2010 Seventh International Conference on Information Technology

Service-Oriented Cloud Computing Architecture

Wei-Tek Tsai*, Xin Sun, Janaka Balasooriya
Department of Computer Science
Arizona State University
Tempe Arizona 85281 USA
*Department of Computer Science and Technology, Tsinghua University, Beijing, China**
{wtsai, xin.sun, janaka}@asu.edu

Abstract

Cloud computing is getting popular and IT giants such as Google, Amazon, Microsoft, IBM have started their cloud computing infrastructure. However, current cloud implementations are often isolated from other cloud implementations. This paper gives an overview survey of current cloud computing architectures, discusses issues that current cloud computing implementations have and proposes a Service-Oriented Cloud Computing Architecture (SOCCA) so that clouds can interoperate with each other. Furthermore, the SOCCA also proposes high level designs to better support multi-tenancy feature of cloud computing.

1. Introduction

Clouds have emerged as a computing infrastructure that enables rapid delivery of computing resources as a utility in a dynamically scalable, virtualized manner. The advantages of cloud computing over traditional computing include: agility, lower entry cost, device independency, location independency, and scalability [**1**].

There are many cloud computing initiatives from IT giants such as Google, Amazon, Microsoft, IBM as well as startups such as Parascale [**2**], Elastra [**3**] and Appirio [**4**]. However, there exist many different interpretations of what cloud computing is. This paper attempts to establish the connections between SOA and cloud computing by presenting related issues, and proposes a Service Oriented Cloud Computing Architecture (SOCCA).

This paper is organized as the follows: Section 2 provides a brief survey on cloud computing hierarchy; Section 3 presents a survey on existing cloud computing architectures and their issues; Section 4 proposes the SOCCA; Section 5 shows an initial prototype and experiment; Section 6 concludes this paper.

2. A Survey on Cloud Computing

2.1. A Hierarchical View of Cloud Computing

Most of the current clouds are built on top of modern data centers. It incorporates Infrastructure as a Service (IaaS), Platform as a Service (PaaS), and Software as a Service (SaaS), and provides these services like utilities, so the end users are billed by how much they used. Figure **1** shows a hierarchical view for cloud computing.

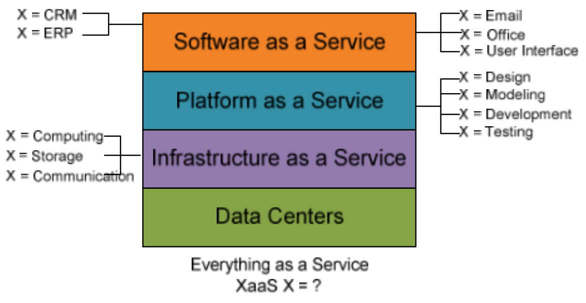


Figure 1: Hierarchical View of Cloud Computing

Data Centers: This is the foundation of cloud computing which provides the hardware the clouds run on. Data centers are usually built in less populated areas with cheaper energy rate and lower probability of natural disasters. Modern data centers usually consist of thousands of inter-connected servers.

Infrastructure as a Service: Built on top of data centers layer, IaaS layer virtualizes computing power, storage and network connectivity of the data centers, and offers it as provisioned services to consumers. Users can scale up and down these computing resources on demand dynamically. Typically, multiple tenants coexist on the same infrastructure resources [**1**]. Examples of this layer include Amazon EC2, Microsoft Azure Platform.

Platform as a Service: PaaS, often referred as cloudware, provides a development platform with a set of services to assist application design, development, testing, deployment, monitoring, hosting on the cloud. It usually requires no software download or installation, and supports geographically distributed teams to work on projects collaboratively. Google App Engine, Microsoft Azure, Amazon Map Reduce/Simple Storage Service are among examples of this layer.

Software as a Service: In SaaS, Software is presented to the end users as services on demand, usually in a browser. It saves the users from the troubles of software deployment and maintenance. The software is often shared by multiple tenants, automatically updated from the clouds, and no additional license needs to be purchased. Features can be requested on demand, and are rolled out more frequently. Because of its service characteristics, SaaS can often be easily integrated with other mashup applications. An example of SaaS is Google Maps, and its mashups across from the internet. Other examples include Salesforce.com and Zoho productivity and collaboration suite.

