by which a client is able to request a specific service from a provider, but not the service itself. Therefore, we understand the term *Web service* as a means to request a service over the Internet, described using agreed standards such as using the Web service to request for e-health service over the Internet[8].

IV WEB SERVICES TECHNOLOGY

As the Internet grew from a forum for sharing information to a marketplace for doing business, a technology matured that allowed Computers to easily transact with each other. Out of these Internet roots, web service technology was born. The general goal of web services is to construct elements of business logic, services, which can be very easily used by other applications. The services themselves hide the complexity of their business logic from the consumers through simple interfaces that allow the services to be reused in many different applications. The service and the consumer are described as being loosely coupled, an approach that allows complex composite solutions to be developed through leveraging multiple web services[1].

Component Technology of Web Service

- eXtensible Markup Language (XML)
- Simple Object Access Protocol (SOAP)
- Web Service Definition Language (WSDL)
- Universal Description, Discovery, and Integration (UDDI)

1) XML (eXtensible Markup Language). This is the core language of web service technology. It is a universally agreed markup meta-language primarily used for information exchange. It provides a platform neutral way to describe the data connected with any service transaction. A good example of a markup language is the Hyper Text Markup Language (HTML)[1]. The beauty of XML lies in the fact that it is extensible. Simply put, XML is a set of predefined rules (syntactical framework) that is needed to follow when structuring your data. For a long time, programmers and application vendors have built applications and systems deployed in an enterprise that processes data that can be interpreted by the enterprise systems—essentially, data structured in a proprietary fashion. But as information exchange between applications and systems across enterprises became prevalent, it became very difficult to exchange data because the systems were never designed to accept data from external, unknown systems. XML provides a standard and common data structure for sharing data between disparate systems. Additionally, XML has built-in data validation, which guarantees that the structure of the data that is received is valid[17]. The importance of these features of XML can not be overemphasized in e-health applications.

2) SOAP (Simple Object Access Protocol). This is the preferred means by which an application invokes a web service. The protocol itself is written in XML. It is the method by which you can send messages across different modules. This is similar to how you communicate with the search engine that contains an index with the Web sites registered in the index associated with the keywords[1]. XML messages provide the common language

by which different applications can talk to one another over a network. To operate a Web service, a user sends an XML message containing a request for the Web service to perform some operation; in response the Web service sends back another XML message containing the results of the operation. Typically these XML messages are formatted according to SOAP syntax. SOAP specifies a standard format for applications to call each other's methods and pass data to one another. Note that other non-SOAP forms of XML messages are possible, depending on the specific requirements of the Web service. But, in any case, the sort of XML message and the specific syntax required can be found in the WSDL file, making the Web service generally available to any client application capable of sending and receiving the appropriate XML messages[3].

It can also be defined as a protocol specification that defines a uniform way of passing XML-encoded data. It defines a way to perform remote procedure calls (RPCs) using HTTP as the underlying communication protocol. SOAP arises from the realization that no matter how nifty the current middleware offerings are, they need a WAN wrapper. Architecturally, sending messages as plain XML has advantages in terms of ensuring interoperability [4].

This is of particular significance when web services applications are designed for e-health delivery services. Health records and information may be exchanged across heterogeneous platform, hence making interoperability a major concern.

3) WSDL (Web Service Definition Language). This is the specification of the interface that a web service exposes to consumers. It describes the set of operations that the service makes available. The WSDL is also written in XML. WSDL file provides a description (written in Web Service Description Language) of how the Web service is operated and how other software applications can interface with the Web service. WSDL file as the instruction manual for a Web service explaining how a user can draw on the resources provided by the Web service. WSDLs are generally publicly accessible and provide enough detail so that potential clients can figure out how to operate the service solely from reading the WSDL file. If a Web service translates English sentences into Yoruba, as in our related work[19], the WSDL file will explain how the English sentences should be sent to the Web service, and how the French translation will be returned to the requesting client. It is the method through which different services are described in the UDDI. See the diagram in Figure. 1, it maps to the actual search [1]. The diagram depicts a typical e-health application. See our related work[18].

