Jordanian Journal of Computers and Information Technology (JJCIT), Vol. 06, No. 02, June 2020.

WEAVESIM: A SCALABLE AND REUSABLE CLOUD SIMULATION FRAMEWORK LEVERAGING ASPECT-ORIENTED PROGRAMMING

Anas M. R. AlSobeh[1], Sawsan AlShattnawi [1], Amin Jarrah[1] and Mahmoud M. Hammad[2]

(Received: 10-Feb.-2020, Revised: 16-Mar.-2020, Accepted: 4-Apr.-2020)

ABSTRACT

Cloud computing service-oriented simulation frameworks are very important tools for modeling and simulating the dynamic behavior of cloud-based software systems. However, the existing service-oriented simulation frameworks lack the ability to measure and control the rapidly changing (adaptive) requirements that span over many modules in cloud-based software systems, such as security, logging, monitoring, ...etc. To address these limitations, this paper presents an efficient framework for reducing the complexity of modeling and simulating the custom and dynamic behavior of cloud-based applications, called WeaveSim. WeaveSim utilizes the aspect-oriented programming (AOP) to encapsulate the complexity of developing the dynamic behavior of cloud-based applications by adding another abstract layer called Context-Aware Aspect Layer (CAAL). CAAL reduces the complexity of using CloudSim to simulate cloud-based software systems. Examples of cross-cutting concerns are data encryption, logging and monitoring. Since implementing a cross-cutting concern on a cloud-based simulator, such as CloudSim, requires modifications, from developers, to many core modules of that simulator. However, using WeaveSim, implementing cross-cutting concerns would be an easy task for developers, since they only need to reuse pre-defined joinpoints and pointcuts without modifying the underlying core modules of the simulator. We evaluated WeaveSim on an academically-scaled system. The results of our experimental evaluations show the benefit of WeaveSim in reducing the complexity of implementing cross-cutting concerns on cloud-based software systems. Hence, the reusability, scalability and maintainability of the cloud-based software systems are increased.

KEYWORDS

Model-based aspect-oriented programming, Cloud simulation, CloudSim, Dynamic modeling, Cloud architecture.

1. INTRODUCTION

Cloud computing provides easy access and reuse of shared resources anytime, everywhere. The appearance of cloud-based applications is often developed across multiple scattered units, some of which are called at run time. To achieve cloud collaboration services in cloud computing, cloud computing architecture must provide better-paired modules, such as cloud data access, monitoring, data compression, security and scheduling concerns [1]. These concerns about the cloud have increased the market value of many customers who want to use the cloud to achieve their organization's goals faster. This space is particularly complex, as these systems generally cost millions of dollars to develop and hundreds of thousands or more in annual cloud deployment costs. For example, security is a common secondary requirement, which is a comprehensive interest in cloud computing. The security application is required to interact with a set of scattered resources, contacts and nested context data objects stored in cloud instances.

By nature, a cloud-based application, or cloud app for short, is a complex and dynamic software system that consists of a set of contextual attributes and communicates with various services over the Internet. These attributes are scattered over different cloud levels: Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS) [2]. The required test of such cross-cutting concerns is very important for cloud users (e.g., service providers and consumers). Hence, the developer needs some crucial insight into how to implement these concerns, such as optimization of application efficiently. The implementation of cross-cutting concerns aims at improving the evolutionarily,

1. A. AlSobeh, S. AlShattnawi and A. Jarrah are with the Department of Computer Information Systems, Yarmouk University, Irbid, Jordan. Emails: `anas.alsobeh@yu.edu.jo`, `Sawsankh@yu.edu.jo` and `amin.jarrah@yu.edu.jo`
2. M. Hammad is with Software Engineering Department, Jordan University for Science and Technology (JUST), Irbid, Jordan. Email: `m-hammad@just.edu.jo`

evolvability, usability, understandability and efficiency of cloud apps.

This paper proposes a scheme that implements an efficient framework for reducing the complexity burden over cloud modeling and simulating the custom dynamic behavior of cloud-based solutions and applications. The proposed framework is structured into an abstract layer and aspects that are stored in the cloud by the simulator along with access to context-aware metadata. CloudSim simulator [3] is the most popular and extensible simulator that enables modeling, simulation and performance evaluation of emerging cloud computing applications [4]-[5] . CloudSim in its native state can be problematic to get up to use. Moreover, design choices and issues that were made and have evolved during its development have created significant issues in terms of its codebase as well as its efficiency and, with respect to certain aspects and details, its scalability and reusability. However, implementing and measuring the behavior of cloud apps against adaptive requirements are challenging tasks and are issues that have not been solved completely yet. Filho et al. [6] redesigned the CloudSim's core code components to increase its scalability and maintenance. In addition, they proposed features to enable dynamic monitoring behavior such as using listeners and performing dynamic operations such as arrival and destruction of virtual machines (VMs), horizontal and vertical VM scaling, fault injection and recovery, dynamic exchange of policies in runtime, …etc. However, this work did not solve the main contextual metadata complexity problem to offer scalability at runtime. In other words, it is not easy and clear to implement secondary requirements, since CloudSim does not provide clear insight for developers into implementing them. This dilemma prevents developers from evaluating the adaptive requirements of their cloud apps as a singular abstract module; i.e., separation of concern (SoC), such as monitoring, security, performance, cryptography, hive, map-reduce, throughput, …etc [1]. This highlights a general lack of sufficient care and accuracy in overreaching in terms of what value CloudSim would actually provide against the overall domain we target. Hence, value exists in creating overlay frameworks and/or abstraction layers so as to make CloudSim more accessible to its potential user community. Using Aspect-Oriented Programming (AOP) makes our work distinguished in that it does not require core code redesign or amendment and might be adaptable over any CloudSim architecture without key changes, viz. obliviousness process.

The significant concern about applying Aspect-Oriented Programming (AOP) into cloud application is dealing with an advanced access control distributed objects; for example, due to different issues such as modification in cloud behavior levels to access the shared resources. Using AOP provides a dynamic and flexible process to add probes for cloud-related cross-cutting concerns and to evaluate the system against related aspects; it encapsulates cross-cutting concerns in first high-level modules. AOP [7] is the appropriate approach for implementing cross-cutting concerns in separate modules in the CloudSim simulator dynamically. To add aspects into CloudSim, we have to extract relevant cloud-related contextual information, introduce helper characteristics and build utility functions, as a promising programming technique that promotes reusability and scalability of software systems through the separation of cross-cutting concerns [8].

The proposed simulation framework is a very abstract cloud collaborative scheme along with an efficient, expanded cloud application layer that leverages AOP to allow developers to efficiently implement and measure the dynamic and complex capabilities associated with cloud environments, viz. WeaveSim. It enables a developer to simulate the ability of the cloud-deployed cross-cutting concern to respond to time-domain variations in the volume and/or nature of the incoming workloads that it is servicing, e.g., which in turn is supported via elastic cloud services, …etc. WeaveSim deals with processing complex cloud- related cross-cutting concerns into separate first-class modules over the cloud. The key contributions of WeaveSim lie in refactoring CloudSim with:

- Supporting AOP-based application-level models for providing the cloud domain attributes,
- Processing cloud-based context-aware aspects using joinpoint-advice model, which's executed based on the creation of multiple VMs as well as a single VM function and
- Facilitating injection of cross-cutting concerns using an abstract pointcut-joinpoint model.

According to our conducted experiments, WeaveSim (1) demonstrated its ability to help developers evolve dynamic requirements of cloud apps with less complicated functionalities and (2) improved the reusability and scalability of cloud apps by creating better SoC efficiently.

The rest of the paper is organized as follows; various methods having been used in the literature as given in Section 2. The overall WeaveSim architecture is given in the methodology in Section 3. To evaluate the efficiency of WeaveSim, Section 4 provides an experimental case study in which we implemented a security cross-cutting concern scenario, then we compared the quality of WeaveSim with CloudSim in terms of scalablity factors: evolvability, functionality, usability, maintainability and efficiency in the results and discussion in Section 5. The conclusions with avenues of future work are depicted in Section 6.

2. RESEARCH WORK: A BRIEF REVIEW

2.1 Cloud Computing Simulation Frameworks

Simulators manage and control the infrastructure of the cloud hardware and software components. They provide information and key performance indicators for both cloud-based platform and applications. Those simulators vary in characteristics, result assessment, result validation metrics and symptoms of cross-cuttingness. In this section, we will introduce CloudSim simulator and the recent simulators built over CloudSim [9].

Kumar et al. [10] have provided a generalized and extensible simulator for modeling and efficient experimentation of cloud infrastructures and application services. It enables simulating the intrinsic distributed environment of cloud providers in a computer and provides a controlled environment that is easy to setup to test the performance of cloud applications. In their work, they formulated the CloudSim architecture to simulate different types of cloud environments (public, private, hybrid and multi-cloud environments) for key issues of cloud-based applications by creating cloudlet instances, which are submitted and processed by VMs deployed in the cloud [11]. CloudSim is built on top of the core engine of grid simulator, GridSim [12], and it is based on Java. It is an open source, event-driven extensible simulator that has diverse cloud features and capabilities and can be extended to include many plug-ins [13]. Due to its extendability, many authors proposed additional features and capabilities to the CloudSim, including [14]-[15]. The proposed CloudSim added many features, such as supporting modeling and simulation for large data center applications, the software architecture and simulation which are energy-aware computational resources, supporting both the network topology and message passing techniques to be compatible with a wide range of applications and supporting the ability to customize the polices for host resources to virtual machines. The existing proposed simulation models are not fully integrated into the requested design and implementation cycle for different applications. The code of some of them is complicated and very difficult to understand [13]-[14], which may prevent adding more functionality. In [15], the authors proposed a system that needs more time to manipulate such that each participant requests only partial information. The scalability of the proposed CloudSim is enhanced by combining increasing the volume of service request technique and deploying multiple instances of the service software. The proposed system maintains the quality of service in terms of average response time, where the scaling behavior is maintained over a long-time scale. Moreover, the proposed model extracts the information and constructs the initial simulation file in an efficient way, where the designer can automatically extract information into the targeted simulation model and run it remotely. This improved proposed system can be applied for many applications and decrease the challenges and drawbacks of the current techniques.

