Modeling, Simulation and Analysis for Software-as-Service in Cloud

by

Wu Li

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved February 2015 by the
Graduate Supervisory Committee:

Wei-Tek Tsai, Chair
Hessam Sarjoughian
Jieping Ye
Guoliang Xue

ARIZONA STATE UNIVERSITY

December 2015

ABSTRACT

Software-as-a-Service (SaaS) has received significant attention in recent years as major computer companies such as Google, Microsoft, Amazon, and Salesforce are adopting this new approach to develop software and systems. Cloud computing is a computing infrastructure to enable rapid delivery of computing resources as a utility in a dynamic, scalable, and virtualized manner. Computer Simulations are widely utilized to analyze the behaviors of software and test them before fully implementations. Simulation can further benefit SaaS application in a cost-effective way taking the advantages of cloud such as customizability, configurability and multi-tendency.

This research introduces Modeling, Simulation and Analysis for Software-as-Service in Cloud. The researches cover the following topics: service modeling, policy specification, code generation, dynamic simulation, timing, event and log analysis. Moreover, the framework integrates current advantages of cloud: configurability, Multi-Tenancy, scalability and recoverability.

The following chapters are provided in the architecture:

- Multi-Tenancy Simulation Software-as-a-Service.
- Policy Specification for MTA simulation environment.
- Model Driven PaaS Based SaaS modeling.
- Dynamic analysis and dynamic calibration for timing analysis.
- Event-driven Service-Oriented Simulation Framework.
- LTBD: A Triage Solution for SaaS.

This thesis would never been completed without the helps from my parents Jinke Li and Xiangying Chen, my grandparents, my labmates, all my committee members and all my friends at ASU computer science department.

I also cherished the time I spent at Arizona State University, it will be one of my most important memories of my life.

Thank you all, my friends. I will embrace the world now.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

Chapter 1

INTRODUCTION

Cloud computing has received significant attention recently as it is a new computing infrastructure to enable rapid delivery of computing resources as a utility in a dynamic, scalable, and virtualized manner. Public clouds are available from Amazon Palankar *et al.* (2008a), Google Google (2014), Microsoft Microsoft (2014), Salesforce Salesforce (2014). Private clouds, in which the cloud software is loaded locally, are available from VMware VMWare (2014), Eucalyptus Eucalyptus (2014), Citrix Citrix (2014), and thousands of vendors offer cloud solutions.

SaaS has become a new delivery model for cloud computing, and it has a four-level maturity model Chong and Carraro (2006) depending on their customization, multi-tenancy architecture (MTA), and scalability capabilities. Researchers have started to investigate the simulated cloud, such as Cloudsim Calheiros *et al.* (2011). A simulated cloud enables applications to run. Note that this paper will not address simulation of a cloud, but provide a new model for simulation as a service. Different from simulated cloud, this paper presents how a cloud can support simulations with unique features such as configuration and MTA.

Simulation in a cloud can take advantage of various resources including physical resources such as processors, cache and logical resources such as in-memory data stores, distributed file systems, and SaaS infrastructure in the cloud. In fact, simulation software can be structured in a SaaS manner in which every resource/software is treated as service, and thus SimSaaS (Simulation Software as a Service) Tsai *et al.* (2011c). Traditional simulation often requires intensive software developments, but SimSaaS can provide significant advantages as it can potentially provide configuration, MTA, and scalability, and it can save notable time and effort in developing simulation software based on the same software base. Furthermore, a cloud offers a computing platform where an organization does not need to acquire its own servers and infrastructure, and thus may save a lot of resources to support simulations.

On top of SaaS and simulation there are lots of goals: First, SaaS providers need to evaluate the timing and events in a SaaS infrastructure, a timing specification Tsai *et al.* (2009) and event infrastructureTsai *et al.* (2010b) need to be defined. Second, as each tenant has its unique configuration, it is difficult to check all the issues at the design time, and thus some constraints will be enforced at runtime by the policy mechanism Tsai *et al.* (2011b). By offering timing, event and policy for SaaS, the SaaS maintainers allow tenant to have a flexible way to maintain, trace, evaluate, analyze and fine-tune the SaaS infrastructure.

Cloud computing often has three principal components, Software-as-a-Service (SaaS)Chong and Carraro (2006), Platform-as-a-Service (PaaS), and Infrastructure-as-a-Service (IaaS). SaaS runs on top of a PaaS that in turn uses IaaS. SaaS is a new approach for delivering software, and it is characterized the capability to support customization Sun *et al.* (2008), Multi-Tendency Architecture (MTA)Tsai *et al.* (2006c), scalability and reliability Tsai *et al.* (2010a). For instance, saleforce.com provides a CRM solution, and it offers a pre-built application that runs on top of Force.com. Well-known SaaS systems include Salesforce.com, Rightnow Technologies, Workday, SuccessFactors, and Corenttech.com. PaaS provides similar functions as a traditional OS with databases and often runs on a massive number of processors with built-in fault-tolerant computing and scalability mechanisms. For example, in GAE (Google App Engine), each write operation will write three independent chunks to ensure reliability, and more chunks if the user requires higher reliability. Furthermore, the system provides scalability by allocating more resources when the workload increases. Well-known PaaS systems include GAE, Amazons EC2, and Microsofts Azure Microsoft (2014). Since the popularity of SaaS infrastructure and the complexity of building the software for SaaS, modeling SaaS and generate code for PaaS based SaaS is important and challeging Li *et al.* (2012b).

Log, used for saving the production behaviors in transaction processing system in a cloud, is widely used to triage production failures. Log has posed challenges for cloud as its large volume in data, unstructured formats and untraceable failures. Even with trustworthy cloud platforms such as Amazon and Google, their services still experience constant downtime Status.aws.amazon.com (2015) Code.google.com (2015). Service downtime on production become frequent, however urgent for enterprise since every minute of failures on production can result in huge loss. Programmers normally put least priority at coding phase for logging, thus causing later on unstructured formats along with the heterogeneous deployment of the different components in cloud. As log data accumulate in cloud at hundreds millions per hour on commercial production sites, they roll fast and become complicate to manage. Lots of existing frameworks from enterprise and open source try to address the logging issues from different aspects. For example Splunk Splunk.com (2015), logstash Logstash.net (2015), uilog Zou *et al.* (2014), IBM SmartCloud Analytics Analysis (2015), Fluentd Project (2015), Google Analytics Google.com (2015), XpoLogXpolog.com (2015) and Nagios Nagios.org (2015) etc. Depending on the different stand point, they all have different usages within their own domains. To handle the unstructured, large data mixed with untraceable failures, none of the existing systems have a complete solution. This paper is to propose a new logging solution for efficient and effective triaging of the production failures with these challenges.

