# Service Oriented Architecture for Cloud Computing

*Impact of Intelligence Layer*

[1]Faraz Saleem, [2]Mr. Manik Chandra Pandey, [3]Mr. Amit Asthana

[1]M. Tech Student, [2]Assistant Professor, [3]Assistant Professor
[1]Department of Computer Science & Engineering,
[1]Subharti Institute of Technology & Engineering, Subharti University, Meerut, India.

*Abstract*—**Cloud computing is a recent trend in IT that moves computing and data away from desktop and portable PCs into large data centers. The main advantage of cloud computing is that customers do not have to pay for infrastructure, its installation, required man power to handle such infrastructure and maintenance. Cloud computing is getting more popular when IT giants such as Google, Amazon, Microsoft, IBM have started their cloud computing infrastructure. There are many characteristics which a cloud should have. Many of the existing platforms and frameworks lack of some of these characteristics, for example most of them cannot really be integrated, all of them just give the user some tools and platforms (such as service bus and development kits), but there is no provided intelligence and automation. This paper proposes a framework that supports all of the mentioned characteristics and some others like low coupling and high integration. The proposed framework not only supports traditional cloud facilities, but also using a service, adds intelligence to cloud to provide an automatic environment. In order to accomplish the above mentioned goals, we have founded our framework on SOA (service oriented architecture) that manages data flow with an event driven approach.**

*Index Terms*— **Cloud computing, Ontology based clouds, Service orientation, Ultra large scale systems, Event driven architecture**

## I. INTRODUCTION

The term "cloud" originates from the world of telecommunications when providers began using virtual private network (VPN) services for data communications [1]. Cloud computing is a recent trend in IT that moves computing and data away from desktop and portable PCs into large data centers [2]. The main goal of cloud computing is to make a better use of distributed resources, combine them to achieve higher throughput and be able to solve large scale computation problems. Cloud computing deals with virtualization, scalability, interoperability, quality of service and the delivery models of the cloud, namely private, public and hybrid.

This paper tries to find a unified architecture and framework for cloud computing relying on the service oriented view point. More precisely, the issue that is aimed in this paper is to propose a cloud computing framework to support all of aforementioned characteristics. To achieve this goal, the paper attempts to find a good relationship between cloud computing, service orientation and some other technologies among these architecture paradigms, such as event driven architecture, semantic web and semantic web service.

