Comparing design pattern compatibility with Cloud Patterns

Apurv Deshpande
15001687
MSc in Cloud Computing

6th September 2017

Abstract

Cloud Computing provides Scalability, Reliability, Pay Per Use features that will benefit organisations by migrating their applications to the cloud. Many Cloud Computing Service Providers has provided patterns to help migration of Applications to cloud. Object Oriented Design patterns is used in Software Industry for a long time and many applications are designed using design patterns. Various terminologies like semantic web ontology has been defined to describe design patterns. This can also extended to define cloud patterns. In this research a score based methodology is used to compare design and cloud patterns. The cloud patterns used are provided by two cloud providers AWS and Microsoft Azure. Also compatibility of design pattens with AWS and Azure is compared.

Contents

1 Introduction

The design patterns has been used for a long time now for software development. With the evolution of technology cloud computing has come as a cheap, reliable, secure source for hosting applications. With the growth of cloud computing, many organisations are moving their business to the cloud. But with this the problem come on how to migrate the legacy applications to cloud that were designed to work in only specific environment. It is not possible to completely redesign the application as it is too costly and time consuming. Also it is difficult to migrate applications that are written in specific languages and those which are platform dependent. Even if different applications has different architecture and design but mostly they share the common design pattern. With the advancement in technology different techniques have been developed to define this design patterns. Mostly the design patterns used are the 23 standard design patterns. defined by gang for four.Gamma (1995). Also as advancement in Cloud computing many cloud vendor and researchers have published patterns to help application portability. In this research the patterns used are published by cloud vendors AWS and Azure. A semantic web ontology has been used to define both design patterns and cloud patterns. This research is dependent on score based methodology as proposed in Di Martino et al. (2015) in which compatibility of design patterns with AWS Patterns is done. With some modifications in formulas the same methodology is used to compare how Microsoft Azure is compatible with design pattern. The main aim of this research is to find out gftwhich cloud patterns of AWS and Azure are more suitable for particular design patterns. Are AWS Patterns more compatible with design patterns or is Microsoft Azure Patterns? The rest of the paper is structured as follows. Section 2 is about literature review in which previous work done in same field are discussed. Section 3 presents the Methodology in which it is explained why particular design and cloud patterns are selected. It also discuss about the chosen methodology and similar methodologies and section concludes with the advantages and limitations of chosen methodology, Section 4 explain the design used with description of designs main components ,Section 5 describes the implementation process in which formulas are defined to check the compatibility of design patterns with cloud patterns. In Section 6 Evaluation is done with case study with selected design and cloud patterns and also Results are explained to decide whether AWS or Azure is good solution.Section 7 concludes the research with future work.

2 Literature Review

In this paperDi Martino et al. (2015) a score based methodology is proposed to find the common characteristics between design patterns and cloud patterns. The design pattern used in case study is observer and AWS cloud patterns are used. This research is dependent on semantic representation of cloud and design patterns as discussed in

Di Martino and Esposito (2013). Owl web ontology, ODOL ontology and OWL-S is used to describe the processing of mapping of design and cloud patterns. The Object Design Ontology Layer (ODOL) is an OWL ontology specifically designed for detailed explanation of design patterns. This ODOL ontology is further expanded to the description of cloud patterns. The main concepts to describe the design and cloud patterns used are Intent, Context, Consequence and Participants of Pattern. While the Motivation behind the pattern, Implementation of Pattern and Source Code of Patterns were discarded. In brief, Intent can be defined as Main Aim of Pattern, Context is how the pattern is Applicable, Consequence is the positive and negative effects of pattern, and participants are the active components of the pattern. The formulas are applied for each of this four concepts as defined below.