The research topic covers the following research areas: service modeling, policy specification, code generation, dynamic simulation, timing, event and log analysis.

Chapter 2

SIMSAAS: SIMULATION SOFTWARE-AS-A-SERVICE

2.1 Introduction

Early exploration of simulation in a cloud has been done by Lanner group Group (2014) and Thomas PaviotPaviot (2010). Lanner group develops a simulator for business process management systems in a cloud. Thomas offers simulation CAD services in a cloud. However, they have not explored the configuration, MTA, and scalability features of SaaS.

This paper explores the potential of using these SaaS features for simulation software. To support configurability at multi-layers, both simulation supporting framework and simulation models need to be included. To support MTA, the system should have the abilities to add/modify/delete the tenants, address the tenants accessibility controls, distinguish tenants simulation interaction message during executions and isolate tenants own specific data. To support scalability, both scale up and scale down need to be considered Wikipedia (2014b). To fully explore these features, one can assume the Platform as a Service (PaaS) such as EC2 Palankar *et al.* (2008a) or GAE Google (2014) can provide relevant features such as scheduling, automated workload detection, web data storage, automated redundancy management, scalability (including scaling up and down, out and in) and recovery, which is indeed supported.

As a summary, the contribution of this paper is as follows:

- Propose SimSaaS framework that incorporates key elements of SaaS features for simulation;
- Design a platform-independent configuration model to support MTA;

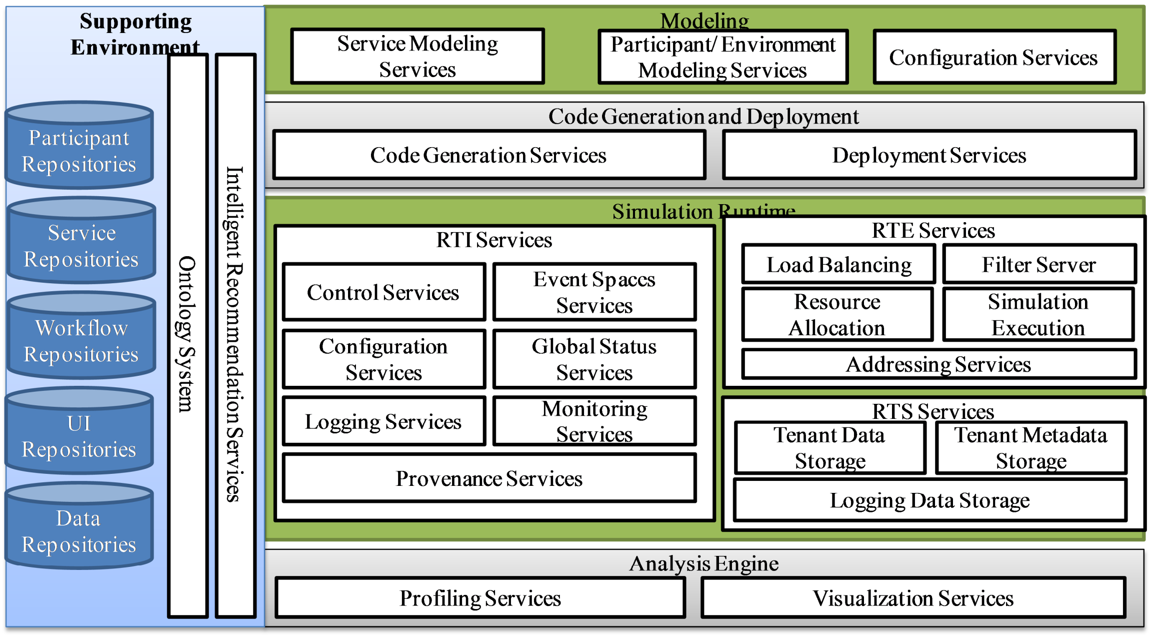


Figure 2.1: Elements of SimSaaS Architecture

- Present a simulation runtime infrastructure that supports MTA;
- Investigate various run-time analysis techniques that can monitor, analyze and calibrate the simulation continuously.

This paper is organized as follows: Section 2 discusses the SimSaaS framework; Section 3 presents a SimSaaS MTA configuration model; Section 4 talks about a MTA simulation runtime; Section 5 presents analysis and calibration of the simulation model; Section 6 presents an example, Section 7 concludes this paper.

2.2 Simulation Architecture

The SimSaaS architecture design is discussed in this section, As shown in Figure 2.1 shows the SimSaaS architecture; and Figure 2.2 shows the relations of the major components in the framework through the life cycle of SimSaaS, including a modeling, code generation/deployment, simulation, and analysis cyclical process.

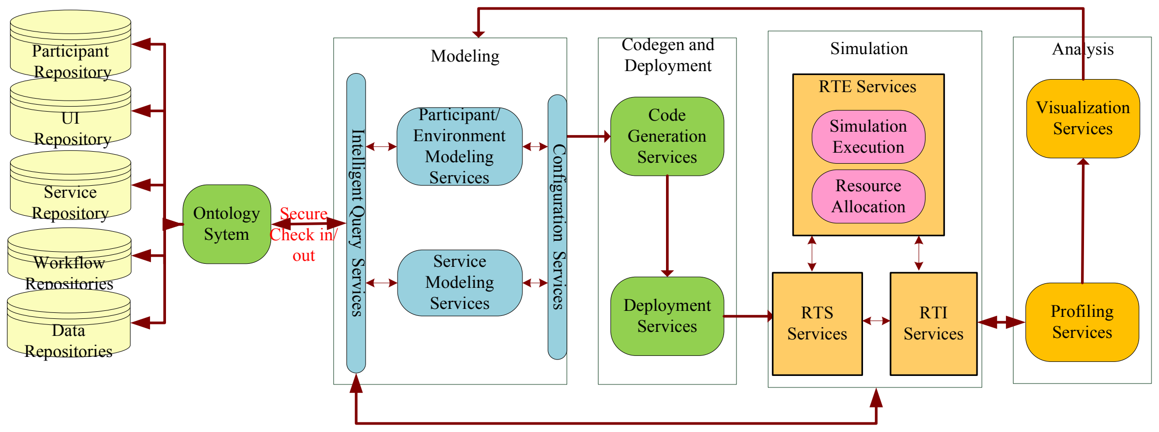


Figure 2.2: SimSaaS Main Flow

As one may find out, the design is similar to the SOA simulation framework Tsai *et al.* (2009, 2010b, 2006c) but with several distinct differences. In modeling phase, configuration services are offered to configure the existing simulation services with MTA abilities. In simulation runtime phase, a new infrastructure is introduced to handle the tenant add, delete, modify operations and the MTA simulation execution including resource allocation, addressing, isolation, accessibility control et al.