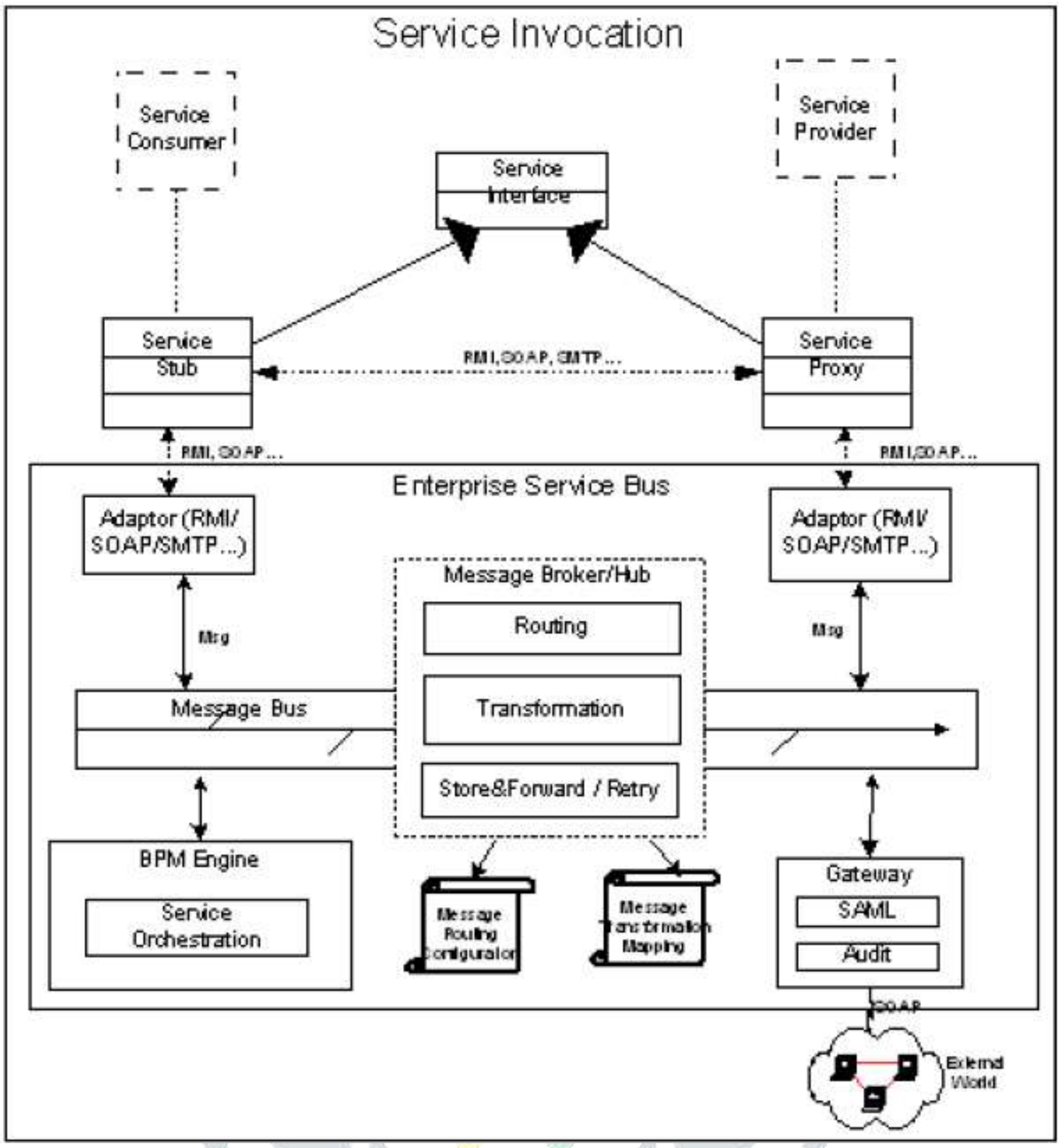
This framework also targets to automate everything in the cloud as far as possible through a new layer of intelligence. This automation would be realized from an event occurrence to output provision, such as SLA provisioning, service composition, business process management and security. In addition, this framework will be managed according to policy and ontology basics, so it can be used in both private and public clouds (It is worth mentioning that there is no public cloud made for internal policies of a particular enterprise). Finally, the framework has an event driven view which makes cloud integration possible in real world.

## II. THE PROPOSED ARCHITECTURE OF SERVICE LAYER

Service orientation and cloud computing are reciprocal [3]. The proposed framework is made of a combination of cloud computing and SOA. In addition, anything happens in the environment will be sent to the cloud as an event; an event can be a user request for any service (from hardware resource to a business process) or it can be detected from environment data transactions.

This framework uses ontology to define a high level and common language. This common language is used to describe the events, business tasks (high level business process to separate business from technology), services and any other resources in the cloud. In fact, ontological view to resources helps users to be decoupled from the cloud and its underlying technologies. In addition, the user can manage and control the cloud through virtualization techniques and access portals.

The main goal of this framework is to provide components which were described by NIST [4]. Figure 1 shows the architecture of the framework.