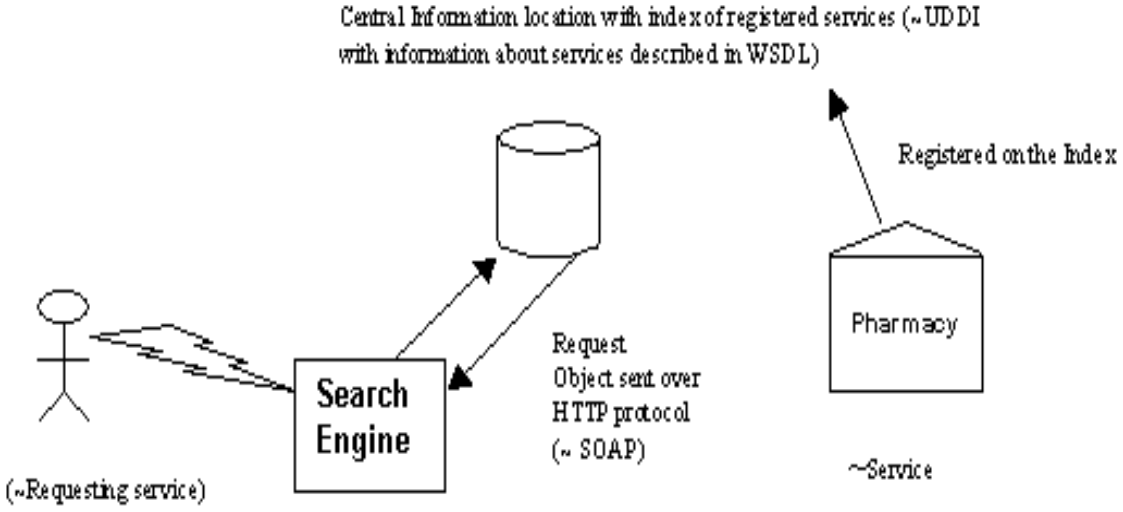

Figure 1: Web Services Components (Source [1])

4) UDDI (Universal Description, Discovery, and Integration). This is the global look up base for locating the services. In the example mentioned earlier, this is analogous to the index service for the search engine, in which all the Web sites register themselves associated with their keywords. It maintains a record of all the pharmacy store locations throughout the country[1]. It is one of the youngest and most rapidly developing standards in the web service family. It is an initiative designed to make it easier for you to locate web services on any server. With discovery files, the client still needs to know the specific URL location of the discovery file. Discovery files may make life easier by consolidating multiple web services into one document, but they don't provide any obvious way to examine the web services offered by a company without navigating to its website and looking for a .disco hyperlink[11].

The goal of UDDI, on the other hand, is to provide repositories where businesses can advertise all the web services they have. For example, a company might list the services it has for business document exchange, which describe how purchase orders can be submitted and tracking information can be retrieved. To submit this information, a business must be registered with the service.

V WEB SERVICES ARCHITECTURE

Figure. 2 is a diagram describing the Web Services Architecture:

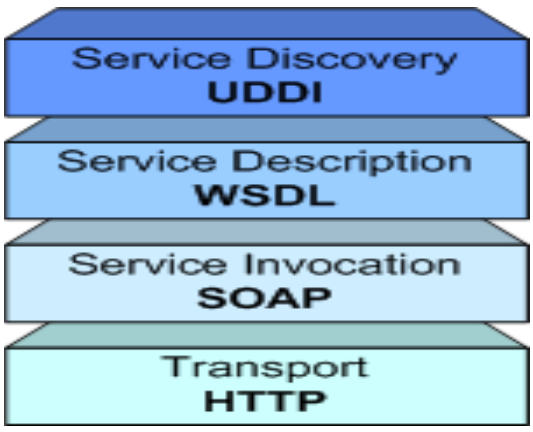

Figure 2: Web Service Architecture

1) Service Discovery: This part of the architecture allows us to find Web Services which meet certain requirements. This part is usually handled by UDDI (Universal Description, Discovery, and Integration). GT3 currently doesn't include support for UDDI. Our work in [18] on semantic healthgrid gives enough explanation about this.

2) Service Description: One of the most interesting features of Web Services is that they are *self-describing.* This means that, once a Web Service is located, one can ask it to 'describe itself' and tell what operations it supports and how to invoke it. This is handled by the Web Services Description Language (WSDL). A pharmaceutical ontology can give a detailed description of itself once discovered[18].