Supporting Environment: This contains repositories and services that facilitate the reuse of simulation models. Service repositories store simulation services, workflows, participants and data; ontology systems classify the models, store their relations and support reasoning function for models; intelligent recommendation services serve as the interfaces to query metadata.

Model information is then passed to the modeling services through intelligent query services. To speed up the searching, continuous indexing and caching system can be used.

Modeling Services: These provide modeling and configuration for simulation using PSML Tsai *et al.* (2007d), simulate service modeling for both atomic and composite services Tsai *et al.* (2007d). Configuration services are offered here as well to support the MTA configuration. Tenant configuration will be discussed in section 3.

Code Generation/Deployment Services: These include services that generate and deploy the code to simulation engine. Similar to DDSOS Tsai *et al.* (2006c), simulation deployment in SimSaaS can support both the on demand and automated way.

Simulation Runtime: This offers the vital infrastructure services, stores necessary information and executes simulation models in cloud. Three major components include Run Time Infrastructure (RTI) services, Run Time Execution (RTE) services and Run Time Storage (RTS) services. More details of Simulation Runtime and MTA support will be discussed in section 4.

Analysis Engine: This tracks the runtime information, analyzes the simulation and displays through a graphic interface. Profiling services analyze the runtime execution information and make predictions to aid the dynamic calibration of simulation. Visualization services display the data from logging, provenance and profiling services. Both timing analysis Tsai *et al.* (2009) and event analysis Tsai *et al.* (2010b) will be supported in SimSaaS.

The whole process is a continuous cycle, in which the analysis results are sent back to the modeling phase for continuous calibration until it meets the desirable goals of the simulation.

Considering the space constraint, this paper will focus on MTA configuration, execution runtime, and the continuous analysis.

2.3 Simulation Multi-Tenancy Configuration

This section designs a configuration model for the MTA support of existing PSML simulation model.

The simulation models used in SimSaaS are described by PSML, which can be executed in cloud platform such as GAE through the code generation abilities. However, PSML does no support of MTA inherently. The desirable features for supporting MTA can actually be configured using a model describing essential requirements for MTA. Then, simulation runtime can read the tenant configuration model, furthermore handle resource allocation, and other related problems in execution runtime. Thus this paper needs to figure out a MTA configuration model first.

Take a smart home simulation example. Suppose a HomeSimulation model is developed using PSML for a room. Before deploying the simulation to cloud, the director changes her mind and wants to replicate the simulation for 100 different families and each room has its unique requirements. One straightforward solution is to make modification to existing model and create 100 instances. In cloud, the 100 different simulations are essentially 100 tenants. And thus this is a MTA problem.

However, several issues are raised: Do we need to change the existing model for MTA? How we differentiate the requirements of tenants? Will hardware, software and storage be shared among the simulations? How does the system manage these simulations with respect to resource allocation, addressing, accessibility control and execution? In the following discussion, our model will solve these issues in turn.

Thus the following requirements are required for support MTA in an existing simulation model:

- Identities and requirements for different simulation tenants.
- Physical and logical address for the simulation models.
- Capability controls of different simulation models.
- Sharing strategies among tenants.

This paper proposes TC (T-Filter Configuration), a configuration model for SimSaas, to address the above MTA requirements.

2.3.1 T-Filter Configuration: The MTA Model for SimSaaS

TC is used to support MTA in PSML simulation model. The goal of TC is to let tenants focus on the simulation modeling without worrying about the tedious MTA concerns at code level.

TC is composed of four parts: S-Model (Simulation Model), T-Filter (Tenant Filter), TE (Tenant Expressions) and hardware requirements. S-Model represents the PSML simulation model. T-Filter stands for tenant filter, which is used as a unique id for filtering different tenants at simulation runtime. TE is tenant expression which defines which T-Filter will be applied to which S-Model as well as the capability controls towards the S-Model.

Formally, corresponds to the four MTA configuration requirements, this paper categorized the definition into three groups: tenant model definitions and addressing, capability controls and sharing strategies.

Tenant Model Definitions and Addressing

The configuration model first needs to define the configuration (TC), the tenant (T-Filter, T-Space), the simulation model(S-Model, Sub-Model), and how to apply the configuration to the simulation model (TE). Then, the end of this section explains using these models for physical and logical addressing.

Definition 1, Given a tenant set T, $T = \{T_1, T_2, ...T_n\}$ there exists a tenant configure set TC, which maps tenant T_i to TC_i, denotes as $T_i \rightarrow TC_i$ (i=1,2,...n). Each TC_i is composed four parts, denotes as $< F_i, M_i, E_i, R_i >$, in which

- F_i is the configures file for TC_i.
- M_i is the simulation model set for TC_i, $M_i = \{M_{i,1}, M_{i,2}, ...M_{i,k}\}$, $\mid k \mid$ is the size of M_i.

- E_i is a set of regular expression. Given a finite alphabet $\sum$, with operations . (concatenation), | (alternation), * (start set union) are defined.

- R_i is the hardware requirements for a specific resource, $R_i = \{R_{i,1}, R_{i,2}, ...R_{i,q}\}$, in which $R_{i,1}$ is request for a recourse q, e.g. cache.

In the following discussion, this paper will define each components in details.

Definition 2 Simulation Model (S-Model): *A simulation M is composed of several sub model, denote as* $M^{(j)}$, $\forall M$, $\exists M^{(1)}$, $M^{(2)}$,..., $M^{(p)}$ such that $M = \bigcup M^{(j)}$. Let S-Model denotes the set of simulation model names. s_i is in S-Model. $s_i = \{sub_1, sub_2, ...sub_n\}$ and $sub_n \in Sub - Model$.

Example 1: BankSimulation and HomeSimulation are two S-Models which contain Sub-Models including workflows, services, participants and data.

Definition 3 (Sub-Model): *Sub model of PSML simulation model. Sub-Models* are identified by a unique id under each *S-Model.* For convenience, the later part of the paper uses names instead of ids.

Let *Sub-Model* denotes the set of sub simulation model names. sub_i is in *Sub-Model* and $sub_i \in$ *participant, UI, data, service, workflow. Sub-Model* can be accessed using . from *S-Model.*

Example 2:

HomeSimulation.UI1

HomeSimulation.Workflow1

Definition 4 Tenant Filter (T-Filter): *T-Filter* is unique name that is used for supporting multi-tenancy in cloud for SimSaaS. *T-Filter* and tenant names are interchangeable.