Service Invocation

**Service Invocation**

The next important requirement to be addressed by the framework is to standardize the service invocation mechanism and provide the infrastructural components with clearly defined Interfaces that shield the service consumers and the service providers from the underlying implementation details. Usually, enterprise architecture teams define the communication policies for applications in an enterprise. The communication policies define the strategies on when to use native protocols, when to use point-to-point communication, and when to use message-oriented communication. A common debate while determining communication policies is whether to use message-oriented communication to invoke a service or to directly invoke the service using its implementation specific synchronous protocol such as RMI. One of the fundamental business requirements is to differentiate the service levels offered to the end customers based on their business value to enterprise, so that it helps achieve the desired client experience business objectives. This is technically possible only if the communication mechanism used in invoking services is controlled and the service invocations can be prioritized. Using message-oriented communication instead of directly invoking the service using its implementation-specific synchronous protocols thus provides the mechanism needed to address this requirement. Strategically, the preferred communication mechanism should be one based on a messaging infrastructure instead of point-to-point invocations.

The infrastructural logical components that are needed for service invocation include:

- Service Stub
- Service Proxy
- Adaptors

- Message Broker
- Message Bus
- Gateways

The Service Stub implements the delegate pattern and provides the service interface to the service consumers, hiding the invocation details.

The Service Proxy implements the proxy pattern and provides the abstraction of the invocation details for the service providers.

The Adaptors provide the technology-specific integration mechanisms for the service stubs and proxies. For a J2EE-based implementation, the adaptor can provide the listener mechanisms that the stubs and proxies can use to receive the messages and the API to send a message.

The Message Broker and the Message Bus provide the transformations, routing, and other such services. The broker and the bus take care of transforming the message representations from the service consumer and service provider internal formats to the Enterprise Message Format and vice versa, They also provide the routing of the messages, store and forward, message retries, prioritizing of messages etc.

The Gateways provide the mechanisms for external integration. The gateways provide the single points of contact for the external partners and transform the invocation protocols and message formats from the external partners to the internal enterprise message formats using a message broker and an adaptor. They also enforce the security checks, audit requirements, etc.

Having clearly defined interface driven components for each of these helps replace implementations with minimal impacts.

**Service Orchestration**

The next important requirement to be addressed is the orchestration of services for the implementation of a business process.

The framework should provide a mechanism based on COTS BPM tools to define, execute, and manage the service orchestration. The framework should define an orchestration Adaptor that helps abstract the interactions with the orchestration implementations (BPM tools) through an adaptor interface with API to initiate processes, get the list of process instances, get the list of activities and their states, and to manipulate the state of activities, list of exceptions, etc. that provide an abstraction over the implementation specifics. The adaptors can be implemented for the selected orchestration implementation. Since they all provide the same API, they can be replaced easily as needed without greatly impacting the services interacting with the orchestration component.

## III. THE LAYERED ARCHITECTURE OF FRAMEWORK

Figure 1 shows that this architecture has 5 layers: IaaS, PaaS, INaaS (Intelligence as a service), SaaS, and environment. Each layer also has some components which will be described.