The dividing lines for the four layers are not distinctive. Components and features of one layer can also be considered to be in another layer. For example, data storage service can be considered to be either in as IaaS or PaaS. Figure **1** suggests a hierarchical relationship among the different layers; however, it does not mean the upper layer has to be built on top its immediate lower layer. For example, a SaaS application can be built directly over IaaS, instead of PaaS.

In the cloud computing environment, everything can be implemented and treated as a service. Figure **1** shows a few examples of what can be treated as a service in different layers.

3. Existing Cloud Computing Architectures

Both academia and industry have been active on cloud computing research, and several cloud computing architectures have been proposed. In [**5**], IBM considers current single-providers cloud as limited resource, and the lack of interoperability among cloud providers prevents deployment across different clouds. A cloud computing architecture named Reservoir was proposed to create a federation from multiple cloud providers which acts as a global fabric of resources that can guarantee the required SLA. In Reservoir architecture, the computational resources within a site are partitioned by a virtualization layer into virtual execution environments (VEEs). A service application is decomposed into a set of software components/services running on VEEs on the same or different VEEs within a site or across from different sites. However, Reservoir architecture does not allow a component/service to run on its duplicates on different VEEs; Moreover, computing resources are abstracted as hosting service which might not be necessarily true for all clouds. In [**6**], a software platform for .NET based cloud computing named Aneka was introduced. Aneka is a customizable and extensible service oriented runtime environment that enables developers to build .NET applications with the supports of APIs and multiple programming models. Aneka is a service-oriented, pure PaaS cloud solution. In [**7**], Rajkumar and his colleagues explained a market-oriented cloud architecture in detail used by Aneka, which regulates the supply and demand of cloud resources to achieve market equilibrium, adds economic incentives for both cloud consumers and providers, and promotes QoS-based resource allocation mechanisms that differentiates service request based on their utility. The key component of this architecture is SLA (Service Level Agreement) Resource Allocator which is consisted of Service Request Examiner and Access Control, VM (Virtual Machines) monitor, Service Request Monitor, and Request Dispatcher. Based on the feedback from VM and Service Request monitors, the dispatcher routes the requests from users/brokers to the cloud resources that can fulfill their QoS requirements. In [**8**], Huang and her colleagues from IBM described a service oriented cloud computing platform that enables web-delivery of application-based services with a set of common business and operational services. The platform supports multi-tenancy feature by utilizing single application instance model. The isolation among tenants is taken care by the underline design. Other services include subscription management, federated ID management, application firewall, etc.

3.1. Issues with Current Clouds

Current cloud computing has following characteristics:

Users are often tied with one cloud provider: Even though up-front cost for a cloud computing deployment is reduced and long term lease is eliminated, much effort and money is spent on developing the application for a specific cloud platform which makes it difficult to migrate the same application onto a different cloud. Often, migration simply may mean redevelopment. For example, applications deployed on Amazon EC2 cannot be migrated easily due its particular storage framework [**9**].

Computing components are tightly coupled: This can be clearly explained using an analogy. Suppose one wants a new computer, this person has the choices of either buying a ready-to-use computer from a manufacturer (buying) or purchasing the components separately and building the computer in a DIY style (building). The advantages of building over buying include wider selection of components, flexibility to customize, and cheaper cost [**10**]. However, as the computing resources over the internet, current cloud implementations do not allow this kind of flexibility. If a customer opts to use Amazon S3 storage service, he is then stuck with other cloud computing services Amazon provides, such as EC2, Elastic Map Reduce.

Lack of SLA supports: Currently, SLA is an obstacle that prevents wide adoption for cloud computing. Cloud computing infrastructure services such as EC2 are not yet able to sign the SLA needed by companies that want to use cloud computing for serious business deployment [**11**]. Moreover, business is dynamic. Static SLA is not able to adapt to the changes in business needs as cloud computing promises to.

Lack of Multi-tenancy supports: Multi-tenancy can support multiple client tenants simultaneously to achieve the goal of cost effectiveness. Currently, one has three types of multi-tenancy enablement approaches: virtualization, mediation and sharing [**12**]. To achieve the full potential of multi-tenancy, three issues remain to be solved [**12**]:

1. ***Resource sharing***: To reduce the hardware, software and management cost of each tenant.
2. ***Security isolation***: To prevent the potential invalid access, conflict and interference among tenants.
3. ***Customization***: To support tenant-specific UI, access control, process, data, etc.