3 Methodology

3.1 Design Pattern

The design patterns used in this research are the 23 standard design patterns defined in Gamma (1995). Design patterns can be defined as the solution for the recurring problems that came while developing a software. This 23 design patterns are divided into three main types Creational, Structural and Behavioral Patterns. "Creational patterns concern the process of object creation. Structural patterns deal with the composition of classes or objects. Behavioral patterns characterize the ways in which classes or objects interact and distribute responsibility." In this research one design pattern from each creational, structural and behavioral is selected as case studies to study the mapping process. Selected patterns include the prototype pattern from creational category, decorator from structural and observer from behavioral properties. The observer and decorator patterns are selected because they are the most common design pattern used in application development. The most common used structural pattern is Singleton but singleton is not used as it is too simple and hence prototype pattern is used. Also prototype has common properties and components to other creational patterns like Abstract Factory pattern.

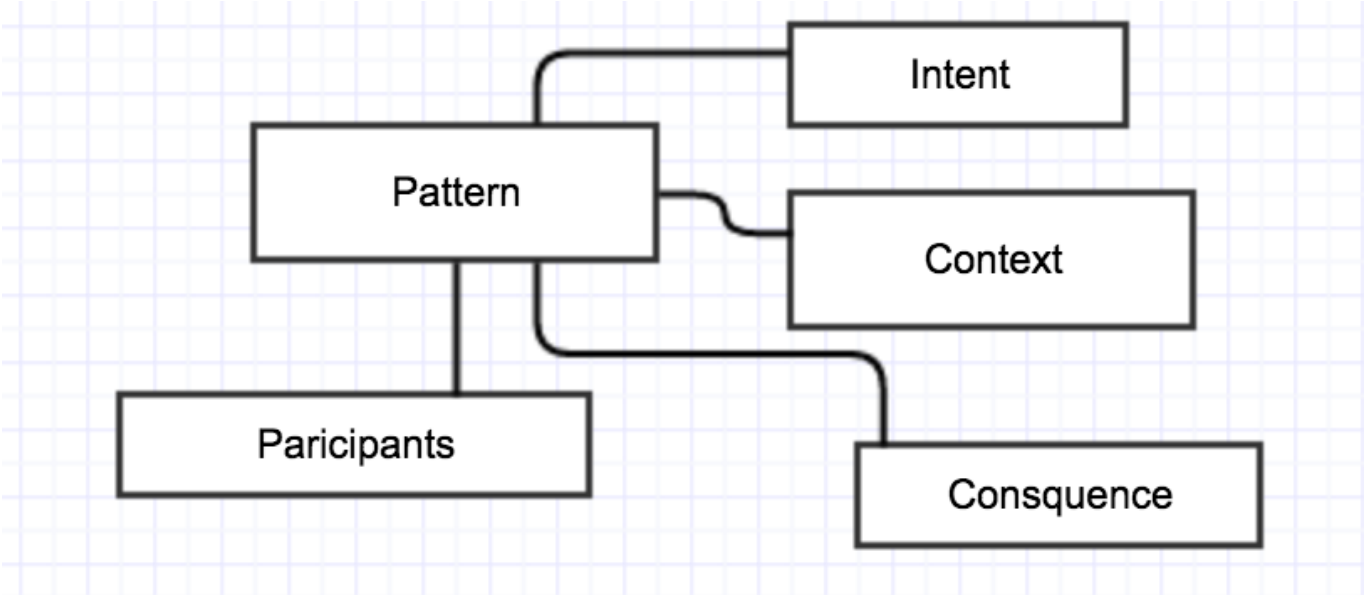
3.2 Cloud Patterns

Cloud patterns can be defined as the solution to recurring problem in cloud computing. Many researchers and cloud vendor has proposed patterns to facilitate application portability to cloud. This patterns are both platform dependent and platform independent. The patterns described by researchers as in are platform independent. Many cloud vendors like AWS and Azure has designed their own patterns specific to their environment. In this research AWS and Azure patterns are used for mapping with design patterns. The results of mapping of AWS patterns with design patterns is compared with mapping of Microsoft Azure patterns with design patterns. In this research the patterns of AWS and Azure are selected with one to one correspondence with each other. The selected cloud patterns are of different types and have different properties from one another. They are broadly classified into basic patterns, Patterns to increase the availability, cloud deployment patterns and patterns to handle the static and dynamic content.

