Received: 30 August 2017 | Revised: 20 October 2018 | Accepted: 26 November 2018

DOI: 10.1002/spe.2674

RESEARCH ARTICLE

WILEY

# Generic input template for cloud simulators: A case study of CloudSim

Manar Jammal[1] | Hassan Hawilo[1] | Ali Kanso[2] | Abdallah Shami[1]

[1]Department of Electrical and Computer Engineering, Western University, London, Ontario, Canada

[2]Thomas J. Watson Research Center, IBM Research, Yorktown Heights, New York

**Correspondence**
Manar Jammal, Department of Electrical and Computer Engineering, Western University, London, ON N6A 3K7, Canada.
Email: mjammal@uwo.ca

**Funding information**
Natural Sciences and Engineering Research Council of Canada, Grant/Award Number: NSERC-STPGP 447230; Ericsson Research

**Summary**

Cloud computing and its service models, such as Platform as a Service (PaaS), have changed the way that computing resources are allocated to Information and Communications Technology enterprises and users. Although multiple cloud providers support dynamic service provisioning, it is necessary to facilitate the management of the cloud infrastructure and applications in order to allow the continuous refinement of cloud models. Therefore, issues are raised regarding the cloud orchestration, including the flexible portability and interoperability of cloud applications among multiple cloud providers. Having said that, there is a need for a standardized design and management of the cloud use cases (during the creation of scenarios, application's deployment, and patching) to ensure efficient applications' migration between different providers. This paper proposes an artifact, GITS, a generic input template for CloudSim and other cloud simulators. GITS can be provided by PaaS offering to manage the creation, monitoring, administration, and patching of infrastructure and applications in the cloud. GITS defines the cloud schema that can be used with conforming cloud models and independent cloud providers; thus, portability and interoperability can be enabled in PaaS cloud models. GITS focuses on the architecture-based modeling for cloud infrastructure and application not only in terms of computational resources but also in terms of high availability properties associated with infrastructure and applications. The main objective of the GITS template is to provide the cloud user with a modular, simple, readable, and reusable model that still supports the essential components and provide them with the ability to control the applications' execution, deployment, and other management needs in addition to the allocation environment. This paper describes GITS usage, specifically as an input template for CloudSim.

**KEYWORDS**

CloudSim, cloud applications, cloud management, cloud simulators, component-based architecture, domain-specific language, failures, GMF, Heat, high availability, JSON, OpenStack, orchestration, recovery, redundancy, software components, XML

## 1 | INTRODUCTION

Nowadays, many enterprises are moving to the cloud to benefit from its applications' availability, elasticity, pay-per-use basis, multitenancy, and resource provisioning. By leveraging the lightweight virtualization, the cloud infrastructure provides virtual machines (VMs) and containers to run multiple applications in private, public, or hybrid cloud environments.[1,2] To leverage these benefits to the users, the cloud resources require well-defined orchestration (resource provisioning, dynamic configurations, interoperability, and portability) within the application domains. This orchestration can then facilitate the management of different application architectures/topologies (VMs, application's components, and their dependencies) on various cloud models (public, private, and hybrid).[3] Having said that, cloud orchestration gives the cloud broker the ability to manage and control underlying physical resources and map them to the cloud applications to efficiently process the users' requests.[4] Thus, it ensures a dynamic deployment of the cloud applications while satisfying the quality of service (QoS) requirements. However, there are different challenges that face cloud orchestration. This is due to the limited capacity of resources (CPU, memory, and network), sudden failures, overload, geographical constraints of the users and providers, and cloud providers' heterogeneity.[5-7] The orchestration of cloud applications generally includes both resource allocation and coordination. Therefore, an efficient application's orchestration approach consists of different building blocks[4]; this includes the need for a scheduling algorithm to map the cloud applications to their hosts while satisfying the performance and other QoS constraints. It also requires a well-defined description of the cloud infrastructure (data centers [DCs], its servers, their geographical locations, and computation resources and other performance-aware granularities), the application's structure (internal and external interactions), and their requirements/configuration settings (such as availability and performance needs).[8-10] Since the main role of the orchestrator is to configure, deploy, and manage cloud applications, it is necessary to support cloud providers' heterogeneity and different provisioning algorithms and metrics. In other words, the dependability on different cloud offerings, including Application as a Service, requires a generic orchestration approach that can be easily customized based on the given cloud properties.[11] Therefore, the orchestration should define the resources of various cloud models in a uniform, standardized, and platform-independent format to facilitate the applications' portability.[12] Since availability is a critical challenge in large distributed systems, mainly cloud, it is necessary to take it into consideration when building a cloud orchestration solution.[13-16] At this point, the paper focuses on one of the cloud orchestration's building blocks, which is the cloud infrastructure description and composition of applications. It defines a generic and unified description of the cloud infrastructure and application while satisfying the high availability (HA) requirements.

When it comes to evaluating different cloud orchestration approaches and ensuring that cloud services meet the QoS requirements, it is important to build a cloud model and simulation that can emulate real cloud behaviors (outages and execute repairing and recovery policies). Using the existing cloud platforms (ie, such as Amazon Elastic Compute Cloud) to model cloud services and execute different orchestration and performance-aware deployment policies is limited by their infrastructure and configuration setting. Therefore, templating and simulations can be the alternative to emulate the cloud behaviors. Discrete event simulators (DESs) can be used in the modeling and assessment of the distributed systems.[17] Multiple cloud-based DESs allow for the modeling and simulation of cloud infrastructure and different scheduling policies, but they discard the design of a generic cloud template that models different scenarios while considering HA granularities. Therefore, using a swivel chair interface to create cloud scenarios that capture a given cloud system is a tedious and error-prone task.[18,19] In this paper, we aim at providing this generic input template for cloud simulators (GITS) to facilitate and evaluate cloud orchestration solutions. GITS defines the cloud infrastructure configuration and application interoperable descriptions, including components, relationships, interdependencies, HA metrics, and computational and latency requirements. GITS facilitates the applications' interoperability and automated orchestration across multiple cloud providers, thus creating cloud scenarios, ensuring configuration and model reusability, supporting availability, and minimizing error, time, and cost-to-value.

This paper provides a multilayer input template. At the frontend layer, a graphic modeling framework (GMF) project is designed to capture the above cloud model. The GMF generates an Extensible Markup Language (XML) file. In order to ensure data reusability, the XML file is inputted to the middle layer where a JavaScript Object Notation (JSON) template is generated. This template captures the specifications and the availability features/attributes of a cloud model in a human-readable format. In the backend layer, the JSON template is mapped to a Unified Modeling Language (UML) class diagram. GITS is applied to CloudSim, one of the well-defined cloud simulators. Therefore, CloudSim is extended to include HA requirements and the hierarchical structure of the cloud infrastructure (DCs, racks, shelves, and servers) and cloud applications (application types and components). It is important to note that the proposed template can be easily mapped to any other cloud simulator while applying a few tuning steps to the transformation stage.

The rest of this paper is organized as follows. Section 2 presents related work for cloud-based orchestration approaches and the input templates for cloud simulators. Section 3 defines the motivation behind GITS, where it describes cloud orchestration, its challenges, and GITS motivation and its applications. Section 4 describes the GITS framework, where it presents GITS graphical and textual models, and the transformation algorithm to translate them to a data format that is understandable by CloudSim. Section 5 discusses GITS implementation in the cloud environment, specifically in CloudSim, the other encoding methods of GITS, and its performance. Finally, the conclusion and future work are discussed in Section 6.

## 2 | RELATED WORK

This paper provides a generic description of a cloud model as part of the orchestration solutions and applies it to cloud simulators to facilitate the evaluation of these solutions before the latter's readiness for the production phase. Therefore, this section discusses several literature studies that address the cloud application orchestration and define the cloud specification models for different simulation tools.[9,20-25]

### 2.1 | Cloud-based orchestration approaches

One of the main aspects of the orchestration approach is to support the deployment of a cloud application given a certain infrastructure setting. In this case, the deployment solution should satisfy different functional (computational resources and latency) and nonfunctional (HA and service interdependency relations) constraints. For this reason, the orchestration needs to include these requirements in the physical resources and cloud applications' component descriptions to ensure real-time monitoring and allocation solutions.[26] Building a flexible cloud application management system also allows creating sophisticated and complex services based on existing ones. At that point, a developer does not need to design cloud applications and their associated composite services from scratch.[27,28]

Most of the existing open-source orchestration solutions are limited by the ecosystem of their cloud management platforms. Amazon Web Services (AWS) provide AWS CloudFormation as its proprietary orchestration approach.[29] It provides a stack, JSON template that specifies AWS resources (Elastic Load Balancer instances and Amazon Relational Database Service instances). The template describes the resources needed to process certain applications, and it is managed as an entity. The AWS CloudFormation provisions resources and manages their creation, deletions, and dependencies. However, the template is a proprietary approach and ignores HA attributes and redundancy models.

OpenStack proposes Heat as their orchestration platform. Heat uses a templating approach, Heat Orchestration Template (HOT), to manage resource creation and management and facilitate portability between multiple cloud environments.[30] The proposed template has a similar structure to the AWS CloudFormation templates and can be integrated with Puppet and Chef. Heat uses YAML files to define its template that supports autoscaling and some HA features, including instance logical grouping, services running in an instance, VMs, or individual instances. Although HOT supports some HA features, it discards the mean time to failure (MTTF), mean time to repair (MTTR), recovery time, and tolerance time as basic HA metrics. Furthermore, it does not describe redundancy models and interdependency between multiple application components. Lastly, it is restricted by its cloud management platform, OpenStack, and depends on its domain-specific language.

The Organization for the Advancement of Structured Information Standards proposes Topology and Orchestration Specification for Cloud Applications (TOSCA).[31] The latter is an industry-based standard developed to define and manage applications and facilitate their portability between different multiple cloud providers, thus minimizing vendor lock-in. Using an XML-based template, TOSCA describes the topology of application components and their interactions, provides an orchestration platform to manage application deployment, and supports interoperability. TOSCA defines topology as a graph with a set of nodes and relationships, with each assigned a type (inheritance, requirements, properties, and implementations). Business Process Model and Notation and Business Process Execution Language are used in TOSCA management plans, which depend on standard workflow settings. On the basis of OpenTOSCA,[32,33] a cloud-based modeling tool, Winery, is designed to define cloud-based application topologies.[34-36] Winery is a web-based graphical model that describes TOSCA cloud service topologies and management plans. Winery consists of a type and template management modules that create and modify components defined in TOSCA. It stores the information in a repository and uses the TOSCA packaging format to import and export them.[35] Similarly, both TOSCA and Winery do not support HA features and redundancy relationships.

Carlson et al proposed Cloud Application Management for Platforms (CAMP).[32] CAMP is an approach that improves cloud interoperability over different cloud infrastructures. With CAMP, the authors were aiming at defining specifications that facilitate cloud application management. CAMP provides a self-service management application program interface (API) that allows a platform implementation layer to control application deployment and platform usage. Through its specifications, CAMP allows the creation of different services that can interact with other platforms, thus facilitating interoperability. Although CAMP aims at facilitating cloud portability, it discards the impact of interoperability on cloud applications' HA. It does not describe service interactions, redundancy models, recovery, and repair policies.

## 2.2 | Input templates for cloud-based simulators

A simulation environment is needed to evaluate any cloud design and associated scenarios given a set of metrics (computational resources, service level agreement [SLA] requirements, and HA parameters).[37,38] Simulation is an important way for the cloud to allow repeatable scenarios in a controlled environment. They can evaluate different cloud parameters, such as outage or security issues. In turn, the evaluation strategies can be used as filters for the failed approaches compared to others or for the approaches that do not meet the QoS. However, a simulation tool requires an input modeling or templating of a cloud system to ensure scenario reusability and system modularity and minimizes error associated during manual input model's generation.

Tian et al proposed CloudSched, which is a Java-based simulation tool to evaluate the resource allocation for cloud-based applications.[39] The tool consists of a graphical user interface (GUI) where the simulation setup and cloud scenarios are created. The specification of cloud scenarios does not follow any standardized format, which hinders the reusability of certain scenarios. The proposed GUI is simple and has restrictions on the number of physical machines and VMs, which limits its functionality in simulating real cloud systems. Although CloudSched provides a GUI to facilitate setup creations, it discards the need for data reusability. Additionally, it does not capture the cloud provider and user models and overlooks any HA attributes or application interactions. Filho and Rodrigues proposed a YAML document as a template to create scenarios to CloudSim.[40] Although the YAML schema ensures simplicity and reusability, the proposed template is proprietary to CloudSim and does not capture any HA metrics. This proprietary hinders the applications' portability across different cloud simulators.

Wickremasinghe et al provided CloudAnalyst, which is an extension to CloudSim.[41] CloudAnalyst describes the cloud applications' workloads, DC resources, and traffic/DC geographical locations. With these features, the request response time, processing time, and other related measures are determined. CloudAnalyst consists of a Java-based GUI to create different cloud setups and a graphical output representation in table and chart forms. It also provides XML-based files to save simulation input data and results. Although CloudAnalyst supports XML files, it does not provide an input model template to minimize the error and time consumed by cloud users.

Jakovits et al and Srirama et al developed the Stratus cloud simulation framework.[42,43] Stratus provides simulations of distributed cloud applications using bulk synchronous parallel models. The bulk synchronous parallel model consists of an iterative algorithm that ensures task parallelism. Stratus also provisions resources to allow for the scaling up and down of cloud scientific applications. However, Stratus does not provide an input template to facilitate the creation of cloud scenarios. It also overlooks the HA features of the cloud model. Guo et al provided a service specification for simulating Software as a Service.[44] The specification describes service-oriented and Software-as-a-Service setups, which generates a metamodel for simulated scenarios. Although the authors proposed a web-based GUI, they did not capture the HA metrics of different cloud elements. Moreover, the GUI does not support the reusability and repeatability features of the cloud scenarios.

While Balmer et al proposed MATSim, a traffic simulator,[45] Behrisch et al provided the Simulation of Urban Mobility tool that can be modified to simulate cloud-based scenarios.[46] Contrary to MATSim, the Simulation of Urban Mobility tool has a GUI, but both simulators use configuration files to import its input parameters and export its simulation results. In both tools, the cloud scenario discards HA characteristics, and its creation requires deep knowledge from the user to the configuration settings of the cloud system. In contrary, the GITS proposes a human-readable and HA-aware template.

While Citron and Zlotnick proposed a simulation and emulation approach to evaluate a cloud management environment,[47] Calheiros et al proposed EMUSIM, which is a CloudSim extension to leverage the automated emulation of a cloud system.[48] The approach of Citron and Zlotnick emulates the cloud infrastructure and simulates their VMs based on different management, development, and deployment policies. As for EMUSIM, the simulator understands the behavior of the cloud application to extract its information and use it as an input model for CloudSim to enhance the simulation accuracy. While both approaches aim at emulating the cloud application, they overlook the availability policy and its impact on the cloud application's configuration and the simulator input model.

DynamicCloudSim extends the CloudSim simulator to model the dynamic changes, inhomogeneity, and instability in the performance of computational resources at runtime as well as the resource failure of the task execution.[49] Fakhfakh et al extended CloudSim with CloudSim for Dynamic Workflow (CloudSim4DWf) to model the dynamic changes in the applications' workflow during execution.[50] For this purpose, CloudSim4DWf aims at including different resource provisioning approaches for dynamic workflows in the cloud. Both DynamicCloudSim and CloudSim4DWf can be used in parallel with GITS, but they discard the HA characteristics of cloud infrastructure and application, in addition to the absence of a generic template that would facilitate the automation of the cloud input model.

De Boer et al proposed an abstract approach to model the resource consumption, performance, and deployment of different cloud scenarios.[51] Using the abstract behavioral specification language, the model can execute extensive simulations and predict the output of multiple load balancing, scheduling, or deployment policies. Zeng et al proposed IoTSim, which is an extension to the CloudSim simulator to model batch processing simulation.[52] IoTSim is designed to automate the configurations of big data processing in the cloud environment. Piraghaj et al proposed ContainerCloudSim, as an extension of CloudSim that models the simulation of containers in a cloud environment.[53] The authors demonstrated the scalability of the proposed extension in terms of the number of containers in a DC, but the simulator does not include a generic input model that would facilitate the evaluation of different simulations' scenarios. GITS can be easily extended to support container architecture and complement ContainerCloudSim not only in terms of the input model template but also to execute HA policies on container deployment in the cloud.

