EVALUATION OF ARCHITECTURE STYLES FOR CLOUD COMPUTING APPLICATIONS

Sanaa Mohammed Abduljalil, Osman Hegazy, Hany Hammam

Faculty of Computers and Information, Cairo University, Cairo, Egypt
Lane Computer Science and Electrical Engineering Department, College of Engineering and Mineral Resources, West Virginia University, Morgantown, West Virginia 26505, USA
E-mail: Sana_jalil1@yahoo.com, Osman.hegazy@gmail.com, hammar@wvu.edu

ABSTRACT

Cloud computing is becoming popular and IT giants have started evolving toward networks and different cloud services that serve different organizations. Software as a service (SaaS) is the most mature category of cloud services [11], since it evolved from the application-service-provider model of software hosting. As our use of SaaS increases, future solutions will likely require more frequent data exchange with lower tolerance for failure. We also look to improve scalability and modifiability, which currently require a great deal of work. These requirements led us to develop a SaaS architecture that will help us shift to a more strategic view of SaaS and enable faster, more standardized implementations. Using the SaaS architecture, we can build the cloud application independent of any platform. The architecture must be both prescriptive and descriptive. In this paper, we present a simple architecture style for cloud computing applications that enhance certain quality attributes. The style combines both the Layered architecture and the Service Oriented architecture styles. This style will help developers in creating future multitenant, on demand, scalable services that are suitable in cloud computing. We illustrate this style on the design of a document management system (DMS) application. We evaluate the scalability, maintainability, and reliability of the resulting application architecture and propose techniques to enhance the quality attributes of the resulting architecture.

Keywords: Cloud computing, Software Architecture Styles. Software as a Service SaaS.

1. INTRODUCTION

There are few questions that arise whenever you start thinking about deploying an application to cloud. Will all applications run in the Cloud? Should you attempt to port all of your existing applications to the Cloud? Should all your new applications be developed in the Cloud? Some applications will be ideal candidates to be ported to a cloud platform, developed on a cloud platform, or hosted on a cloud infrastructure, while other applications will be poor cloud candidates. In this case, the standard architectural answer “it depends” can be applied to all of the preceding questions. Practically every application potentially could exist either partially or fully in the cloud; the only caveat to this are the trade-offs in an application's attributes and, possibly, functionality that you might be willing to make to move it to the Cloud.

The software industry is currently adopting the Software-as-a-Service deployment model in many application domains. SaaS applications are hosted on Internet servers by a provider instead of being downloaded and installed locally on the user’s computer as shown in figure 1. SaaS applications typically provide license on a pay-per-use basis, instead of being bought by a user. A special kind of SaaS offering is a multi-tenant software application. It serves multiple tenants (e.g., companies or non-profit groups) from a single application instance. Furthermore, for all tenants the software runs on the same infrastructure (i.e., containers, virtual machine, operating system, hardware), runs from the same code base, and can thus be maintained centrally.

The contributions of this paper are as follows:
1. Propose an architectural style that combines SOA, layered architecture style, and multi-tenants architecture style.
2. Apply this style to an open source application
3. Evaluate the instantiated architecture based on the quality attributes such as scalability, usability, and reliability.
SOA lets you design, build, deploy and integrate services independent of applications and the computing platforms on which they run [1]. These services are then linked together through defined business processes to form composite services, applications and composite applications to perform complete business functions. Service-Oriented Architecture (SOA) is not a new concept but a different approach to designing and building systems that are flexible and adaptable to support a dynamic business environment.

This paper is organized as follows: Section 2 briefly describes the concepts of architecture styles with some examples. Related work is summarized in section 3. Section 4 presents the proposed architecture style. In section 5, we describe a case study of instantiating the proposed style. Section 6 provides an evaluation of the instantiated architecture. Conclusion and future work are presented in section 7.

2. BACKGROUND

Software-as-a-Service (SaaS)
A single copy of software can be made available to consumers on demand as a shared service accessible over remote network location and charged on subscription or pay-per-use basis [9].