3.3 Selection of Method Applied

Many different techniques has been used to describe design patterns. The UML diagrams and textual descriptions of patterns have been explained in design pattern bookGamma (1995). This book gives all the information about the design patterns like Intent of Pattern, Context, Implementation of Pattern, components of design pattern and effects of design pattern. This properties can be extended to be applied to cloud patterns. Many cloud providers vendors like AWS and Azure has proposed their own pattern which will help in application portability to cloud. This cloud patterns can be explained in same way as design patterns to find the intent, context ,consequences and components of cloud patterns. By describing the cloud patterns in the same way as design patterns it is possible to know the common Aim(Intent), Context and Consequences of design and cloud patterns. The next section describes the design and its specification.

4 Design Specification



The research is based on semantic representation of patterns. The concepts included are from semantic web ontology are Intent, Context, Consequence and Participants as described in Di Martino et al. (2015) and Di Martino and Esposito (2013) This four concepts are used to describe both design and cloud patterns. The concepts are as follows: 1) Intent defines the main purpose and objectives of the patterns. Intent can also be defined as the main Aim for which pattern is described. It is connected to Pattern concept via hasIntent property.
2) Context defines the applicability of the solution that is described by the pattern. Context is important property because it help in modification of patterns. It is connected to pattern concept via a usedInContext property.
3) Consequence is defined as the result or effect of the pattern.It should fulfil the goals presented by Intent Concept derived from the Intent property of the pattern. The results can be both positive and negative. By using Consequence property we can know the benefits, limitations and side effects of the pattern.It is connected to pattern concept through hasConsequence property.

4) Participant can be defined as the entities who actively participate to implement the functionalities of the pattern. They are the core components of patterns on which pattern implementation, design and working of the pattern depends. It is connected to pattern concept through hasParticipant Property.

5 Implementation

As stated above, the main goal of this research is to find about the relationship between design patterns and Cloud patterns and then comparison of results between two different cloud vendors AWS and Azure. A score based methodolgy is proposed to implement this. The implementation can be done in two ways. In the first part we are using the Intent, Context and Consequence properties of the pattern to get the rank for AWS and Azure Patterns. In the next part we are using the participant concept for finding the matching score between the participants of design patterns and cloud patterns.

5.1 Selection of Subset Of Patterns

The intent, context and consequence properties can be used to shortlist the patterns that are going to be used in actual mapping as proposed in Di Martino et al. (2015)
1.Using the intent OWl it is possible to know which design and cloud patterns share the same intent.
2. Using the context concept it is possible to know which design and cloud patterns share the same solutions. 3. Using the Consequence concept the list of patterns can be shortlisted as to know about the positive and negative effects of cloud patterns on design patterns.

Note that one design pattern may be compatible with different cloud patterns. It is many to many relationship between the design and cloud patterns. The formula for intent, context and consequence concept are as follows as defined in Di Martino et al. (2015).
1. The formula for Intent calculation is as follows:
Intent Score= CorrIntent/ NIntensDp. where
CorrIntent are the common intent share by both cloud and design patterns and NintentsDp are the total number of intents of design pattern.
2. The formula for context calculation is same as that of intents
Context Score= CorrIntent/ NContextsDp
where CorrContext are the common context share by both cloud and design patterns and NContextDp are the total number of context of design pattern. 3. The formula for consequence is as follows:
ConseqScore= CorrConseq-AddedNegConseq/ NConseqDp + NconseqCP
where CorrConseq are the common consequences defined in both design and cloud patterns. AddedNegConseq are the total no of negative consequences that appear in cloud pattern which are not present in design patterns. NConseqDp are the number of consequences of design Patterns and NConseqCp are the number of consequence of Cloud Patterns. The above formulas are defined in this research Di Martino et al. (2015). The formulas for intent and context are going to be same. But I am proposing a slightly different formula for consequence. As we are calculating score comparing cloud patterns on the basis of design pattern hence the score should be calculated on the basis of design patterns. The proposed formula is:

ConseqScore= CorrConseq-AddedNegConseq/ NConseqDp.
The range of intent and context scores are from 0 to 1. If the ConseqScore is less than 0 between the patterns then the pattern is rejected as cloud pattern will have more negative impact on particular design pattern. Once we get the results of intent, context and consequence scores the following formula can be applied to calculate the matching score called Rank.
Rank=a*IntentScore+b*ContextScore+c*ConsequenceScore.
here a,b,c are constants. The value of a,b,c will depend on user as some users want only the intent concept and discard context and Consequence. But the basic condition is a+b+c=1 as defined in Di Martino et al. (2015). In this research to keep things simple we are using a=b=c=1/3
If the consequence score is zero or positive then the next steps are used to match the participants of design and cloud patterns.

5.2 Design and Cloud Patterns Participants Matching

Participant are the actual components of the patterns. They are the basic buildings blocks of both design and cloud patterns. The formula for Matching the Participants of Cloud and Design pattern as defined in Di Martino et al. (2015) is as follows.
MatchScore= MatchingParticipants/NParticipantsDp
where Matchingparticipants are the common components in both design and cloud patterns and NParticipantsDp are the total number of participants Of Design pattern.

6 Evaluation

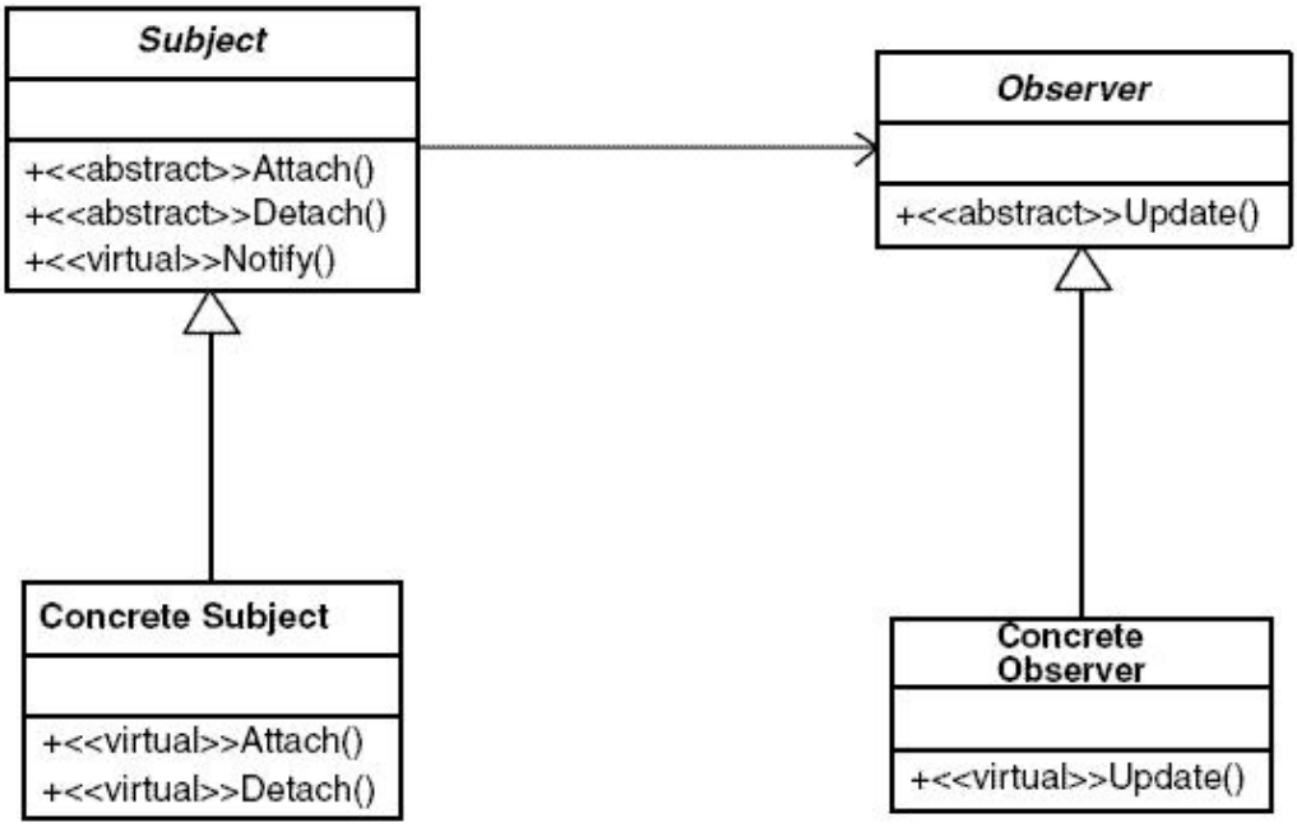
In this section the above procedure is matched to a set of design patterns and cloud patterns. The design pattern used is Observer pattern.