In this paper, we distinguish our work from the previous initiatives by developing a generic input template as one of the building blocks of the cloud orchestration solution that can be used by any cloud simulator, mainly CloudSim. GITS has a graphical modeler where the Eclipse GMF model is created. In the middle layer, a Parser engine is developed to ensure the generation of a JSON-based template from XML-based files. The JSON template ensures the input models' repeatability and reusability. It also captures the cloud environment from both the provider and user sides. It maps the cloud infrastructure with cloud applications using lightweight virtualization technology, VMs or containers. One of the main features of GITS is the HA-aware specifications. It describes the redundancy models of cloud applications, recovery/repair policies, interdependency between applications, MTTF, MTTR, and other HA metrics. With GITS, any cloud scenario can be created, which includes the needed infrastructure configuration files (resources and HA metrics) and the cloud application structure information.

## 3 | BACKGROUND AND MOTIVATION

Kratzke et al have shown that the percentage of the proposed cloud orchestration (ie, standardization) solution has been decreasing due to the quick emergence of new cloud offerings compared to the pace of standardizing the existing cloud services.[54] However, the European Commission has discussed the need for Open Standards in their interoperability platform mainly for the government services.[55] Furthermore, the need for cloud standards, mainly in the public acquisition services, has been proposed by the United States National Institute of Standards and Technology and the United Kingdom Government Cabinet Office.[56,57] To align with the continuous need for a well-defined orchestration solution with cloud standardization, GITS focuses on developing a unified cloud template model that would not only facilitate the service portability and platform-agnostic deployment solutions but also provide an HA-aware description as HA is now one of the key parameters for the success of any cloud-based solution.[58-62] In this section, we discuss cloud orchestration and GITS contributions and its application.

### 3.1 | Cloud orchestration

According to Fujitsu, a hybrid cloud is the new benchmark in the Information and Communications Technology area.[63] However, many of these systems do not support cloud orchestration approaches.[63] Cloud orchestration is gaining a lot of momentum nowadays as it is expected to perform and facilitate the deployment of various complex architectures of cloud services in different domains, including industry, e-government, and science.[64]

Cloud orchestration refers to the management, at the infrastructure and application levels, and the flexible deployment of the cloud virtualized resources while satisfying the operational objectives and QoS of the cloud providers and users. It provides visibility, monitoring, and control of the physical sources and software services while satisfying the business requirements. It is necessary to note that cloud orchestration approaches can be implemented either at the cloud application (software) service level or at the cloud infrastructure (hardware) service level.[65]

- *Cloud application orchestration*. This case manages the cloud application's components as executable instances that can interact with each other and with different applications. Each of these instances is associated with its physical host on which they will execute.
- *Cloud infrastructure orchestration*. This case orchestrates the computational resources of the cloud infrastructure (server's memory, CPU, etc) in order to process the user's requests and satisfy the SLA of the cloud provider. The orchestration approach can also allow dynamic workload reallocation due to sudden changes in the cloud model; thus, it can consolidate multiple physical cloud resources and reduce the operational costs.

In order to maintain the value and the need of cloud, it is necessary to deploy a dynamic cloud management approach that can scale up and down the resources, react to sudden changes/demands, and facilitate the standardized application's configurations and portability/interoperability between different cloud models. However, with the emergence of Big Data and Internet of Things (IoT), the challenges of cloud management and orchestration are not limited to the existing traditional architectures (client-server model) but are associated with the need for dynamic models. Additionally, the cost of managing cloud resources and applications is very high.[66] Therefore, cloud orchestration approaches are critical needs to, first, facilitate an effective design and management of large-scale DCs and applications in the cloud and, second, to minimize the work of the information technology team to keep up with the SLAs and the needs of the rapidly changing technologies and business model.[67]

## 3.2 | GITS contributions

This section addresses the different cloud issues and the corresponding contributions proposed by the GITS framework.

1. ***Cloud-based application interactions***

   Cloud-based applications consist of multiple components that interact with each other due to dependency-redundancy relations. When modeling these applications, it is necessary to describe the interaction relations between their components and their dependency on the cloud infrastructure. In GITS, a component-based application is modeled to capture different redundancy and dependency relations among different components and their dependencies on a given cloud infrastructure.
2. ***Cloud provider-user partition***

   Vendor lock-in is one of the challenges faced by users when choosing the corresponding cloud providers.[68] Many cloud-based applications are provided and offered by the same vendors, such as the Google office suite application[69] or the Salesforce Customer Relationship Management application.[70] This creates user dependency on certain providers. Therefore, it is necessary to separate the cloud providers and cloud applications, which allows users to select the appropriate provider and application vendors. Besides, this partitioning allows the provider and the user to enlarge their customer numbers and share a standardized cloud provider-user template.

   GITS provides a well-defined cloud template that ensures a separation between the cloud provider and the cloud user while mapping them through a virtualization engine, where VMs/containers are instantiated according to a given allocation objective. The proposed template does not depend on certain technologies or cloud simulators.
3. ***HA for cloud applications***

   The dependency on multiple cloud services does not only increase the number of customers but requires systems that are always available anytime and anywhere. Building a highly available cloud environment becomes inevitable. Although different HA-aware policies for cloud applications have been proposed,[71] they are proprietary approaches, which hamper the application's interoperability across different providers. Furthermore, the literature has many cloud simulations with predefined input models, but up to our knowledge, the literature does not propose an input template that captures availability metrics. These metrics include MTTF, MTTR, redundancy models, failover policies, and other HA attributes.[72-76]

   HA modeling is one of the main contributions of GITS. Different redundancy models, availability parameters, failure types, and HA middleware attributes are captured in the GITS template.
4. ***Customization of applications***

   Cloud users have different functional and nonfunctional requirements on applications. In order to satisfy their needs, these applications should be well defined, modeled, and evaluated while considering modularity and portability. Building a modular and portable template allows the cloud users to adjust the application to meet certain objectives and a given cloud infrastructure. Template customization requires prompt variability adaptation without violating application interactions. To ensure customization and modularity, generic cloud templates should be designed, which can run on any cloud environment.

The GITS template is generically designed to model cloud providers and applications. With the appropriate transformation algorithm, GITS can capture the input model of CloudSim and any other cloud simulator or environment.

5. ***Unique template for cloud management***

   Each cloud provider has its own APIs and tools to set up applications and infrastructures and provision them. It is necessary to define a rubric that describes the necessary attributes for a cloud provider-user management module. An architecture of a unification layer that syntactically and functionally unifies the APIs of different providers and their provisioning infrastructure is needed. This unification layer is the basis for describing the provider-independent provisioning scripts for applications.

   GITS provides a generic specification that provides a unified view of requirements, parameters, and interactions between different entities of a cloud. This specification facilitates the description of management and provisioning unit for cloud infrastructure and applications.

   Cloud applications currently are hosted in VMs and containers. These virtualization technologies can scale up and down based on the user needs. However, scaling the applications depends on a central information system, such as Netflix Eureka[77] and Kubernetes.[78] GITS aims at modeling the application while taking these requirements into consideration. It models the cloud applications to facilitate the decisions of the applications' deployments while defining their dependencies. Thus, GITS initiates a database (DB) instance before creating an App instance since the latter depends on DB, ie, App requires a DB instance to function normally especially if the tolerance time of App is less than the recovery time of a DB instance.

   Additionally, the GITS model can be integrated with various load balancer technologies and gateway APIs that exist in the cloud. For instance, GITS can support the Netflix cloud stack, such as Zuul[79] and Ribbon[80] load balancing, to generate decisions on routing the user requests to a specific application's components.

## 3.3 | Application of GITS

GITS can be used for different purposes.

1. *Integration with the orchestration approach.* GITS is used as a building block in the orchestration approach. In terms of the application stack, it exists on top of the orchestration manager, where the latter can collect updates on the cloud resources and services. It communicates with both the provider and orchestrator building blocks (provisioning, monitoring, and patching) to execute specific cloud scenarios and implement the corresponding placement algorithm. With GITS, generic configurations, different deployment scenarios, and QoS-aware policies can be implemented not only to support performance constraints but also to execute and evaluate multiple HA-aware deployment approaches.
2. *Input model for different cloud deployment solutions in the simulator environment.* As multiple placement solutions can be implemented in the cloud simulators, it is necessary to have a templating for generating multiple scenarios to evaluate these solutions. GITS replaces the “hard-coded” input of any cloud-based simulator, particularly CloudSim. GITS is applied on CloudSim since it is one of the most mature and well-developed cloud simulators. However, GITS can fit with the objective of any other cloud simulator while applying some tweaking to the transformation algorithm that maps a GITS scenario to the simulator input model. Before running any simulation, the user can use either the GITS textual or graphical model to input the cloud scenario while overlooking the mapping between the input model and the used cloud simulator. GITS will then execute the transformation algorithm to map the scenario to the CloudSim's input format.
3. *Support HA objectives.* Although cloud computing provides benefits to different players in its ecosystem and makes services available anytime, anywhere, and in any context, many concerns arise as to the HA of cloud-based applications, where business is expected to be running in the occurrence of any disruptive incident or sudden failure.[81,82] The absence of an application protection plan has a tremendous effect on business continuity and information technology enterprises. For example, Amazon has faced an outage due to a “human-typo” on February 28, 2017, which has affected many services and websites, including Quora, Sailthru newsletter provider, Business Insider, Slack filesharing, and various connected IoT hardware.[83] Moreover, the number of active Facebook users, for instance, has grown to 1.5 billion, which requires an immune system to ensure data delivery anytime and anywhere.[84] The International

Data Corporation expects that the IoT market will reach $1.7 trillion in 2020.[85] This will cause a spurt growth in the cloud, and the key solution is to integrate HA-aware objectives in the cloud solutions. As a start point, GITS provides an HA-aware description and a hierarchical structure of the cloud infrastructure and application that would facilitate the execution of different HA-aware solutions (load balancing, elasticity, and failover).

## 4 | GITS FRAMEWORK

Generating scenarios for CloudSim and other cloud simulators requires knowledge in the development and simulator environment. It is not always the case that the cloud users are aware of the code needed to generate these scenarios. This can hinder the process of application deployment and cloud system evaluations because of the complexity challenges associated with cloud scenario creation. For instance, if cloud-based applications are deployed while considering the impact of the dependency relationship among them, a new code should be added, which reflects this interaction. This can be an exhausting task, which is not only an erroneous process but also requires prerequisite knowledge in the tool used.

With GITS, cloud brokers, providers, and users can easily create different sophisticated cloud infrastructure and applications with various topologies and structures. In other cases and due to sudden changes that might affect the cloud application performance (outage, malware, overload, etc), the applications might change its behavior.[86] GITS also minimizes the exposure of cloud users to the development process and provides an intuitive way to generate any cloud scenario that depicts applications, infrastructure, and their interfaces. The objective of this template is to separate the cloud providers from the cloud users and ensure their mapping through a virtualization layer. In addition to performance metrics (computational resources and latency), modeling HA parameters allows considering the availability as an objective during cloud application deployment, where applications are mapped to the providers that protect them according to a given redundancy model and other related metrics.

Creating such a template is achieved using a UML model to build a cloud model, a JSON file to have a human-readable template, and, finally, a graphical interface to maintain the GITS user-friendly feature. In the following, we describe the details of GITS design.

### 4.1 | CloudSim simulator

CloudSim is a Java-based open-source framework for modeling, simulating, and evaluating cloud environments.[18,19] It is proposed by the GRIDS laboratory and developed on top of SimJava, which is a DES.[87] CloudSim is used to model cloud infrastructure (DCs and servers), cloud broker, cloud information system, and multiple time- and space-based allocation and scheduling policies. Besides, it models container/VM processing, including instantiation, provisioning, and destruction. With CloudSim, large cloud systems can be processed using different scheduling approaches, where VMs/containers are mapped to the hosts that satisfy their computational resources. Although CloudSim is a self-contained tool that models cloud architecture, it does not support the simpler creation of cloud scenarios. The latter is manually created or hard-coded, which can be a tedious and error-prone job. Therefore, in this paper, a generic input template is proposed that models cloud infrastructure and applications. This template can be easily tuned to model scenarios for different cloud simulators.

### 4.2 | Component-based architecture

Component-based architecture (CBA) provides well-designed complex systems with different methods, properties, and events. CBA decomposes the system design into logical or functional subsystems, where each compromises a specific partition of the whole communicating system. With this system abstraction, CBA allows designing the cloud models using software entities of commercial off the shelf offered by multiple cloud providers. Adapting CBA in building cloud templates ensures component reusability, replaceability, extensibility, and modularity. Each entity, known as a component, of the CBA encapsulates certain properties, methods, and behaviors that represent, for example, an element of the cloud-based model (application and infrastructure). For example, SaaS offers services as Internet-based applications, including multiple component-based applications.[88] These applications communicate using a message exchange engine that is based on protocols, such as Web Services Description Language or the Simple Object Access Protocol. Different platforms can be used to build CBA, including JavaBeans,[89] Common Object Request Broker Architecture,[90] Distributed Component Object Model,[91] and Microsoft.NET. With component-based applications, the cloud can offer scalable, interoperable, and highly available services.

To this end, CBA can be adopted to design a generic template for cloud simulators, which aims at enhancing system quality, evaluation, and building scenarios from standard elements instead of redesigning the wheel. Additionally, it replaces the manual design of cloud entities through an automated generation of scenarios that can be parameterized according to a given cloud environment (provider and/or simulator objective).[92] Thus, entities needed for particular application and infrastructure are instantiated from the proposed template. This describes a framework that includes the structural entities, methods, and parameters of the cloud system and the interconnection behaviors between its different elements. The main idea is to design an interoperable cloud-based template through well-defined connections that can be easily customized to meet a given cloud management system, such as OpenStack (HOT) and AWS Elastic Compute Cloud (CloudFormation template), and the associated SLA requirements.

## 4.3 | GITS UML model

The initial phase in designing a generic template for cloud simulators is abstracting the cloud through system modeling. The objective of abstract modeling is to define different cloud characteristics, faulty points, redundancy model, recuperation phases, and other availability and performance attributes. In this paper, a UML class diagram is used to provide an abstract view of the cloud model. The diagram can be partitioned into cloud infrastructure, cloud application, and virtualization layers. Figure 1 shows the Ecore model of the cloud UML diagram.