3) Service Invocation: Invoking a Web Service (and, in general, any kind of distributed service such as a CORBA object or an Enterprise Java Bean) involves passing messages between the client and the server. SOAP (Simple Object Access Protocol) specifies how we should format requests to the server, and how the server should format its responses. In theory, we could use other service invocation languages (such as XML-RPC, or even some *ad hoc* XML language). However, SOAP is by far the most popular choice for Web Services. This explains how the clients and server ends of our work on Doctor-Patient interaction in [19] communicate.

4) Transport: Finally, all these messages must be transmitted somehow between the server and the client. The protocol of choice for this part of the architecture is HTTP (HyperText Transfer Protocol), the same protocol used to access conventional web pages on the Internet. Again, in theory we could be able to use other protocols, but HTTP is currently the most used one[10].

VI WEB SERVICE AND ITS APPLICATIONS

Now that we have an idea of what Web Services are, programming Web Services will certainly be the next in mind to do. But before doing that, one needs to know that despite having a lot of protocols and languages floating around; Web Services programmers usually never write a single line of SOAP or WSDL. Once we've reached a point where our client application needs to invoke a Web Service, we *delegate* that task on a piece of software called a *client stub*. The good news is that there are

plenty of tools available that will generate client stubs automatically for us, usually based on the WSDL description of the Web Service.

A Web Services client doesn't usually do all those steps in a single invocation. A more correct sequence of events would be the following:

1. We locate a Web Service that meets our requirements through UDDI.
2. We obtain that Web Service's WSDL description.
3. We generate the stubs *once*, and include them in our application.
4. The application uses the stubs each time it needs to invoke the Web Service.

Programming the server side is just as easy. We don't have to write a complex server program which dynamically interprets SOAP requests and generates SOAP responses. We can simply implement all the functionality of our Web Service, and then generate a *server stub* (the term *skeleton* is also common) which will be in charge of interpreting requests and *forwarding* them to the service implementation. When the service implementation obtains a result, it will give it to the server stub, which will generate the appropriate SOAP response. The server stub can also be generated from a WSDL description, or from other interface definition languages (such as IDL). Furthermore, both the service implementation and the server stubs are managed by a piece of software called the *Web Service container*, which will make sure that incoming HTTP requests intended for a Web Service are directed to the server stub.

The steps involved in invoking a Web Service are described in Figure. 3.

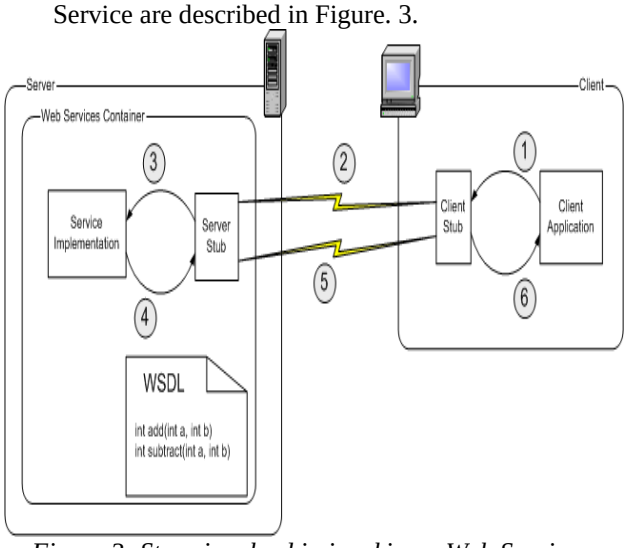

Figure 3: Steps involved in invoking a Web Service

Supposing a Web Service has been located, and the client stubs generated from the WSDL description. Furthermore, the server-side programmer will have generated the server stubs.