Cloud Computing
Cloud provides a dynamically scalable, abstracted computing and storage infrastructure that is typically based on a virtualized, distributed, fault tolerant, parallel computing architecture. Via Cloud, users can leverage the power of highly distributed Internet or the high computing power made available by grid engines [8][12]

Layered Architecture
Layered architecture focuses on the grouping of related functionality within an application into distinct layers. Functionality within each layer is related by a common role or responsibility. Communication between layers is explicit and loosely coupled. Layering your application appropriately helps to support a strong separation of concerns that in turn, supports flexibility and maintainability.

Service Oriented Architecture
A service-oriented architecture is essentially a collection of services. These services communicate with each other. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. Some means of connecting services to each other is needed.

3. RELATED WORK

Research on cloud computing have focused on the architecture of cloud environments to support the notion of everything as a service. Few recent research articles focused on architectural styles for cloud computing applications [2][3]. The authors in [2] proposed the SPOSAD style, an architecture style for multi-tenants cloud computing applications. The SPOSAD style, presented in [2] for multi-tenants cloud computing, requires the architect to use a data architecture where resources are shared. For high scalability, the style also requires the architect to use a data partitioning scheme to physical servers that best allows for scaling out large amounts of data. This can for example involve tenant specific partitions or local partitions for different user groups of a single tenant. Architects have to make several decisions and trade-offs when developing multi-tenant applications. They have to define the degree of customization that the application should support. More customizability implies more complicated development and makes the use of shared resources more difficult. Thus, highly customizable applications are not well suited for a multi-tenant architecture [3].

The development of cloud computing application architectures is a complex process. The architecture of a software system is almost never limited to a single architectural style. It is often necessary to have a combination of architectural styles that make up the complete system. For example, you might have a SOA-based design composed of services developed using a layered architecture approach and a multi-tier architecture style. A combination of architecture styles is also useful if you are building cloud application, where you can achieve effective separation of concerns by using the layered architecture style. This will separate your presentation logic from your business logic and your data access logic. Your organization’s security requirements might force you to deploy the application using either the 3-tier deployment approach, or a deployment of more than three tiers. The presentation tier may be deployed to the perimeter network, which sits between an organization’s internal network and an external network. On your presentation tier, you may decide to use a separated presentation pattern (a type of layered design style). You might also choose a SOA architecture style, and implement
message-based communication, between your Web server and application server. When developing cloud computing applications, many factors will influence the architectural styles you choose. These factors include the capacity of your organization for design and implementation; the capabilities and experience of your developers; and your infrastructure and organizational constraints. There is a need for cloud computing applications architecture styles that can facilitate the process developing applications that can be implemented and deployed on multiple cloud computing platforms.

4. CLOUD APPLICATION ARCHITECTURE DESIGN

The architecture of cloud applications can differ significantly from traditional application models and, as such, implementing cloud applications can require a fundamental shift in application-design thought processes. Cloud applications can often eliminate the need to install and run the application locally, thereby reducing the expenditure required for software maintenance, deployment, management, and support. This type of application would be considered Software as a Service (SaaS) application. An alternative to this would be the Software plus Services (S+S) model. This is the hybrid between traditional application development and a full SaaS implementation. S+S applications typically use rich client applications that are installed on a client’s side as an interface into externally hosted services. S+S often includes the ability to interact with an application in an offline mode, and sync back to a central service when required. The core of this paper is to propose an application architecture style that combines Service Oriented architecture and multitier architecture styles. Deployment on the cloud needs a well-defined flexible and yet simple architecture style especially “Application as A service”. In its most general form, an SOA application is composed of multiple services that communicate with each other via messages over a distributed ESB infrastructure platform. Cloud enabled SOA applications have a number of characteristics that impose particular requirements on the underlying ESB infrastructure over which SOA applications are deployed [13]. Figure 2 provides a description for a generic architecture style and shows the noticeable relation between multitier architecture styles that improves the separation of concerns, and enables updating separate tiers, and SOA that links services together to perform a complete business function. In software engineering, multi-tier architecture (often referred to as n-tier architecture) is logically separate processes [3]. For example, an application that uses middleware to service data requests between a user and a database employs multi-tier architecture. On the other hand, SOA causes flexible and reusable services. By breaking up an application into services, developers only have to modify or add a specific service, rather than have to rewrite the entire application over. [14].