Let *T-Filter* be the set of tenant filters, let *Tenant* be the set of tenants. $\forall tenant_i$ in *Tenant*, $\exists$ $t\text{-}filter_i$ in *T-Filter*. $tenant_i \in string$.

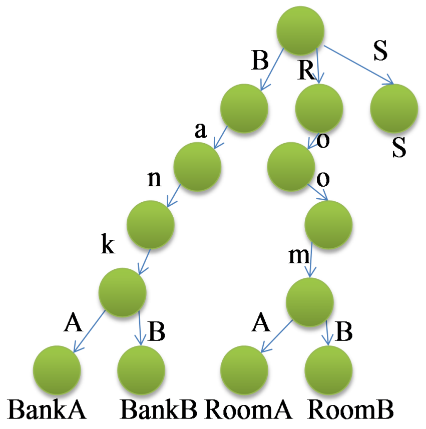


Figure 2.3: T-Space Implementation using TRIE

Example 3:

RoomA, RoomB can be two different *T-Filters*

Definition 5 T-Filter Space (T-Space): *T-Space* stores and guards the uniqueness of the T-Filters in a simulation environment. It is a TRIE Willard (1984) thus it eliminates the naming conflicts in simulation runtime. TRIE is a prefix tree in which the value of the node is stored on the edge. Every leaf of the prefix tree represents a T-Filter. A lemma of prefix tree is that every leaf will be different since the paths from root to the leaves are different.

Example 4:

In Figure 2.3, the sample T-Space has four different T-Filters and each of them is a leaf in the TRIE, thus no naming conflicts will occur.

Definition 6 T-Filter EMPTY Set ($\emptyset$): Defines the case that no *T-Filter* is applied to an *S-Model*. It is represented as $\emptyset$. Empty set implies no tenant will use a specific *S-Model*.

Definition 7 Tenant Expression (TE): *TE* is the tenant expression to describe how to apply *T-Filters* to *S-Models*. It is composed by a number of *T-Filters*, *S-Model* and *T-Filter* operators including "Capability", "Negation", "Union", "Intersection"

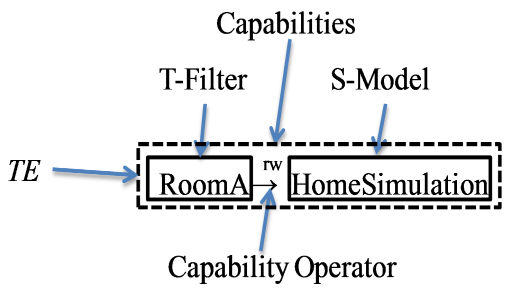


Figure 2.4: A Sample Tenant Expression (TE)

as defined below. *Capability* and *Negation* tells whether and how the *S-Model* is accessed. Union and Intersection tells how the *S-Model* is shared.

Let *TE* be the set of tenant expressions. Each te_i in *TE* is a 4-tuple $< T-Filters, S-Model, Operators, Capabilities >$ where *T-Filters* denote the set of tenant filters to te_i, *S-Model* the simulation model to te_i, *Operators* the operation from *T-Filter* to *S-Model*, and *Capabilities* the read, write, execute capabilities to te_i. Figure 2.4 shows a *TE* example.

The six definitions are also used in addressing the resource in execution runtime. More specifically, *T-Filter*, *Req* in *TC* are used later in physical addressing. *T-Filter*, *TE*, *Sub-Models* and *S-Models* are used in logical addressing.

Capability Controls

Capability-based systems Levy (1984) has been proposed in hardware system to apply access control. This paper uses capability to solve tenant access control problems in a MTA simulation runtime. Meanwhile, capability system is also used in the logical addressing in runtime.

Operation 1 Capability Operator ($\rightarrow$): Defines the capability operation from a *T-Filter* to a specific *S-Model*. By default, capability operator implies to have all capabilities unless addressed specifically.

Let C be a tuple $< T - Filter, S - Model >$ where *T-Filter* denotes tenant filter, and *S-Model* denotes simulation model. c_i is in C. t is a *T-Filter* and s is a *S-Model*. Operation $t \rightarrow s$ means that *T-Filter* t will be applied to *S-Model* s in c_i with default capabilities.

Example 5:

$$\text{RoomA} \rightarrow \text{HomeSimulation}$$

$$\text{RoomB} \rightarrow \text{HomeSimulation}$$

Indicate two tenants which logically share the same *S-Model HomeSimulation*, but physically separate in implementation. Two *T-Filters RoomA* and *RoomB* are used to isolate tenant information between the two model implementations.

Example 6:

$$\text{RoomA} \rightarrow \text{HomeSimulation}$$

$$\text{RoomA} \rightarrow \text{TemperatureControl}$$

Indicate one tenant with a *T-Filter RoomA* contains two different *S-Models*.

Operation 2 Negation Operator (¬): Negation operator defines the *T-Filter* that will not be applied to an *S-Model*.

Example 7:

$$\neg\text{RoomA} \rightarrow \text{HomeSimulation}$$

The expression means that *T-Filter RoomA* cannot be applied to *HomeSimulation* model.

Definition 8 Tenant Capability (Read, Write, and Execute): Defines the Read, Write and Execute capability for the capability operator. Let a be a tenant filter and b be a simulation model.

$a \xrightarrow{r,w,e} b$ denotes that *T-Filter* a will be applied to *S-Model* b with *Read(r)*, *Write (w)* and *Execute (e)* capability. Thus this expression will be identical to a $\rightarrow$ b.

$a \xrightarrow{\neg r,\neg w,\neg e} b$ denotes that *T-Filter* a will be applied to *S-Model* b without *Read*, *Write* and *Execute* capability. This will be identical to the $\neg a \rightarrow b$.

Capability control to the *Sub-Model* of the simulation will give flexibility for the accessibility of *S-Models*.

However, this might result in some validity problems since not every *Sub-Model* in *S-Model* can be simply assigned with a capability. A way to go around is to list out *Sub-Models* that can be configured. And thus it will be safe to configure the *S-Model* with the capabilities.

Model Sharing Strategies

Operation 3 Union Operator ($\cup$): Union operator defines the set of *T-Filters* that can be applied to the same *S-Model*. It can concatenate *T-Filters*. Often, it is used for abbreviation purpose.

Let t_n be the tenant filter, s be the simulation model. $(t_1 \cup t_2 \cup ...t_n) \rightarrow s$ denotes $t_1, t_2, ...t_n$ all share the same logical model s.

For instance, the aforementioned example 5 can be expressed as

Example 8:

$$(\text{RoomA} \cup \text{RoomB}) \rightarrow \text{HomeSimulation}$$

Operation 4 Intersection Operator ($\cap$): Intersection operator defines the set of tenants that share the same *S-Model*. The difference between Intersection and Union is that Union keeps separate execution instances for different tenant while Intersection shares the same execution instance.