CloudSim has a layered architecture which is built on top of the GridSim [12] simulator. It consists of four layers [3]: (1) Simjava layer that implements the core functionalities required for higher-level simulation; (2) GridSim layer which supports high-level software components for modeling multiple grid infrastructure; (3) Management layer that manages the data centers, hosts, CPU units and VMs; (4) Application layer which allows users to write a code to configure the functionalities of a host, applications, VMs and broker scheduling policies.

Figure 1 shows the core components of the CloudSim, consisting of a base platform for simulation and research. The components form the infrastructure on which CloudSim is based and their relationships, which show the dependency arising from the interactions and overlap between these core components, as well as the oscillation resulting from the distribution of common objects between them [16] as follows: DataCenter component is a module that implements the core infrastructure level provided by the providers of cloud resources. DataCenterBroker is a class that models the relationship between end-users and service providers to allocate resources according to certain quality of service (QoS) requirements. Cloudlet contains a set of modules for cloud apps services deployed in data centers. VM is a module that runs different instances of VMs that are considered parts of the host. The host can instantiate several VMs and allocate the cores according to the internal scheduling policy, which is extended from the abstract component called VMScheduling. In [17] survey, the authors show that CloudSim emerges as a platform for simulation research and shows a lack of support for distributed implementation tools.

Despite CloudSim is written in Java, an object-oriented (OO) language, the framework barely relies on the intrinsic OO message-passing mechanism; i.e., the utilization of relationships between entities to perform method calls across such entities. CloudSim is an even-driven simulation framework that relies on custom message queue mechanism to enable communication between entities, such as Datacenters, Hosts, VMs, Brokers, …etc. The implemented mechanism transmits data inside events using the object raw type, instead of enabling type-safe message passing. Despite that, you can store anything inside an object variable, which is error-prone and usually leads to runtime exceptions, making tracing the root cause difficult.

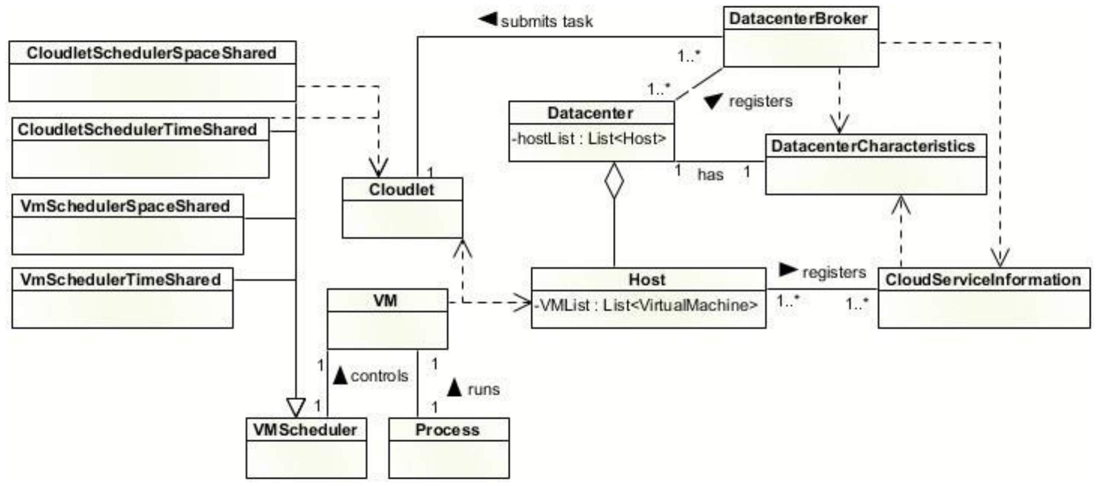


Figure 1. Key components of the CloudSim.

Despite all the aforementioned CloudSim features, the complexity of design scheme in master control and deep computation to control context attributes of cloud services makes it difficult to understand which type of context data each different event has to send. It allows the sender to transmit a data type different from that the receiver is expecting, leading to runtime exceptions if the receiver doesn't check whether or not the type of the received data is correct. Even if the receiver checks the data, there is a waste in processing to receive and discard an invalid event which shouldn't have been sent in the first place. This challenge prevents its scheme from being extensible and scalable effectively.

Wickremasinghe et al. [18] proposed CloudAnalyst simulator, which is derived from CloudSim, extended some capabilities and focused on evaluating performance and cost of large-scale Internet applications in a cloud environment. CloudAnalyst presents a significant challenge for a huge user workload. In attribute-based cloud applications, this issue is even more difficult, since each context data is conceivably scattered and tangled by multiple objects. This is why the current mechanism implemented in CloudSim is error-prone and not easy to understand and extend. This way, creating a data retrieval mechanism on top of this inappropriate CloudSim's message passing mechanism is questionable, mainly when one of your goals is to favor extensibility. Hence, it does not allow for separation of service abstractions and resources required by cloud applications. This presents a significant challenge for users who simulate cross-cutting concerns on the cloud. The next sub-section discusses how AOP copes with this challenge dynamically.

CloudSim4DWF is another simulator based on CloudSim to provide a new resources' provisioning policy for dynamic workflow applications. CloudSim4DWF added three modules to the CloudSim: (1) a graphical user interface (GUI) module that enables users to manage different types of VMs and to provide the inputs needed for simulation; (2) an event injection module aiming to trigger some events according to the dynamic workflow application; and (3) a resources' provisioning module for ensuring efficient resource provisioning for dynamic workflow. CloudSim4DWF injects some events that change the workflow during the runtime and adapts the actions to meet Quality of Service (QoS) requirements [19]. Dynamic CloudSim extends CloudSim to simulate instability and dynamic performance changes in virtual machines (VMs) during runtime. It determines whether a task succeeds or fails [20]. All of the presented simulation tools do not address cross-cutting concerns to improve cloud application evolvability. To address their limitations, we have designed and implemented WeaveSim framework that leverages AOP concepts to implement cross-cutting concerns efficiently.

Some simulators are presented in literature, such as GreenCloud [21] and CloudNetSim++ [22] for energy-aware cloud computing data centers. These simulators are designed to capture the energy consumed by data center components, such as servers, switches and links. GreenCloud is developed as an extension of a packet-level network simulator built on top of Ns2. The authors analyzed the network behavior of various data center network architectures over a designing data center simulator. They didn't consider the distributed data centers. CloudNetSim++ is built on the top of OMNeT++. It is designed to utilize the computing power of the data centers considering the distributed nature of data centers.

Other simulations rely primarily on CloudSim, as it is the basic infrastructure for other simulations (such as CloudSim plus, dynamic CloudSim, …etc.). These simulations suggest just reworking the code without showing any kind of updates with the same code limitations. So, our work is compared to become a competitor to CloudSim and is adapted to be more scalable in the future. Moreover, Byrne et al. in [23] reviewed 33 cloud-based tools. It identifies the emergence of CloudSim as a de facto base platform for simulation development and research. 18 of the platforms analyzed were derivatives or extensions of CloudSim. This is not surprising given the early mover advantage that CloudSim had, the eminence of the researchers involved and the quality and timeliness of the release of the simulator platform.

2.2 Cross-cutting Concerns in AOP Concepts

Cross-cutting concerns are aspects of a software system that span over many modules and affect the entire system. For example, monitoring, authentication, authorization and data encryption are crucial cross-cutting concerns that affect the entire cloud applications. The implementation of such concerns are either scattered (duplicated), tangled (significant dependencies between modules) or both.

In cloud application, the encryption concerns work as a service invoked when any message is sent or received [24]. Unfortunately, the above cross-cutting security concerns cannot be efficiently captured using the current implementation of CloudSim. To solve this problem, Filho et al. [6] proposed major restructuring and refactoring of the entire code base of the CloudSim, called CloudSim plus. They made a comprehensive re-engineering process to fix lots of existing issues in CloudSim. CloudSim plus was in fact focused on fixing lots of issues in CloudSim to promote extensibility. Changing core classes to create a customized framework could make it very difficult to incorporate improvements in new official versions of the base framework. Such a solution is costly and is not completely backward compatible with the cloud applications that have been built using CloudSim. In addition, there are major adjustments that are required to enable running a CloudSim's application in CloudSim plus. Due to those issues, they really re-structured CloudSim and provide a new general-purpose framework that can be easily extended (without forcing the researcher to change core classes). But, our framework intends to insert a layer over CloudSim instead of reforming the classes. Therefore, AOP concepts can solve this problem by implementing the aforementioned cross-cutting security concerns as separate aspects.

Even though AOP is a very powerful technique that can solve such a problem efficiently, no previous work in the literature has leveraged AOP for cloud simulation. WeaveSim is the first cloud simulation framework that leverages AOP to solve such a problem.

WeaveSim identifies a set of joinpoints in the CloudSim architecture and injects the code of each aspect module which changes the behavior of CloudSim at execution time without altering its core code. The added behavior; i.e., cross-cutting concern logic, is called advice [25]. These aspects encapsulate each concern in a special class, which alters the behavior of the base class by applying the advice at defined points on the original code. These points are called joinpoints; when a query matches at a point, this is called a pointcut [25]. Therefore, WeaveSim extends CloudSim by adding the necessary functions to support the separation of cross-cutting concerns across the core implementation of cloud apps in order to assess cloud services.

3. Overall Methodology: WeaveSim Model and Architecture

Changing the simulations' model of cloud applications changes the actual system's reusability or scalability. In particular, ensuring scalability is retained a critical concern within cloud-deployed software systems as cloud applications evolve and need to service expanding workload demands. This work addresses such issues and claims that it does. The proposed model presents a highly adaptive and collaborative scheme along with an efficient architecture in cloud computing based on CloudSim; i.e., WeaveSim. As described in Figure 2, WeaveSim shows the layered architectural design model of the WeaveSim framework. The architecture consists of six layers which can be applied to any data-driven cloud application that involves cloud-related cross-cutting concerns. The architecture includes: the CloudSim core layer, cloud resource layer, cloud service layer, user interface layer, cloud-aware aspect layer (CAAL) and application- level code layer. These layers are structured as a collection of loosely coupled services providing a solution for inter-service between cloud components in cloud-based applications.

The lower three layers (the CloudSim core, resource and user interface layers) are inherited from the CloudSim architecture. These layers provide infrastructures that facilitate communication, resources and services, respectively, needed to build cloud applications [4]. These layers depend on each other and are tightly coupled, which increases the complexity of implementing cloud-related cross-cutting concerns, e.g., monitoring, management, security, …etc.