Lack of Flexibility for User Interface: UI is an important part of the application, and user experience can be a major evaluation factor for a business application. However, cloud/SaaS users are limited with UI choices because UI composition frameworks, such as the one proposed in [**13**], have not been integrated with cloud computing.

4. Service Oriented Cloud Computing Architecture (SOCCA)

4.1. Cloud Computing and SOA

SOA and cloud computing are related, specifically, SOA is an architectural pattern that guides business solutions to create, organize and reuse its computing components, while cloud computing is a set of enabling technology that services a bigger, more flexible platform for enterprise to build their SOA solutions. In other words, SOA and cloud computing will co-exist, complement, and support each other.

There have been several initiatives at attempting bridging SOA and cloud computing. Noticeably, the works in [**6**] [**8**] have more service-oriented features than the other mentioned in section 3.

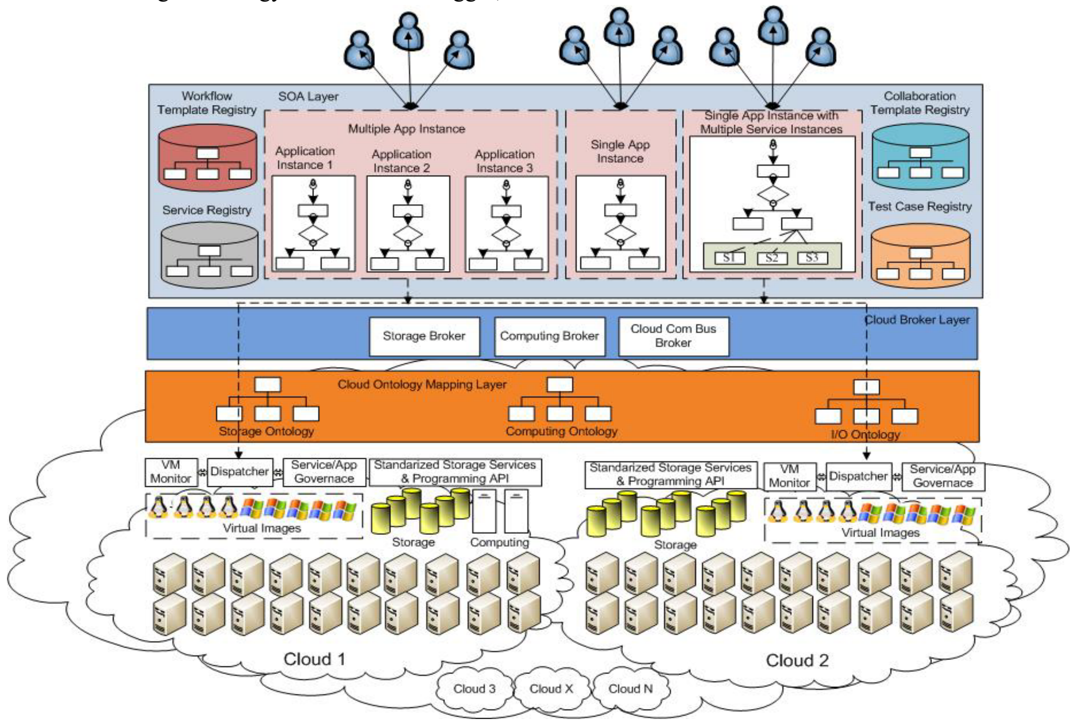


Figure 2: Service-Oriented Cloud Computing Architecture

4.2. Layered Architecture of SOCCA

Our SOCCA is a layered architecture shown in Figure **2**:

Individual Cloud Provider Layer: This layer resembles the current cloud implementations. Each cloud provider builds its own data centers that power the cloud services it provides. Each cloud may have its own proprietary virtualization technology or utilize open source virtualization technology, such as Eucalyptus [**14**]. Similar to Market-Oriented Cloud Architecture proposed in [**7**], within each individual cloud, there is a request dispatcher working with Virtual Machine Monitor and Service/App Governance Service to allocate the requests to the available recourses. The distinction from current cloud implementations is that the cloud computing resources in SOCCA are componentized into independent services such as Storage Service, Computing Service and Communication Service, with open-standardized interfaces, so they can be combined with services from other cloud providers to build a cross-platform virtual computer on the clouds. In order to achieve maximum interoperability, uniform standards need to be implemented. For example, SQL is de facto standard for RDBMS data management, and many database vendors have their own implementations. A cloud version of SQL needs to be defined, so data manipulation logic of an application that works on one cloud can also work other clouds. A distributed computing framework standard to unify all different implementations of Map/Reduce is also in need for the same reason.