Let t_n be the tenant filter, s be the simulation model. $(t_1 \cap t_2 \cap ...t_n) \rightarrow s$ denotes $t_1, t_2, ...t_n$ all share the same s in execution.

Example 9:

$$(\text{RoomA} \cap \text{RoomB}) \rightarrow \text{HomeSimulation}$$

The example means the same *HomeSimulation* model is shared between tenant *RoomA* and tenant *RoomB*. This is different to the meaning of example 5. In example 5, *HomeSimulation* has two copies of models at runtime; in this example, only one copy of model is shared between tenants at runtime.

Whether the intersection operator can be applied to simulation is depend on the cloud platform implementation.

Next chapter of the paper will discuss applying these configurations for MTA simulation.

2.3.2 Discussions

As discussed earlier the goal of TC is to support MTA in PSML simulation model in a cloud platform such as GAE. With TC, GAE can 1) integrate the physical addressing from the cloud and the logical addressing from PSML; 2) GAE will know how to apply the resource sharing strategies to simulation; 3) apply the fine grained capability controls to the PSML simulation model.

2.4 SimSaaS Execution Runtime

This section designs a simulation runtime that be able to handle MTA simulation in cloud. MTA support is enabled in simulation runtime through TC. Other than the runtime infrastructure, three main issues exist for supporting MTA. 1) To configure the simulation tenants in real-time; 2) To allocate and address the tenant resource

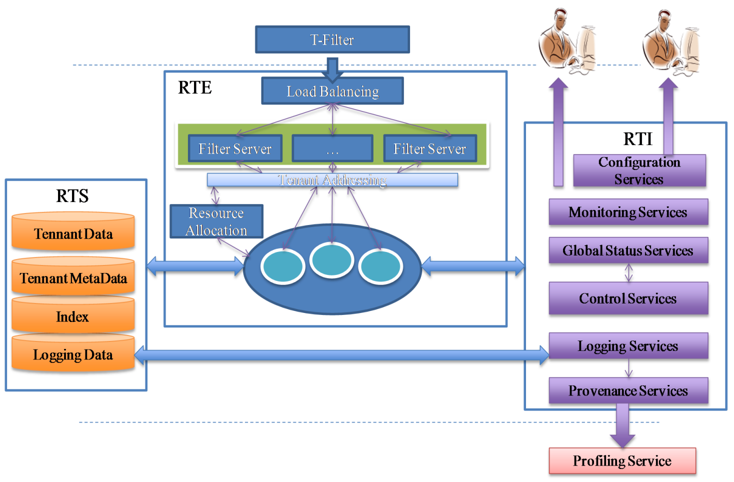


Figure 2.5: Simulation Runtime

among the tenants; 3) To execute the model, share and isolate resources, and apply the capability controls to the model.

2.4.1 Simulation Runtime Infrastructure

Simulation runtime includes three major responsibilities: storing simulation model and related information for the simulation; executing the model; offering fundamental services for the execution of the model. In this paper, Runtime Storage Services (RTS), Runtime Execution Services (RTE) and Runtime Infrastructure Services (RTI) are introduced to solve the three responsibilities. Figure 2.5 shows an architecture view of simulation runtime.

RTS store the necessary information for the simulation engine in cloud. The data include tenant data, tenant metadata, tenant index and simulation logging data.

RTE allocate resource according to individual simulation model and tenant configuration and then execute it from server side. They support adding, modifying, deleting tenants; addressing tenants; and executing the simulation for tenants.

RTI manage the simulation models and support the communication within each individual tenant. A list of services are offered for execution of simulation including control services for the synchronization among the simulation parties, event space services for categorizing the events, global status services for sharing the current statuses of the variables, runtime monitoring services for watching the current execution status, logging services for recording the simulation procedures, configuration services for runtime and provenance services track the origin of simulation data from logging services.

Simulation models and configurations are delivered to the RTS first. RTE read data from RTS and then execute the simulation model. To handle events, resources, timing and synchronization issues of individual simulation, RTI is then required to offer infrastructure level support.

2.4.2 Real-Time Tenant Configuration

After modeled *S-Model* and defined the *TC* for the model, the next step is how to apply the configuration to the simulation runtime. This section will discuss three basic operations: add, delete and modify.

S-Model is stored in RTS as tenant metadata thus when creating a new tenant for an existing model, there is no need to upload the *S-Model* for multiple times. The simulation developers just need to operate upon the *TCs*.

The following algorithm explains how to do tenant configuration. The inputs are the *S-Model* and the type of the operation. Different outputs will be yielded depending on the operations.

Add Operation: This adds a new tenant by offering a new *tenant configuration*. In the algorithm, it involves the new allocation of computing and storage resource, and the recording of the current TC.

Algorithm 1 Real-Time Tenant Configuration Algorithm

```
Input:
 1: Tenant configuration TC, Operation type (add, delete, modify)
Output:
 2: Create, delete or update the virtual tenant according to TC and operation type
 3: LoadBalancer loadBalancer = DeliverTC2LoadBalancer(TC);
 4: FilterServer filter = loadBalancer.RoutTC2FilterServer(TC);
 5: if add_tenant then
 6:     filter.getResourceAllocater.allocateComputingResource(TC);
 7:     filter.getResourceAllocater.allocateStorageResource(TC);
 8:     addTCToRuntime(TC);
 9:     startSimulationModel(TC);
10: else if delete_tenant then
11:     stopSimulation(TC);
12:     filter.getResourceAllocater.removeComputingResource(TC);
13:     filter.getResourceAllocater.removeStorageResource(TC);
14:     removerTCFromRuntime(TC);
15: else if modify_tenant then
16:     if s_model_accessed then
17:         restartSimulation(TC);
18:     else
19:         filter.updateTenantTC(TC);
20:         proceedSimulation();
21:     end if
22: end if
```

Delete Operation: This deletes an existing tenant from simulation runtime. In the algorithm, it involves the removal of computing resource, storage resource and TC from the existing system.

Modify Operation: modifying an existing tenant configuration in runtime. In the algorithm, it is related to the TC modification rules.

For each capability of the *S-Model* (*Sub-Model*), it is marked with a boolean flag as stated in *S-Model/Sub-Model* capability table. Two rules that will change the Real-Time behavior when modifying the simulation:

Restart Rule: If the model has been accessed when being modified, a restart of simulation is required and the runtime will have to clean/differentiate the existing data associated with the current simulation execution.