The architecture involves an attributed-based context layer, where the context problem aspects from the analysis are embedded in the cloud-aware aspect layer (CAAL). This layer adds advice behaviors as structured activities in the CloudSim. The main reason for using the AOP is lying in making the objects' distribution and cross-cutting services encapsulated, extensible and accessible with less computational expenses with the layer that is executed into a machine.

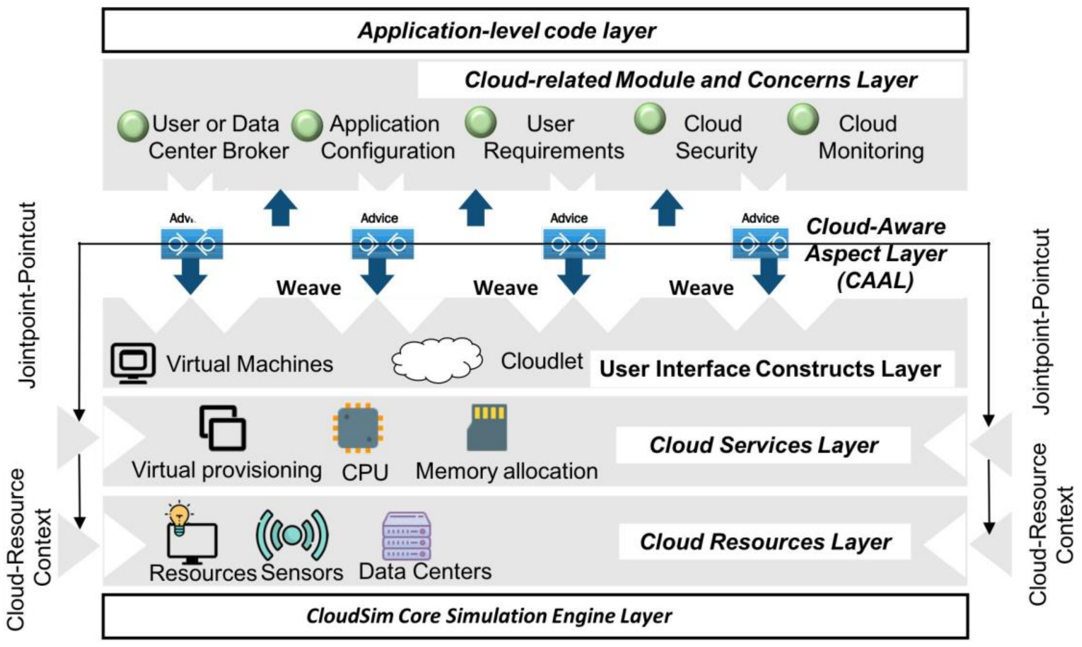


Figure 2. Layered architecture of WeaveSim.

CAAL includes an abstract library which offers services and interfaces to cloud apps. It integrates CloudSim APIs with contextual application-level components to make a cloud app more flexible and dynamic. It is modularized into small services that provide a well-defined functionality using well-accepted design principles. The layer offers the benefit of SoC to build scenarios that can handle the motioned challenge. In design, different packages have been used to make the design dynamic-oriented. Each package has aspects that are related to each cloud component. One of the interesting design decisions is that the system has an application-level code layer, the topmost layer, which contains aspects and methods for cloud objects. These aspects provide the ability to easier implement secondary functional requirements (i.e., cross-cutting concerns) necessary for cloud apps.

3.1 Cloud-Aware Aspect Layer (CAAL)

Cloud-Aware Aspect Layer is broken into a set of components (packages) communicated together. It is concerned with the aspects and mechanisms that are present in the cloud context data problem. It presents monitoring, WeaveSim and CloudService subsystems that monitor and simulate different cloud components in real time. They also make the CloudSim architecture and its functionalities available to developers as an abstract and reusable framework. CAAL is implemented pragmatically by extending the core functionalities of CloudSim. It allows users to select the cloud services and monitor the selected services based on user-defined characteristics. Technically, CAAL implements the cross-cutting concerns as aspects in the cloud architecture. It has been using different WeaveSim's joinpoints that are invoked/executed by aspects' pointcuts as shown in Figure 2. Each joinpoint has a crucial role in the simulation process. To simulate cloud-related cross-cutting concerns in a flexible manner, the CloudSim's meta-data components should be parameterized properly. CAAL extracts heterogeneous meta-data that supports comprehensive constructs of cloud-related cross-cutting concerns (e.g., managing cloud resource allocation and services). Such basic parameterization allows weaving of abstract cross-cutting concerns that might bind multiple cloud components together obliviously. Basically, CAAL implementation introduces a new abstraction level for aspects to overcome the dynamic weaving limitations on the cloud application code, recall Section 3, and it provides the adequate aspect representation in a woven structure based on context information, helper characteristics and utility function concepts.

The class diagram in Figure 3 depicts the structure of the cloud-based model of the dynamic weaving implementation within the CAAL layer. CAAL's structure is designed to embed the behavior of each advice with the weaving process as the structured event-based activities in the simulation process of the CloudSim. These events are being used in distribution service. They expose the context information about cloud components and are marked-up as general abstract classes and aspects to show how different values will be displayed at each joinpoint (e.g., BaseCloudServiceAspect, CloudServiceJP, CloudServiceJP). If new cloud services are added to the cloud application, those services' context information must be captured in such joinpoint which can improve the framework extensibility.

Through this layer, WeaveSim model leverages the intrinsic AOP communication mechanism to pass some context data around; i.e., method calls across a chain of objects. For instance, to pass a message to know in which data center a cloudlet is running, one has to simply cloudlet service joinpoint; i.e., CloudMessageService, which encapsulates message context data and provides a cheap message operation that returns immediately because it creates a SendEventJP and ReceiveEventJP. They cut-through simply call cloudlet.getVm().getHost().getDatacenter(), so there is no need to load a message to a queue, to be processed after a while and then return the response of the sender in an asynchronous manner.

This package is a library for services. It works as a monitoring component that performs low-level tracking such as instantiate, start, stop, resource allocation, management, scheduling, …etc. It automates the monitoring of a cross-cutting concern as a service. Moreover, its implementation determines the properties of various entities of cloud services, their communications and cloud applications to simulate the logical behavior of cross-cutting concerns. The issue with CloudSim is that any service that needs to be processed around goes into the CloudServiceRegistry, making the registry process really not expensive in large-scale simulations.

The package monitoring contains classes for both CloudSim components and cloud services. In Figure 3, CloudServiceJPTracer and BoundsimPoint are aspects extending the core components of the CloudSim by implementing specific new properties which are required to investigate and simulate cloud-related cross-cutting concerns. They cut-through the core components of the CloudSim to pull low-level contextual data characterizing the simulation of cloud-related concerns. Such context data includes services, network communication, VM, resources, device profile, location, calendar and time information.

Figure 4 shows that the CloudserviceJPTracer is an aspect object that cuts-through the CloudSim library components to connect CloudSim classes with defined distributed service (e.g., BaseCloudServiceAspect, ConfigureCloudlet, CloudSimBeginOnInitiator, CloudSimEndOn-Initiator, CloudMessageService, …etc.). It communicates with CloudSim's components via their joinpoints. It is responsible of exposing the spacious need for context data the effect of which manifests many separated parts of the cloud simulations. The collected service data is allocated in CloudsServiceRegistry. This repository uses the template parameter pattern to create a container for the advice methods for dynamically woven services.

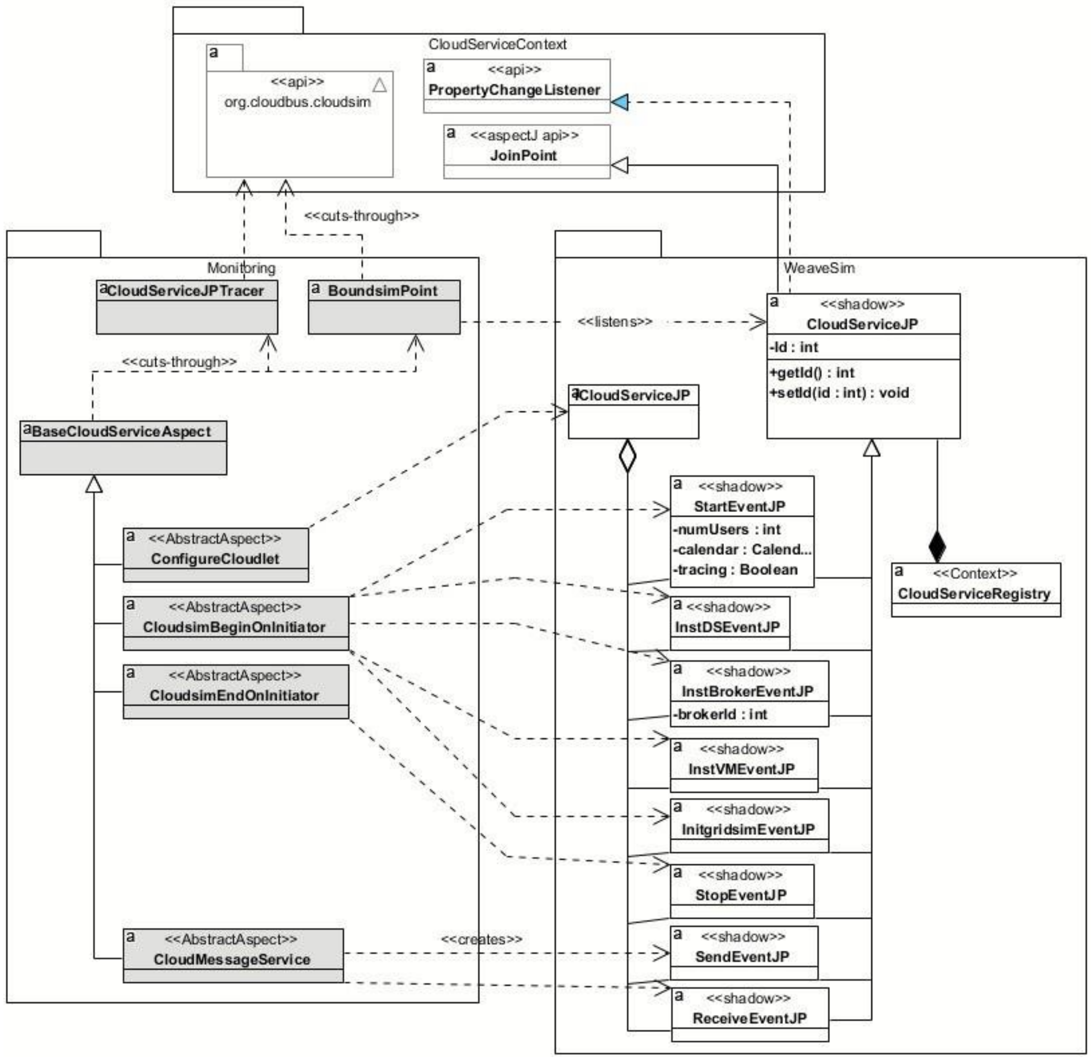


Figure 3. Structure of the cloud-based model of the dynamic weaving implementation within the CAAL layer.[1]

The BoundsimPoint is responsible for CloudSim's context which listens to parameter properties that hold a reference to an instance of CloudServiceJP, as shown in Figure 5.