A. Environment Layer

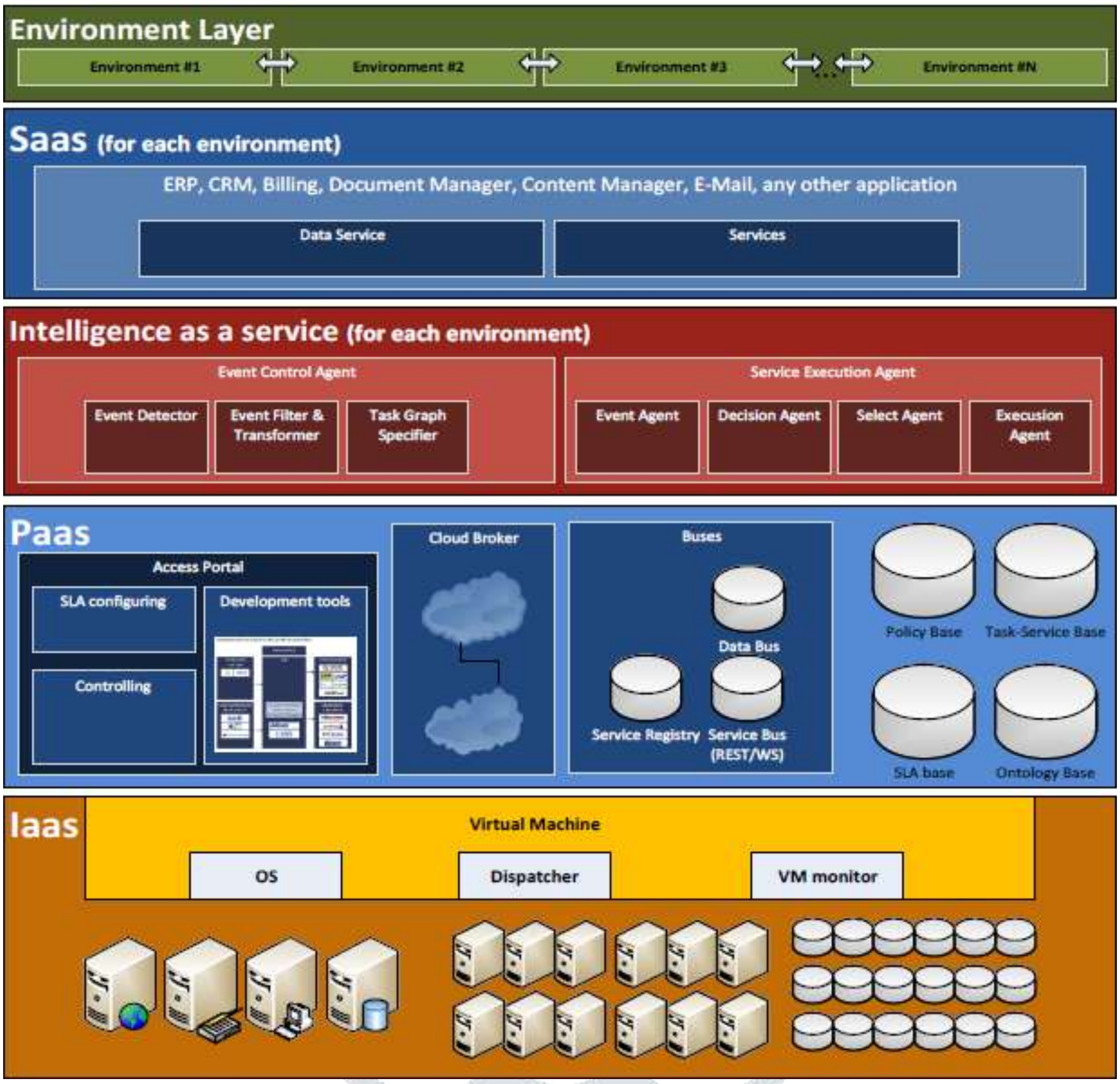


Figure 1.The proposed cloud computing architecture

Everything in an environment can be public or private to other environments; this feature helps privacy for each environment. An environment can have some legacy systems whose data or services could be in the cloud; to integrate with these systems, their services must be available for the cloud service bus. The environments can be in relation with each other. Type of this relationship must be defined by the user; for instance, the relationship can happen through a XML file, a message based connection, a web service, an event, etc. So as to keep the privacy, users and their access level can be defined in each environment. This layer is a virtual layer and does not really exist in the framework. The framework provides virtual environments through its PaaS and IaaS layers. The environment is an atmosphere that will be defined for the cloud. Each customer can have one or more environment; the environment could be a cloud, an enterprise or something else that the customer defines to the cloud.

***B. Software as a Service Layer***

This is the layer where an environment can access the applications, services and data services. Any application like ERP, CRM, SCM, Financial Apps, Web services, and any other software services which the user (those users which are defined in environments) are allowed to access, can be accessible through this layer. Each environment has its own SaaS layer. The abstraction of environments plus SaaS enables the cloud to be multi-tenant. Each environment can have its own application, configuration, customization and data. No other environment can access this configuration and data without permission.