Cloud Ontology Mapping Layer: Cloud providers might not conform to the standards rigidly; they might also have implemented extra features that are not included in the standards. Cloud Ontology Mapping Layer exists to mask the differences among the different individual cloud providers and it can help the migration of cloud application from one cloud to another. Several important ontology systems are needed:

1. ***Storage Ontology*****:** It defines the concepts and terms related to data manipulation on the clouds, such as data update, date insert, data delete, and data select, etc.

2. ***Computing Ontology***: It defines the concepts and terms related to distribute computing on the clouds, such as Map/Reduce Framework.
3. ***Communication Ontology***: It defines the concepts and terms related Communication Schema among the clouds, such as data encoding schema, message routing.

Cloud Broker Layer: Cloud brokers serve as the agents between individual cloud providers and SOA layer. Each major cloud service has an associated service broker type. Generally, cloud brokers need to fulfill the following tasks:

1. ***Cloud Provider Information Publishing***: Individual cloud providers publish specifications and pricing info to the cloud brokers. Important provider information includes:

Cloud Provider Basic Information: Company Name, Company Address, Company Website, Company Contact Info, etc.

Resource Type and Specifications: Whether it is computer/storage/communication resource and its specification and limitation. For example, for the data storage service, the data transmission rate can be as high as 2Gb/s.

Pricing Information: How the services charge. This varies the most among different cloud providers. For example, currently, Google does not charge for the first 500MB storage, and $0.15 per GB of data after, while Amazon charges $0.11 per GB-month for its EBS Volumes service. Even within a cloud provider, the pricing info might change as the market's dynamic changes.

2. **Ranking**: Like the service brokers in SOA, cloud brokers also rank the cloud resources published. Services can be ranked in several categories such as price, reliability, availability, and security, etc. Ranking can be achieved through user voting or historical service governance records.
3. **Dynamic SLA Negotiation**: Business is often dynamic, and the IT infrastructure has to be adaptive to accommodate the business needs, therefore to achieve the optimal ROI (Return of Investment). It's often the case that the IT resources a business demands can be predicted. Cloud service brokers can help cloud users and cloud providers negotiate on a SLA dynamically.
4. **On-Demand Provision Model**: Most services experience seasonal or other periodic demand variation as well as some unexpected demand bursts due to external events. The only way to provide "on-demand" services, is to provision for them in advance. Accurate demand prediction and provision become critical for the successful of the cloud computing, which reduces the waste of utility purchase and can therefore save money using utility computing. We are investigating a demand prediction model and model the evolution of multi-tenant as a discrete time stochastic process. We have investigated several macroeconomic factors in a real mortgage service platform [**15**][**16**], and the initial results show that the underlying stochastic process may depend on a number of external factors such as macroeconomic variables, as well as service internal features. Some analysis results from real applications demonstrate the effectiveness of our models. Due to the space limitation, more details can be found in [**15**]. Specifically, the process needs to answer the following question: What is the forecast of tenant m days into the future? How to predict the workload distribution at different services? How to optimize the service provision process and minimize customers' dissatisfaction?

SOA Layer: This layer fully takes the advantages of the existing research and infrastructure from traditional SOA. Many existing SOA frameworks, such as CCSOA [**17**], UCSOA [**18**], GSE [**19**] and UISOA [**13**] can be integrated into this layer. Figure **2** shows a possible SOA layer for SOCCA. Similar to CCSOA, not only services but also many other artifacts can be published and shared, such as workflow templates, collaboration templates and test cases. The registry for each type of artifacts is indexed and organized by its according ontology. The fundamental difference of the SOA layer of SOCCA from traditional SOA is that the service providers no longer host the published services anymore. Instead, they publish the services in deployable packages, which can be easily replicated and redeployed to different cloud hosting environments. Application developers can decide which clouds they want to these services to run based a set of criteria. The details will be discussed in section 4.4. Another major improvement is multi-tenancy support that allows more flexibility, which will be discussed in section 4.3. SOA layer of SOCCA allows more flexibility than traditional SOA; it further separates the roles of service providers and cloud providers, and the service logics and its running environments.