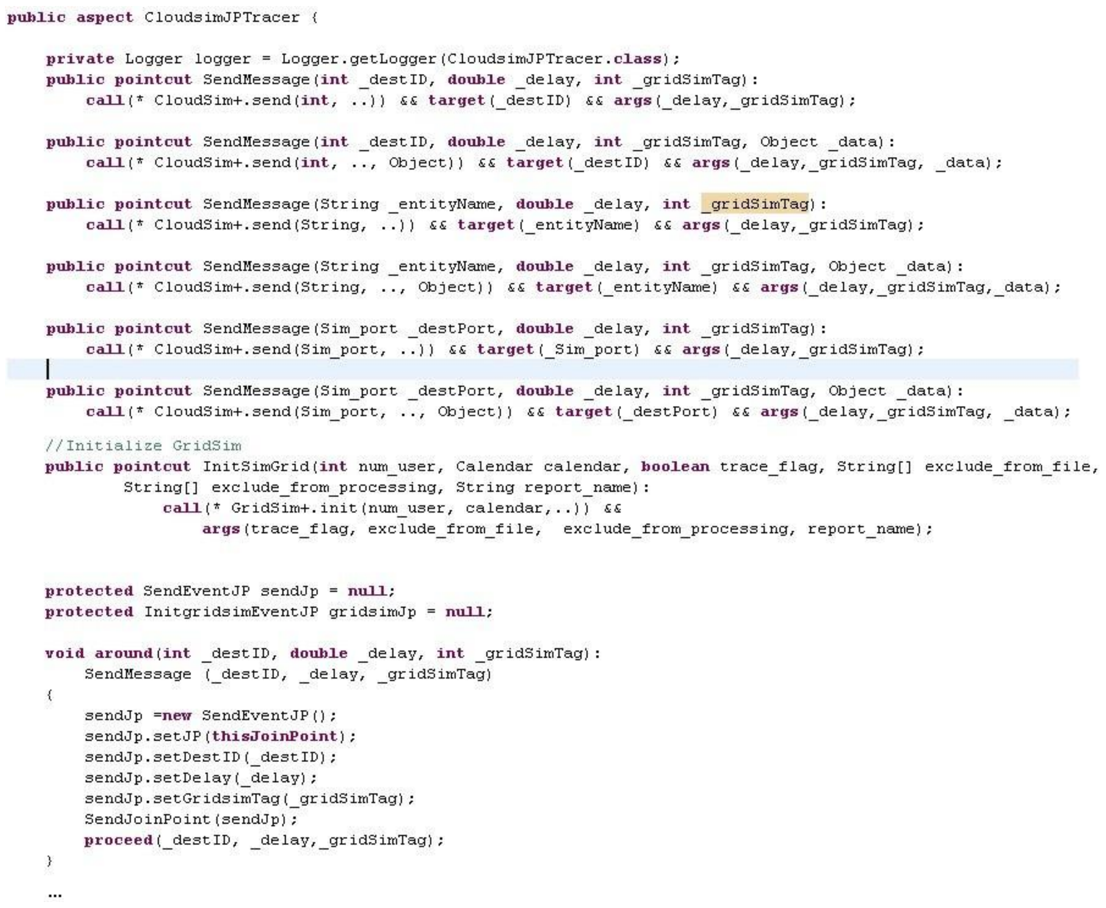
CloudServiceJP offers a direct mapping to the parameters of an advice for realizing joinpoint to add cross-cutting concerns related to a component-based cloud application. In addition, it supports the behavior of the aspects to work together, which can support the retrieval of the related context information and advice.

BaseCloudServiceAspect captures the auxiliary aspects which allow developers to handle interactions between different CloudSim components. It leads to the way in which different aspects cross-cut each other. All context parameters can be extended with additional properties to decide when and where crosscutting concerns are woven in order to perform the desired behavior.

The desired behavior is defined as a set of inherited subaspects, e.g., ConfigureCloudlet, CloudSimBeginOnInitiator, CloudSimEndOnInitiator and CloudMessageAspect. Such aspects can be woven before, after and during the calling and/or execution of joinpoints. In our approach, the binding of context parameters is represented in abstract pointcuts, which permits the reuse of pointcut signatures.

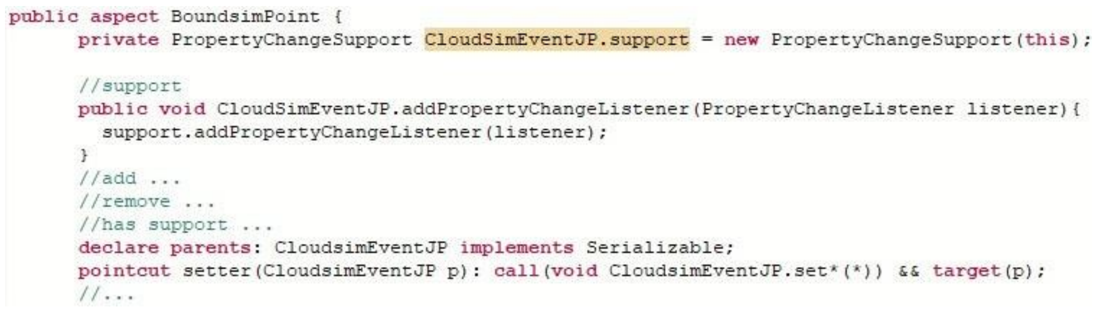
[1] Source code available on Github: https://github.com/aalsobeh .

We use explicit pointcut signatures which basically use generic constructs such as create, construct, initiate, start, stop, send, receive, …etc. The pointcuts pick out joinpoints and expose data from the execution context of joinpoints. The context provided by the selected joinpoints is validated regarding the general event handler class, the CloudServiceJP.



```
public aspect CloudsimJPTracer {

    private Logger logger = Logger.getLogger(CloudsimJPTracer.class);
    public pointcut SendMessage(int _destID, double _delay, int _gridSimTag):
        call(* CloudSim+.send(int, ..)) && target(_destID) && args(_delay,_gridSimTag);

    public pointcut SendMessage(int _destID, double _delay, int _gridSimTag, Object _data):
        call(* CloudSim+.send(int, .., Object)) && target(_destID) && args(_delay,_gridSimTag, _data);

    public pointcut SendMessage(String _entityName, double _delay, int _gridSimTag):
        call(* CloudSim+.send(String, ..)) && target(_entityName) && args(_delay,_gridSimTag);

    public pointcut SendMessage(String _entityName, double _delay, int _gridSimTag, Object _data):
        call(* CloudSim+.send(String, .., Object)) && target(_entityName) && args(_delay,_gridSimTag,_data);

    public pointcut SendMessage(Sim_port _destPort, double _delay, int _gridSimTag):
        call(* CloudSim+.send(Sim_port, ..)) && target(_Sim_port) && args(_delay,_gridSimTag);

    public pointcut SendMessage(Sim_port _destPort, double _delay, int _gridSimTag, Object _data):
        call(* CloudSim+.send(Sim_port, .., Object)) && target(_destPort) && args(_delay,_gridSimTag, _data);

    //Initialize GridSim
    public pointcut InitSimGrid(int num_user, Calendar calendar, boolean trace_flag, String[] exclude_from_file,
            String[] exclude_from_processing, String report_name):
                call(* GridSim+.init(num_user, calendar,..)) &&
                    args(trace_flag, exclude_from_file,  exclude_from_processing, report_name);


    protected SendEventJP sendJp = null;
    protected InitgridsimEventJP gridsimJp = null;

    void around(int _destID, double _delay, int _gridSimTag):
        SendMessage (_destID, _delay, _gridSimTag)
    {
        sendJp =new SendEventJP();
        sendJp.setJP(thisJoinPoint);
        sendJp.setDestID(_destID);
        sendJp.setDelay(_delay);
        sendJp.setGridsimTag(_gridSimTag);
        SendJoinPoint(sendJp);
        proceed(_destID, _delay,_gridSimTag);
    }
    …
```

Figure 4. A snippet of abstract CloudSimJPTracer module.



```
public aspect BoundsimPoint {
      private PropertyChangeSupport CloudSimEventJP.support = new PropertyChangeSupport(this);

      //support
      public void CloudSimEventJP.addPropertyChangeListener(PropertyChangeListener listener){
        support.addPropertyChangeListener(listener);
      }
      //add ...
      //remove ...
      //has support ...
      declare parents: CloudsimEventJP implements Serializable;
      pointcut setter(CloudsimEventJP p): call(void CloudsimEventJP.set*(*)) && target(p);
      //...
```

Figure 5. A snippet of BoundsimPoint code.

Finally, the CloudServiceJP allows for every CloudSim action to be validated according to the expected context parameters provided by advises and pointcuts explicitly. Indeed, events have to be extended to support new joinpoints without changing the validity of the CloudSim functionalities, as shown in Figure 3.

3.2 Joinpoint Shadows

Sub-aspects are structured for each CloudSim component to hold additional information for discharging and to be woven automatically. Depending on the level of a subaspect abstraction, the mapping refers to the execution of region range from simple joinpoints to complicated joinpoints. At any given point of time, only one abstract joinpoint can execute a region of a particular cross-cutting concern. This joinpoint contains a complete set of parameters for executing a cloud-related concern, where the advice associated with the aspect may be executed.