Precede Rule: If the model has not been accessed being modified, the modification will be marked as safe. The simulation can proceed without restarting.

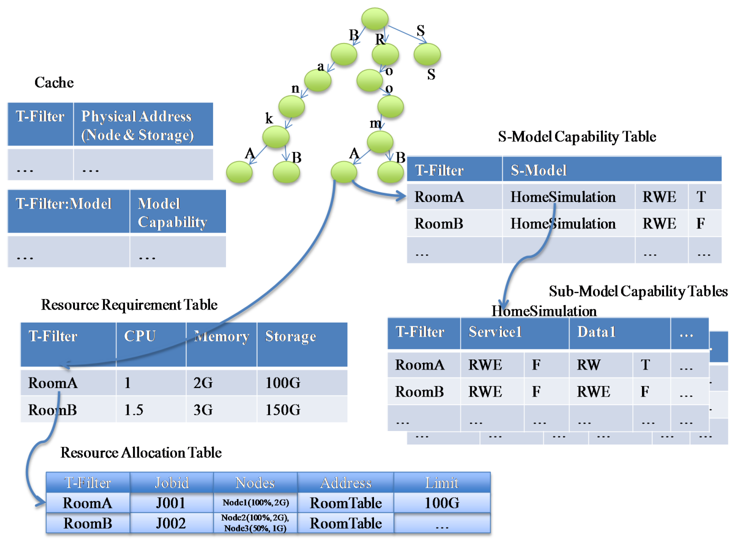
Using the above algorithm, people can add tenant, delete tenant and modify an existing tenant at real time.

2.4.3 Tenant Physical and Logical Addressing

Another important issue is to find the resources for different simulations in the same simulation runtime. The addressing includes both physical and logical addressing.

Resource physical addressing is related to resource allocation since when allocating the physical resource to different tenants, it will be specified with the details of the hardware resource.

Resource allocation is done through the Resource Allocator in RTE as shown in Figure 2.5. When a new tenant configuration arrives, the defined physical resource requirement (Figure 2.6) will be stored to the resource requirement. Physical resource



Cache

T-Filter	Physical Address (Node & Storage)
…	…

T-Filter:Model	Model Capability
…	…

S-Model Capability Table

T-Filter	S-Model		
RoomA	HomeSimulation	RWE	T
RoomB	HomeSimulation	RWE	F
…		…	…

Resource Requirement Table

T-Filter	CPU	Memory	Storage
RoomA	1	2G	100G
RoomB	1.5	3G	150G

Sub-Model Capability Tables

HomeSimulation

T-Filter	Service1		Data1		…
RoomA	RWE	F	RW	T	…
RoomB	RWE	F	RWE	F	…
…	…	…	…	…	…

Resource Allocation Table

T-Filter	Jobid	Nodes	Address	Limit
RoomA	J001	Node1(100%, 2G)	RoomTable	100G
RoomB	J002	Node2(100%, 2G), Node3(50%, 1G)	RoomTable	…

Figure 2.6: Resource Addressing Example

for each tenant such as CPU, memory and storage requirements will be specified. The resource allocator will then interact with PaaS such as GAE for the required physical resources. Each T-Filter will point to an entry in resource allocation table, in which the physical address for the allocated resource including JobId, node names, storage table address and limits for the storages are specified. The arrows in the left part of Figure 2.6 show how to address the physical resources.

Logical addressing is related to *TE*. As stated earlier, each simulation model has a *TE*. It defines the capability control from *T-Filters* to the *S-Models* and *Sub-Models*. That in fact is also used for addressing the logical resource. The addressing is divided in two steps. First, *S-Model* level: it can be addressed by *T-Filter* since each of them is different in tenant space. Second, *Sub-Model* level: it can be addressed by the ids of the *Sub-Model*. The arrows in the right hand side of Figure 2.6 explain how to address the logical resource.

Note that each *Sub-Model* is part of *S-Model*, thus the addressing for a *Sub-Model* will involve the addressing for the *S-Model* first and then for the *Sub-Model*. For each

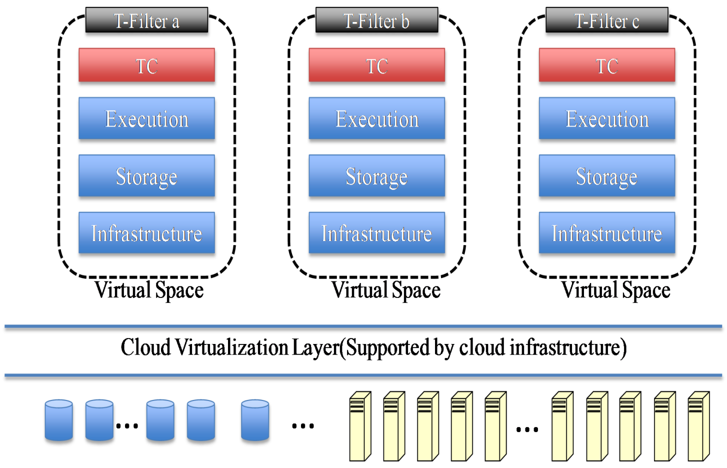


Figure 2.7: Virtual Space for Simulation Execution

type of *S-Model*, it will have its own *Sub-Model* capability table since they contain different *Sub-Models*.

Caches are also used for speeding up the lookup in a relational table. LRU, MRU and other algorithms can all be applied for the cache services.

2.4.4 Resource Isolation and Capability Control

MTA introduces two new issues in the simulation runtime. 1) Share and isolate the resources; 2) Apply the different capability controls to the models deployed in runtime.

Logical isolation of the resource is done by *T-Filter*. Each tenant creates a virtual space by *T-Filters*. Logically, these virtual spaces are independent since none of them will duplicate.

In fact physically, the execution environment can share models and resources. As stated in section 3.1.3 the sharing happens at two levels.

The first is at resource level. Sharing is confined to *S-Models*, hardware and storage. Sharing *S-Model* is obvious, since through *TC*, it is easy to create two tenants out of the same PSML simulation model. Sharing hardware can actually be handled by the PaaS in a synchronized or sequential way. Sharing storage can be done through proper data storage schema design. In Tsais book Chen and Tsai (2010) 6 different schema designs for supporting MTA in cloud are provided. All of them can be used in SimSaaS design. Other than that in the case study part, the paper listed a *T-Filter* based design for storing the tenant data in Bigtable.