4.3. Multi-tenancy Architecture (MTA)

As shown in Figure 2, SOCCA allows 3 different main multi-tenancy patterns. In [**8**], the authors discussed the left two multi-tenancy patterns: Multiple Application Instance (MAI) and Single Application Instance (SAI). The authors pointed out, the former does not scale as well as the latter, but it provides better isolation among different tenants. Within SOCCA, a new multi-tenant pattern becomes possible: Single Application Instance and Multiple Service Instances (SAIMSI). The motivation behind this pattern is that the workloads are often not distributed evenly among application components, and the performance of the single application instance is limited by the application components having lower throughput. Moreover, to enhance scalability, we want to reduce unnecessary duplications as much as possible as opposed to Multiple Application Instances pattern. Figure **3** shows a simplified example. The example application is composed by A, B, C, three services with C being the computing intensive component. With C being the bottleneck to support multiple tenants, 3 instances of C are created to balance the workloads. Note that the 3 instances of the services can also reside on different clouds.

Better scalability is not only benefit from the SAIMSI pattern, easy customizability is another gain. Suppose in the sample application, C is a payment service. Different tenants might have different payment method requirements, such as credit card, Paypal, or check. The application runtime environment (not described in this paper) will direct users of each tenant to the correct service instance according to tenants' individual configuration. In the case that a future tenant has a payment requirement that cannot be met by the existing service instances, say money order, an according service instance can be easily plugged into the existing service instances group. The upcoming papers on multi-tenancy from our research group will provide more details on this topic.

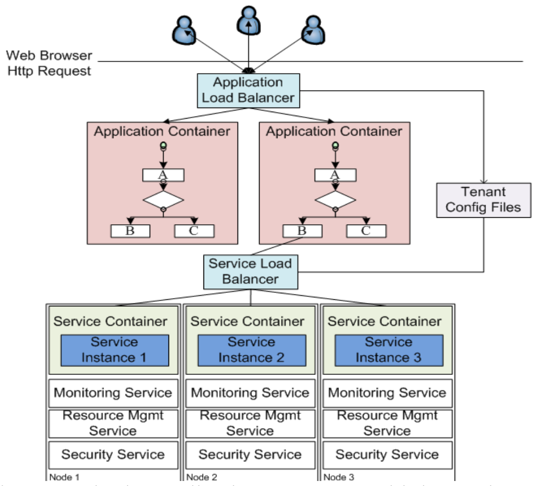


Figure 3: Single Application Instance Multiple Service Instances

4.4. Application Development on SOCCA

4.4.1. Service Package

Service providers of traditional SOA develop the logic of a service and provide its running environment. In SOCCA, services are published as re-deployable packages, namely service package. A service package contains the following required/ optional information and files:

Compiled Code: If service providers only use the standard APIs and protocols, a single version of complied code is enough; if service providers optimize the performance of their services by utilizing some platform unique APIs and features, complied code for each platform is needed.

Source Code: This is optional. It is useful to help its user to understand the service better, also gives the freedom to its users to tweak the services to accommodate their specific requirement.

Configuration File: Services might use external basic services. For example, a computing intensive scientific service which also uses a lot of storage might deploy its computing logic on a cloud that provides high performance computing power, but use the cheaper storage service provided by another cloud. This requires a configuration file which specifies the external service's locations, partner link, etc. This can also be achieved in a BPEL manner, however, since basic services such as storage services, have a widely adopted standards, and are frequently used, so it is more efficient to handle in a database connection configuration file style.

Resource Files: Any resource files that the service depends on, such as images, documents.