1. ***Cloud infrastructure***
   The cloud infrastructure consists of a DC network, where each DC has its computational resources (CPU and memory) and associated HA metrics, such as MTTF and MTTR. Each DC has one or many rack(s). Similarly, each rack is

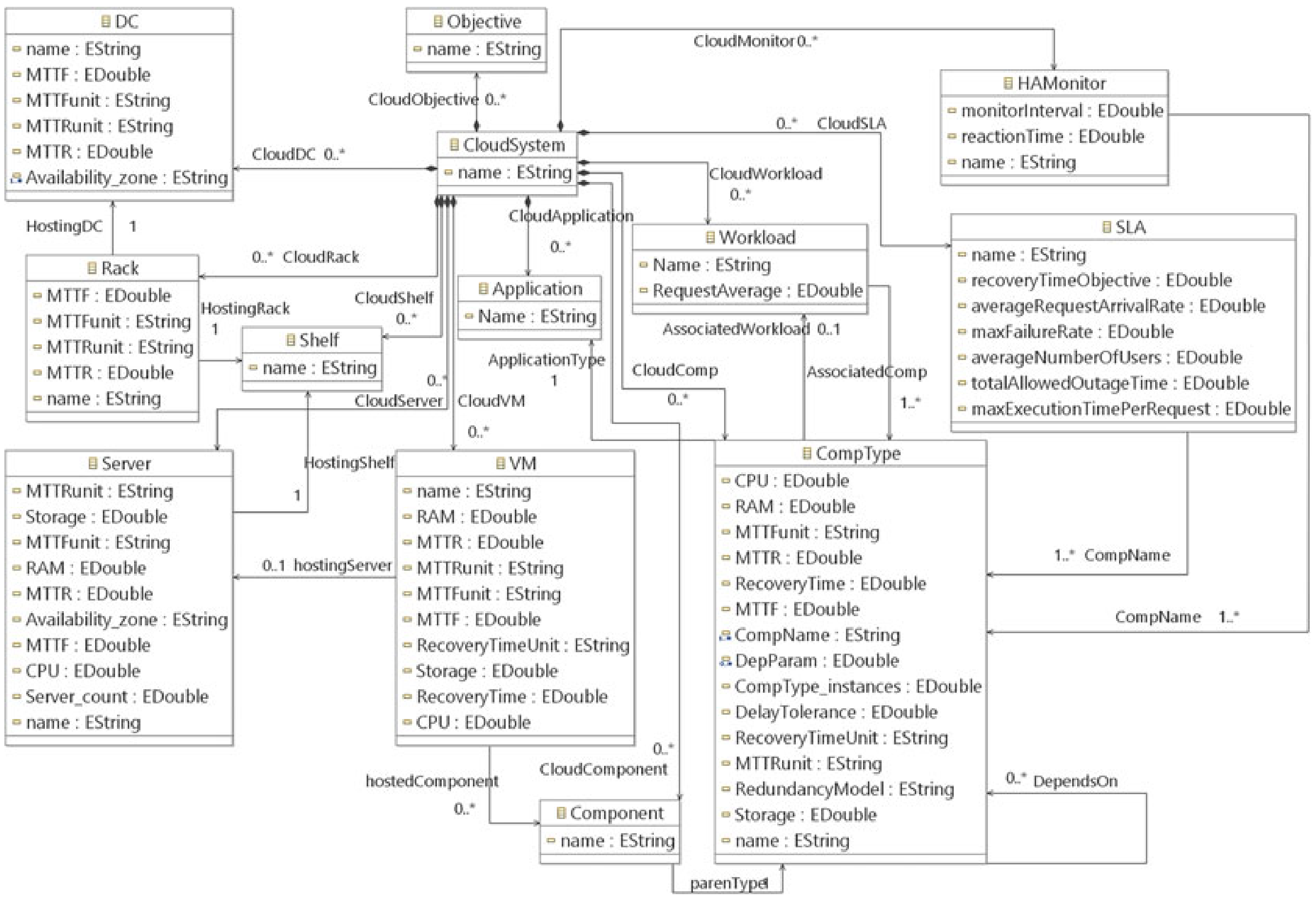


**FIGURE 1** Unified Modeling Language Ecore diagram of the generic input template for cloud simulators model [Colour figure can be viewed at wileyonlinelibrary.com]

characterized by its available resources and HA attributes. Multiple shelves can be grouped in one rack, where each shelf has one or more server(s). Each server is defined through its resources and HA parameters. Each of these elements has other attributes defined in the class diagram to be used for designing and scheduling purposes.

2. ***Cloud application***

The cloud-based application consists of multiple component types. Each type has multiple components, where one component is considered active, whereas the others represent the redundant ones. Each component type requires specific computational resources to be properly processed. These resources are captured as flavors in the proposed diagram. Flavors can be repressed as disk resources (disk size or speed), operating system specification, CPU requirements (core, cache size, speed), memory (RAM size or speed), and/or network (bandwidth or latency). Each component type has its own failure types that define failure scope (impact of component type failure), MTTF, MTTR, and recovery time.

Each component type consists of multiple redundant components, which forms a protection or redundancy group. Each group determines the number of active, standby, and/or spare components depending on the used redundancy model. In order to maintain HA, each component follows a sequence of operations during its life cycle. The state model of an HA solution is captured in the UML diagram. Figure 2 depicts the above component state model. During this life cycle, an HA orchestrator or middleware monitors the health of one or more components. Each orchestrator is characterized by a monitoring frequency and a response time. Upon failure detection, the faulty component is isolated and fails over to its redundant component(s). The failover time depends on the redundancy model (active/active, active/standby, or active/spare). Figure 3 depicts the impact of the redundancy model type on the failover time calculation. For instance, if the redundancy model is active/spare, the failover time is the summation of the instantiation time, fetch state delay, parsing state delay, recuperation duration, execution time, and termination duration. Each of these duration depends on the flavors of the component's host. Each component type can interact with other types

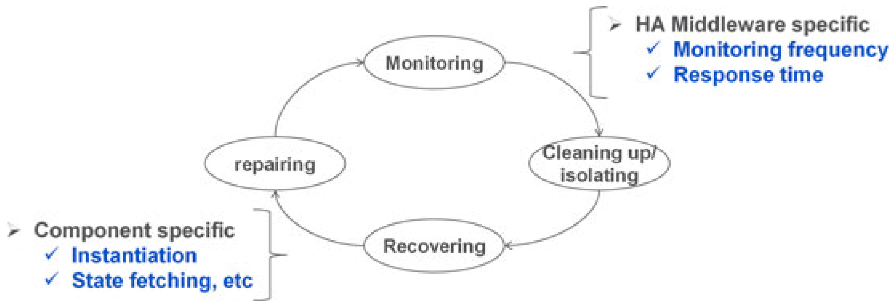


**FIGURE 2** High availability (HA) solution state model [Colour figure can be viewed at wileyonlinelibrary.com]

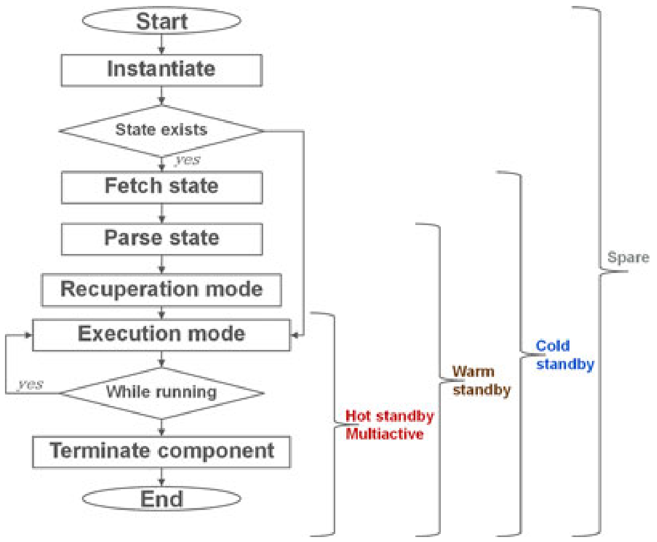


**FIGURE 3** Effect of the redundancy model on failover time [Colour figure can be viewed at wileyonlinelibrary.com]

through dependency relation. A type can sponsor or depend on another type. A three-tier web application can be an example of cloud-based applications. A web application consists of a Hypertext Transfer Protocol Secure server at the frontend, which processes user requests and forwards them to an App server. The latter generates the required content and, in turn, depends on the backend DB server that stores the users' data. The communication between these different component types forms the functional path that a request should follow to be successfully processed. The dependency between different types is characterized by a delay tolerance that represents the allowed delay between them and a tolerance time that determines how much a dependent component can tolerate the absence of its sponsor(s). These metrics have an important role when selecting dependent and sponsor placements. Each component is associated with an SLA that determines the allowed outage time, average request arrival rate, recovery time objective, and other HA, performance, and scheduling attributes.

3. ***Virtual mapping***

   The applications' components are mapped to the servers that can satisfy their computation needs, HA, and other performance objectives. Once the allocator finds the best server that can host a given application component, a virtual mapping is generated between the server and the component. The virtual mapping can be a VM or a container. This mapping forms the glue between the cloud provider and the user.

## 4.4 | GITS JSON file

The UML class diagram provides an abstract view of the cloud model, but it does not ensure simplicity in scenario creation and repeatability. For this purpose, we use JSON to represent a cloud template. JSON is a simple, lightweight, human-readable, and universal representation language.[93] JSON does not depend on the programming language type and can support multiple data types (numbers, arrays, objects, strings, Boolean, and null) and deep-level hierarchical data. Moreover, JSON has many extensions that enable cyclic relation implementation, such as dojox in the Dojo toolkit used in Google Content Delivery Network.[94] To this end, we use a JSON data format to represent a readable and reusable cloud setting. Figures 4, 5, and 6 show the JSON files for the cloud infrastructure, application, and virtualization layers.

The GITS JSON template consists of the following.

1. ***Objective***

   It represents the data type that represents the goal behind using the template. It can be either evaluation or scheduling. In the case of the "evaluation" objective, the template is inputted to a cloud simulator, such as CloudSim, to evaluate certain application deployment in terms of availability or other performance. For this purpose, the template is populated with deployment information of a certain application, such as the hosts of the application components. In the case of the "scheduling" objective, the template is inputted to a cloud simulator to schedule the application components.

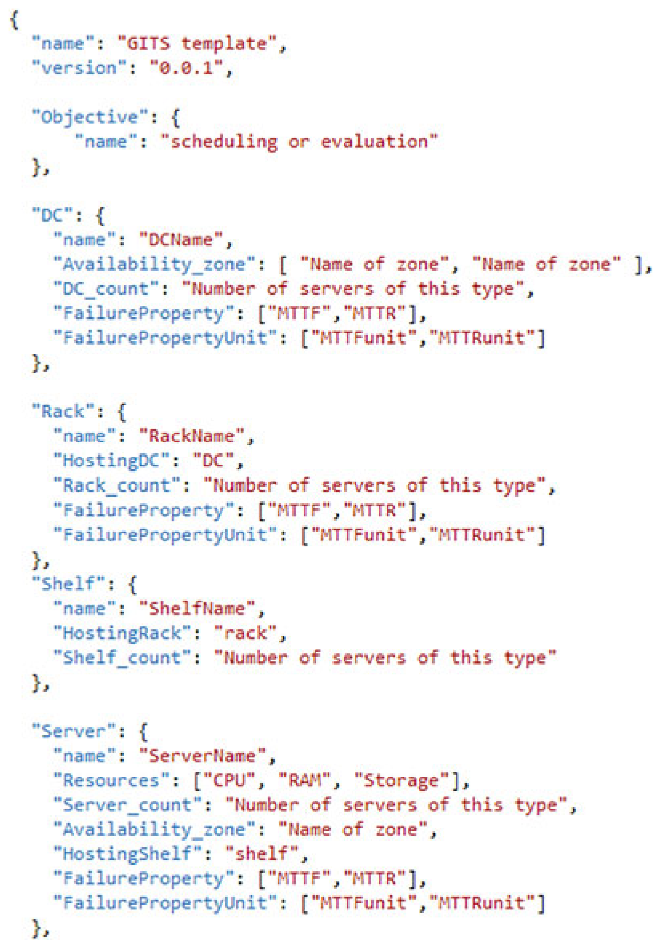
2. ***Cloud infrastructure information***

   *DC, rack, and shelf*. It represents the DC, rack, and shelf details of a given cloud infrastructure. It includes the *name* (string type), *Availability_zone* (array of strings) that hosts its servers, *FailureProperty* (array of numbers) that represents its MTTF and MTTR, and *FailurePropertyUnit* (array of strings) that reflects units of MTTF and MTTR. If multiple DCs/racks/shelves hold similar characteristics, a "count" attribute can be defined to automate their generation and avoid repetition.

   *Server*. It represents the server details of a given cloud infrastructure. It includes the server *name* (string), *Resources* (array of numbers) that represents its computational resources, *Availability_zone* (string) that hosts it, *HostingShelf* (shelf object) that represents the shelf hosting the corresponding server, *FailureProperty* (array of numbers) that represents its MTTF and MTTR, and *FailurePropertyUnit* (array of strings) that reflects units of MTTF and MTTR. If multiple servers hold similar characteristics and reside on the same shelf, a *Server_count* can be defined to automate their generation and avoid repetition.

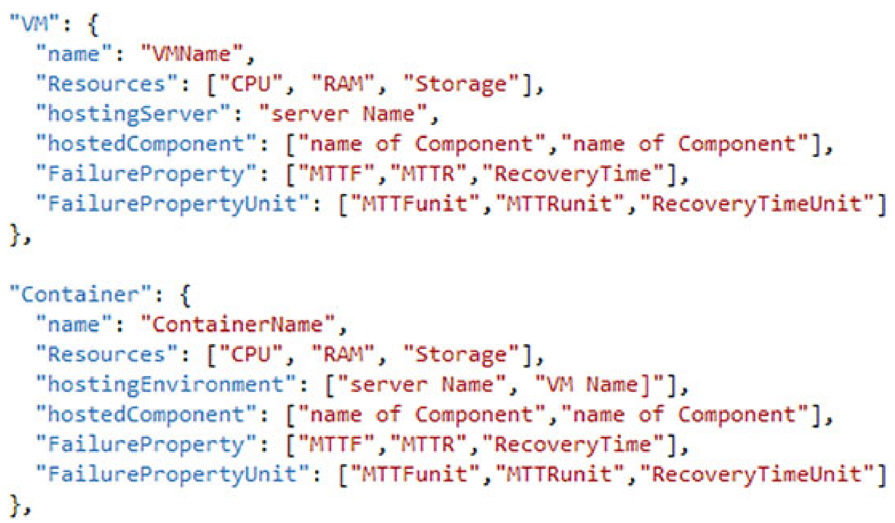
3. ***Virtualization layer information***

   *VM*. It represents the VM details of a given cloud environment. It includes the VM *name* (string), *Resources* (array of numbers) that represents its computational resources, *HostingServer* (server object) that represents the server hosting the corresponding VM, and *HostedComponent* (array of multiple component objects) that represents the components hosted on this VM. *HostingServer* and *HostedComponent* properties are populated only if the objective is "evaluation." Moreover, the VM includes *FailureProperty* (array of numbers) that represents its MTTF, MTTR, and recovery time and *FailurePropertyUnit* (array of strings) that reflects units of MTTF, MTTR, and recovery time.



```
{
  "name": "GITS template",
  "version": "0.0.1",

  "Objective": {
      "name": "scheduling or evaluation"
  },

  "DC": {
    "name": "DCName",
    "Availability_zone": [ "Name of zone", "Name of zone" ],
    "DC_count": "Number of servers of this type",
    "FailureProperty": ["MTTF","MTTR"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit"]
  },

  "Rack": {
    "name": "RackName",
    "HostingDC": "DC",
    "Rack_count": "Number of servers of this type",
    "FailureProperty": ["MTTF","MTTR"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit"]
  },
  "Shelf": {
    "name": "ShelfName",
    "HostingRack": "rack",
    "Shelf_count": "Number of servers of this type"
  },

  "Server": {
    "name": "ServerName",
    "Resources": ["CPU", "RAM", "Storage"],
    "Server_count": "Number of servers of this type",
    "Availability_zone": "Name of zone",
    "HostingShelf": "shelf",
    "FailureProperty": ["MTTF","MTTR"],
    "FailurePropertyUnit": ["MTTFunit","MTTRunit"]
  },
```

