WEB SERVICES SOLUTION ON REPLICATION ORIENTED ARCHITECTURE (ROA)

G.O. Ekuobase and E.A. Onibere
Department of Computer Science,
University of Benin, 300001,
Benin City, Nigeria

ABSTRACT
ROA an acronym for Replication Oriented Architecture is speculated as capable of attenuating the scalability defect of Web Services and help application programmers build scalable Web Services solution. For this claim to be considered, it is ordinarily necessary to establish that Web Services solution can be built on ROA; and this is the essence of this paper. We have introduced the implementation equivalent of ROA for a Web Service using Java technologies. The Java technologies were Servlet, Enterprise Java Bean (EJB), Java Messaging Service (JMS) and SOAP with Attachment API for Java (SAAJ). A prototype ATM system has been built using the Java implementation equivalent of ROA for a replica size of three. The development environment was NetBeans Integrated Development Environment version 6.0 (NetBeans 6.0). Sun Java System Application Server Platform Edition 9 was the build and runtime environment for the three service replicas as well as the replication logic; all in the same computer system. The successful deployment and consumption of the prototype system shows that Web Services solution can be built on ROA.

INTRODUCTION
Ekuobase and Onibere, (2011), asserted that the poor scalability of Web Services, if not properly addressed, may not only pose a serious threat to its survivability but may also lead the Information Technology (IT) community into shambles as Web Services has the potential of becoming the ubiquitous platform technology for next-generation computing systems, having an unprecedented number of active systems dependent on it, and hence introduced ROA and claimed that ROA can attenuate the scalability defect of Web Services and help application programmers build scalable Web Services solution. It is however imperative to establish that web services solutions can be built on ROA, for this claim to be given any consideration. This paper concerns itself with demonstrating how Web Services solution can be realized on ROA. We were careful to use a test (small-to-medium sized) software development project of an application with little or no computational capability because of the fears raised in Ekuobase and Onibere, (2011).

The Automated Teller Machine (ATM) System is the application system of choice. The ATM system was selected because it requires less computational capability, easy to appreciate, conversational, and incorporates the use of back end system (database). Besides, the ATM system is in reality of high activity and great economic importance. Also, with the proliferation of hand
Wed services solution on replication oriented architecture  

Ekuobase and Onibere

held devices, emergence of ubiquitous systems and the move towards cashless society, the ATM system reflects two important domains – financial transaction (be it commerce or governance) and ubiquitous systems domains – that will be fully dependent on Web services.

MATERIALS AND METHODS

We began the software development project by first selecting an appropriate software development methodology (SDM) for it. We observed that no SDM is a failure and none is a silver bullet; they all have their strengths and weaknesses and project domain or nature of software projects they suite most, (Ekuobase, 2006), as summarized in Table 1. From table 1, it became obvious that the agile software development approach, (Ekuobase, 2006; Agile Alliance, 2001), is most appropriate for the software development task. The tailored agile approach, depicted in Figure 1, was adopted as the software development process.

The software development tools used were basically software and hardware tools. These tools and the roles they played in the research are highlighted as follows:

**Hardware Tools:** the tools in this category were the computer and its peripherals. In particular, Dell notebook (Intel® CPU T2080 @ 1.73 GHz x 2, Duo Core; 794 MHz, 2.0 GB RAM and 150 GB Hard Disk) was used for developing the prototype system. It also served as host to the software tools used in this research.

**Software Tools:** we choose to discuss software tools under Operating System, Development Platform, Language, Integrated Development Environment (IDE), and Packages.

**Operating System:** We settled for Microsoft Windows XP Professional Version 2002 Service Pack 2 which work seamlessly with the other software tools used in the research and for which our institution holds a multi-user license. The Operating System enabled our applications and other software tools used interact with the machine and tap its computational and peripheral resources.