The other sharing is at the execution level, in which two or more tenants share the same execution instance. This will require a redesign of the simulation model and it will be PaaS dependent. For instance, GAE allows sharing the same execution instance from different tenants. *S-Model* of SimSaaS can also implement simulation in this way; however, this requires a redesign of the simulation model since namespace switch will be extensively used in GAE. To support this, a code generator needs be introduced to insert pieces of code into the model. In the resource allocation table in figure 6, physical addressing will point to the same set of resource, while it will have a controller to budget the resource usage according to the requirements. Instance sharing may also include resource sharing including *S-Models*, computing and storage resources.

Sharing of the models tells the common parts of different simulation, while capability controls specify the differences among the different tenants. With capability control, one set of *S-Model* can behave differently since they are configured with different accessibilities. Problems are raised in checking the capability in a MTA since it allows the capabilities to be considered in a MTA environment.

Various designs can be applied to solve the capability control problems. Considering MTA, we summarized two ways below with their own pros and cons.

- Inject: Require every *Sub-Model* maintain a capability attribute along with the model. The capabilities of the model are injected to the model when the virtual model is initiated. This works pretty efficiently since the capability is with the

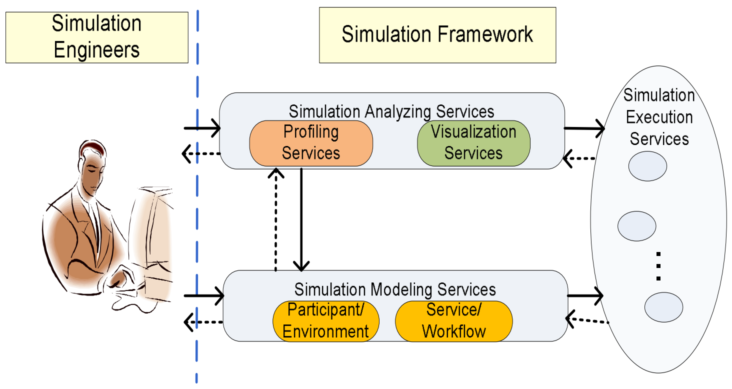


Figure 2.8: Simulation Process

model. However, it has potential problem in MTA since the model needs to be redesigned to record the different *T-Filters* and its capabilities.

- Force Look Back: Update the capability in *TC*, force look back the *TC* whenever accesses the model. It will cost a bit more since an external look up is required. However, it requires no change towards the existing simulation model and the MTA problem is solved by offering a capability lookup table as shown in Figure 2.6. This solution is taken in SimSaaS.

2.5 Real-Time Analysis and Continuous Calibration

This section talks about how to do real-time analysis and continuous calibration in SimSaaS.

Continuous calibration of simulation includes the interferences of simulation users, simulation analyzing services, simulation modeling services and simulation runtime. And it is done through the dynamic cyclical process as shown in the Figure 2.8.

While static analysis provides useful data, many analyses can be done only via dynamic simulation. Simulation modeling services are useful to make the process dynamic. By monitoring the execution data from the simulation analyzing services, a simulation engineer can change the model.

Profiling services in the simulation analyzing services are also useful to make the calibration process dynamic. Based on the information from the provenances services and logging services in RTI, profiling services can control the selection of services by following the predefined algorithms in the services.

Simulation model can evolve by calibrations. In Jins paper Zeng *et al.* (2009) he addresses some patterns for service evolvement such as add, replace. Since these operations can be strictly described and proved, it is easy to derive that the whole process can in fact be automated. For instance, the calibration process can automatically use the intelligent query service and then use the query results to replace the current model with the other candidates.

A report is generated at the end and thus it is easy to compare the simulation models by viewing them visually.

T-Filter still applies in analysis engine since it isolates data collection from the Logging Services and Provenance Services. Analysis and reports are generated by different *T-Filters* and presented to end users.

2.6 Example

Figure 2.9 shows a smart home Device Control simulation which requires controlling the devices in a smart home. The whole process is divided in five steps. The goal of this case is to create two tenants RoomA and RoomB which use the same S-Model. In execution time, the two tenants will create two execution instances with their data stored in one Bigtable.

Step1: An S-Model template is created using PSML model. The S-Model contains a workflow and a table which contains the device lists.

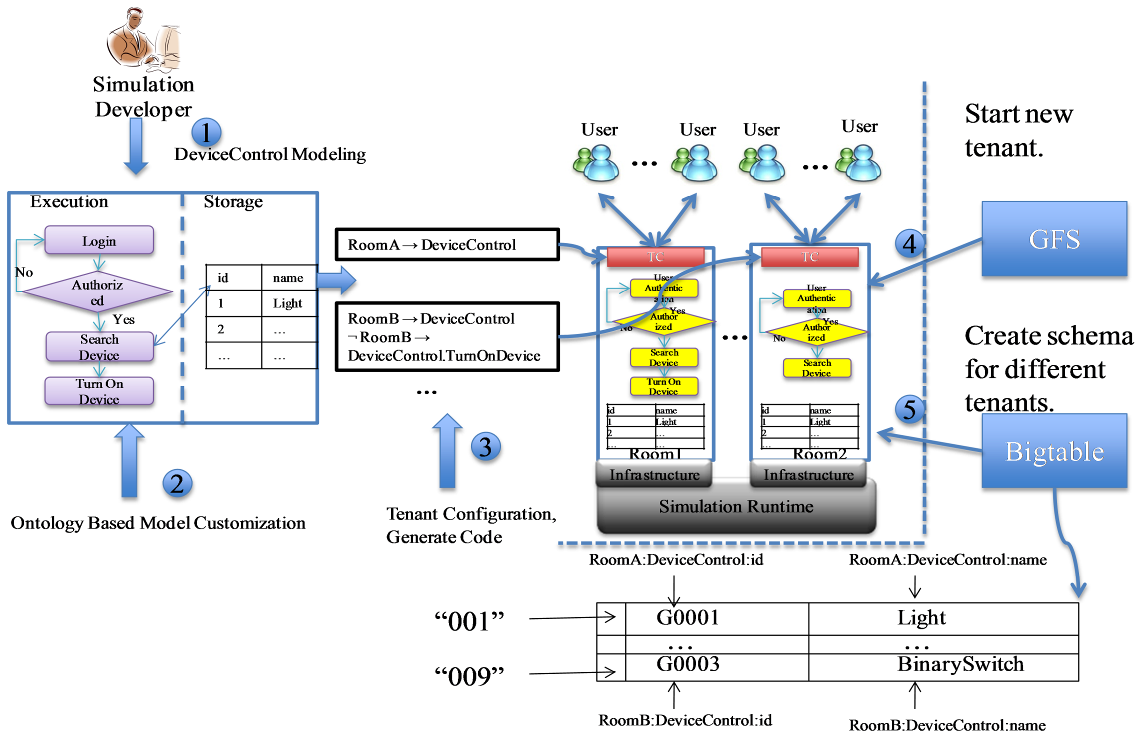


Figure 2.9: Creating Device Control Simulation to Support Multi-tenancy

Step 2: An S-Model is created through customization. In customization, the simulation developers will design the ontology system for the desired UIs, services, workflows, data and participants.