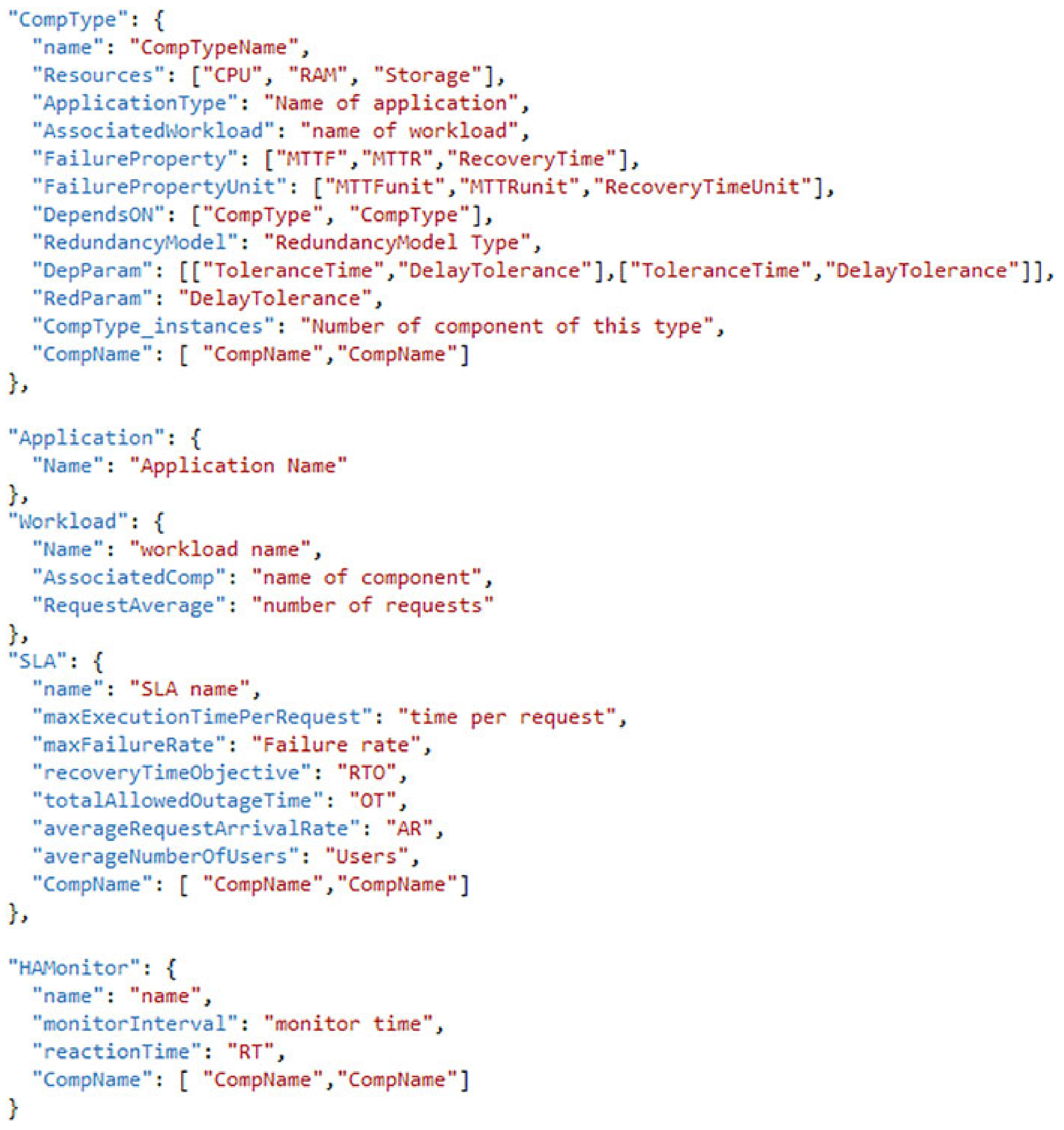
**FIGURE 4** JavaScript Object Notation–based cloud infrastructure template [Colour figure can be viewed at wileyonlinelibrary.com]



```
"VM": {
  "name": "VMName",
  "Resources": ["CPU", "RAM", "Storage"],
  "hostingServer": "server Name",
  "hostedComponent": ["name of Component","name of Component"],
  "FailureProperty": ["MTTF","MTTR","RecoveryTime"],
  "FailurePropertyUnit": ["MTTFunit","MTTRunit","RecoveryTimeUnit"]
},

"Container": {
  "name": "ContainerName",
  "Resources": ["CPU", "RAM", "Storage"],
  "hostingEnvironment": ["server Name", "VM Name]"],
  "hostedComponent": ["name of Component","name of Component"],
  "FailureProperty": ["MTTF","MTTR","RecoveryTime"],
  "FailurePropertyUnit": ["MTTFunit","MTTRunit","RecoveryTimeUnit"]
},
```

**FIGURE 5** JavaScript Object Notation–based provider-user mapping template [Colour figure can be viewed at wileyonlinelibrary.com]

*Container*. It represents the container details of a given cloud environment. It has the same characteristics as the VM except that the *Host* (server and/or VM object) represents the server hosting the corresponding container. As for the *VM* object, it is populated if the container is hosted on a VM; otherwise, it is null.



```
"CompType": {
  "name": "CompTypeName",
  "Resources": ["CPU", "RAM", "Storage"],
  "ApplicationType": "Name of application",
  "AssociatedWorkload": "name of workload",
  "FailureProperty": ["MTTF","MTTR","RecoveryTime"],
  "FailurePropertyUnit": ["MTTFunit","MTTRunit","RecoveryTimeUnit"],
  "DependsON": ["CompType", "CompType"],
  "RedundancyModel": "RedundancyModel Type",
  "DepParam": [["ToleranceTime","DelayTolerance"],["ToleranceTime","DelayTolerance"]],
  "RedParam": "DelayTolerance",
  "CompType_instances": "Number of component of this type",
  "CompName": [ "CompName","CompName"]
},

"Application": {
  "Name": "Application Name"
},
"Workload": {
  "Name": "workload name",
  "AssociatedComp": "name of component",
  "RequestAverage": "number of requests"
},
"SLA": {
  "name": "SLA name",
  "maxExecutionTimePerRequest": "time per request",
  "maxFailureRate": "Failure rate",
  "recoveryTimeObjective": "RTO",
  "totalAllowedOutageTime": "OT",
  "averageRequestArrivalRate": "AR",
  "averageNumberOfUsers": "Users",
  "CompName": [ "CompName","CompName"]
},

"HAMonitor": {
  "name": "name",
  "monitorInterval": "monitor time",
  "reactionTime": "RT",
  "CompName": [ "CompName","CompName"]
}
```

**FIGURE 6** JavaScript Object Notation–based cloud application template [Colour figure can be viewed at wileyonlinelibrary.com]

4. ***Cloud application information***

*CompType*. It represents the component type details of a given cloud application. It includes the component type *name* (string), *Resources* (array of numbers) that represents its computational resources, and *ApplicationType* (string) and *AssociatedWorkload* (string) that determine the names of the component type application and workload. Additionally, the component type has *FailureProperty* (array of numbers) that represents its MTTF, MTTR, and recovery time and *FailurePropertyUnit* (array of strings) that reflects units of MTTF, MTTR, and recovery time. The interaction between component types is also reflected in the template. It has *RedundancyModel* (string) that determines the type of redundancy model (ie, active/active) and *RedParam* (number) that shows the allowed delay tolerance between the redundant components. It also has *DependsON* (array of multiple component type objects if applicable) that determines the sponsor(s) of the corresponding component type and *DepParam* (array of numbers) that shows the tolerance time of the corresponding component type and the allowed delay tolerance between the dependent components. The number of *DepParam* subarrays is the same as the size of *DependsON*. Finally, the *CompType* determines the number and the names of the components of the same type by populating *CompType_instances* (number) and *CompName* (arrays of strings).

*Application*. It represents the applications deployed in the cloud and has one property, ie, *name* (string).

*Workload*. It represents the workload details associated with each component. It includes the workload *Name* (string), the name of associated components, *AssociatedComp* (string), and a number of average requests, *RequestAverage* (number).

*SLA*. It represents the SLA details associated with application components. It includes the SLA *name* (string), allowed time per request, *maxExecutionTimePerRequest* (number), acceptable failure rate, *maxFailureRate* (number), allowed recovery time objective, *recoveryTimeObjective* (number), allowed outage time, *totalAllowedOutageTime* (number), average arrival number of requests, *averageRequestArrivalRate* (number), average number of users for each component, *averageNumberOfUsers* (number), and list of monitored components, *CompName* (array of strings).

*HAMonitor*. It represents the details of the HA monitor for application components. It includes monitor *name* (string), the frequency of monitoring, *monitorInterval* (number), reaction time to handle faulty node(s), *reactionTime* (number), and a list of monitored components, *CompName* (array of strings).

## 4.5 | GITS graphical interface

In GITS, it is possible to generate the cloud use cases not only using a textual format but also using a graphical format. In the latter case, users can create scenarios through a graphical interface that implements the syntax of the cloud model at the infrastructure and application levels. The graphical interface is the only interaction with the user, and the transformation of the scenarios to the proper data format of the used cloud simulator happens behind the scene. While the JSON template is a simple data representation and exchange format that constructs the cloud scenarios, the proposed interface allows the users to graphically build their cloud use cases with GMF interaction. With GMF, a graphical representation of a domain-specific language can be created and mapped to a graphical and textual concrete syntax.[95]

On the basis of the Graphical Editing Framework and the Eclipse Modeling Framework, a GMF project provides a model-driven process for developing graphical editors in Eclipse. It has a Model-View-Controller architecture that isolates the graphical interface from the domain model, which provides the diagram and domain model, permitting better quality, productivity, and design independency. Figure 7 shows the GMF overview. The required graphical editor has been created using model-to-model transformation and model-to-text transformation.

To generate the GITS graphical editor, a domain, tooling, graphical, mapping, and generator models are defined to build a functional graphical interface based on the GMF runtime.[96] The domain model is based on an Ecore model of the above UML class diagram. Once the Ecore generator model is created, a graphical definition model is created, which defines the cloud nodes (DC, rack, server, component types, and other nodes) and the connections between these nodes (relationships

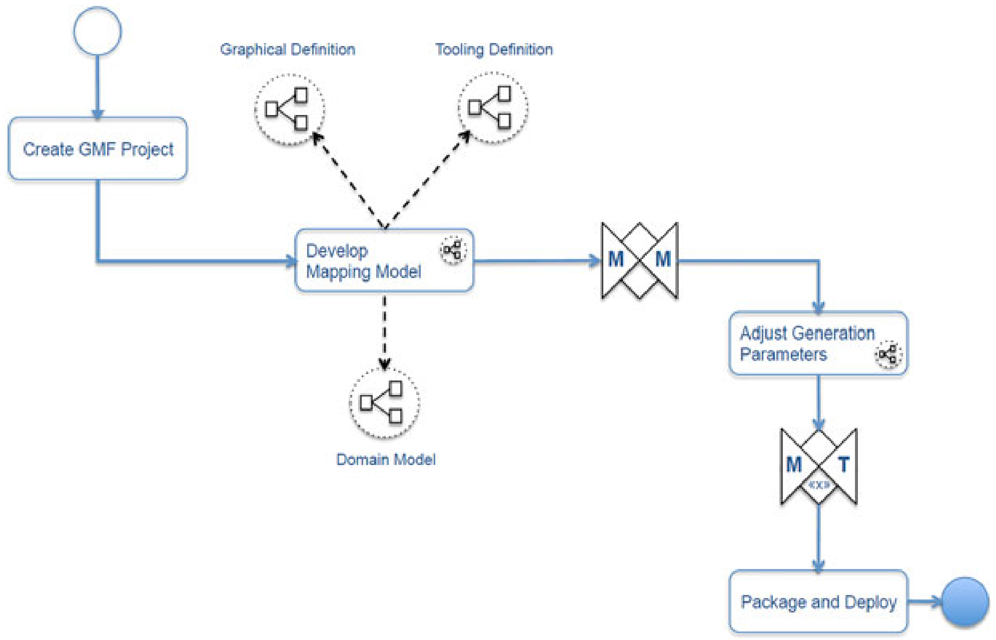


**FIGURE 7** Eclipse graphic modeling framework (GMF) overview [Colour figure can be viewed at wileyonlinelibrary.com]

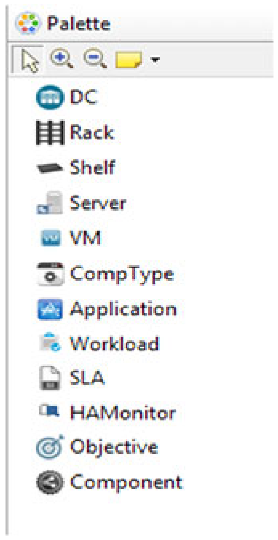


**FIGURE 8** Generic input template for cloud simulator tool palette [Colour figure can be viewed at wileyonlinelibrary.com]

defined in the Ecore model). When the cloud nodes and links are determined, the tool palette of the graphical editor can be created using the tooling definition model (TDM). The TDM describes the cloud elements, their names, and their descriptive icons in the editor palette. Figure 8 shows the GITS tool palette.

The domain model, the graphical definition model, and the TDM are combined to generate the mapping model. The latter is the base of a GMF diagram because it generates the mapping between the nodes, links, and corresponding icons. A successful mapping enables the generation of the desired GMF generator model that produces an extensible graphical diagram based on the GMF runtime.[97] The latter is an industrial application framework that bridges the Graphical Editing Framework and the Eclipse Modeling Framework to create graphical editors. It provides reusable elements such as tool palette, connection handles, and element property menu. The GITS graphical editor is shown in Figure 9. It is a user-friendly interface where cloud elements can be dragged and dropped from the tool palette, and their corresponding properties and links are populated in the property panel. Once the cloud element properties and links are defined, the connections between them are generated automatically.

The GMF simplifies the development complexity of a GUI and reduces the maintenance and testing life cycle. It is simple to generate Java codes from the corresponding editor, and the model is stored as an XML file, a standard data exchange format.

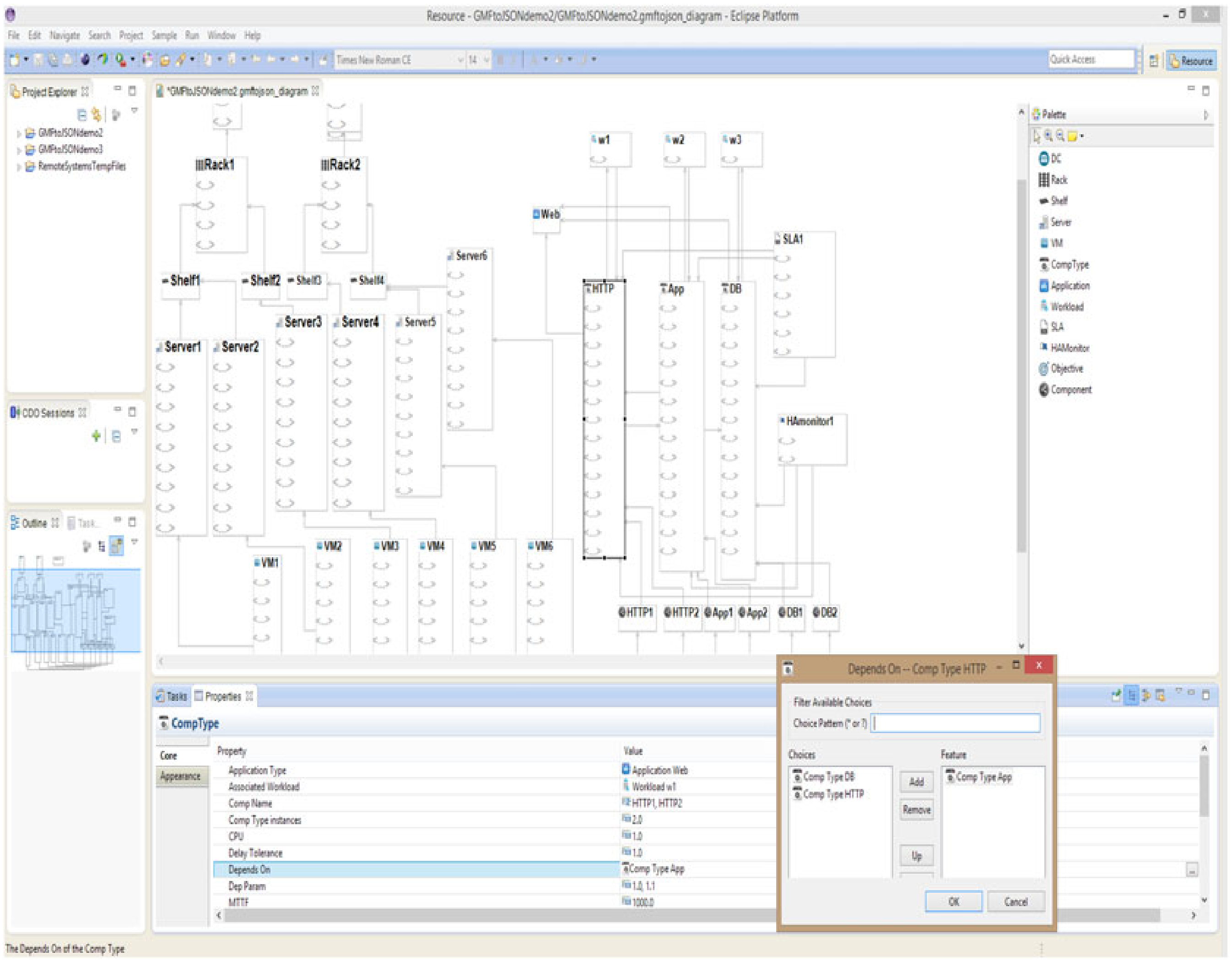
## 4.6 | GITS transformation algorithm

GITS aims at generating a user-friendly, reusable, and interoperable cloud topology. For this purpose, the users populate the GMF editor with a cloud scenario, and the transformation algorithm ensures the mapping of the graphical scenario into a readable data format by the employed cloud simulator. In this paper, CloudSim is the cloud simulator that is extended with GITS. The transformation algorithm consists of multiple subalgorithms to map the graphical input model (using GMF) to the input format of the underlying simulator. Figure 10 shows the GITS transformation algorithm. This algorithm is the automation process that does not require any human intervention, and thus, the user does not need deep knowledge of any language of the simulator input.