**Development Platform:** The choice of Java EE (Jendrock et al., 2006) as our Development platform for building the Web services solution, is not only because we were biased towards the platform as a result of our priori comfortable Java programming experience but also because as it stands today, Java EE

Table 1: Software Projects and the basic Software Development Approaches

<table>
<thead>
<tr>
<th>Agile Software Development Approach</th>
<th>Traditional Software Development Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Visible systems</td>
<td>Legacy/embedded systems</td>
</tr>
<tr>
<td>Low risk projects</td>
<td>High risk projects</td>
</tr>
<tr>
<td>Blurred and unstable requirements</td>
<td>Explicit and fairly stable requirements</td>
</tr>
<tr>
<td>Small to medium sized projects</td>
<td>Large and complex projects</td>
</tr>
<tr>
<td>Time-to-market driven</td>
<td>Product quality driven</td>
</tr>
</tbody>
</table>

Fig. 1: An Agile Implementation Model

- **Operating System:** We settled for Microsoft Windows XP Professional Version 2002 Service Pack 2 which work seamlessly with the other software tools used in the research and for which our institution holds a multi-user license. The Operating System enabled our applications and other software tools used interact with the machine and tap its computational and peripheral resources.

- **Development Platform:** The choice of Java EE (Jendrock et al., 2006) as our Development platform for building the Web services solution, is not only because we were biased towards the platform as a result of our priori comfortable Java programming experience but also because as it stands today, Java EE

and Microsoft.NET platforms remain the most dominant application developer’s platform for enterprise applications in general and Web Services solutions in particular (Birman, 2005; Williams, 2003; Vawter and Roman, 2001) but Microsoft.NET platform is proprietary.

- **Language:** Here, we mean programming language, modelling language and Database Management System (DBMS). Obviously, our core programming language of implementation is Java 5.0 (van der Linden, 2004; Horstmann and Cornell, 2005a, 2005b; Deitel and Deitel, 2010) since it is the only language our choice platform supports. Java was used for coding. The Structured Query Language (SQL) being the de facto language for interacting with DBMS was adopted. Our choice of Java DB as our DBMS for building and managing the databases was based on seamless compatibility with both Java and NetBeans – our Integrated Development Environment of choice. The Unified Modelling Language (UML) was used to design the applications but not without some modifications since object oriented analysis and design which UML stands for, fall short of the concept of service orientation which requires additional elements for it to be realized (Zimmermann et al., 2004; Arsanjani, 2004).

- **Integrated Development Environment (IDE):** We have several IDEs that support Java. They include JBuilder, JCreator, Eclipse, and NetBeans. We choose NetBeans (NetBeans 6.0) on the ground of familiarity though it is not in any way less powerful than the others. Normally, IDE makes programming easy, nimble and interesting.

- **Packages:** the software package – Argo UML (Ramirez et al., 2006; Tolke and Klink, 2006) was used for the UML design.

The following assumptions about the ATM system under construction were made:

- a) The ATM system does not concern itself with the working and nature of the ATM terminals (screen, card reader, keypad, cash dispenser and deposit slot).
- b) The ATM system serves as a base station for several ATM terminals with the sole duty of interfacing with the user.
- c) The ATM system does not include security features sufficient for real life deployment.
- d) Although ATM system provides four basic services as depicted in Figure 2 and Figure 3, only the Fund Transfer service was built since it encompasses the other basic ATM services.

![Fig. 2: Use Case Diagram for the ATM System](image)

The service equivalent of the ATM system is modeled in Figure 4. Observe that the ATM was replaced in the service model by two SOAP routers - Web Services client and Web services solution that communicate in a SOAP request-response manner. These in turn communicate with backend (logic) objects directly as in the object model and thus same internal state as its object equivalent is maintained. The implication of this is that the logic state of a Web Services solution is basically the same as that of its object counterpart and can be modeled in the same manner. A significant difference is however evident during communication with external (Web Services) systems, when it has to perform the ritual of invocation, serialization/deserialization and deployment (Hansen, 2007). At this stage UML modeling symbols proved inadequate and we had to adopt the SOAP router class symbol in Hansen, (2007), which is reproduced in
Figure 5. This compound symbol aggregates UML symbols with the exception of the SOAP symbol which emphasises the basic transportation mode of Web Services. The UML symbols usage is consistent with the original UML definitions (Rumbaugh et al., 1999).