6.1 Observer Pattern

Definition: "Observer pattern is a software design pattern in which an object, called the subject, maintains a list of its dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods." The intent, context, consequence and participants of observer pattern is described in the table. The structure of observer pattern is as follows.

6.2 AWS Patterns

The following is the list of AWS Patterns used.
1.Scale out pattern: How to automatically add/remove virtual resource according to computational load as needed.
2.Priority Queue pattern: Assigning of job according to priority by using queue
3.Queueing Chain Pattern: It helps the sequential task to decouple into independent tasks by using queue.
4.Multi-Datacenter Pattern: By using Multi-Datacenter it is possible to improve availability and help in disaster recovery. 5.Deep Health Check Pattern: It checks the status of services and helps in application monitoring.



6.Inmemory DB Cache Pattern : This pattern help in improving the performance of application by caching the data that is used frequently .

This is just brief introduction of the patterns. Detailed description of patterns is found on website of AWS.

6.3 Azure Patterns

The following is the list of AWS Patterns used.
1. Autoscaling guidance: Dynamic allocation and deallocation of resources as needed by application
2. Priority Queue Pattern: Assigning of job according to priority by using queue
3. Queue-Based Load Leveling Pattern: Improve the performance and availability of application by using queue by managing the resources during peak time and normal time
4. Multiple Datacenter Deployment Guidance: Increase the availability and gain better user experience by deploying your application to multiple Datacenters.
5. Health Endpoint Monitoring Pattern: This pattern helps in verifying that services and application are performing correctly.
6. Cache-Aside Pattern: This pattern improve performance and help in maintaining the consistency of data both in cache and data storage.
This is just brief introduction of the patterns. Detailed description of patterns can be found here Homer et al. (2014)

6.4 Figure

The following diagram shows the characteristics of patterns

Pattern	Intent	Consequence	Context	Participant
Observer Pattern	Loose Coupling Automatic Update Monitoring State Tracking	Loose Coupling(p) Component Reuse(p) Parallel Processing(p) Liability management(p)	Notification System Change Propogation Monitoring System	Subject Observer Concrete Observer Concrete Subject
Scale Out Pattern (AWS)	Load Balancing System Monitoring Resource Scaling Service Decoupling	Parallel Processing(p) Increase Scalability(p) Service Reuse(p)	Notification System Change Propagation Monitoring System Automatic Scaling AMI	Elastic Load Balancer Cloud Watch Auto Scaling
Queuing Chain Pattern(AWS)	Service Decoupling Parallel Processing Failure Resiliency	Service Decoupling(p) Parallel Processing(p) Failure Resiliency(p)	Asynchronous Messaging	Elastic Compute Cloud Simple Queue Service
Priority Queue Pattern(AWS)	Service Decoupling Parallel Processing Failure Resiliency	Service Decoupling(p) Parallel Processing(p) Failure Resiliency(p)	Asynchronous Messaging Priority Management	Elastic Compute Cloud Simple Queue Service
Autoscaling(Azure)	System Monitoring Resource Scaling	Parallel Processing(p) Resource Mangement(p) Horizontal Scaling(p)	Change propagation Dynamic Allocation	Windows Azure Autoscaling
Queue based Load Levelling (Azure)	Load Balancing System Monitoring Increase Avalibality	Increase Scalability(p) Autoscaling(p)	Scalability Asynchronous Messaging Decoupling	Microsoft Azure Web Role Message Queue
Priority Queue Pattern(Azure)	System Monitoring Resource Scaling Parallel Processing	Parallel Processing(p) Dynamic Allocation Change Propogation	Asynchronous Messaging	Windows Azure Service Bus

The following diagram show the scores of cloud patterns while comparing with observer design pattern .