### C. Intelligence as a Service Layer

This layer is the heart of the framework architecture. This is where the events are detected or received, related tasks are extracted, required services are invoked and the response is provided at last. Everything will be as an event, such as adding, deleting and updating services (and applications), changing the SLA, updating users, defining environments and anything else. Every event type must be defined (its header and body attributes); each event has its own Event Processing Network (EPN) which specifies event transformation process [5] and event subscribers. EPNs can be defined either according to the first order predicate logic [6] or modeling graphs; this model is independent from software, hardware and infrastructure technologies, so it can separate the user from low level concerns. Every event should also have some corresponding tasks and services to make it executable. Main elements of this layer will be described in the following.

1) Event Control Agent. The purpose of this agent is to recognize events or event patterns. An event pattern can have a semantic which events (input events) cannot have one by one [5]. Figure 2 shows the working flow of this component. This component has some subcomponents which are described in the following.

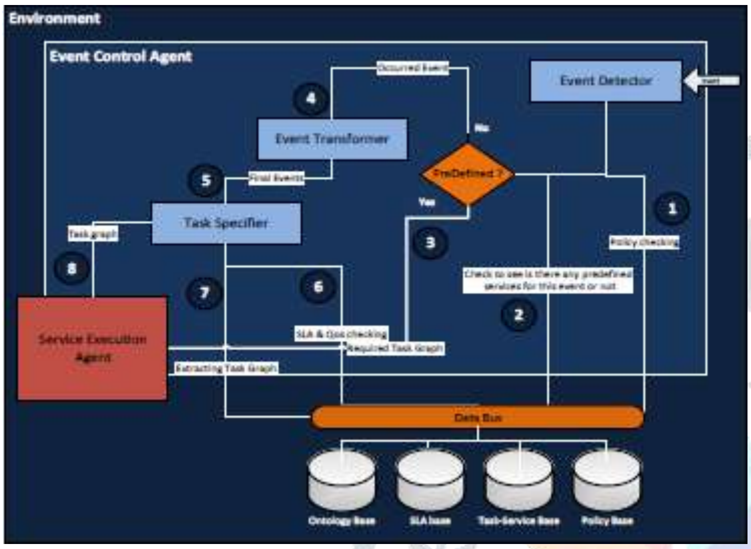


Figure 2. An Abstraction of the Event Control Agent

Event Detector: This component is in charge of receiving and detecting events. Sometimes events are sent to the component on behalf of a human or software agent, and some other times events are detected by the component; it is possible that the user is not aware of the event occurrence.

As the events are received or detected, first of all, it will be checked whether message data is sufficient or not (message header and body attributes). If the data is not sufficient, the operation will be suspended as long as the data is completed by the event producer. A system that involves event producers which produce huge number of events needs to filter the events. This component is also in charge of filtering the events. It filters events according to the policy base. After performing these processes, the task-service base will be searched (through data bus) for the received event. If the event is found, so its corresponding registered task graph will be sent to the service execution agent; if not, it will continue to the next step.

Event Transformer: This component is in charge of event transformation according to the process which has been specified in EPN (for each particular event). Transformation process includes event aggregation, splitting, composition and translation, which the last one includes event enrichment and projection. For more information about these notions, you can see [5].

Task Specifier: In this component, required tasks (Tasks are very high level definitions of any services which can be provided in any environment) will be acquired from enterprise ontology*; then, according to their relationships, a task graph will be extracted. After extracting the task graph, it should be specified that which users can get the execution results; these users are event subscribers [7]. This means that the event execution output does not only go to the event producer. It is possible that many other users receive the results, and each one of these results can cause other event occurrence. After task graph acquiring is completed, this graph will be submitted to the service execution agent.

2) Service Execution Agent. The purpose of this agent is to execute required services of events. This agent starts with separating the tasks of the task graph, then it finds the best service composition for each task, and then it finds the most optimized service orchestration. Finally it executes selected services to reach the final result. Figure 3 shows this agent and its sub-components and their relationships.