Java Application Programming Interfaces (APIs) were selected to design the implementation equivalent of ROA, depicted in Figure 7, for a single set of service replica. The Java EE programming models selected were Servlet, Enterprise Java Bean (EJB), Java Messaging Service (JMS) and SOAP with Attachment API for Java (SAAJ) (Crawford and Farley, 2006; Jendrock et al., 2006).

The EJB-endpoint is a service replica – the Web Services endpoint. As figure 7 shows, the replication logic of ROA is implemented as Java Messaging Service (JMS) on Java Servlet. Besides strength of realization, these APIs ensures portability, loose coupling, and very low overhead of WSTPRL – the virtual WS-Server (Crawford and Farley, 2006). In particular, a Servlet is persistent i.e. instead of shutting down at the end of each request the servlet can remain loaded, ready to handle subsequent requests (Crawford and Farley, 2006; Jendrock et al., 2006); and this guarantees a continuous active state of the WSTPRL. As depicted in figure 7, servlet accepts request from the WS-Client and passes it (as JMS message producer) to the JMS messaging domain (Queue) from where the EJB-endpoints, which also serves as JMS message consumer, fetches messages asynchronously. This fetching by an EJB-endpoint is done as soon as it has no “job on hand”. Clearly, the JMS provider can handle any (increasing) number of service replicas as though it is just one i.e. seamlessly accommodates replica growth.

We also made use of a small sized database built using Java DB – an open source Relational Database Management System (RDBMS) for 20 ac-
Fig. 4: Equivalent Conventional Service Model for our ATM system using Modified UML.
Web services solution on replication oriented architecture

Ekuobase and Onibere

Fig. 5: Symbol for modeling SOAP Router Class

Data-base

User:  ATM:  Fund Transfer:  Screen  Input-Stream  Database  Account

Fig. 6: Sequence Diagram for Fund Transfer ATM Service

Wed services solution on replication oriented architecture

Ekuobase and Onibere

Fig. 7: ROA Implementation Equivalent using Java Technology

Fig. 8: AccountsTable of BankDatabase in Netbeans (Before service consumption by client)
count holders with varying fictitious amount in the accounts; as captured in figure 8. Each account has three fields: account number, account pin and available amount.

Clearly, we needed to consume the ROA’s service and have the transferred amount subtracted from the sender account and added to the receiver’s account. This we taught sufficient to establish that Web Services solution can be built on ROA. The consuming client, a simple Java application, was built using the JAX-WS and Java Server Faces API for Java (Deitel and Deitel, 2010; Hansen, 2007; Crawford and Farley, 2006; Jendrock et al., 2006).

RESULTS AND DISCUSSION
The Web Services client was launched and the Web Services Solution built on ROA (codes will be supplied on request due to limited paper size) already deployed to the server was successfully consumed. The correct transfer of 2000 from account 12198 (1st row) to account 21298 (2nd row) is captured in figure 8.
row) as required by the client (compare Figure 8 and Figure 9) established this fact. This shows that ROA is not only implementable but also a realistic server side architecture for building scalable Web Services solution.

CONCLUSION

ROA is a server side architecture capable of attenuating the scalability defect of Web Services. The feasibility of building Web Services solution on ROA is now established. This was demonstrated using Java technology. In particular, we exposed specific Java APIs that can be used to build Web Services solution on ROA. These APIs are Java Servlet, EJB, JMS and SAAJ. The Web Services solution built on ROA successfully deployed and was consumed by a service client application. ROA is a realistic server side architecture for building Web Services solution.

REFERENCES