4.4.2. SOCCA Applications

Application development in SOCCA is similar to the development in CCSOA. Developers first search if there is a workflow template that matches the requirement. A workflow template is composed of service stubs/specifications, which specify the functionalities and interfaces of services. Later a service stub is bound with a service package. Depending on the QoS requirements and the budget for the application, cloud brokers will negotiate with cloud providers on SLA, and deploy the service packages on one or multiple clouds. An algorithm for request dispatching for a service across its deployments on different clouds needs to be applied. Due to the page number constraints, it will be discussed in our upcoming papers. Figure **4** shows a typical application architecture on SOCCA.

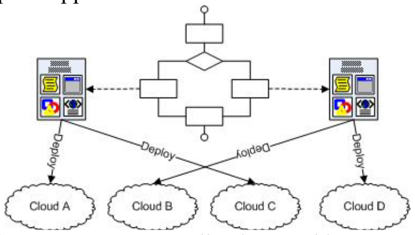
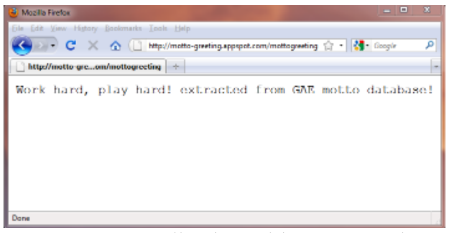


Figure 4 SOCCA Application Architecture

5. Initial Prototype and Experiment


Mozilla Firefox
File
Edit
View
History
Bookmarks
Tools
Help
http://motto-greeting.appspot.com/mottogreeting
Google
http://motto-gre...om/mottogreeting
Done

```
Work hard, play hard! extracted from GAE motto database!
```

Figure 5: Motto Application with GAE Database

This section shows an initial prototype and experiment to demonstrate the possibility of SOCCA.

The demonstration web application has one easy requirement: When each user visits the web page, he/she will be greeted by a random message retrieved from a motto database.

1. We developed the application by using Google App Engine. Note that a number of mottos are retrieved and stored from and to Google cloud by using Datastore with JDO. Figure **5** shows a screenshot of motto application running on Google App Engine.

2. We created a web service that wraps the Azure SQL service that allows retrieving and storing mottos from the motto databases deployed on Azure.

3. We utilized the Web Service Connector (WSC) tool provided by [**20**] to generate Google App Engine compatible client code in java to access the web service developed by step 2. WSC is a code-generation tool that takes a WSDL file and generates an optimized java library that provides access to the web service.

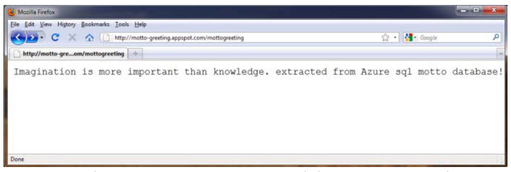
4. We created a class that implements "javax.jdo.PersistenceManagerFactory", which is the interface GAE Datastore uses to manipulate the data on the cloud.

5. We created an instance of SQLPersistenceManagerFactory that implements "javax.jdo.PersistenceManagerFactory", which is the interface GAE Datastore uses to manipulate the data on the cloud. Figure **6**, a code snippet, shows that depending on the config file, the application will be connected with either GAE Datastore database or Azure SQL database. Figure **7** shows that after changing the config file, the motto app is now connected with Azure SQL database. More implementations of PersistenceManagerFactory can be added.

```
private PMF() {
    persistenceManagerFactoryMap.put("Google", gaePMFInstance);
    persistenceManagerFactoryMap.put("Azure", sqlPMFInstance);
}

public static PersistenceManagerFactory get(String type) {
    return persistenceManagerFactoryMap.get(type);
}
```

Figure 6: Code snippet for Database Service Configuration


Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://motto-greeting.appspot.com/mottogreeting
Google
http://motto-gre...om/mottogreeting
Done

```
Imagination is more important than knowledge. extracted from Azure sql motto database!
```

Figure 7: Motto App with Azure Database

The prototype demonstrates that a service package deployed on one cloud can be configured to collaborate with services from other clouds. However, it does not show that a service package can be redeployed on a different cloud and the instances for the same services can live on multiple clouds. This is due to that currently, different clouds support different language sets, and there is no powerful modeling language to support developments for multiple platforms. We are currently developing this feature using a modeling language PSML [**21**].