The transformation algorithm uses the concept of object modeling and different modules to execute the parsing from GMF to JSON. It then maps the JSON template to the object instances of the cloud-based UML model. Once the instance of this model is populated, it can then be mapped to the input format of any cloud-based simulator. This transformation process executes a library file that will be imported to any cloud simulator's building environment to populate the cloud scenarios. To achieve the data transformation, the details of the different transformation subalgorithms are as follows.

1. ***GMF2JSON***

   The model generated by the graphical editor can be stored as an XML file. Studies have shown that the JSON files can be efficiently parsed in comparison to XML, and it can replace the XML as the data exchange format used in web applications.[98] Figure 11 shows the GMF2JSON transformation. Once the XML file is generated, it is inputted to an



Tasks | Properties

CompType

Core | Appearance

| Property | Value |
|---|---|
| Application Type | Application Web |
| Associated Workload | Workload w1 |
| Comp Name | HTTP1, HTTP2 |
| Comp Type instances | 2.0 |
| CPU | 1.0 |
| Delay Tolerance | 1.0 |
| Depends On | Comp Type App |
| Dep Param | 1.0, 1.1 |
| MTTF | 1000.0 |

Depends On -- Comp Type HTTP

Filter Available Choices

Choice Pattern (* or ?)

Choices: Comp Type DB, Comp Type HTTP

Feature: Comp Type App

Add | Remove | Up | OK | Cancel

**FIGURE 9** Generic input template for cloud simulator graphical editor [Colour figure can be viewed at wileyonlinelibrary.com]

XML-JSON parser to generate the desired JSON template discussed above. A document object model parser is used to create the document builder and examine the nodes, links, and attributes. Jackson 1.x is then used to convert the generated Java objects into JSON data format. In order to create the above JSON template, a mapper algorithm is used to implement the preferred JSON data structure. To this end, the graphical model is transformed into a user-readable and reusable data format.

2. ***JSON2UML***

   In this section, the JSON template is mapped to the above UML class diagram. Using Papyrus, an open-source UML tool is used to build the cloud UML model. The JSON template is used to populate an instance of the UML model. The objects in the JSON file are mapped to Java objects and then mapped to the cloud objects defined in the UML diagram. Figure 11 shows the GMF2UML transformation.

3. ***GITS2CloudSimInput***

   The GITS library file is used to populate the CloudSim input. The JSON template and the library file are inputted to the CloudSim building environment. The CloudSim input can then be populated using the given template. The CloudSim DCs and hosts are populated from the GITS DC and server information. As for the application level, CloudSim does not model the cloud applications, but it captures the VM/container generation. CloudSim is extended to model the cloud applications and their components. The latter is populated from the applications and their component type data defined in GITS. The VM/container in CloudSim is populated from the GITS VM and container information. As for the other data, such as the SLA and HA monitor, their associated information can be accessed by any scheduling or allocation policy to evaluate certain deployment in terms of performance or availability intentions. Since CloudSim does not support any HA algorithms (failover, redundancy, etc), some of the template parameters are

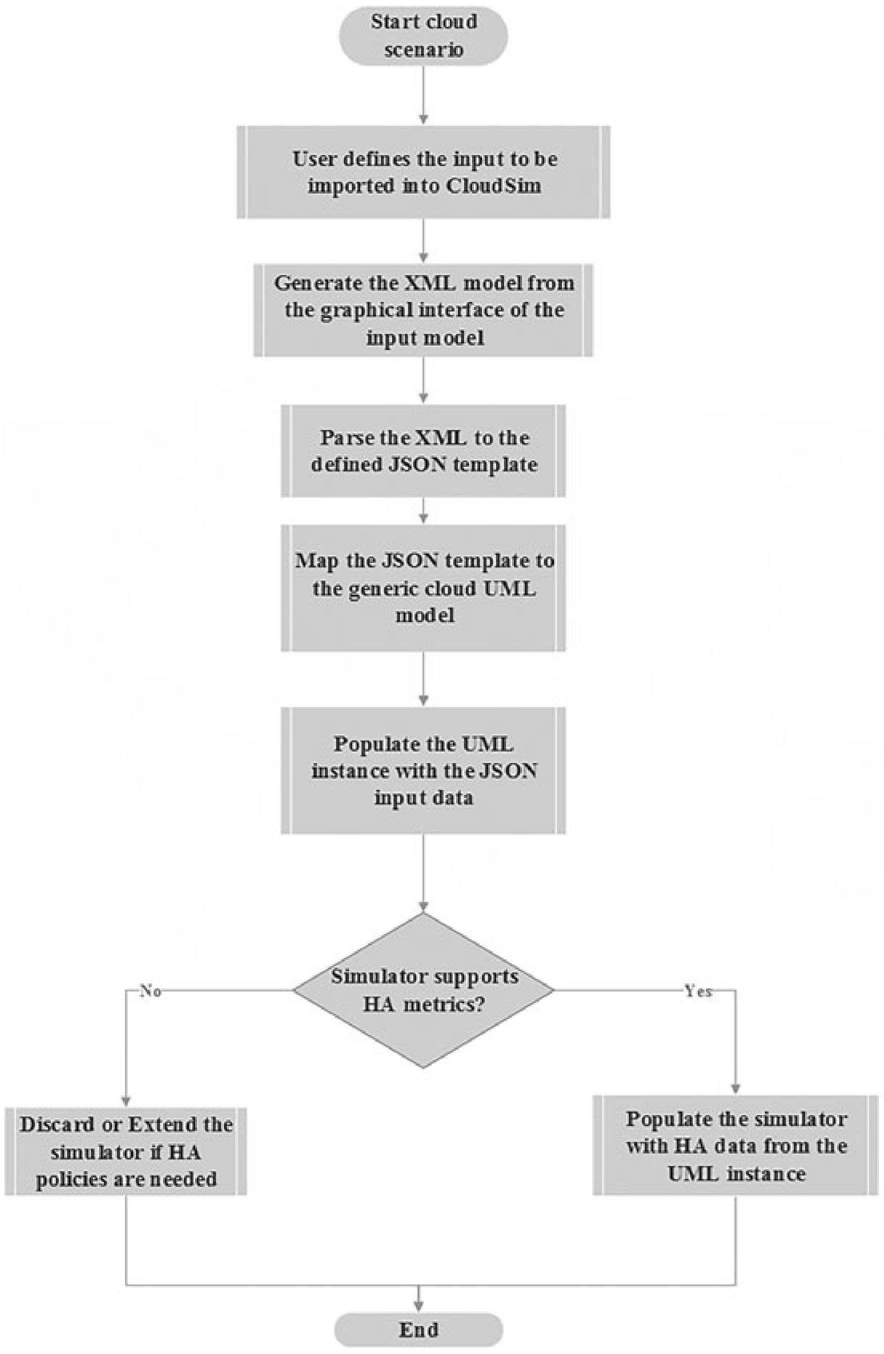

Start cloud scenario
User defines the input to be imported into CloudSim
Generate the XML model from the graphical interface of the input model
Parse the XML to the defined JSON template
Map the JSON template to the generic cloud UML model
Populate the UML instance with the JSON input data
Simulator supports HA metrics?
No
Yes
Discard or Extend the simulator if HA policies are needed
Populate the simulator with HA data from the UML instance
End

**FIGURE 10** Generic input template for cloud simulator transformation algorithm flowchart. HA, high availability; JSON, JavaScript Object Notation; UML, Unified Modeling Language; XML, Extensible Markup Language

not used. However, the template and the GITS library file can be imported to any cloud simulator that supports performance and/or HA objectives, and the simulator input model is populated accordingly. It is important to note that the library file and the JSON template of GITS will be available upon request.

## 5 | GITS DISCUSSION: A CASE STUDY OF CLOUDSIM

This section discusses the implementation and performance of GITS in a cloud environment, particularly CloudSim.

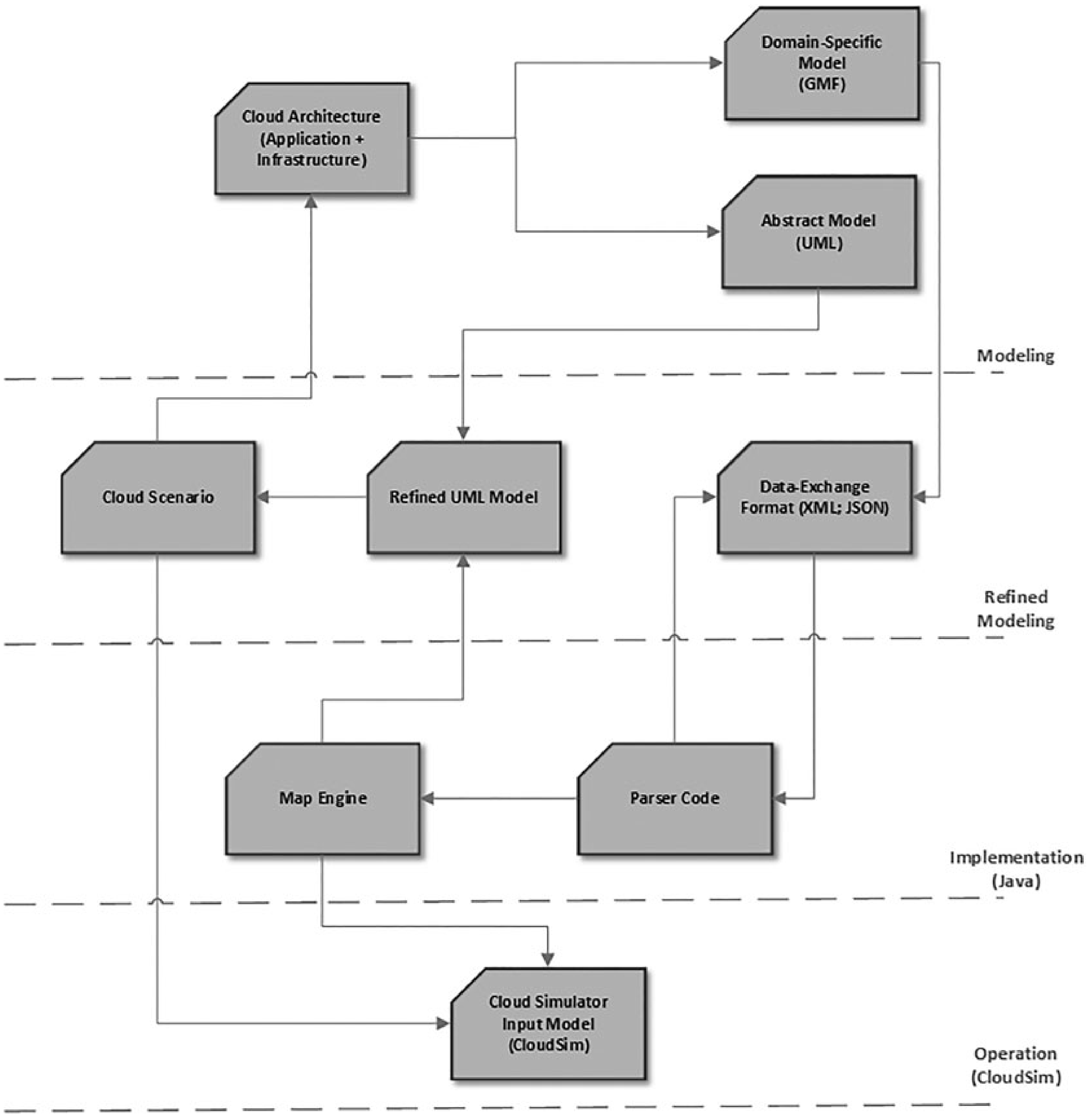

Domain-Specific Model (GMF)
Cloud Architecture (Application + Infrastructure)
Abstract Model (UML)
Modeling
Cloud Scenario
Refined UML Model
Data-Exchange Format (XML; JSON)
Refined Modeling
Map Engine
Parser Code
Implementation (Java)
Cloud Simulator Input Model (CloudSim)
Operation (CloudSim)

**FIGURE 11** Generic input template for cloud simulator transformation approach. GMF, graphic modeling framework; JSON, JavaScript Object Notation; UML, Unified Modeling Language; XML, Extensible Markup Language

## 5.1 | GITS implementation in cloud

Cloud computing is one of the enablers of network function virtualization (NFV) technologies. GITS can be used to represent the NFV platform, where it can define the different components of the platform and represent different relationships among NFV components, ie, virtual network function components, to form the service function chain. To support the different cloud-enabled emerging platforms, GITS is designed to support different tools, including Java and multiple autoparsing methods. GITS can then search for dependency between the components of the underlying scenario and then map the JSON to the simulator input using different libraries, Apache XML and Jackson, to represent the application structure. Given this generation process of the simulator input, GITS can support any type of cloud-enabled platform including NFV.

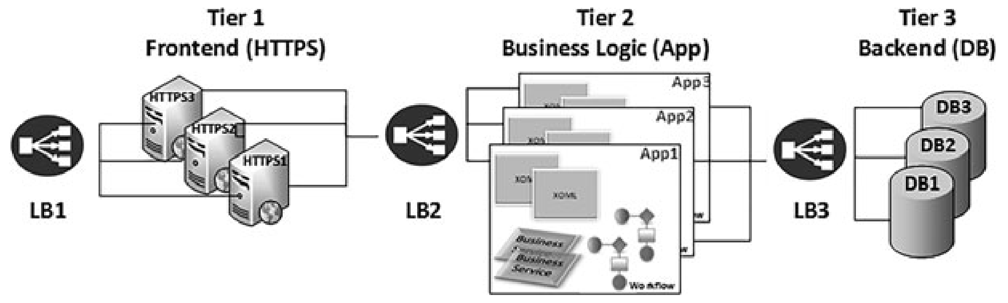


**FIGURE 12** Three-tier web application. DB, database; HTTPS, Hypertext Transfer Protocol Secure

## 5.2 | GITS as a case study of CloudSim

GITS is designed to facilitate the process of generating and evaluating different cloud scenarios. Its objective tends to minimize the implementation complexity and give the user the freedom to focus on designing algorithms to enhance and address different cloud challenges, such as scalability, availability, security, and other SLA requirements. With the era of Big Data, IoT, and NFV, the cloud is considered the enabler of such technologies, and thus, the automation of any configuration in the cloud is a necessity.

In order to show the GITS interface, this section provides an example of using GITS to create a three-tier web application. The Amazon web application can be an example.[99] The web application consists of an active Hypertext Transfer Protocol Secure server, App logic, and a DB, as shown in Figure 12. Each of these component types is backed up with redundant component(s). The functional and protection chains between different component types are also captured as links between different nodes. The MTTF, MTTR, and recovery time are the HA measures of application components (VMs), failure, and recovery times.

Note that the availability $A_C$ of an application's component $C$ is calculated as follows,[1] where 8760 is the number of hours per year:

$$A_C = \left( \frac{8760 - downtime_C}{8760} \right) \times 100. \tag{1}$$

GITS can then be used to facilitate the creation and evaluation of different HA-aware policies for applications' deployments in the cloud as the following scenarios.

- Assess different HA-aware placement techniques.
- Execute different HA policies to examine the resiliency of a given cloud model.
- Support multiple HA-aware analyses of cloud applications' deployment approaches. With the data defined in GITS configurations, this analysis will not only detect failures and execute their corresponding recovery solutions but also determine the availability of a cloud system under different stochastic incidents (failure, overload, etc).
- Assess the impact of redundancy models and failure injection times to extract HA-aware lessons.

GITS is implemented in Eclipse on a Linux VM running Ubuntu 12.04. In order to generate cloud scenarios in GITS, the user should define either the JSON template or the GMF project. The results are evaluated on a network of two DCs, two racks, and six servers. For evaluation purposes, each DC has a rack with two shelves and three servers. Following the cumulative distribution function generated in the work of Garraghan et al,[100] the repair times of the DCs/racks/servers can be generated using a lognormal distribution, ranging from 15 seconds to 4 days. Similarly, the failure times of the DCs/racks/servers can be generated using a Weibull distribution, ranging from 576 to 696 hours.[100] As for the cloud application's components, the repair and failure times follow lognormal and Weibull distributions, ranging from 25 minutes to 8 days for repairing and from 380 to 450 hours for the failures, respectively.[100-102] Table 1 shows the GITS resources' configuration, where the VMs are configured in different instances (small, medium, large, etc).[103]

The GMF is designed using the Eclipse modeling tool, Kepler version. In the following, we create cloud scenarios using GITS, test it in CloudSim, and discuss the steps to map GITS to other cloud templates. Figure 13 shows a sample of the cloud scenario generated using GITS GMF.