4.1 Logical View:

Figure 2 shows the logical view of the proposed architecture style. The architecture layers and the components in each layer are described in the following paragraphs.

4.2 Components:

Client Layer:
Users access the system through the client tier and access the application and database tier through the presentation tier, which is responsible for providing the user interface.

Application Logic Layer:
For business transactions, all computations are made in the application tier, which retrieve and persist the involved data in the database tier. The components of application logic tier are as follows:

- **Client Controller**: It allows software to request data and computations from one or more services through a direct or indirect interface. Cloud APIs most commonly expose their features via REST and/or SOAP. Vendor specific and cross-platform interfaces are available for specific functions. Cross-platform interfaces have the advantage of allowing applications to access services from multiple providers without rewriting, but may have less functionality or other limitations vs vendor-specific solutions [4].

- **Internet Service Bus**: This is similar to an enterprise service bus, but with applications hosted in the cloud instead of on an enterprise network. ISB provides a platform for creating and deploying composite applications that integrate services that other sites provide. A core concept of ISB is the use of Uniform Resource Identifiers (URIs), URIs present application integration points. The team develops an ISB application by associating policy and function with URIs. The composite application is a set of URIs, policies, and functions. The ISB provides an identity and access function for controlling which messages can be sent to a URI,
and by whom. Moreover, the policy controls the routing of logic through applications and services in the cloud [5].

- **Load Balancer**: Load balancing is the key component to a successful cloud based implementation. The goal of a cloud-based architecture is to provide some form of elasticity, the ability to expand and contract capacity on-demand. The implication is that at some point additional instances of an application will be needed in order for the architecture to scale and meet demand. That means there needs to be some mechanism in place to balance requests between two or more instances of that application. The mechanism most likely to be successful in performing such a task is a load balancer. As is required by even the most basic of cloud computing definitions, the end user is abstracted by the load balancer from the actual implementation and needs not care about the actual implementation. The load balancer makes one, two, or two-hundred resources - whether physical or virtual - appear to be one resource; this decouples the user from the physical implementation of the application and allows the internal implementation to grow, to shrink, and to change without any obvious affect on the user.[7]

- **Web Services**: A web service is the externally visible part of a specific business function that is well-defined, self-contained, and does not depend on the context or state of other services and exposed to the web via a well-defined interface and invoked via standard web protocols.

- **Controller**: is in charge of the interaction between the data model and web services. Controller checks the rights in order to preserve security and call data model methods to get a response from it.
  
  (a) **Data Management Layer**
  
  It retrieves tenant-specific metadata from the database and adapts the application accordingly; on the other hand the data needs a special arrangement to enable updates in multi-tenant DB [2].

  (b) **Multitenant Database Layer**
  
  The multi-tenant database store business data as well as meta-data persistently. To maximize resource sharing, a single database should host the data of all tenants that avoids overheads for memory consumption and administration [2].

5. **THE QUOTERO CASE STUDY**

This part shows a re-construct for an application as service named Quotero based on Service Oriented Architecture and N-tier architecture. Quotero is fully open source DMS based on service oriented architecture. It is a document management system (DMS) whose aim is to provide its users with the easiest as well as the most powerful document production application. The principle mission of Quotero is to make operations like producing and accessing documents, which are highly frequent tasks in modern organizations easier. [15] Quotero improves and speeds up the procedure of processing and sending documents to the colleagues, customers and partners in a easy, safe and reliable way thus helping the customer save the time that is usually wasted in retrieving, editing and sending these documents. Quotero offers new ways of using document management systems and supplies the user powerful and trustworthy solutions for simplifying document management. Here I will describe the architectural elements of the Quotero architecture intent of the proposed architectural style described in the previous section. Client contains the user oriented functionality responsible for managing user interaction with the system, and generally consists of components that provide a common bridge into the core business logic encapsulated in the business layer. Client Controller is controller which allows any clients to call remote methods from web services. Internet Service Bus the composite application is a set of URIs, policies and functions. The ISB provides an identity and access function for controlling which messages can be sent to a URI, and by whom. The identity and access function is an example of associating a policy with a URI. Load Balancer the load balancers maintain an array of resources that can be requested on demand. When a user requests a resource, the load balancer chooses the highest resource based on the resource priority calculated, where resources are continually monitored checking availability (based on some messaging techniques to ensure the presence information, whether the service is available or unconnected, as well as the common format for services to establish connection) and performance (closest ip address that may serve the requester).