1. Whenever the client application needs to invoke the Web Service, it will actually call the client stub. The client stub will turn this 'local invocation' into a proper SOAP request. This is often called the *marshaling* or *serializing* process.
2. The SOAP request is sent over a network using the HTTP protocol. The Web Services container receives the SOAP requests and hands it to the server stub. The server stub will convert the SOAP request into something the service implementation can understand (this is usually called *unmarshaling* or *deserializing*)
3. The service implementation receives the request from the service stub, and carries out the work it has been asked to do. For example, if we are invoking the int add (int a, int b) method, the service implementation will perform an addition.
4. The result of the requested operation is handed to the server stub, which will turn it into a SOAP response.
5. The SOAP response is sent over a network using the HTTP protocol. The client stub receives the SOAP response and turns it into something the client application can understand.
6. Finally the application receives the result of the Web Service invocation and uses it[10].

Our e-health work in [19] follows the procedures outlines above.

VII CREATING WEB SERVICE

Web services have the potential to dramatically simplify the way distributed applications are built. They might even lead to a new generation of applications that seamlessly integrate multiple remote services into a single web page or desktop interface. However, the greatest programming concept in the world is doomed to fail if it isn't supported by powerful, flexible tools that make its use not only possible but also convenient. Fortunately, ASP.NET doesn't disappoint. It provides classes that allow creating a web[11]. And Microsoft .NET marketing has created a huge hype about its Web Services[13]. Also, The IBM has given a lot of codes to the Apache group called the Apache AXIS which features better performance and makes it to play important role in the creating of

Web Service and the Java 2 Platform code among others.

A. Limitations of Web Service Solutions

Web Services solutions provide functional interoperability using standards like SOAP and WSDL. Web Services are described using WSDL, but WSDL does not support operational semantic. Web Services do not support message semantic [20]. In order to support operational and message semantics, semantic web services is required

B. Semantic Web Service Solution

Semantic Web Service Solution use Semantic mediation/reconciliation to convert healthcare messages defined in one standard format into another as realized with the scope of the Artemis project [20]. Archetype based semantic interoperability of EHR standards as realized within the scope of Artemis is a viable solution approach. HL7 categorization of healthcare events are used to annotate web services functionality because it exposes the business logic as proposed in Artemis. Web Services messages can be semantically enriched using Archetypes as proposed in Artemis [20].

VIII OUR PROPOSED MODEL OF WEB SERVICES FOR E-HEALTH SERVICES PROVISIONING

Figure 4 shows the overall system architecture of the proposed infrastructure. The diagram shows how the components are logically and functionally related. The design is such that applications in each module could invoke the services linked to it without human intervention. Modules indicated as Web Services would be hosted by our local server while External Web Services may exist as Internet services to be invoked at runtime. The overall functional structure of the framework is summarised as follows: E-health services are automatically detected by user's mobile devices. A patient requests E-health services and selects from the list of services offered. The capability of the patient to pay for service through the National or Regional health insurance scheme is ascertained either by authenticating the user's credentials in the health insurance database or by registering him/her as a new insurance holder. If the patient is not registered and cannot be newly registered on the health insurance scheme, the request for e-health services is rejected. If insurance authentication is valid, the e-health service is granted. The e-health services could be as simple as a patient-doctor consultation interaction and drug recommendation and delivery from the pharmacy to the patient's doorstep (the parcel delivery services handles this). Services could involve some other healthcare practitioners such as medical laboratory diagnosis. In this case the patient would have to visit a medical lab or the service is brought to him. The results of any such secondary care would be loaded to patient's online medical record to enable next level of medical care. It is assumed that patient's medical history is available for anywhere/anytime access to facilitate the healthcare services.

(Figure 4: Model of the Proposed Web Services Oriented E-health Framework at page 100)

IX CONCLUSION

Currently, first-generation web services are being used to bridge the gap between modern applications and older technologies. For example, an organization might use a web service to provide access to a legacy database. Internal applications can then contact the web service instead of needing to interact directly with the database, which could be much more difficult. Similar techniques are being used to allow different applications to interact. For example, web services can act as a kind of "glue" that allows a payroll system to interact with another type of financial application in the same company. Second-generation web services are those that allow partnering companies to work together. For example, an e-commerce company might need to submit orders or track parcels through the web service provided by a shipping company. Second-generation web services require two companies to work closely together to devise a strategy for exposing the functionality they each need. Second-generation web services are in their infancy but are gaining ground quickly.