Shadows are places on the source code implemented as event handler classes to be responsible for the handling of joinpoint execution. The joinpoint context model is realized by defining a set of shadow points (reflective events) to which an advice can be bound (e.g., StartEventJP, InstDSEventJP, InstBrokerEvenJP, InstVMEventJP, InitgridsimEventJP, StopEventJP, SendEventJP and RecieveEventJP), as shown in Figure 3. In addition, with expanding the CloudServiceJP, the shadow classes provide utility functions related to encapsulate code constructs for each joinpoint simulator.

StartEventJP, InstDSEventJP, InstBrokerEvenJP, InstVMEventJP and InitgridsimEventJP encap- sulate the creation of the introduction region through initiating and starting of a simulation process. These classes added methods and properties to an advised cloud component to simplify tracking a target object at instantiating. StopEventJP encapsulates the creation of the region that spans after starting and before stopping a simulation. CloudletJP implements the creation of the entire region of a complete simulation service or task that spans before instantiating until after stopping. SendEventJP encapsulates the creation of the region that spans through establishing a message request containing a remote event to the end-user. The ReceiveEventJP encapsulates the creation of the region that spans through establishing a message response from an end-user. These shadows implement the entire region of the message-passing task that spans from before sending a request to after receiving a response.

In WeaveSim, each event class supports the shadow of those joinpoints and involves references to the CloudSim aspects' bindings to at least one pointcut that matches at a given event. In a nutshell, these references are used to pull the context information needed for weaving processes either before, after or around the joinpoint shadow.

4. Experiment: Simulation and Evaluation

Despite the successful practice in cloud-based systems, solutions are still unable to deal with the dynamic context changes. It is therefore necessary to provide context-aware innovative cloud services in which AOP-based services operate as a dynamic simulation service. To evaluate WeaveSim on real-world cloud apps, we have implemented the encryption service-oriented cross-cutting concern as a simulation service [26]-[27]. Figure 4 shows the quality of implementing cloud-related cross-cutting concern that begins from tracking states to the VM, broker and data center to ensure that the process of optimization becomes service-aware and well implemented. Adaptive WeaveSim based on the AOP increased the quality of the models regarding size, coupling, cohesion, obliviousness, complexity, response time separation of concerns and ease of change. Compared to Filho et al. [6], lots of issues in CloudSim are presented regarding these quality measurement features. One of the major issues in CloudSim and CloudSim plus is that some of these principles are not followed, such as correct inheritance and composition. Lots of subclasses in CloudSim just duplicate codes from the base class, what goes against inheritance. That increases code duplication, leading to software erosion and degrading maintainability. CloudSim 4.0 increased code duplication by 300%. That directly impacts all these quality measurement features.

WeaveSim ignores and improves some of these measurement features without adding additional code issues related to CloudSim and simply creates a separate layer over that original framework obliviously, as discussed in Section 5.

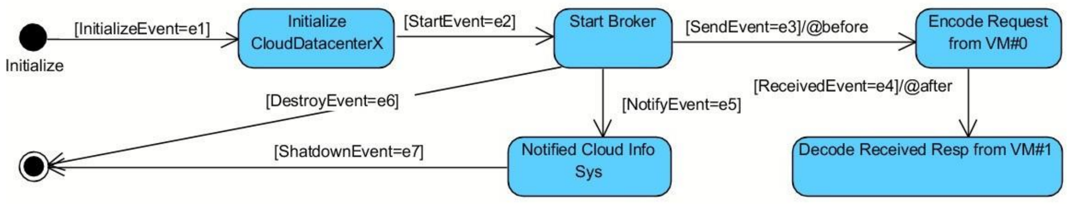


Figure 6. Example: the state machine causes weaving encryption solution.

Eventually, the simulation service using AOP-based service provides an integrated process simulation environment to optimize context-aware services for reduced time and resource consumption and increased quality and productivity. Hence, the extensibility, design stability, configurability, time behavior, resource behavior, code reducibility, understandability, complexity, learnability, reusability, changeability, modularity and scalability of platform-independent cross-cutting concerns could be improved in cloud application scenarios. In section 5, we examine such quality factors.

4.1 Cross-cutting Concern Case Study

In the proposed model, the central encryption cross-cutting concern is utilized to handle the communication security acting as a trusted application. Here, the encryption is significant to stay away from the security attacks amid the season of data modification. The access control policies are effectively taken care of in the proposed encryption feature. A real-world cloud-based weather system was used to conduct our experiments. This system indicates weather patterns and changes. The cloud application context data and encryption parameters are set up to the top application service level. The encryption parameters and collaboration services overhead increased the complexity of simulation by encryption and decryption message in real time. Therefore, the encryption process slows down the cloud's process significantly. Unfortunately, encryption is not well-structured into CloudSim components as a separate module; rather, it is intertwined into many CloudSim's components. This is the result of the interlocking process resulting from the implementation of such a concern. So, there is a need for a mechanism that allows developers to add such concern in the simulation process away from complexity and tangling caused by CloudSim.

Implementing such an aspect allows developers to evaluate the efficiency of various encryption algorithms. To show the benefit of WeaveSim, we have implemented the encryption service as a separate security aspect and compared that with a conventional implementation of the encryption service using CloudSim.

In our experiment, we expect the communication messages to be secured under attribute secret keys. The WeaveSim simulates encryption by injecting the secret keys on VMs at data centers. When a simulation process in WeaveSim starts, a Broker in the host requests, through a VM, a key to encrypt the Cloudlet message. The host is allocated in the datacenter and the VM encrypts the Cloudlet message data, creates a key, encapsulates the key in the Cloudlet response message and sends it back to the data center, as shown in Figure 6.

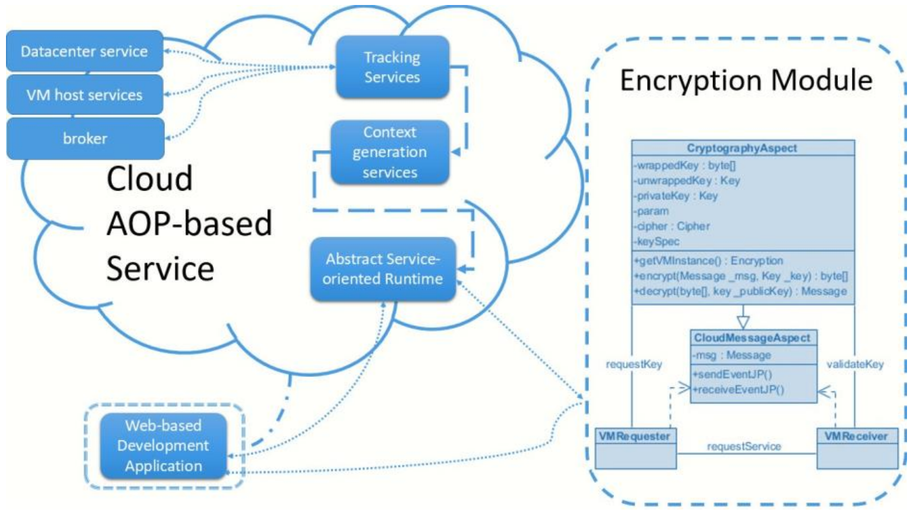


Figure 7. AOP-based model for the encryption service in the cloud weaving solution.

To exchange a secret key, Figure 6 shows a finite state machine for the encryption process. The behaviors of VM#0 and VM#1 are considerably simplified. The VM instance authenticates the requester instance, creates a key, encapsulates it in the response message and then sends it to the requester. The receiver instance also creates a new key, which sends the key request to the VM instance. The VM instance again authenticates the receiver instance, creates a key, encapsulates it in key Cloudlet message and sends it to the receiver instance. WeaveSim starts a VM process on Datacenter, which addresses the key requests from the requester instance before invoking the send method and after receiving a response from the receiver instance. In this approach, to ensure the integrity of the message data, we hash cloud message using the public key to obtain a new version with mapping all bits of the message data and offer an error-detection capability. Thus, encoding and decoding involve a combination of message data, a hash key and an encryption-decryption key.

To implement the encryption aspect in a way that would prevent the cloud's client from grabbing sensitive information, we extended CloudMessageAspect with CryptographyAspect.Cryptography-Aspect provides asymmetric key encryption constraints throughout the execution of the message; i.e., the send and/or the receive primitives. Figure 7 demonstrates the process of exchanging public-private key cryptography message codes, which apply to encrypt and decrypt advices at runtime, whereas AspectJ applies at sendEventJP and receiveEventJP, respectively.

5. EVALUATION

We selected measurement features relative to the AOP and OO approaches, which effectively set up a strawman comparison of improvements on the basis of a selected set of measurements that are tuned to the software engineering approach. The measurements take a software code base, in CloudSim and WeaveSim and then overlays an ease-of-use methodology that an easier to evaluate system results.

We conducted a comparative experiment of the encryption cross-cutting concern implementation on both CloudSim and WeaveSim environments. Seven times of simulation runs were performed to measure such results. This is accomplished by comparing the result produced by the simulation weather application with data measured on the CloudSim and WeaveSim. The WeaveSim model has been desired with some specified degree of certainty (e.g., 95%) context parameters, as discussed below within acceptable confidence intervals for each hypothesis.

The evaluation process focuses on the quality principles of cloud applications. We mainly measure on these quality measurement features: (1) evolvability and functionality (Section 5.1) (2) usability and maintainability (Section 5.2) and (3) efficiency (Section 5.3). These quality measurement features are the most relevant measurement features for measuring the quality of cloud-based models [28]. These models are extended with further concepts, extra features and effective software quality measurements. To relate the results with the AOP concepts, we related the calculated results of each quality measurement feature with AOP concepts, such as aspect, pointcut, joinpoint and advice.

The specific set of concrete quality metrics to measure such attributes represents the number of AOP features, concerns and modules for cloud-related cross-cutting concerns. These metrics were adapted to measure the simulator application to be closer for the real-world implemented application. The simulation evaluation is completely isolated from the internal structure of the simulation itself. Our measurements measure the internal structure of the implemented applications. It's possible to implement a cloud application using a framework designed to run applications in a real cloud environment. These metrics will produce the same results in a simulated and actual cloud environment, because we measure high-level abstraction of applications for test results in the cloud.