Based on these two factors, the resource table is updated calculating new priorities for resources in order to update the priority table. Figure 3 shows the flowchart of load balancer. **Quotero Web Services** are the layer which allows the clients make calls to remote methods. These methods should be as simple as possible and just call the controller.
The web services that Quotero contains are:

- **Administration Service** is used to manage role, authentication source, users, locked, documents, documents owners and sessions.
- **Log Service** is used to get some reports about document hits, document transactions, and entity information or user actions.
- **Search Service** is used to make searches in order to find documents into Quotero.
- **Security Service** is used to get authentication sources, users and groups. It's also used to check privileges for each web services call.
- **Studio Service** is used to manage document types, metafeeds and workflows.
- **Workflow Service** is used to make actions over workflows, as accept or reject request.

**Quotero Controller** is in charge of the interaction between the data model and web services. Controller checks the rights in order to preserve security and call data model methods to get a response from it.

**Quotero Data Model** contains the business code and the database calls through the Hibernate layer.

6. **EVALUATION**

To provide an initial form of evaluation in the scope of this paper, we have focused on quality attributes that are enhanced by this style.

- **Modifiability**

SOA promotes loose coupling between service users and providers. Services are self-contained, modular, and accessed via cohesive interfaces. These characteristics contribute to the creation of loosely coupled SOAs where there are few, well-known dependencies between services. Modifiability has been achieved by using loosely coupled services where bugs and updates can be fixed with a degree of ease. For example changes in the search service in order to fix an error, will not have an impact on the other listed services shown in figure 3 because the design of the services are loose coupling which increase flexibility and facilitate replacement and reusability. At the same time the cost of modifying the implementation of services is reduced and the overall system modifiability increases. If service interfaces need to be changed, the change may create problems because once service interfaces are published and used by applications, it can be difficult to identify who is using a service and what impact changing its interface will have.

Extensibility is a special case of modifiability. Extensibility is the ease with which the services’ capabilities can be extended without affecting other parts of the system. It is an important quality because the cloud environment in which a software system lives is continually changing. These changes mean changes in the service users, service providers and the messages exchanged among them. Extending an SOA may represent:

- Adding new services. SOAs allow for the easy addition of services (or new versions of services) because of the dynamic binding between service users and providers.
- Extend existing services without changing the interfaces. Because services are loosely coupled, adding capabilities that do not require a change in the service interface can be done without affecting service users.
- Extend existing services with changes to interfaces. New capabilities that require changes to the service interface can have a broad impact on the software. In many cases, the implementation of the service users needs to change due to the interface modification. In other cases, however, additions to a given interface can be made without breaking existing service users [10].

**Open Issue:** further research is needed in processes and techniques to deal with identifying the impact of updating services and incorporating new versions of service into a cloud environment.

- **Scalability**

The architecture ensures scalability by running multiple application threads concurrently. The load balancer distributes client requests to these threads. It distributes subsequent requests of the same client to different application thread. On the other hand, the load balancer is also responsible for auto-scaling as it starts a new thread upon increasing workload and stops running threads upon decreasing workloads, this is elasticity. In this way it can scale up to support millions of users. The major issue in SOA scalability is the ability of the site where the services are located to accommodate an increasing number of service users without performance degradation.

**Open Issue:** further research is needed in how scalability can be handled without effecting performance in a negative way.
• Reliability

Several aspects of reliability are important within an SOA, particularly the reliability of the messages that are exchanged between service users and providers, and the reliability of the services themselves.

➤ Message Reliability

Services are often made available over a network with possibly unreliable communication channels. Connections break and messages fail to get delivered or are delivered more than once or in the wrong sequence. Although techniques for ensuring the reliable delivery of messages are reasonably well understood and available in some messaging middleware products today, messaging reliability is still a problem. In this architecture the queue in the application logic layer pend requests that implement store and forward or cached message-based communication systems that allow requests to be stored when the target system is unavailable, and replayed when it is online so the probability that a system will not fail and that it will perform its intended function for a specified time interval is low [10].