Here two scores of consequence are there one based on old formula in Di Martino et al. (2015) and one based on new formula as defined in previous sections of the research. There are some modifications done as service decoupling in cloud patterns means loose coupling in design patterns and also liability management is same concept as resource management. This discrepancies can be solved by defining a common vocabulary for both design patterns and cloud patterns.

6.5 List of AWS Participants

The list of Participants are already describedDi Martino et al. (2015) and used as are as follows: 1.Simple Queue Service(SQS): SQS is a queue management service to store data, messages while travelling from one component to other,
2.Elastic Compute Cloud(EC2): EC2 is a service that provides scalable computing capacity in the form of virtual machine in the cloud.
3. CloudWatch: It is monitoring service used to check applications and services in cloud. Auto feature of notification can be set to do automatic check in application.
4. Autoscaling: It helps in scaling up and down EC2 instances automatically according to the need of services.
5. Elastic Load Balancer (ELB): It is used to automatically balances the load between

Pattern comparison with observer	Intent Score	Context Score	Consequence Score(new formula)	Consequence Score (old formula)	Rank (new consequence score)	Rank old consequence score
Scale Out Pattern (AWS)	0.5	1	0.5	0.3	0.66	0.60
Queuing Chain Pattern(AWS)	0.25	0	0.5	0.3	0.25	0.18
Priority Queue Pattern(AWS)	0.25	0	0.5	0.3	0.25	0.18
Autoscaling (Azure)	0.25	1	0.5	0.3	0.58	0.52
Queue based Load Levelling (Azure)	0.5	0.33	0.25	0.14	0.36	0.33
Priority Queue Pattern(Azure)	0.25	0.66	0.25	0.14	0.39	0.34

EC2 instances according to need of application.
further information can be found on AWS website.

6.6 List of Azure Participants

1.Microsoft Windows Azure Monitoring Services Management Library: It is an API that that provides Automatic Autoscaling and also provides automatic notifications and metrics to check the load of the application
2. Microsoft Azure Web Role: It provide separate storage for data while multiple instance are running concurrently to access the same data store.
3. Message Queue: Message queue is used to balance the load between Web Role and storage services. 4. Windows Azure Service Bus: It provide a mechanism to handle the priority request of the queues.

6.7 Results Evaluation

Here two scores of consequence are there one based on old formula in Di Martino et al. (2015) and one based on new formula as defined in previous sections of the research. There are some modifications done as service decoupling in cloud patterns means loose coupling in design patterns and also liability management is same concept as resource

management. This discrepancies can be solved by defining a common vocabulary for both design patterns and cloud patterns. If the consequence score is less than zero then the cloud pattern is not compatible with design patterns. The positive effects of cloud pattern are shown by(p) and negative effects are shown by(n). Here there is one to one dependence between selected AWS pattens and Azure cloud patterns. The Scale Out Pattern of AWS is quite similar to Autoscaling Pattern of Azure. Similarly Queuing Chain Pattern of AWS is similar to Queue based Load Levelling pattern and Priority Queue Pattern of AWS is similar to Priority Queue Pattern of AWS. The most promising results are for Scale Out Pattern (AWS) and Autoscaling Pattern of Azure while Scale Out Pattern of AWS is slightly better than (8 percent) than Autoscaling Pattern for Azure. Also Queue based Load Levelling and Priority Queue Pattern of Azure are far better(almost double) in terms of rank than Queuing Chain Pattern and Priority Queue Pattern of AWS. Also the context score is zero for Queuing Chain Pattern and Priority Queue Pattern of AWS. As the rank of both AWS and Autoscaling is similar to one another then further competency can be checked by using the participants of Patterns. The concept of Participants can be used when users want to know more about the competency of design patterns with cloud pattern based on the components of patterns.