5.1 Evolvability and Functionality Qualities

Evolvability is the ability of a cloud application to easily evolve in order to continue serving its users in a cost-effective manner. Since evolvability is a cross-cutting quality measurement feature of the system, it can also be considered as a non-functional requirement of the system. Thus, we treat evolvability as a functionality quality of the system [29]. To quantify these qualities, we considered a set of factors that cause the application to evolve. The identified factors are: extensibility, design stability, configurability, scalability and changeability. These factors can be measured by measuring how a system meets certain cross-cutting concerns' quality. Alongside, some of the related methods and advices might be adapted to measure the number of components (i.e., class or aspect) executed in response to a message received by a given feature (e.g., method and advice); these are triggered whenever a pointcut is matched. We define an application program as follows:

$$P = M_1(F_1, F_2, \dots, F_n), \dots, M_j(F_1, F_2, \dots, F_n) \quad (1)$$

where, P is a cloud app, Fi is a list of feature elements in a module Mj. M includes classes C, interfaces I, aspects A and cross-cutting concerns cc. F includes attribute/field/inter-type declaration field, advice adv, class shadow, method m, joinpoint and pointcut. These factors are measured by a set of respective measurement features.

- Degree Diffusion Pointcuts (DDPs): corresponding to the number of features defined in an aspect module. DDP measures the outgoing coupling connections (i.e., fanout) which helps the refactoring process and affects extensibility and scalability [30].
- In-Different Concerns (INCs): corresponding to the number of different concerns to which a module element is participating. INC is an AOP measurement feature for creating complex aspect code through extracting context information which is heavily dependent on the underlying code. This measurement feature affects the design stability and changeability factors [31].
- Feature Cross-cutting Degree (FCD): it corresponds to the number of modules that are crosscut by all elements of an advice in a concern and that are cross-cut by the inter-type declarations. It affects the application design stability and its configurability [32].

$$FCD = count\big(C \rightarrow m,\ C \rightarrow constructors,\ C \rightarrow field,\ C \rightarrow shadows\big(pointcuts\big(adv_{A(F_i)}\big)\big)\big) \quad (2)$$

Advice Crosscutting Degree (ACD): it corresponds to the number of classes that are exclusively crosscut by the method of advice in a concern. It affects the application design and stability [33]-[34].

$$\text{ACD} = \text{count}(\text{Mj}(\text{shadows}(\text{pointcuts}(\text{advA}(\text{Fi}))))) \quad (3)$$

where A(Fi) represents the indices of aspect functions having been injected at the execution or call time.

- Program Homogeneity Quotient (PHQ): it corresponds to the summation of the homogeneity quotients of all features in a cloud app, divided by the number of features (FCD). It affects the application design stability and its configurability [35]-[36].

$$PHQ = \frac{\sum((\forall P, g.HQ(g,P)))}{FCD} \quad (4)$$

where,

$$HQ(Fi, P) = \frac{Count(ACD)}{FCD}$$

λ: is a computation based on aspect abstraction and application using AOP variable binding and execution.

- Class and Aspect Complexity due Number of Children (CACNoC): it corresponds to the number of inherited methods and advices of sub-classes and sub-aspects, respectively, of the parent class or aspect. It gives an idea regarding the effect of class and aspect on the overall design and implementation. The CACNoC value indicates the extensibility and the design stability, since inheritance is a form of code reuse [33].

$$\text{CACNoC} = \sum_{j=0}^{n} \text{Mj} + \sum_{i=0}^{m} \text{Ai} \quad (5)$$

where, M_j and A_i are methods of class and aspect, respectively, at level *i,j*.

- Number of Methods (NoM) or Number of advices (NOA): it corresponds to the number of method and advice signatures in both classes and aspects in a cloud app, respectively. The complexity of (NoM) and (NoA) is obtained by counting the number of parameters in each operation and advice, assuming that an operation or advice with more parameters is a more complex than one with less parameters. They affect the code changeability level [37].
- Percentage of Advised Modules (ADM): it corresponds to the percentage of class and aspect modules that are interwoven, where a joinpoint shadow might be determined among all modules in the simulated cloud apps. It includes method-execution, method-call, constructor-call, constructor- execution, field-get and field-set pointcuts. It measures the extensibility and the

scalability quality factors [38].

- Coupling on Advice Execution (CAE): it corresponds to the number of modules declaring fields that are accessed by a given module. It provides an overall estimate regarding the effects of aspects in other modules (classes or other aspects), in terms of how many modules an aspect affects and how many aspects affect a given module. It affects the capacity to extend and change a cloud app's components.
- Lack of Cohesion (LoC): it corresponds to the number of different methods and advices within a class and an aspect that refer to an invoked particular joinpoint. To reduce the possibility of errors during the development process, high cohesion value decreases complexity. LoC affects on the code design stability [39].
- Obliviousness (Obl): it corresponds to the number of inter-type declarations (ITDs) in the aspects and the number of times they are being used, which also includes the number of modules affected by pointcuts in a given aspect DDP. Moreover, it takes into account scattered aspects over cloud components INC. Obl indicates the tangling of aspects in a cloud app's components. It makes a simulated less reusable and less scalable cloud app.

$$\mathrm{Obl} = \mathrm{count}(\mathrm{IT\ D}) + \mathrm{count}(\mathrm{DDP}\ ) + \mathrm{count}(\mathrm{IN\ C}) \quad (6)$$

These aforementioned measurement features measure how much time and effort are required to evolve and maintain the functionality of a cloud app. The greater the value of each quality measurement feature, the more complex the program would be to evolve; i.e., the lower the better [16]. Figure 8 shows the values of these quality measurement features obtained to measure finer-grained constructs of cloud apps with different configurations on both CloudSim and WeaveSim.

The DDP, INC, FCD, ACD and PHQ measurement features measure the application tangling and scattering in AOP components [40]. Figure 8 clearly demonstrates that the number of changes required to evolve the functions of a cloud app is significantly reduced using WeaveSim in comparison to CloudSim. The results show that WeaveSim offers better encapsulating cross-cutting concerns with less scattering compared to CloudSim.

On the other hand, CACNoC, NOA, ADM and Obl provide additional insights into method and advice-level tangling of cloud applications inside aspect components [41]. As shown in Figure 8, WeaveSim's values are less than CloudSim's values, which indicates that the functions of cloud apps have lower coupling with the cross-cutting concerns. However, the figure shows that, using CloudSim, the concerns are tightly coupled with the cloud app's functions, increasing the complexity of evolving and reusing the functionalities.

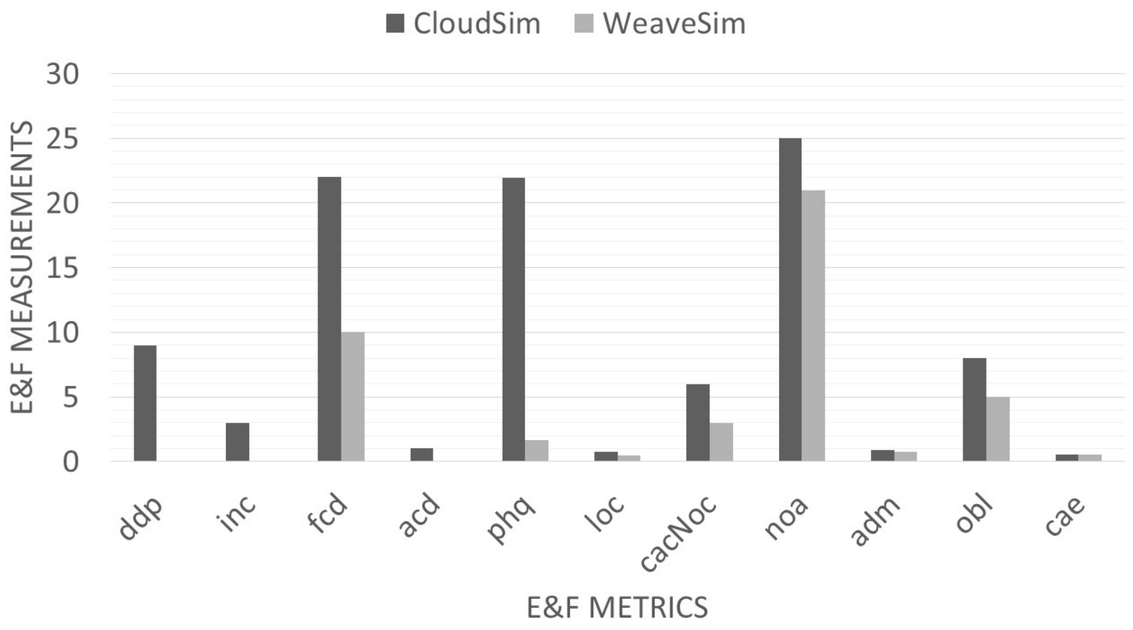


Figure 8. Evolvability and functionality measurement features for CloudSim and WeaveSim (the

lower the better).

CAE and LoC measurement features point out the complexity of cloud components in terms of advices, which should be tailored to the specific context information [30]. Figure 8 shows that CAE and LoC values for WeaveSim's application components have a higher coherence of a single cross-cutting concern logic compared to the tangled implementation in CloudSim.

Consequently, the presented result gives the confidence that in practice, the developer does not have a hard time trying to implement such concerns using WeaveSim compared to the extension built on top of CloudSim that was abandoned after a while. We can confidently conclude that WeaveSim provides a neater and cleaner set of pointcuts and joinpoints to cloud abstractions, which helps developers evolve the cross-cutting concerns with less complicated functionalities and implement some features on the top easily.

5.2 Usability and Maintainability Quality

Maintainability and usability measure the ability of a software product to be easily modified. Modifications may include corrections, improvements or adaptations of a software to adapt (change) to environments, requirements and functional specifications [42]. We measured these qualities with a set of factors, such as: code reducibility, understandability, complexity, learnability and reusability. These factors affect the SoC and ease of change, where SoC is an indication of concerns diffused in a cloud app and ease of change indicates the number of changes made to maintain a cross-cutting concern in the application [30].