**Entry Agent:** This agent is in charge of receiving task graph and refining it into its basic tasks. For the reason that an event might have more than one task graph, therefore the most optimized task graph need to be selected and executed. Task graphs will be sent to the decision agent one by one, and the decision agent will find the best service composition, based on service QoS and user SLA, for each of them. This will be repeated for all graphs, one by one, and eventually, the one will be selected which is fully matched to the user SLA.

**Decision Agent:** This agent is in charge of receiving task graphs and finding the most appropriate service orchestration for the graph. This agent, in fact, is not responsible for executing the services; when it finds the service orchestration, sends it back to the entry agent. This agent calls the search agent for each unit task; the search agent will find the appropriate service (or service composition) for the task, and finally, the decision agent aggregates the selected services for each task, and generates a service orchestration for the received task graph.

**Search Agent:** This agent receives a task, and finds an appropriate service (or service composition) for the task. It first investigates the task-service base: if no pattern is found, it will try to find a service composition. As it is possible that more than one service composition be found, the most optimized one should be selected (according to the user SLA). After finding the most suitable composition, it will be stored in the task-service base so as to accelerate finding compositions in future.

**Execution Agent:** This agent receives a service composition from the entry agent and executes the services through the ESB.

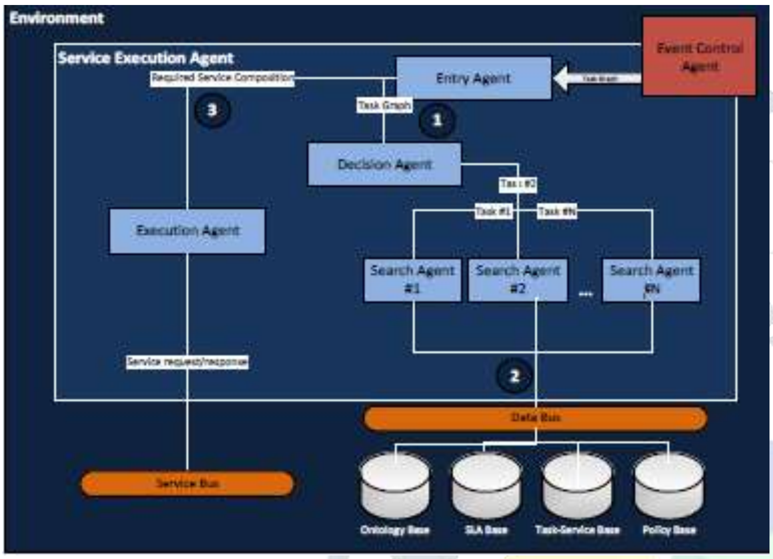


Figure 3. An Abstraction of the Service Execution Agent

## D. Platform as a Service Layer

**1) Policy Base.** The policies of how to respond and utilize the events will be stored in this base. For instance, grouping and categorizing the users, user types, user authentication and authorization information, and security policies are some kinds of policies. Every environment can have its own policies. When an event occurs, these policies will be reviewed in the first place, and if there is nothing mismatched, the response process will carry on. Thus the policy base helps the framework to be useful for both private and public cloud computing.

**2) Task-Service Base.** This base is a repository which helps the event control and service execution agents to remember what the last task graph or service composition has been for a particular event (according to different SLAs), so they can refuse repetitive operations.

As the event control agent finds a task graph for a particular event, it stores the graph in the task-service base to remember it in future. The execution agent will also store some information in this base, such as, service compositions for a task and service orchestrations for a task graph.

If any change is made on a service (like changing QoS, service deletion, service updating, etc.), it will affect the task-service base. Any of these changes is defined as an event in the cloud, and they are handled by their corresponding services. For instance, if a service is deleted, the service deletion event occurs, and its corresponding service will be executed.

**3) Ontology Base.** Ontology documents are stored in this base. These documents have some different types:

a) Some documents are related to event models and EPNs. There are also some documents about tasks, relationships between tasks and events, and task relationships (to extract task graph).

b) Some other information in this base is the ontology about business processes. By analyzing these documents, it can be specified which services are required for a task.