6.8 Participant Concept

The following figure shows the participant and it scores. Here there can be one to many

	A	B	C	D	E
	Observer Pattern	**Scale Out Pattern(AWS)**	**Autoscaling (Azure)**	**Priority Queue Pattern(AWS)**	**Priority Queue Pattern(Azure)**
	Subject				
	Concrete Subject	CloudWatch EC2	Azure Application Insights	SQS EC2	Windows Azure Service Bus
	Observer				
	Concrete Observer	ELB CloudWatch Autoscaling	Azure Application Insights Autoscaling		
	Matching Score	0.5	0.5	0.25	0.25

dependencies between the components of design pattern with Cloud pattern. It is dependent on how participant play their role during the execution. For example Concrete Subject Component is equivalent to CloudWatch and EC2 participants of Scale Out Pattern(AWS). As subject notifies the observer so Cloud watch is a service used .And to execute it you need computing power hence EC2 component is equivalent to Subject of observer Pattern. But this two are compared as one unit of Matching Participant as both are matched with concrete subject while applying formula. The Matching Score Of Participants is similar both in Scale Out Pattern(AWS) and Autoscaling (Azure) Also Priority Queue Pattern(AWS) and Priority Queue Pattern(Azure). It is also similar in

third set of patterns. Hence we can infer that participant of cloud patterns(both AWS and Azure) will not help in choosing which pattern is more suitable AWS and Azure for observer design pattern. It can be also extended to other design patterns. As services in AWS and Azure are similar as defined in . Hence the rank system previously explained based on intent, context and consequence property only can be used to infer which design patterns are suitable for which cloud patterns of AWS or Azure. Sometimes some patterns AWS will have more compatibility with particular design pattern and sometimes Azure will have more compatibility pattern. In this case study Scale Out Patten(AWS) is more compatible with design pattern than Autoscaling(Azure) while Queue based Load Levelling and Priority Queue Pattern of Azure are better than Queuing Chain Pattern and Priority Queue Pattern of AWS. Hence it depends on particular type of design or cloud pattern. It can't be concluded that AWS is superior than Azure or vice versa.

7 Conclusions and Future Work

In this Research Paper two types of methodologies based on semantic web ontology are proposed to find which cloud Pattern AWS or Azure is more compatible with design patterns based on the characteristics, properties and components of design and cloud patterns. In the first one a score based methodology was used in which intent, context and consequence of design and cloud pattern were taken into consideration and result was found that every design and cloud design pattern has their unique characteristics and it can't be decided whether particular AWS or Azure Cloud Provider patterns is more compatible with design patterns. In some cases as explained above AWS patterns are more compatible with design patterns than Azure patterns while in some cases Azure patterns are more compatible with design patterns than AWS pattern. From the second type of methodology in which components called as participants are mapped with design patterns it is found that components(participants) of both AWS Cloud and Azure are quite similar and hence components(participants) of patterns can't be a factor between the services of AWS and Azure.
The whole process is conducted manually and it can be automated by using a software that can parse the given description of pattern and able to find it properties and components automatically. The mapping process between design and cloud patterns also can be automated using matchmaking algorithm.

References

Di Martino, B., Cretella, G. and Esposito, A. (2015). Mapping design patterns to cloud patterns to support application portability: a preliminary study, *Proceedings of the 12th ACM International Conference on Computing Frontiers*, ACM, p. 50.

Di Martino, B. and Esposito, A. (2013). Towards a common semantic representation of design and cloud patterns, *Proceedings of International Conference on Information Integration and Web-based Applications & Services*, ACM, p. 385.

Gamma, E. (1995). *Design patterns: elements of reusable object-oriented software*, Pearson Education India.

Homer, A., Sharp, J., Brader, L., Narumoto, M. and Swanson, T. (2014). *Cloud Design Patterns: Prescriptive Architecture Guidance for Cloud Applications*, Microsoft patterns & practices.