- Line-of-Code (LoCC): it corresponds to the quantity of the executable source codes in terms of classes, interfaces and aspect elements in a correlative manner. The overall complexity of the simulated application will increase as the app's size grows. This affects the code complexity and understandability of the system [30]. LoCC can be computed as follows, where i is composed of several elements M1, ..., Mi of modules M.

$$\sum_{i=1}^{n} LoCC(Mi) \tag{7}$$

- Classes, Interfaces and Aspects (CIA): it corresponds to the number of occurrences (NOOs) of classes, interfaces and aspects, as well as LoCC associated with each other. It points out whether aspects (as opposed to classes and interfaces) are a small or a large fraction of the modularization mechanisms used in a cloud app, then the implementation of cross-cutting concern will be a significant part or only a small part of the base code of the simulated application. It affects the code complexity and understandablity [43].
- Weighted Advice in Aspect (WAA): it corresponds to the number of adv and M's methods in a given aspect that indicates different weights to various advises with internal complexity. In other words, it is an indicator of how much effort is required to develop and maintain a particular cross-cutting concern. A high value denotes that the aspect is more complex and therefore harder to reuse and maintain, which may affect the code complexity and reusability.
- Code Replication Reduction (CRR): it corresponds to the reduction in the amount of LoCC when using homogeneous advises and inter-type declarations, rather than the amount of LoCC resulting from the use of traditional object-oriented approach. CRR is roughly the number of affected joinpoints, multiplied by the LoCC associated with them [35]. CRR affects the code reducibility, complexity and reusability [35].
- Degree of Focus (DoF): it corresponds to the variances of the dedication of a component to every concern with respect to the worse case. The average degree of focus gives an overall picture of how well concerns are separated in the program [43]. It affects the code complexity and learnability.
- Inter-type Declaration (ITD): it corresponds to the number of injected new data members into the core code to add states or behaviors to a particular class. A small number indicates less coupled modules increasing the code maintainability. It also affects the code reusability [30].

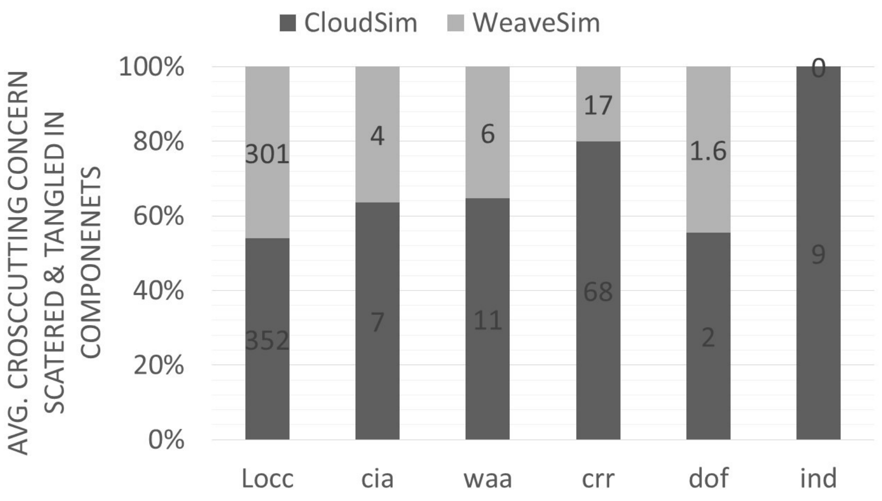


Figure 9. Usability and maintainability measurement features using CloudSim and WeaveSim (the higher the better).

An analytical evaluation for each of the aformentioned measurement features presents an overall sight of the size of the implemented cross-cutting concern in terms of modules. Figure 9 depicts the values of these quality measurement features for WeaveSim and CouldSim. LoCC and CIA are usually used as indicators of effort and productivity [44]. In the measurement of effort and productivity, Figure 9 clearly shows that the efforts for developing and maintaining a concern using WeaveSim are significantly less than those used to perform the same task with CloudSim. WeaveSim reduces the efforts, since it reduces the difficulty of redefining several core code behaviors in CloudSim. The flexibility of implementing such non-functional requirements in cloud apps is indicated by the high-level of SoC.

Figure 9 also shows that the measurements of WAA, CRR, DoF and ITD measurement features of the CloudSim app are higher than those of the WeaveSim app. The reason behind that is that CloudSim app contains tangled concern code with a few extension points which compromise ease of changes. In contrast, WeaveSim abstracts the aspects and provides context states that deal directly with cross-cutting concerns obliviously. Therefore, it is expected that the ability to modularize such concerns will improve the reusability and maintainability with a better SoC.

5.3 Efficiency Quality

The Response Time of Aspect (RFM) is a well-known factor that affects efficiency [45]. This measurement feature is used to measure the number of methods and advices executed in response to a message received by a given module triggered whenever a pointcut is matched. The results in terms of the change in response time for CloudSim and WeaveSim are shown in Figure 10.

To evaluate the performance of cloud apps, results were simulated on virtual instances configured with Window 10 (64-bit), 2.7, core i5 processor and 8GB RAM. The development environment was as follows: Java programming language (JDK 8), AspectJ to realize AOP concepts and Eclipse Oxygen IDE. As shown in Figure 10, we have conducted three rounds of execution where each round was run with different simulation parameters, as shown in Table 1. Table 1 shows the experimental design in terms of the involved userbases, user requests generated from each regional userbase and requests that are simultaneously processed by a virtual machine (VM).

As shown in Figure 10, CloudSim has a slightly higher efficient execution than WeaveSim, since CloudSim does not require any aspect. However, WeaveSim still provides an efficient weaving process for cross-cutting concerns. This Figure shows that WeaveSim extends the capabilities of CloudSim without degradation in efficiency. Based on our experimental results, WeaveSim provides reasonable efficient provisioning for large-scale cloud apps, allowing users to manage various types of VMs and provide the inputs needed for simulation without any knowledge of the core code of CloudSim.

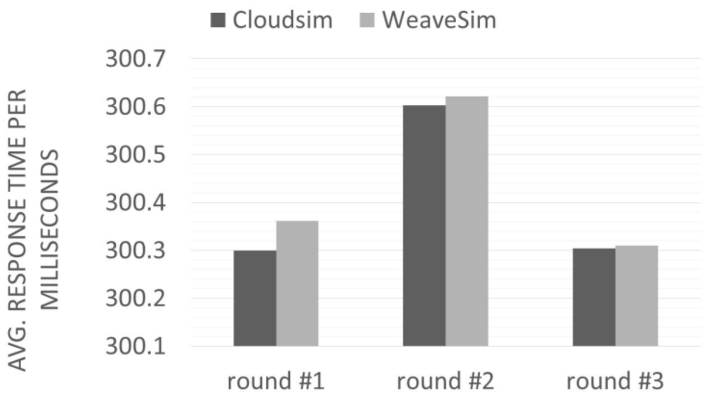


Figure 10. Overall average response time, in milliseconds, of encryption process using CloudSim and WeaveSim (the lower the better).

Table 1. The simulation experiment configuration.

Userbase	Region	TimeZone	PeakHours	Simultaneous
UB#1	0-N. America	GMT 6.00	08:00 - 10:00 pm	300,000
UB#2	Africa	GTM 4.00	10:00 - 14:00 pm	100,000
UB#3	Asia	GMT 2.00	10:00 - 16:00 pm	150,000

6. CONCLUSION

This paper delves into the factors that make cloud services largely scalable through bridging the gap between simulated cloud-based applications and real-time cloud applications. It introduces a novel cloud- based simulation framework called WeaveSim. WeaveSim extends CloudSim with AOP concepts to enable developers to simulate non-functional requirements for cloud computing applications easily. It extracts relevant contextual meta-data that can be applied to all cloud-related aspects. This framework effectively provides a high-level abstraction that encapsulates cross-cutting concerns in executable cloud structure in separate first-class aspects. It provides developers with a set of well-defined joinpoints and abstract pointcuts to pick the targeted cloud's modules. In the future, we will conduct more experiments with different types of cross-cutting concerns, such as monitoring, transaction, quality of service, service level agreements and synchronization. It is sought to target the entire space inclusive of industry-scale cloud- deployed solutions. In addition, we will upgrade the base code of the AspectJ's weaver which allows managing and handling the weaving process more efficiently.

REFERENCES

[1] T. Grance and P. Mell, "The NIST Definition of Cloud Computing," NIST Special Publication, pp. 800–145, 2011.

[2] H. T. Dinh et al., "A Survey of Mobile Cloud Computing: Architecture, Applications and Approaches," Wireless Communications and Mobile Computing, vol. 13, no. 18, pp. 1587–1611, 2013.

[3] R. Buyya, R. Ranjan and R. N. Calheiros, "Modeling and Simulation of Scalable Cloud Computing Environments and the CloudSim Toolkit: Challenges and Opportunities," Proc. of IEEE International Conference on High Performance Computing & Simulation (HPCS'09), pp. 1–11, 2009.

[4] R. N. Calheiros et al., "CloudSim: A Novel Framework for Modeling and Simulation of Cloud Computing Infrastructures and Services," arXiv preprint arXiv:0903.2525, 2009.

[5] H. Mei and X.-Z. Liu, "Internetware: An Emerging Software Paradigm for Internet Computing," Journal of Computer Science and Technology, vol. 26, no. 4, p. 588. DOI: 10.1007/s11390-011-1159-y, [Online], Available: http: //jcst.ict.ac.cn/EN/abstract/article_1768.shtml.

[6] M. C. Silva Filho et al., "CloudSim Plus: A Cloud Computing Simulation Framework Pursuing Software Engineering Principles for Improved Modularity, Extensibility and Correctness," IFIP/IEEE Symposium on Integrated Network and Service Management (IM), IEEE, pp. 400– 406, 2017.

[7] G. Kiczales et al. "Aspect-oriented Programming," Proc. of European Conference on Object-oriented Programming, Springer, pp. 220–242, 1997.

[8] R. S. Pressman, Software Engineering: A Practitioner's Approach, Palgrave Macmillan, 2005.

[9] S. K. Sood, "A Combined Approach to Ensure Data Security in Cloud Computing," Journal of Network and Computer Applications, vol.35, no. 6, pp. 1831–1838, 2012.

[10] R. Kumar and G. Sahoo, "Cloud Computing Simulation Using CloudSim," arXiv Preprint arXiv:1403.3253, 2014.

[11] P. Humane and J. N. Varshapriya, "Simulation of Cloud Infrastructure Using CloudSim Simulator: A Practical Approach for Researchers," Proc. of IEEE International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM), pp. 207–211, 2015.

[12] R. Buyya and M. Murshed, "GridSim: A Toolkit for the Modeling and Simulation of Distributed Resource Management and Scheduling for Grid Computing," Concurrency and Computation: Practice and Experience, vol. 14, no. 13-15, pp. 1175–1220, 2002.