**4) SLA Base.** Here is the place where the information about approved SLA for any user of any environment is registered. Search agents use this base to take the user SLA into account while searching for suitable services.

**5) Service Registry.** This registers service descriptions and their QoS information. ESB uses this registry to search and select services.

**6) Buses.** The cloud has two types of buses: service bus and data bus. Data bus is used by internal components of the cloud when they need to access some data from each base (in fact, it is used to facilitate accessing these data). Service bus could be used by both internal components and users. It is in relation with service registry, invokes the services and replies the results.

**7) Cloud Broker.** This is the part that makes relationships between environments. As said before, other clouds, legacy systems, a service bus and any other external places can be defined as an environment. Cloud broker is the fusion of these environments. The user defines the type of relationship (a XML file, a message based connection, a web service, an event, etc.), and then the cloud handles the integration by itself. For example, if another cloud is defined as an environment, the relationships between these two clouds happen through events. Any request from a cloud to other one will be sent as an event.

**8) Access Portal.** This part interacts with privileged users directly. One part of this portal is development tools which are in charge of defining environments and users, developing services, creating ontologies and documents, defining events and tasks. Another part is SLA configuration which allows the user to configure needed SLA for each user/group of the environment. Finally, the other part is in charge of controlling and monitoring resources.

### E. Infrastructure as a service

This layer provides the hardware resources, virtual machines, dynamic provisioning, virtual machine manager, resource scheduler, dispatcher and any underlying requirements.

## IV. DISCUSSION

In the first section of the paper, we underlined some cloud computing characteristics, and we have incorporated all of these characteristics in the proposed framework of this paper. Now we have discussed each of these characteristics and showed how they were supported by the framework.

Security characteristics can be attained through policies which play the filter role in the cloud. Every enterprise (which has a cloud environment) can scale up/down its resources through virtualization technologies which are provided in the cloud computing.

The cloud broker is the operator of making the cloud integrated with other clouds and software systems. This framework also utilizes event driven approach to make an easier way to communicate with other systems

Finally, the degree of coupling is low because the cloud users just deal with events, and the cloud use ontology to select the appropriate services.

## V. CONCLUSION AND FUTURE WORK

This paper proposed a framework for cloud computing. This framework is a service oriented framework with an event driven view to the environment. The framework has four real layers and one abstract level which are really provided through those other four ones. It tries to provide standard cloud activities and components. In the future we will implement this framework and deploy it on a data center.

## VI. Acknowledgment

I would like to extend my gratitude to many people but I will mention few. Firstly, I am grateful to Dr. Jayant Shekhar for guiding me in the right direction. I am also thankful to my friends and other lecturers for helping me to sort out various problems.

## References

[1] John Harauz, Lorti M. Kaunan. Bruce Potter, "Data Security in the World of Cloud Computing", IEEE Security & Privacy ,Copublished by the IEEE Computer and Reliability Societies, July/August 2009.

[2] Marios D. Dikaiakos, George Pall is, Dimitrios Katsaros, Pankaj Mehra, Athena Vakali, "Cloud computing : Distributed Internet Computing for IT and Scientiic Research", IEEE Internet Computing, Published by the IEEE Computer Society, September/October 2009.

[3] D.S. Linthicum, Cloud Computing and SOA Convergence in Your Enterprise, Addison-Wesley, 2010.

[4] Bhaskar Prasad Rimal, Eunmi Choi, "A txonomy and survey of cloud computing systems", 2009 Fith International Joint Conference on INC, IMS and IDC, published by IEEE Computer Society.

[5] O. Etzion, P. Niblett, “Event Processing in Action”, MANNING 2011

[6] EUCALYPTUS Cloud, http://www.eweek.com/c/a/Cloud-omputing/Eucalyptus-Offers-OpenSource-VMwareBased - Cloud-Platform-100923/

[7] J.L. Maréchaux , “Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus,” IBM Journal, 2006, http://www.ibm.com/developerworks/library/ws-soa-eda-esb