➤ Service Reliability

The main issue to be dealt with is managing the transactional context in order to preserve data integrity during failures and concurrent access. Transaction management is more difficult in such a distributed, loosely coupled context for two reasons. Firstly, services are usually implemented in a stand-alone fashion, and transactions begin and end within the service. Therefore, transactions that involve the composition of services require either nested transactions or a redesign of transaction demarcation. Secondly, agents performing data changes (i.e., the service providers) are distributed, and, hence, a distributed transaction model is needed. Because services may be implemented in different languages and platforms, the implementation of distributed transactions using two phase commit, for example requires compatible transaction agents in all end points that interact using the same protocol.

Open Issue: further research is needed into how different mechanisms and approaches can help to guarantee reliability in cloud applications and the implications of their use on other quality attributes such as performance.

• Usability

It allows different tenants to create their own branding for an application by changing the data of the application because it stores all common data between the tenants in a central table while only tenant specific extensions are realized by additional tables. Table 1, 2 and 3 show some examples.

### Table 1: Central table for all common tenants’ data

<table>
<thead>
<tr>
<th>Account – Extension</th>
<th>Tenant ID</th>
<th>Row</th>
<th>AID</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>27</td>
<td>0</td>
<td>1</td>
<td>ABC</td>
</tr>
<tr>
<td></td>
<td>27</td>
<td>1</td>
<td>2</td>
<td>DEF</td>
</tr>
<tr>
<td></td>
<td>33</td>
<td>0</td>
<td>1</td>
<td>GHI</td>
</tr>
</tbody>
</table>

### Table 2: Tenant 2 specific table

#### Industrial Account

<table>
<thead>
<tr>
<th>Tenant ID</th>
<th>Row</th>
<th>Robot</th>
<th>Speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>27</td>
<td>0</td>
<td>X</td>
<td>20</td>
</tr>
<tr>
<td>27</td>
<td>1</td>
<td>Y</td>
<td>50</td>
</tr>
</tbody>
</table>

#### Telecommunication Account

<table>
<thead>
<tr>
<th>Tenant ID</th>
<th>Row</th>
<th>Lines</th>
</tr>
</thead>
<tbody>
<tr>
<td>46</td>
<td>0</td>
<td>12</td>
</tr>
</tbody>
</table>

Open Issue: there is a need for mechanisms to provide service users with effective feedback or control over communications. These are currently lacking in the standards.

7. IMPLEMENTATION

**Force.com**

- N-tier [presentation, application, and data],
- Each tenant has his own instance from the code base,
- Tenants can customize the interface, workflow, and tables (arbitrary data) by changing the Meta data stored in the UDD (i.e. applications are polymorphic).
• All tenants access the same logical Db which can be portioned among multiple machines.
• A universal table layout provides a single table for storing data by all tenants. This layout might be more efficient, because there is no need for expensive joins to reconstruct the logical schema.
• Metadata table stores information about tenant specific fields.

Microsoft Windows Azure:
• Three tier architecture.
• The application tier with user interface is implemented as web roles.
• The background application is implemented as worker roles.
• Load balancers distribute requests among instances.
• Tenants can implement UI customizations.

8. CONCLUSIONS

This paper proposed an architectural style for cloud computing applications based on the multi-tenants, SOA, and layered architecture styles. We defined the proposed architectural style layers and components in each layer, and presented an instantiation of this style on the open source Quotero document management application. We evaluated the quality of the proposed architecture with respect to the most important quality attributes in cloud computing, namely, scalability, reliability, and usability. We also discussed the implementation of this architecture on the two cloud computing environments of Force.com and MS Windows Azure. We proposed techniques to enhance the quality attributes of the resulting architecture, and also highlighted several open issues for future research.

9. ACKNOWLEDGMENT

This research was made possible by NPRP grant # [NPRP 09 - 1205 - 2 - 470] from the Qatar National Research Fund (a member of Qatar Foundation). The statements made herein are solely the responsibility of the authors.

REFERENCES


Figure 1: Conceptual Model

Figure 2: Logical View for Application As A service
Figure 3 Quotero components in the architecture.
Figure 4: Load Balancer Flow Chart