The third generation of web services will allow developers to create much more modular applications by aggregating many different services into one application. For example, you might add a virtual hard drive to your web applications using a third-party web service. You would pay a subscription fee to the web service provider, but the end user wouldn't be aware of what application functionality is provided by you

and what functionality relies on third-party web services. This third generation of web services will require new standards and enhancements that will allow web services to better deal with issues such as reliability, discovery, and performance. These standards are constantly evolving, and it's anyone's guess how long it will be before third-generation web services begin to flourish, but it's probably just a matter of time. Already, one can use third-party web services from companies such as eBay, Amazon, and Google.

These web services act as part of a value-added proposition and may eventually evolve into separate cost based services. E-health care delivery system will thrive well on third generation web services. Currently, we are developing a number of web services applications to facilitate e-health care delivery especially for the African continent. Interoperability, state management and Security/privacy among others issues are of serious concern in this research direction.

REFERENCES

[1] Lakshmi Ananthamurthy (2008): "Introduction to Web Service." Available at http://www.developer.com/services/article.php/1485821

[2] Tony Baer, Ron Schmelzer: "The Elements of Web Services" (Application Development Trends, 2 December 2002), available at http://adtmag.com/articles/2002/11/30/the-elements-of-web-services.aspx

[3] http:www.dev2dev.bea.com. Official website for the Developer.

[4] V. Vasuderan (2001): A Web Services Primer, available online at http://webservices.xml.com/pub/a/ws/2001/04/04/webservices/

[5] Wikipaedia (2008)

[6] M. Paolucci et al. (2002): "Semantic Matching of Web Services Capabilities," *Proc. Int'l Semantic Web Conf.* (ISWC), LNCS 2342, Springer Verlag, 2002, pp. 333-347.

[7] Dieter Fensel, Uwe Keller, Holger Lausen, Axel Polleres and Ioan Toma (2005): "***WWW*** *OR* ***WHAT IS WRONG WITH WEB SERVICE*** DISCOVERY," Position Paper for the Workshop on Frameworks for Semantics in Web Services, Innsbruck, Austria, June 2005, available at http://www.w3.org/2005/04/FSWS/Submissions/50/WWW_or_What_is_Wrong_with_Web_service_Discovery.pdf.

[8] C. Preist (2004): "A Conceptual Architecture for Semantic Web Services," in Proceedings of the International Semantic Web Conference 2004 (ISWC 2004), November 2004 available at http://www.hpl.hp.com/techreports/2004/HPL-2004-215.pdf.

[9] Z. Baida, et 'al (2004): "A Shared Service Terminology for Online Service Provisioning," Proceedings of the Sixth International Conference on Electronic Commerce (ICEC04), Delft, The Netherlands.

[10] http://ww.gdp.globus.org

[11] M. MacDonald(2005): Beginning ASP.Net 2.0 in C#

[12] Mc Grawhill Company Inc.(2005): "What the heck are Web Services?" Business Week

[13] C. Peiris (2005): "Creating a .net Web Service," Caulfield, Australia.

[14] D. Almaer (2002): "Creating Web Services with Apachel Axis," O' Reilly Inc.

[15] Miyoo Tsanang Yves Stephan (November 2006): "Entwurf eines Web Services basiertes Workflow Management System (WfMS)", available at http://iaks-www.ira.uka.de/calmet/stdip/DA-Miyoo-final.pdf

[16] S. Geric, et' al (2006): "Prerequisites for successful Imlementation of Service-Oriented Architecture," Varazdin, Croatia.

[17] D, Hunter, et' al (2007): Beginging XML, 4th Edition, Wiley, Indiana, Canada.

[18] Emuoyibofarhe O.J., Lasisi O.A. and Oladosu J.B., "Towards the Semantic HealthGrid: A Pharmaceutical Ontology Development," in proceedings of Second International Conference on Application of Information and Communication Technologies to Teaching, Research and Administration; 2007; pp. 14 – 22.

[19] John B. Oladosu, John T. Bamigbala, Samuel O. Adetutu and Justice O. Emuoyibofarhe, (2008): "A Yoruba-English Language Translator for E-Health Care Delivery System via Doctor Patient Mobile Chat Application on Mobile Devices (Pdas)," in under review.