6. Conclusion

This paper proposed a service-oriented cloud computing architecture SOCCA that allows an application to run on different clouds and interoperate with each other. The SOCCA is a 4-layer architecture that supports both SOA and cloud computing. SOCCA supports easy application migration from one cloud to another and service redeployment to different clouds by separating the roles of service logic provider and service hosting/cloud providers. It promotes an open platform on which open standards, ontology are embraced. The paper also introduced related topics for future research, such as service demand prediction and SLA negotiation, and service request dispatching algorithms. More work will also be conducted to devise the ontology systems needed for a working SOCCA, and a prototype with all features discussed.

7. References

[1] Wikipedia - Cloud Computing. [Online]. http://en.wikipedia.org/wiki/Cloud_computing

[2] Parascale. [Online]. http://www.parascale.com/

[3] Elastra. [Online]. http://www.elastra.com/

[4] Appirio. [Online]. http://www.appirio.com/

[5] Rochwerger B et al., "The RESERVOIR Model and Architecture for," *IBM Systems Journal*, 2009.

[6] Christian Vecchiola, Xingchen Chu, and Rajkumar Buyya, "Aneka: A Software Platform for.NET-based Cloud Computing," in *High Speed and Large Scale Scientific Computing*, 2010.

[7] Rajkumar Buyya and Chee Shin Yeo, "Cloud Computing and Emerging IT Platforms: Vision, Hype, and Reality for Delivering Computing as the 5th Utility," *Future Generation Computer Systems*, pp. 599-616, 2009.

[8] Ying Huang et al., "A Framework for Building a Low Cost, Scalable and Secured Platform for Web-Delivered Business Services," , 2009.

[9] Bernard Golden. (2009, January) Computer World. [Online]. http://www.computerworld.com/s/article/9126620/The_case_against_cloud_computing_part_one

[10] Mark Kyrnin. About.com. [Online]. http://compreviews.about.com/od/general/a/BuildvsBuy.htm

[11] Galen Moore. (2009, Jan) Mass High Tech. [Online]. http://www.masshightech.com/stories/2009/01/05/weekly10-Cloud-computings-SLA-obstacles-clear-in-2009.html

[12] Bo Gao, Changjie Guo, Zhihu Wang, Wenhao An, and Wei Sun. (2009, March) Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 3: Resource sharing, isolation and customization in the single instance multi-tenant application. [Online]. http://www.ibm.com/developerworks/webservices/library/ws-multitenant/index.html

[13] Wei-Tek Tsai, Qian Huang, Jay Elston, and Yinong Chen, "Service-Oriented User Interface Modeling and Composition," in *International Conference on e-Busines Enginerring*, Xi'an, 2008, pp. 21-28.

[14] Eucalyptus System. [Online]. http://www.eucalyptus.com/

[15] Qihong Shao et al., "Ranking Mortgage Origination Applications using Customer, Product, Environment and Workflow Attributes.," in *IEEE Proceedings of Congress on Services*, 2009.

[16] Wei-Tek Tsai et al., "Forecasting Loan Volumes For the Mortgage Orgination Process," , under review.

[17] W.T. Tsai, Bingnan Xiao, Ray A Paul, and Yinong Chen, "Consumer-Centric Service-Oriented Architecture: A New Approach," in *SEUS-WCCIA*, 2006, pp. 175-180.

[18] Mark Chang, Jackson He, W.T. Tsai, Bingnan Xiao, and Yinong Chen, "UCSOA: User-Centric Service-Oriented Architecture," in *IEEE International Conference on e-Business Engineering*, 2006, pp. 248-255.

[19] Wei-Tek Tsai, Bingnan Xiao, Ray Paul, Qian Huang, and Yinong Chen, "Global Software Enterprise: A New Software Constructing Architecture," in *International Conference on E-Commerce*, 2006, pp. 55-55.

[20] force.com. [Online]. http://developer.force.com/appengine

[21] W.T. Tsai et al., "Modeling and Simulation in Service-Oriented Software Development," *Simulation*, vol. 83, no. 1, pp. 7-32, 2007.

[22] Carl Osipoy, German Goldszmidt, Mary Taylor, and Indrajit Poddar. (2009, May) Develop and Deploy Multi-Tenant Web-delivered Solutions using IBM middleware: Part 2: Approaches for enabling multi-tenancy. [Online]. http://www.ibm.com/developerworks/webservices/library/ws-multitenantpart2/index.html