[13] R. Lakshminarayanan and R. Ramalingam. "Usage of Cloud Computing Simulators and Future Systems for Computational Research," arXiv preprint arXiv:1605.00085, 2016.

[14] F. Fakhfakh, H. Hadj Kacem and A. Hadj Kacem, "Simulation Tools for Cloud Computing: A Survey and Comparative Study," Proc. of the 16th IEEE International Conference on Computer and Information Science (ICIS), pp. 221–226, 2017.

[15] P. Kathiravelu and L. Veiga, "Concurrent and Distributed CloudSim Simulations," Proc. of the 22nd IEEE International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS), pp. 490–493, 2014.

[16] R. N Calheiros et al., "CloudSim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms," Software: Practice and Experience, vol. 41, no. 1, pp. 23–50, 2011.

[17] J. Byrne et al., "A Review of Cloud Computing Simulation Platforms and Related Environments," CLOSER, pp. 651–663, 2017.

[18] B. Wickremasinghe and R. Buyya, "CloudAnalyst: A CloudSim-based Tool for Modelling and Analysis of Large Scale Cloud Computing Environments," MEDC Project Report, vol. 22, no. 6, pp. 433–659, 2009.

[19] F. Fakhfakh, H. Hadj Kacem and A. Hadj Kacem, "CloudSim4DWf: A CloudSim Extension for Simulating Dynamic Workflows in a Cloud Environment," Proc. of the 15th IEEE International Conference on Software Engineering Research, Management and Applications (SERA), pp. 195–202, 2017.

[20] M. Bux and U. Leser, "Dynamic CloudSim: Simulating Heterogeneity in Computational Clouds," Future Generation Computer Systems, vol. 46, pp. 85–99, 2015.

[21] D. Kliazovich, P. Bouvry and S. U. Khan, "GreenCloud: A Packet-level Simulator of Energy-aware Cloud Computing Data Centers," The Journal of Supercomputing, vol. 62, no. 3, pp. 1263-1283, 2012.

[22] A. W.Malik, K. Bilal, K. Aziz, D. Kliazovich, N. Ghani, S. U. Khan and R. Buyya, "Cloudnetsim++: A Toolkit for Data Center Simulations in Omnet++," Proc. of the 11th IEEE Annual High Capacity Optical Networks and Emerging/Enabling Technologies (Photonics for Energy), pp. 104-108, 2014.

[23] J. Byrne, S. Svorobej, K. M. Giannoutakis, D. Tzovaras, P. J. Byrne, P. O. Östberg and T. Lynn, "A Review of Cloud Computing Simulation Platforms and Related Environments," CLOSER, pp. 651-663, April 2017.

[24] B. Fateh et al., "Secure Inverted Index Based Search over Encrypted Cloud Data with User Access Rights Management," Journal of Computer Science and Technology, vol. 34, no. 1, p. 133. DOI: 10.1007/s11390-019-1903-2,[Online],Available: http://jcst.ict.ac.cn/EN/abstract/article_2500, 2019.

[25] A. M. R. AlSobeh, R. Hammad and A.-K. Al-Tamimi, "A Modular Cloud-based Ontology Framework for Context-aware EHR Services," International Journal of Computer Applications in Technology, vol. 60, no .4, pp. 339–350, 2019.

[26] A. M. R. Alsobeh, A. A.-R. Magableh and E. M. AlSukhni, "Runtime Reusable Weaving Model for Cloud Services Using Aspect-Oriented Programming: The Security-related Aspect," International Journal of

Web Services Research (IJWSR), vol. 15, no. 1, pp. 71–88, 2018.

[27] A. A.-R. Magableh and A. M. R. AlSobeh, "Securing Software Development Stages Using Aspect-Orientation Concepts," International Journal of Software Engineering & Applications (IJSEA), vol. 9, no. 6, 2018.

[28] E. Kessler Piveta et al., "An Empirical Study of Aspect-oriented Metrics," Science of Computer Programming, vol. 78, no. 1, pp. 117–144, 2012.

[29] S. Ciraci and P. van den Broek, "Evolvability as a Quality Attribute of Software Architectures," The International ERCIM Workshop on Software Evolution, pp. 6–7, 2006.

[30] A. M. R. AlSobeh and S. W. Clyde, "Transaction-aware Aspects with TransJ: An Initial Empirical Study to Demonstrate Improvement in Reusability," ICSEA, p. 59, 2016.

[31] H. Ossher and P. Tarr. "Multi-dimensional Separation of Concerns and the Hyperspace Approach," Software Architectures and Component Technology, Springer, pp. 293–323, 2002.

[32] E. Figueiredo et al., "On the Maintainability of Aspect-oriented Software: A Concern-oriented Measurement Framework," Proc. of the 12th IEEE European Conference on Software Maintenance and Reengineering (CSMR), pp. 183–192, 2008.

[33] R. E. Lopez-Herrejon and S. Apel, "Measuring and Characterizing Cross-cutting in Aspect-based Programs: Basic Metrics and Case Studies," Proc. of International Conference on Fundamental Approaches to Software Engineering, Springer, pp. 423–437, 2007.

[34] E. Alemneh, "Current States of Aspect Oriented Programming Metrics," International Journal of Science and Research, vol. 3, no. 1, pp. 142–146, 2014.

[35] S. Apel, D. Batory and M. Rosenmüller, "On the Structure of Cross-cutting Concerns: Using Aspects or Collaborations," Proc. of GPCE Workshop on Aspect-Oriented Product Line Engineering (AOPLE), 2006.

[36] S. Garg, K. S. Kahlon and P. K. Bansal, "Testability Analysis of Aspect Oriented Software," International Journal of Computer Theory and Engineering, vol. 2, no. 1, pp. 119–124, 2010.

[37] G. Kiczales et al., "Getting Started with AspectJ," Communications of the ACM, vol. 44, no. 10, pp. 59–65, 2001.

[38] R. Akiladevi, "Aspect Oriented Refactoring for Software Maintenance," International Journal of Emerging Trends & Technology in Computer Science (IJETTCS), vol. 2, pp. 79–84, 2013.

[39] L. Badri, M. Badri and F. Toure, "An Empirical Analysis of Lack of Cohesion Metrics for Predicting Testability of Classes," International Journal of Software Engineering and Its Applications, vol. 5, no. 2, pp. 69–85, 2011.

[40] A. Przybyłek, "An Empirical Study on the Impact of AspectJ on Software Evolvability," Empirical Software Engineering, vol. 23, no. 4, 2018.

[41] C. Driver and S. Clarke, "Distributed Systems Development: Can We Enhance Evolution by Using AspectJ?," Proc. of International Conference on Object-oriented Information Systems, Springer, pp. 368–382, 2003.

[42] I. Fleming, "Defining Software Quality Characteristics to Facilitate Software Quality Control and Software Process Improvement," Software Quality Assurance, Elsevier, pp. 47–61, 2016.

[43] F. E. Ritter, G. D. Baxter and E. F. Churchill, "User-centered Systems Design: A Brief History," Foundations for Designing User-centered Systems, Springer, pp. 33–54, 2014.

[44] K. Z. Ne Win, "Measuring and Evaluating Sustainability and Design Stability Software Qualities for Long-living Aspect-oriented Applications," Proc. of the 10th International Conference on ASEAN Community Knowledge Networks for the Economy, Society, Culture and Environmental Stability, Republic of the Union of Myanmar, pp. 8–12, 2014.

[45] M. Schalk, Response Time Measurement System and Method, US Patent 8,990,779, Mar. 2015.

ملخص البحث:

تُعدّ أُطر العمل الخاصة بالمحاكاة الموجّهة نحو الخدمات (service-oriented) في الحوسبة السحابية أدواتٍ مهمّةً جداً لنمذجة السلوك الديناميكي لأنظمة البرمجيات القائمة على الحوسبة السحابية ومحاكاته. ومع ذلك، فإن أُطر المحاكاة القائمة الموجّهة نحو الخدمات تحوزها القدرة على قياس المتطلبات سريعة التغير والتحكم بها؛ تلك المتطلبات التي تمتد لتشمل العديد من أنظمة البرمجيات القائمة على الحوسبة السحابية، مثل: الأمان، والتسجيل، والرّصد، ...الخ. ولمعالجة هذه المحدِّدات، تقدم هذه الورقة إطاراً فعالاً للحدّ من تعقيد النمذجة والمحاكاة فيما يتعلق بالسلوك الديناميكي للتطبيقات القائمة على الحوسبة السحابية. ويعرف الإطار المقترح بإسم (WeaveSim)، وهو يستفيد من البرمجة الموجّهة نحو المظاهر (AOP) للحدّ من التعقيد الذي يرتبط بتطوير السلوك الديناميكي للتطبيقات المرتكزة على السّحابة، وذلك عبر إضافة طبقة مجردّة أخرى تسمى طبقة المظاهر الواعية للسياق (CAAL).

وتعمل الطبقة المضافة (CAAL) على التقليل من التعقيد المرتبط باستخدام إطار المحاكاة المعروف باسم (CloudSim) عند محاكاة أنظمة البرمجيات القائمة على الحوسبة السحابية. ومن الأمثلة على قضايا التعارض التي تتم معالجتها هنا: سرّية البيانات، وتسجيلها، ورصدها. ونظراً لأن تنفيذ مسألة من مسائل التعارض باستخدام نظام محاكاة قائم على السحابة (مثل CloudSim) يتطلب تعديلاتٍ من المطوِّرين على وحدات أساسية من ذلك النظام، فإن استخدام الإطار المقترح (WeaveSim) يسهّل مثل هذه المهمّة على المطوِّرين الذين لا يحتاجون في هذه الحالة سوى إلا إعادة استخدام نقط الوصل والقطع المعرّفة مسبقاً دون الحاجة الى إجراء تعديلاتٍ على الوحدات الأساسية المكوِّنة للمحاكي.

لقد تم تقييم الإطار المقترح (WeaveSim) للتحقق من أنه يحدّ من التعقيد المرتبط بتنفيذ مسائل التعارض، وجرى ذلك التقييم على نظام موزون أكاديمياً؛ إذ أثبتت النتائج فعالية الإطار المقترح في خفض التعقيد. وبذلك تم تحسين أنظمة البرمجيات القائمة على الحوسبة السحابية من حيث قابلية إعادة الاستخدام، وقابلية رفع الدرجة، وقابلية الاستمرار.