1. ***Cloud scenario creation***

   The GMF project is run as a Java application to create a cloud example, but the user can populate the JSON template directly as well. The cloud infrastructure consists of two DCs, each characterized by computational resources and

**TABLE 1** Computing metrics of the generic input template for cloud simulator evaluation

| Scenario Metrics | Server | VM |
|---|---|---|
| CPU (cores) | 16-32 cores | 1-2 core(s) |
| RAM | 25-35 GB | 256-1024 MB |

Abbreviations: CPU, central processing unit; VM, virtual machine.

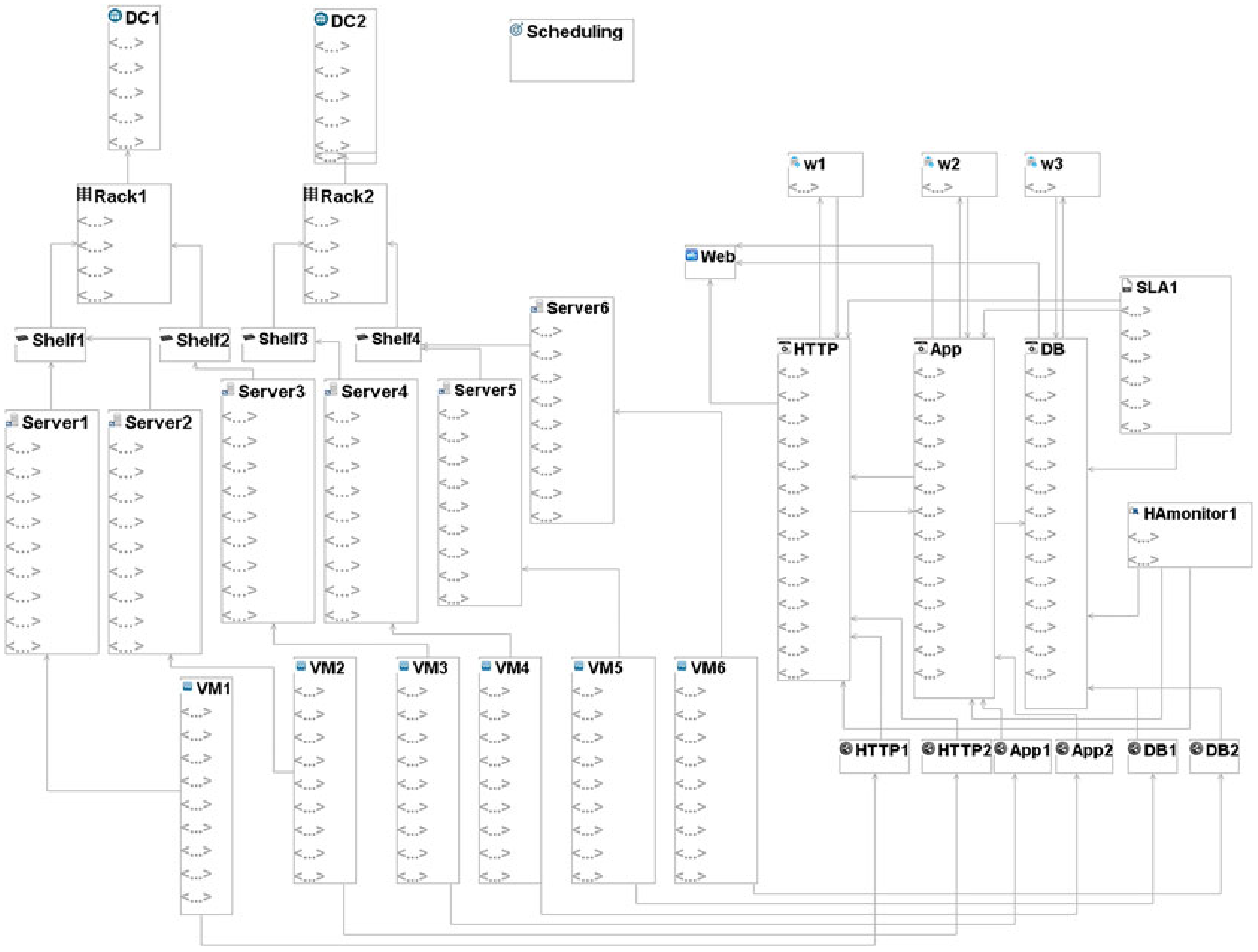
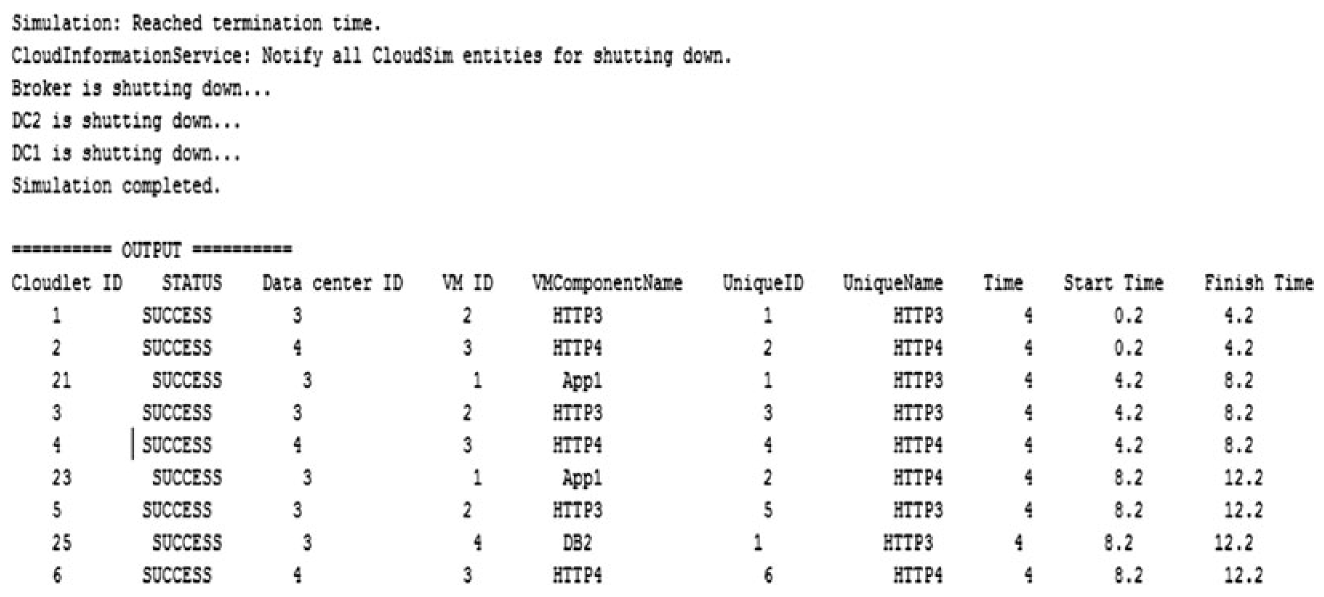


**FIGURE 13** Generic input template for cloud simulator cloud scenario. DC, data center; DB, database; HTTP, Hypertext Transfer Protocol; VM, virtual machine [Colour figure can be viewed at wileyonlinelibrary.com]

availability metrics (MTTF in hours per year and MTTR in seconds). The availability zone is “Z1,” which means that all the servers of this DC are located in the availability zone “Z1.” The *DC_count* is one indicating that only one DC with specified attributes is created. Similarly, the rack is populated with resources and HA features, and each rack should determine its DC. Each of these racks has two shelves, and each shelf has three servers. The server has CPU (cores), RAM (MB), and storage (GB).

In this example, a VM is used to represent the virtual mapping between the cloud infrastructure and the cloud applications. Since the objective of this scenario is “scheduling,” the hosting server and the hosted component of the VM are populated as “Null.” These properties are populated after triggering an allocation policy of the cloud applications.

The cloud application consists of the HTTPS component at the frontend, App logic, and DB at the backend. The App server sponsors the HTTPS, and consequently, the HTTPS component type has the *DependsON* property as “App” and *DepParam* are populated with the corresponding tolerance time and delay tolerance. The same applies to the App component type. Since DB does not have sponsors, its *DependsON* and *DepParam* are “Null.” Each of these types has a



```
Simulation: Reached termination time.
CloudInformationService: Notify all CloudSim entities for shutting down.
Broker is shutting down...
DC2 is shutting down...
DC1 is shutting down...
Simulation completed.

========== OUTPUT ==========
Cloudlet ID   STATUS    Data center ID   VM ID   VMComponentName   UniqueID   UniqueName   Time   Start Time   Finish Time
    1        SUCCESS        3              2        HTTP3              1          HTTP3       4        0.2         4.2
    2        SUCCESS        4              3        HTTP4              2          HTTP4       4        0.2         4.2
    21        SUCCESS        3              1        App1              1          HTTP3       4        4.2         8.2
    3        SUCCESS        3              2        HTTP3              3          HTTP3       4        4.2         8.2
    4       |SUCCESS        4              3        HTTP4              4          HTTP4       4        4.2         8.2
    23        SUCCESS        3              1        App1              2          HTTP4       4        8.2         12.2
    5        SUCCESS        3              2        HTTP3              5          HTTP3       4        8.2         12.2
    25        SUCCESS        3              4        DB2              1         HTTP3       4       8.2        12.2
    6        SUCCESS        4              3        HTTP4              6          HTTP4       4        8.2         12.2
```

**FIGURE 14** Generic input template for cloud simulator scenario in CloudSim

redundancy model to back them up upon failure. The redundancy relation is described using *RedundancyModel*, *RedParam*, *CompType_instances*, and *names*. Each component type has its own workload characteristics. It is monitored by an HA middleware and follows an SLA.

If the user describes the cloud scenarios using GMF, the transformation algorithm, which consists of GMF2JSON, JSON2UML, and GITS2CloudSimInput, is triggered to automate CloudSim population. If the user describes the cloud scenarios using the JSON template, the transformation algorithm, which consists of JSON2UML and GITS2CloudSimInput, automates the CloudSim population.

The described cloud example is tested in CloudSim. GITS does not only simplify scenario creation and model repeatability, it also captures HA properties, including redundancy models and HA metrics. We have extended CloudSim to include these HA features and an HA-aware allocation policy.[104-106] In the extended CloudSim, application components, their dependents, and redundants are modeled. Therefore, the GITS template is evaluated using the extended CloudSim since it supports the application and HA modeling. Figure 14 shows the successful completion of simulation using the above GITS scenario. As seen in Figure 14, each component is hosted on a VM. The extended CloudSim supports the functional chaining; therefore, multiple cloudlets can be created on a VM. The finish time of the processed cloudlet is the start time of the waiting cloudlet in the queue. As the simulation time increases, more cloudlets (requests) are created to model the requests being processed by the application components.

The CloudSim input model is graphically and textually designed as a readable and reusable scenario. CloudSim users do not require experience in the simulator environment and can focus on the simulator features and scheduling extensibility to design solutions that overcome other cloud challenges.

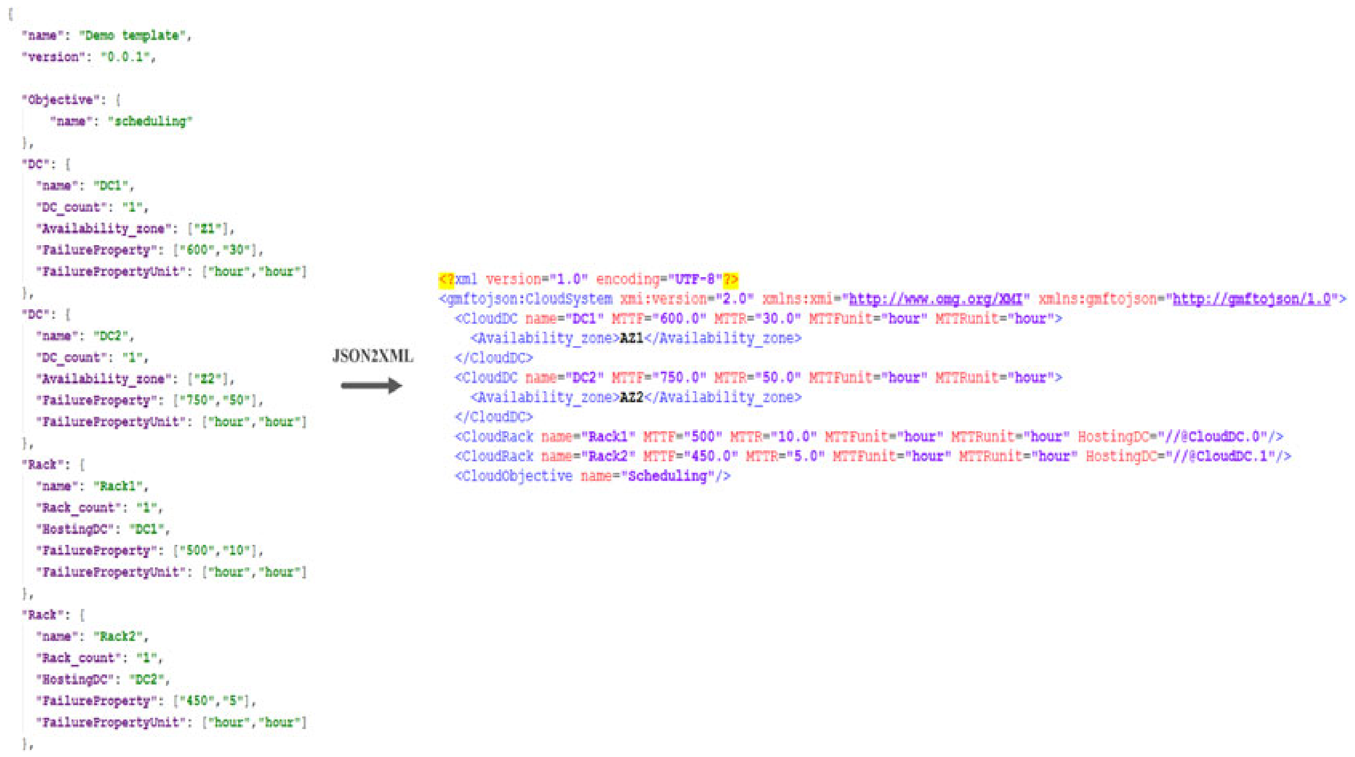
2. ***GITS encoding***

GITS uses JSON as the data exchange schema to define a cloud model. However, this schema can be easily mapped to another encoding format, such as XML and YAML files. Figure 15 shows the JSON2XML translation. This ensures the ability to use this template not only for CloudSim input, but it can also be adapted to other cloud providers, simulator, and cloud management systems, such as OpenStack Heat. Heat is an orchestration service for OpenStack that uses a template mechanism and controls cloud resource groups.

In order to translate GITS to other cloud templates (OpenStack HOT), some key points should be considered.

It is not necessary to use GITS as the base Heat data exchange scheme because the proposed template of Heat can be translated to/from GITS.