[20] Olugbara, O.O., (2007): "*Requirements Engineering Framework for Information Utility Infrastructure for Rural e-Healthcare Service Provisioning*", Ph.D. Research Proposal Centre for Mobile e-Services for Development, Department of Computer Science, University of Zululand, RSA.

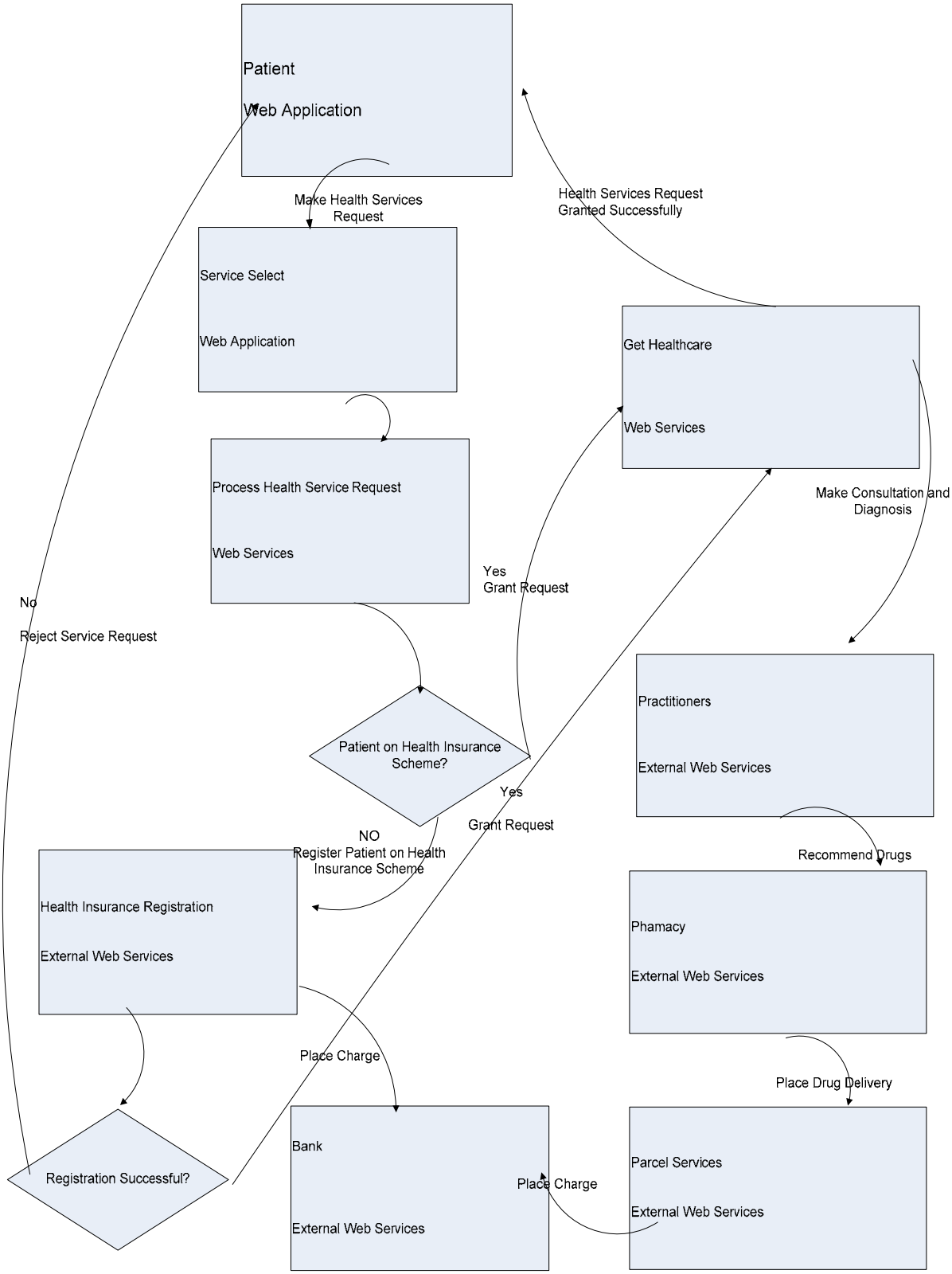

Figure 4: Model of the Proposed Web Services Oriented E-health Framework