- The GITS template can be reshaped to meet the standards of HOT. For example, when assigning server resources, a mapping can be generated between resource number and resource description in HOT (tiny, small, medium, and large instances).
- Multiple JSON-YAML parsers can be adopted in the GITS-HOT translator.



```
[
  "name": "Demo template",
  "version": "0.0.1",

  "Objective": {
     "name": "scheduling"
  },
  "DC": [
   "name": "DC1",
   "DC_count": "1",
   "Availability_zone": ["Z1"],
   "FailureProperty": ["600","30"],
   "FailurePropertyUnit": ["hour","hour"]
  },
  "DC": [
   "name": "DC2",
   "DC_count": "1",
   "Availability_zone": ["Z2"],
   "FailureProperty": ["750","50"],
   "FailurePropertyUnit": ["hour","hour"]
  },
  "Rack": [
   "name": "Rack1",
   "Rack_count": "1",
   "HostingDC": "DC1",
   "FailureProperty": ["500","10"],
   "FailurePropertyUnit": ["hour","hour"]
  },
  "Rack": [
   "name": "Rack2",
   "Rack_count": "1",
   "HostingDC": "DC2",
   "FailureProperty": ["450","5"],
   "FailurePropertyUnit": ["hour","hour"]
  },
```

```
<?xml version="1.0" encoding="UTF-8"?>
<gmftojson:CloudSystem xmi:version="2.0" xmlns:xmi="http://www.omg.org/XMI" xmlns:gmftojson="http://gmftojson/1.0">
  <CloudDC name="DC1" MTTF="600.0" MTTR="30.0" MTTFunit="hour" MTTRunit="hour">
    <Availability_zone>AZ1</Availability_zone>
  </CloudDC>
  <CloudDC name="DC2" MTTF="750.0" MTTR="50.0" MTTFunit="hour" MTTRunit="hour">
    <Availability_zone>AZ2</Availability_zone>
  </CloudDC>
  <CloudRack name="Rack1" MTTF="500" MTTR="10.0" MTTFunit="hour" MTTRunit="hour" HostingDC="//@CloudDC.0"/>
  <CloudRack name="Rack2" MTTF="450.0" MTTR="5.0" MTTFunit="hour" MTTRunit="hour" HostingDC="//@CloudDC.1"/>
  <CloudObjective name="Scheduling"/>
```

**FIGURE 15** Example of generic input template for cloud simulator encoding to an Extensible Markup Language schema [Colour figure can be viewed at wileyonlinelibrary.com]

- It is also necessary to determine the relation between stack and OpenStack resources because each module (Nova, Cinder, and Compute) requires different properties defined in the stack parameter section.

The encoding method of a cloud model can be easily modified to meet certain cloud systems. As long as the template captures the properties needed to manage cloud infrastructure and applications, the translation method can be straightforwardly implemented.

## 5.3 | GITS performance

GITS can model any multitier application to be executed in the cloud environment. It can reflect the different relationships between the applications' components while minimizing the mapping complexities. GITS is generated as a library with size of approximately 3 MB. This file can be added to the building environment of any cloud-based simulator. Given the small size of GITS, it does not only save on the memory of any solution but also facilitates the application autoscaling, adding dependencies between components, and executing different redundancy models and other HA-aware policies. Additionally, GITS can be used with different service registrars, including Kubernetes and Netflix Eureka. Given this 3-MB library file, GITS can represent more than 10 000 records of servers and application components. In the proposed template, a user can either use the "count" variable to generate a set of servers, DCs, racks, shelves, and application components of the same characteristics or include a recursive function to generate the infrastructure and application of different types/resources.

In terms of the transformation algorithm, it consists of different parsing modules. Using Eclipse memory analysis, Figure 16 shows the memory consumed when executing the GITS transformation algorithm to generate its JAR library that can be used in CloudSim. The algorithm uses Hashmap because it has a high load factor. In other words, the higher is the load factor, the less memory space is consumed. This characteristic aims at supporting the scalability requirement of the cloud environment, especially with the era of Big Data and IoT where any node or subnode of the cloud model can be accessed without the need to an iterative script.

Table 2 shows the need for GITS to facilitate the creation of input models for CloudSim. Different attributes are compared for creating cloud scenarios of CloudSim with and without GITS. The comparison is done according to the middleware used to model the desired scenarios, the programming abstraction, model generality, scalability, repeatability,

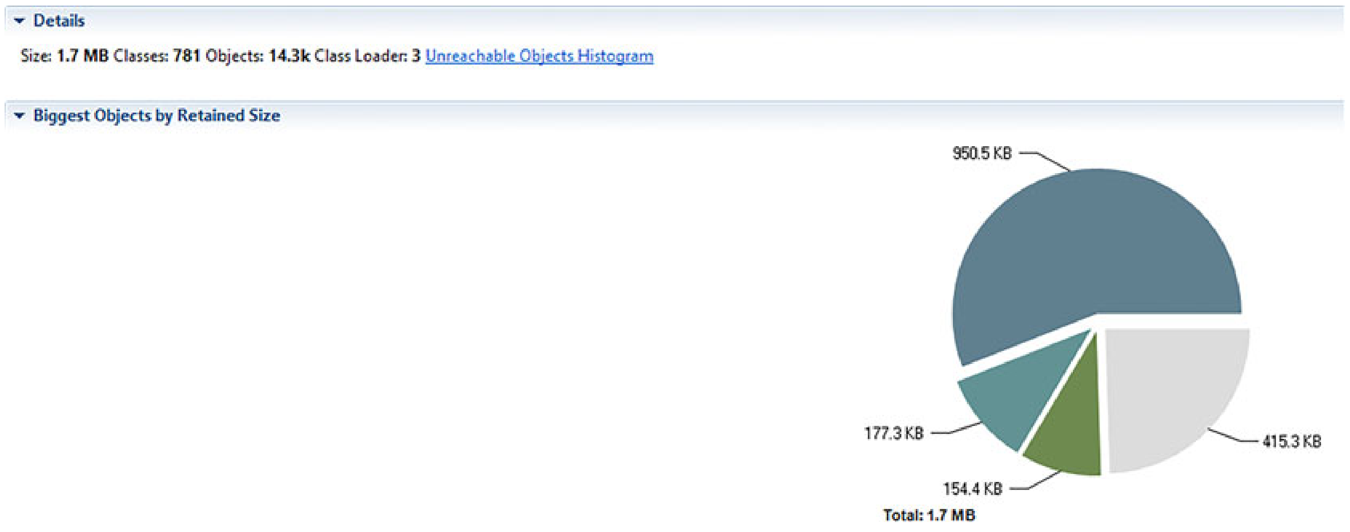

Details
Size: 1.7 MB Classes: 781 Objects: 14.3k Class Loader: 3 Unreachable Objects Histogram
Biggest Objects by Retained Size
950.5 KB
177.3 KB
154.4 KB
415.3 KB
Total: 1.7 MB

**FIGURE 16** Availability of each deployed component using the generic input template for cloud simulators [Colour figure can be viewed at wileyonlinelibrary.com]

**TABLE 2** Generic input template for cloud simulators (GITS) performance

| **Descriptor** | **CloudSim Input Model** | |
|---|---|---|
| | **With GITS** | **Without GITS** |
| **Underlying Technology** | Java | Java |
| **Programming Abstraction** | Supported | Not Supported |
| **Syntax** | Abstract, concrete, and serialization | None |
| **Modeling** | Graphical and textual | Hard-coded |
| **Modeling Generality** | Supported | Not Supported |
| **Dynamic Modeling** | Supported | Not Supported |
| **Static Modeling** | Supported | Supported |
| **Implementation Complexity** | Soft-coded | Hard-coded |
| **Scenario Reusability** | Supported | Not Supported |
| **Scenario Repeatability** | Supported | Not Supported |
| **Scalability** | Supported | Supported |
| **Structure** | Application's Component Deployment | VM Deployment |
| **Cloud Service Target** | XaaS (Everything as a Service; supports availability policies simulation) | IaaS (Infrastructure as a Service) |

Abbreviation: VM, virtual machine.

implementation complexity, and support of cloud services. In terms of the programming abstraction, the user can generate the input template using JSON and add it with the GITS JAR library to a CloudSim environment; the corresponding scheduling or allocation algorithms would run them without adding any changes to the CloudSim environment. It is necessary to note that the developer/user does not need to do any changes to the transformation algorithm in GITS as it is executed in the JAR file; the user/developer has only the task of defining the cloud scenario in JSON and saving this template to a CloudSim workplace. If the user needs to test availability in CloudSim, she/he should extend CloudSim with HA-aware placement. In terms of the modeling generality, GITS can model any kind of multitier cloud application with any infrastructure topology.

In terms of the implementation complexity, GITS can be easily used to create any cloud scenario using either its textual or graphical model while ensuring the scenarios' reusability and repeatability to evaluate or reassess any QoS policies. In terms of the scalability, the generality and dynamic characteristics of GITS allow the scaling up and down of cloud applications and infrastructure due to overload, failure, or sudden change in the cloud model. In terms of the support of cloud service, GITS can support the modeling of Everything as a Service and does not only evaluate different scheduling and deployment models but also assess HA policies and availability-aware placements of cloud applications.

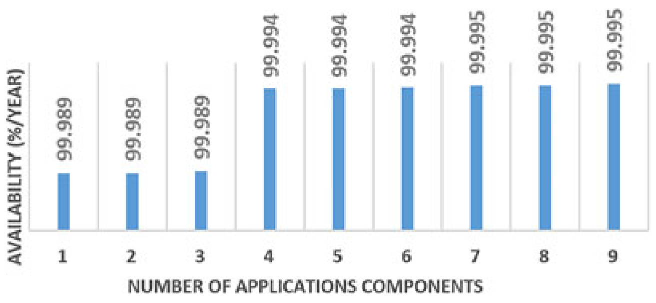


**FIGURE 17** Availability of each deployed component using the generic input template for cloud simulators [Colour figure can be viewed at wileyonlinelibrary.com]

To emphasize the need for GITS especially on HA-aware objectives, it is tested on our HA-aware extension of CloudSim, ACE.[107] ACE extends CloudSim with an HA-aware placement solution, failure injection, load balancing, and recovery/repair modules. The HA-aware placement algorithm generates deployments of the applications' components on the servers while maximizing their availability. Figure 17 shows the components' availability of the three-tier web application. As GITS has extended CloudSim specification with availability metrics and a hierarchical structure of the cloud infrastructure and applications, we are able to execute ACE and implement different HA-aware solutions.

## 6 | CONCLUSION AND FUTURE WORK

The design of a cloud template that provides simplicity, understandability, repeatability, and interoperability is a paramount step in cloud design to fully exploit its benefits. To this end, it is necessary to define a CBA that describes the cloud infrastructure and application parameters and enables application configurability between different cloud platforms. This architecture leverages the challenge of cloud scenario development, testing, and maintenance. Therefore, in this paper, we proposed GITS to enable the above features and reduce the complexity of understanding different cloud technologies. GITS provides a standard interface and tool that facilitates the cloud management solutions (performance and deployment). It supports availability consistency and templating that can be used with any model to ensure compliance with the HA requirements and cloud structure. GITS allows microsegmentation that improves the HA modeling through its granular and generic design, where applications can be easily scaled up or down given the cloud profile. GITS also offers portability of workload, thus providing the user with the freedom to run a cloud application in the appropriate cloud profile and minimizing the impact of the vendor lock-in issue. Lastly, GITS is mapped to CloudSim using a transformation algorithm, but it can be easily translated to fit any cloud simulator or cloud management input. In the future work, GITS will be extended to include a reporting tool to allow the users to collect, consolidate, analyze, and report availability results, performance trends, application's deployments, and utilization measures.

## ACKNOWLEDGEMENT

This work was partially supported by the Natural Sciences and Engineering Research Council of Canada (NSERC-STPGP 447230) and Ericsson Research.

## ORCID

*Manar Jammal* https://orcid.org/0000-0002-4833-7644

## REFERENCES

1. Jammal M, Kanso A, Shami A. High availability-aware optimization digest for applications deployment in cloud. Paper presented at: 2015 IEEE International Conference on Communications (ICC); 2015; London, UK.
2. Jammal M, Kanso A, Shami A. CHASE: component high-availability scheduler in cloud computing environment. Paper presented at: IEEE International Conference on Cloud Computing (CLOUD). 2015; New York, NY.
3. SDx Central. 2018 cloud orchestration and DevOps report - download. 2018. https://www.sdxcentral.com/reports/cloud-orchestration-devops-download-2018/. Accessed September 17, 2018.
4. Kratzke N. A brief history of cloud application architectures. *Appl Sci*. 2018;8(8):1368.

5. Dillon T, Wu C, Chang E. Cloud computing: issues and challenges. Paper presented at: 24th IEEE International Conference on Advanced Information Networking and Applications; 2010; Perth, Australia.
6. Wei Y, Blake MB. Service-oriented computing and cloud computing: challenges and opportunities. *IEEE Internet Comput*. 2010;14(6):72-75.
7. Baryannis G, Garefalakis P, Kritikos K, et al. Lifecycle management of service-based applications on multi-clouds: a research roadmap. In: Proceedings of the International Workshop on Multi-Cloud Applications and Federated Clouds (MultiCloud); 2013; Prague, Czech Republic.
8. Juhnke E, Dornemann T, Bock D, Freisleben B. Multi-objective scheduling of BPEL workflows in geographically distributed clouds. Paper presented at: IEEE 4th International Conference on Cloud Computing; 2011; Washington, DC.
9. Liu C, Van der Merwe JE, Mao Y, Fernández MF. Cloud resource orchestration: a data-centric approach. Paper presented at: 5th Biennial Conference on Innovative Data Systems Research (CIDR); 2011; Asilomar, CA.
10. Wang X, Liu Z, Qi Y, Li J. LiveCloud: a lucid orchestrator for cloud datacenters. Paper presented at: 4th IEEE International Conference on Cloud Computing Technology and Science Proceedings; 2012; Taipei, Taiwan.
11. Carella GA. *An Extensible and Customizable Framework for the Management and Orchestration of Emerging Software-based Networks* [thesis]. Berlin, Germany: Berlin Institute of Technology; 2018.
12. Yusoh ZIM, Tang M. Composite SaaS placement and resource optimization in cloud computing using evolutionary algorithms. Paper presented at: IEEE Fifth International Conference on Cloud Computing; 2012; Honolulu, HI.
13. Hawilo H, Kanso A, Shami A. Towards an elasticity framework for legacy highly available applications in the cloud. Paper presented at: IEEE World Congress on Services (SERVICES); 2015; New York, NY.
14. Ayoubi S, Zhang Y, Assi C. A reliable embedding framework for elastic virtualized services in the cloud. *IEEE Trans Netw Serv Manag*. 2016;13(3):489-503.
15. Hawilo H, Shami A, Mirahmadi M, Asal R. NFV: state of the art, challenges, and implementation in next generation mobile networks (vEPC). *IEEE Netw*. 2014;28(6):18-26.
16. Ayoubi S, Chen Y, Assi C. Towards promoting backup-sharing in survivable virtual network design. *IEEE/ACM Trans Networking*. 2016;24(5):3218-3231.
17. Boteanu A, Dobre C. A simulation model for fault tolerance evaluation. *UPB Sci Bull Series C*. 2011;73(1):13-26.
18. Calheiros RN, Ranjan R, Beloglazov A, De Rose CAF, Buyya R. CloudSim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Softw: Pract Exper J*. 2011;41(1):23-50.
19. Buyya R, Ranjan R, Calheiros RN. Modeling and simulation of scalable cloud computing environments and the CloudSim toolkit: challenges and opportunities. Paper presented at: International Conference on High Performance Computing & Simulation; 2009; Leipzig, Germany.
20. Strauch S, Andrikopoulos V, Bachmann T, Leymann F. Migrating application data to the cloud using cloud data patterns. In: *Proceedings of the 3rd International Conference on Cloud Computing and Services Science - Volume 1: CLOSER*. Setúbal, Portugal: Science and Technology Publications; 2013:36-46.
21. OASIS. Topology and orchestration specification for cloud applications. Version 1.0. 2013. http://docs.oasis-open.org/tosca/TOSCA/v1.0/TOSCA-v1.0.html. Accessed December 2016.
22. Antonescu A-F, Robinson P, Braun T. Dynamic topology orchestration for distributed cloud-based applications. Paper presented at: Second Symposium on Network Cloud Computing and Applications (NCCA); 2012; London, UK.
23. Juve G, Deelman E. Automating application deployment in infrastructure clouds. Paper presented at: 2011 IEEE Third International Conference on Cloud Computing Technology and Science (CloudCom); 2011; Athens, Greece.
24. Daur H. IBM SmartCloud orchestrator: architected for extensibility. IBM; 2013. http://www.iaas.uni-stuttgart.de/lehre/vorlesung/2013_ws/vorlesungen/smcc/materialien/SCOrchestrator%20Extensibility%20Architecture%201105.pdf. Accessed December 2016.
25. IBM. *Orchestration Simplifies and Streamlines Virtual and Cloud Data Centre Management*. White paper. ITProPortal; 2015. https://goo.gl/8dEqle. Accessed December 2016.
26. Zhu J, Zheng Z, Zhou Y, Lyu MR. Scaling service-oriented applications into geo-distributed clouds. Paper presented at: 2013 IEEE Seventh International Symposium on Service-Oriented System Engineering; 2013; San Francisco, CA.
27. Gutierrez-Garcia JO, Sim K. Self-organizing agents for service composition in cloud computing. Paper presented at: IEEE Second International Conference on Cloud Computing Technology and Science; 2010; Indianapolis, IN.
28. Zeng C, Guo X, Ou W, Han D. Cloud computing service composition and search based on semantic. Paper presented at: IEEE International Conference on Cloud Computing; 2009; Beijing, China.
29. Amazon. AWS CloudFormation templates. 2017. https://aws.amazon.com/cloudformation/aws-cloudformation-templates/. Accessed February 2017.
30. Heat Orchestration Template (HOT) Guide. 2013. http://docs.openstack.org/developer/heat/template_guide/hot_guide.html. February 2013.
31. Binz T, Breiter G, Leymann F, Spatzier T. Portable cloud services using TOSCA. *IEEE Internet Comput*. 2012;16(3):80-85.
32. Carlson M, Chapman M, Heneveld A, et al. Cloud application management for platforms. 2012. https://www.oasis-open.org/committees/download.php/47278/CAMP-v1.0.pdf.
33. OpenTOSCA. Open Source TOSCA Ecosystem. 2016. http://www.iaas.uni-stuttgart.de/OpenTOSCA/. Accessed February 2017.
34. OpenTOSCA. Winery tool. 2013. http://stable.winery.opentosca.org/servicetemplates/. Accessed January 2017.

35. Kopp O, Binz T, Breitenbücher U, Leymann F. Winery—A modeling tool for TOSCA-based cloud applications. Paper presented at: 11th International Conference on Service-Oriented Computing; 2013; Berlin, Germany.
36. Eclipse. Eclipse Winery. 2017. https://projects.eclipse.org/projects/soa.winery. Accessed February 2017.
37. Altevogt P, Denzel W, Kiss T. Cloud Modeling and Simulations. IBM research report. Rüschlikon, Switzerland: IBM Research - Zurich; 2013.
38. Armbrust M, Fox A, Griffith R, et al. *Above the Clouds: A Berkeley View of Cloud Computing*. Technical report. Berkeley, CA: University of California, Berkeley; February 2009.
39. Tian W, Zhao Y, Xu M, Zhong Y, Sun X. A toolkit for modeling and simulation of real-time virtual machine allocation in a cloud data center. *IEEE Trans Autom Sci Eng*. 2015;12(1):153-161.
40. Filho MCS, Rodrigues JJPC. Human readable scenario specification for automated creation of simulations on CloudSim. In: *Internet of Vehicles - Technologies and Services: First International Conference, IOV, Beijing, China, September 1-3, 2014. Proceedings*. Cham, Switzerland: Springer International Publishing; 2014;345-356.
41. Wickremasinghe B, Calheiros RN, Buyya R. CloudAnalyst: a CloudSim-based visual modeller for analysing cloud computing environments and applications. Paper presented at: 24th IEEE International Conference on Advanced Information Networking and Applications; 2010; Perth, Australia.
42. Jakovits P, Srirama SN, Kromonov I. Stratus: a distributed computing framework for scientific simulations on the cloud. Paper presented at: IEEE 14th International Conference on High Performance Computing and Communication & IEEE 9th International Conference on Embedded Software and Systems; 2012; Liverpool, UK.
43. Srirama S, Batrashev O, Vainikko E. SciCloud: scientific computing on the cloud. Paper presented at: 10th IEEE/ACM International Conference on Cluster, Cloud and Grid Computing; 2010; Melbourne, Australia.
44. Guo S, Bai F, Hu X. Simulation software as a service and service-oriented simulation experiment. Paper presented at: IEEE International Conference on Information Reuse & Integration; 2011; Las Vegas, NV.
45. Balmer M, Rieser M, Meister K, Charypar D, Lefebvre N, Nagel K. MATSIm-T: Architecture and Simulation Times. In: *Multi-Agent Systems for Traffic and Transportation Engineering*. Hershey, PA: IGI Global; 2009:57-78.
46. Behrisch M, Bieker L, Erdmann J, Krajzewicz D. SUMO-simulation of urban mobility: an overview. Paper presented at: SIMUL 2011: The Third International Conference on Advances in System Simulation; 2011; Barcelona, Spain.
47. Citron D, Zlotnick A. Testing large-scale cloud management. *IBM J Res Dev*. 2011;55(6):6:1-6:10.
48. Calheiros RN, Netto MAS, De Rose CAF, Buyya R. EMUSIM: an integrated emulation and simulation environment for modeling, evaluation, and validation of performance of Cloud computing applications. *Softw: Pract Exper*. 2012;43(5):595-612.
49. Bux M, Leser U. DynamicCloudSim: simulating heterogeneity in computational clouds. In: Proceedings of the 2nd ACM SIGMOD Workshop on Scalable Workflow Execution Engines and Technologies (SWEET); 2013; New York, NY.
50. Fakhfakh F, Kacem HH, Kacem AH. CloudSim4DWf: a CloudSim-extension for simulating dynamic workflows in a cloud environment. Paper presented at: IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA); 2017; London, UK.
51. De Boer FS, Hähnle R, Johnsen EB, Schlatte R, Wong PYH. Formal modeling of resource management for cloud architectures: an industrial case study. In: Proceedings of European Conference on Service-Oriented and Cloud Computing (ESOCC); 2012; Bertinoro, Italy.
52. Zeng X, Garg SK, Strazdins P, Jayaraman PP, Georgakopoulos D, Ranjan R. IOTSIm: a simulator for analysing IoT applications. *J Syst Archit*. 2016;72:93-107.
53. Piraghaj SF, Dastjerdi AV, Calheiros RN, Buyya R. ContainerCloudSim: an environment for modeling and simulation of containers in cloud data centers. *Softw: Pract Exper*. 2016;47(4):505-521.
54. Kratzke N, Quint P-C, Palme D, Reimers D. Project cloud TRANSIT - or to simplify cloud-native application provisioning for SMEs by Integrating already available container technologies. In: *European Project Space on Smart Systems, Big Data, Future Internet-Towards Serving the Grand Societal Challenges*. SciTePress; 2017.
55. IDABC. European interoperability framework for pan-European eGovernment services. European Communities; 2004. http://ec.europa.eu/idabc/servlets/Docd552.pdf?id=19529
56. Pritzker P, Gallagher PD. NIST cloud computing standards roadmap. National Institute of Standards and Technology; 2013. https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-291r2.pdf
57. UK Government Cabinet Office. Open Standards principles. Policy paper. 2018. https://www.gov.uk/government/publications/open-standards-principles/open-standards-principles
58. Koski K, Hormia-Poutanen K, Chatzopoulos M, Legré Y, Day B. European open science cloud for research. Position paper. EUDAT; 2015. https://eudat.eu/position-paper-european-open-science-cloud-for-research
59. Lipton P, Lauwers C, Tamburri D. OASIS topology and orchestration specification for cloud applications (TOSCA) TC. OASIS; 2018. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=tosca
60. Garcia AL, Fernández del Castillo E. Analysis of scientific cloud computing requirements. Paper presented at: 7th Iberian Grid Infrastructure Conference (IBERGRID); 2013; Madrid, Spain.
61. Garcia AL, Fernández del Castillo E, Fernández PO. Standards for enabling heterogeneous IaaS cloud federations. *Comput Stand Interfaces*. 2016;47:19-23.
62. Teckelmann R, Reich C, Sulistio A. Mapping of cloud standards to the taxonomy of interoperability in IaaS. Paper presented at: IEEE Third International Conference on Cloud Computing Technology and Science; 2011; Athens, Greece.

63. Herbert N. The state of orchestration: 2017. Market report. 2017. http://www.fujitsu.com/uz/Images/21637_Orchestration_Research_Report_v4_ONLINE.pdf. Accessed August 23, 2018.
64. Caballer M, Zala S, López García Á, Moltó G, Fernández PO, Velten M. Orchestrating complex application architectures in heterogeneous clouds. *J Grid Comput*. 2018;16(1):3-18.
65. Peltz C. Web services orchestration and choreography. *Computer*. 2003;36(10):46-52.
66. ITPP. *Orchestration Simplifies and Streamlines Virtual and Cloud Data Centre Management*. White paper. 2014. https://www.itproportal.com/2014/06/19/whitepaper-orchestration-simplifies-and-streamlines-virtual-and-cloud-data-centre-management/. Accessed May 2018.
67. Ranjan R, Benatallah B. Programming cloud resource orchestration framework: operations and research challenges. *Semantic Scholar*. 2012.
68. Wang X. *Concept and Implementation of a Graphical Editor for Composite Application Templates* [thesis]. Stuttgart, Germany: University of Stuttgart; 2010. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.465.1373&rep=rep1&type=pdf
69. Google. G Suite. 2017. https://gsuite.google.com/. Accessed February 2017.
70. Salesforce. Bring your CRM to the future. 2016. https://www.salesforce.com/crm/. Accessed February 2017.
71. Salehi P, Hamoud-Lhadj A, Colombo P, Khendek F, Toeroe M. A UML-based domain specific modeling language for the availability management framework. Paper presented at: IEEE 12th International Symposium on High Assurance Systems Engineering; 2010; San Jose, CA.
72. Santos GL, Endo PT, Gonçalves G, et al. Analyzing the IT subsystem failure impact on availability of cloud services. Paper presented at: IEEE Symposium on Computers and Communications (ISCC); 2017; Heraklion, Greece.
73. Torres E, Callou G, Andrade E. A hierarchical approach for availability and performance analysis of private cloud storage services. *Computing*. 2018;100(6):621-644.
74. Fernandes S, Santos M. SDN dependability: assessment, techniques, and tools. Paper presented at: IETF 93; 2015; Prague, Czech Republic. https://datatracker.ietf.org/meeting/93/materials/slides-93-sdnrg-4
75. Snow A. Tutorial: statistical methods for system dependability: reliability, availability, maintainability and resiliency. Paper presented at: The International Symposium on Advances in Software Defined Networking and Network Functions Virtualization; 2017; Venice, Italy. https://www.iaria.org/conferences2017/filesCTRQ17/AndySnow_NexComm2017_Tutorial.pdf
76. APC. Availability and reliability theory. 2003. http://www.fazekas-andras-istvan.hu/9_12_bme/DFAI_RELIABILITY_01.pdf
77. Baeldung. Introduction to Spring Cloud Netflix - Eureka. 2011. http://www.baeldung.com/spring-cloud-netflix-eureka. Accessed April 19, 2018.
78. Kubernetes. Production-grade container orchestration. 2018. https://kubernetes.io/. Accessed May 3, 2018.
79. DZone. Microservices communication: Zuul API gateway. 2017. https://dzone.com/articles/microservices-communication-zuul-api-gateway-1. Accessed February 10, 2018.
80. Spring-Pivotal. Client side load balancing with ribbon and spring cloud. 2018. https://spring.io/guides/gs/client-side-load-balancing/. Accessed April 21, 2018.
81. Jayasinghe D, Pu C, Eilam T, Steinder M, Whally I, Snible E. Improving performance and availability of services hosted on IaaS clouds with structural constraint-aware virtual machine placement. Paper presented at: IEEE International Conference on Services Computing (SCC); 2011; Washington, DC.
82. Dean J, Barroso LA. The tail at scale. *Commun ACM*. 2013;56(2):74-80.
83. Tech Crunch. Amazon AWS S3 outage is breaking things for a lot of websites and apps. 2017. https://techcrunch.com/2017/02/28/amazon-aws-s3-outage-is-breaking-things-for-a-lot-of-websites-and-apps/. Accessed March 4, 2018.
84. Ponemon Institute. Cost of data center outages. 2016. http://files.server-rack-online.com/2016-Cost-of-Data-Center-Outages.pdf. Accessed September 9, 2016.
85. The Wall Street Journal. Internet of Things market to reach $1.7 trillion by 2020: IDC. 2015. http://blogs.wsj.com/cio/2015/06/02/internet-of-things-market-to-reach-1-7-trillion-by-2020-idc/. Accessed July 15, 2018.
86. Subashini S, Kavitha V. A survey on security issues in service delivery models of cloud computing. *J Netw Comput Appl*. 2011;34(1):1-11.
87. Hillston J. SimJava. 2002. http://www.inf.ed.ac.uk/teaching/courses/ms/notes/note12.pdf
88. Truyen E, Vanhaute B, Joosen W, Verbaeten P, Jørgensen BN. Dynamic and selective combination of extensions in component-based applications. In: Proceedings of the 23rd International Conference on Software Engineering (ICSE); 2001; Toronto, Canada.
89. Oracle. JavaBeans Spec. 2017. http://www.oracle.com/technetwork/articles/javaee/spec-136004.html. Accessed February 2017.
90. CORBA. OMG Specifications. 2017. http://www.omg.org/spec/#MW. Accessed February 2017.
91. Microsoft. Distributed component object model. 2012. https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-2000-server/cc958799(v=technet.10). Accessed February 2017.
92. Petritsch H. *Service-Oriented Architecture (SOA) vs. Component Based Architecture*. White paper. Vienna, Austria: Vienna University of Technology. 2006. http://www.petritsch.co.at/download/SOA_vs_component_based.pdf
93. Eriksson M, Hallberg V. *Comparison Between JSON and YAML for Data Serialization* [bachelor thesis]. Stockholm, Sweden: School of Computer Science and Engineering, KTH Royal Institute of Technology; 2011. http://www.csc.kth.se/utbildning/kth/kurser/DD143X/dkand11/Group2Mads/victor.hallberg.malin.eriksson.report.pdf
94. Zyp K. dojox.json.ref. dojo; 2017. http://dojotoolkit.org/reference-guide/1.10/dojox/json/ref.html. Accessed February 2017.

95. Gronback RC. *Eclipse Modeling Project: A Domain-Specific Language Toolkit*. Longman, UK: Addison-Wesley; 2009. https://sisis.rz.htw-berlin.de/inh2009/12371395.pdf
96. Eclipse. Graphical Modeling Project (GMP). 2017. http://www.eclipse.org/modeling/gmp/. Accessed February 2017.
97. Plante F. Introducing the GMF Runtime. 2006. http://www.eclipse.org/articles/Article-Introducing-GMF/article.html
98. Peng D, Cao L, Xu W. Using JSON for data exchanging in web service applications. *J Comput Inf Syst*. 2011;7(16):5883-5890.
99. Amazon Web Services. AWS Template Format. 2010. https://s3-us-west-2.amazonaws.com/cloudformation-templates-us-west-2/AutoScalingMultiAZWithNotifications.template. Accessed March 2016.
100. Garraghan P, Townend P, Xu J. An empirical failure-analysis of a large-scale cloud computing environment. Paper presented at: IEEE 15th International Symposium on High-Assurance Systems Engineering; 2014; Miami Beach, FL.
101. Reliability HotWire. Availability and the different ways to calculate it. 2007. http://www.weibull.com/hotwire/issue79/relbasics79.htm. Accessed September 20, 2016.
102. EventHelix. System reliability and availability. 2014. http://www.eventhelix.com/RealtimeMantra/FaultHandling/system_reliability_availability.htm#.WKz9jVUrKUk. Accessed September 20, 2016.
103. Microsoft. How to: change the size of a Windows Azure virtual machine. 2013. https://msdn.microsoft.com/en-us/library/dn168976(v=nav.70).aspx. Accessed September 20, 2016.
104. Jammal M, Kanso A, Heidari P, Shami A. Availability analysis of cloud deployed applications. Paper presented at: IEEE International Conference on Cloud Engineering (IC2E); 2016; Berlin, Germany.
105. Jammal M, Kanso A, Heidari P, Shami A. A formal model for the availability analysis of cloud deployed multi-tiered applications. Paper presented at: IEEE International Conference on Cloud Engineering Workshop (IC2EW); 2016; Berlin, Germany.
106. Jammal M, Hawilo H, Kanso A, Shami A. Mitigating the risk of cloud services downtime using live migration and high availability-aware placement. Paper presented at: IEEE International Conference on Cloud Computing Technology and Science (CloudCom); 2016. Luxembourg.
107. Jammal M, Hawilo H, Kanso A, Shami A. ACE: availability-aware CloudSim extension. *IEEE Trans Netw Serv Manag*. 2018. To appear.

**How to cite this article:** Jammal M, Hawilo H, Kanso A, Shami A. Generic input template for cloud simulators: A case study of CloudSim. *Softw: Pract Exper*. 2019;49:720–747. https://doi.org/10.1002/spe.2674