Step 3: Two tenants are created by specifying the tenant configuration defined follows:

$$\text{RoomA} \rightarrow \text{DeviceControl}$$

$$\text{RoomB} \rightarrow \text{DeviceControl}$$

$$\neg\text{RoomB} \rightarrow \text{DeviceControl.TurnOnDevice}$$

This TC configured two tenants RoomA and RoomB which shared one simulation S-Model while executes separately.

Step 4: The TC will be added to simulation runtime. Simulation runtime invoke the resource allocator and allocate necessary computing resource through GFS according to the hardware resource requirements. Two execution instances are named after RoomA and RoomB.

Step 5: a Bigtable is created for the storage of the data according to the schema. In implementation, it used Bigtable to store the data, in which each column name contains the logical addressing of the data. This helps isolating the data from tenant RoomA and RoomB. For instance, the id of DeviceControl for RoomA will have a column name RoomA:DeviceControl:ID. Thus it is easy to store and look up data for different tenants.

2.7 Related Work

CloudSim is a simulation project that simulates a cloud platform. It has two features: 1) to allow the cloud developers to test the provisioning and service delivery policies before the actual deployment of the application; 2) to support tuning up the performance bottlenecks before deploying on real clouds. CloudSims focus of the cloud simulation focused on the backend infrastructure, they simulate data centers, virtual machine availabilities, and ability for picking up the virtual processing cores for virtualized services.

Early explorations of simulation in cloud has been done by Lanner group Group (2014), an business process improvement company. Lanners simulator simulates the business process management systems. L-SIM 2.0 is configured as a RESTful service in the cloud and thus the end user can do the business process simulation in the cloud. In his presentation, Thomas Paviot (2010) used open source project and open standard and built a CAD simulation on cloud to reduce the need for expensive hardware, costly licensing scheme and to solve the interoperable problems of using the CAD software. However, most of these early explorations are still at a very primitive state. For instance, instead of considering customization, MTA, and scalability, both

L-SIM 2.0 and Thomas CAD simulation simply deployed their current simulator from a local server to a remote cloud by using the web services protocols. Obviously, there are more challenges for doing simulation in cloud. How to model simulation, how to reuse simulation, how to execute simulation and how to do the analysis and calibration have become ever important to solve to run simulation on cloud and makes fully use of cloud.

Different cloud platform has different ways to support MTA. For instance, saleforce.com uses a database approach, IBM uses Corent to solve MTA Chate. (2010), and Google enforces namespaces in Google App Engine to support MTA Google (2014).

Configuration is widely used in software Cons and Poznanski (2002) and hardware system to add flexibility to existing models. With proper configurations, one can modify the existing model easily and satisfy diverse requirements. SimSaaS uses tenants configuration to address the MTA requirements and then applies the configurations to the simulation runtime.

2.8 Conclusion

This paper proposed a MTA simulation framework SimSaaS by following the SaaS architecture in cloud computing. Furthermore, this paper uses a filter system to perform access control similar to the capability systems proposed earlier, but this feature is used in MTA in cloud platform. The design of SimSaaS on GAE will be presented in the near future.

Chapter 3

MODEL-DRIVEN TENANT DEVELOPMENT FOR PAAS-BASED SAAS

3.1 Introduction

The conventional approach to develop SaaS application on GAE is through code based approach in that the programmers are responsible to develop the entire application using a programming language. However, in fact, SaaS application can be modeled in a model driven way. In that, SaaS application can be modeled in the service modeling language and then customized with the multi-tenancy architecture, scalability, and redundancy & recovery techniques, finally generate code with these models directly for the target PaaS.

SaaS systems have been classified into four categories Tsai *et al.* (2009):

1. **Integration with databases:** In this approach where software applications are fully integrated with a database. Salesforce.com is one such example.

2. **Kernel-based approach:** In this approach, software applications will run on top of kernel that runs on top of databases, and any communication between software applications and databases is via the kernel. Corenttech.com is an example of this approach.

3. **Service-oriented SaaS:** In this approach, a SaaS system is designed in a service-oriented manner with service components that communicate via a service bus. EasySaaS is one such example Tsai *et al.* (2011a,e).

4. **PaaS-based systems:** In this approach, a SaaS system is developed on top of a commercial PaaS system such as GAE, Azure Microsoft (2014), and EC2

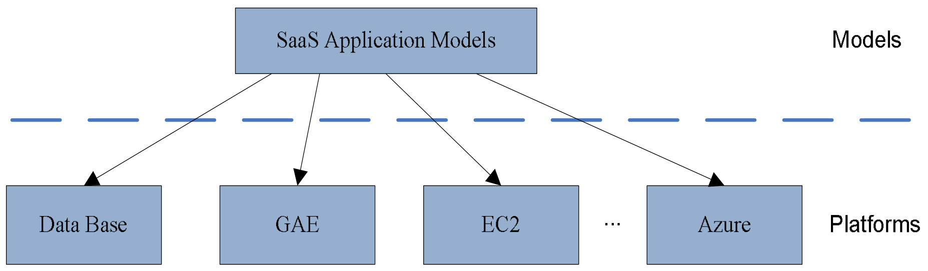


Figure 3.1: PaaS Independent SaaS Application Development

Amazon (2014); Palankar *et al.* (2008a). The following figure shows a PaaS independent SaaS application development model.

The customization, MTA, and scalability mechanisms have been compared in T-sai *et al.* (2012). Using a commercial PaaS system for developing a SaaS system has its advantages as well as disadvantages. The most notable advantages are that a commercial PaaS system already has built-in fault-tolerant capabilities, scalability mechanisms such as provisioning and automated migration, and database support. A new SaaS system can use these PaaS capabilities and save significant effort. However, these are also key disadvantages, as a PaaS system often controls these features directly, and a SaaS may not have indirect control only. Thus, a PaaS-based SaaS system may not be efficient as it does not control its key features directly, but relies on the underlying PaaS mechanisms for implementation. These PaaS mechanisms may be efficient for scalable Web applications, but they may not necessarily be the best solution for customizable and scalable SaaS with MTA.

3.2 PaaS Support for SaaS Implementation

GAE is a well-known PaaS system and this section will use it to illustrate PaaS support for SaaS. GAE provides a seemingly unlimited computing resource and virtualizes applications across multiple servers and data centers. Its infrastructure allows its hosted Web applications to scale easily, and frees developers from hardware

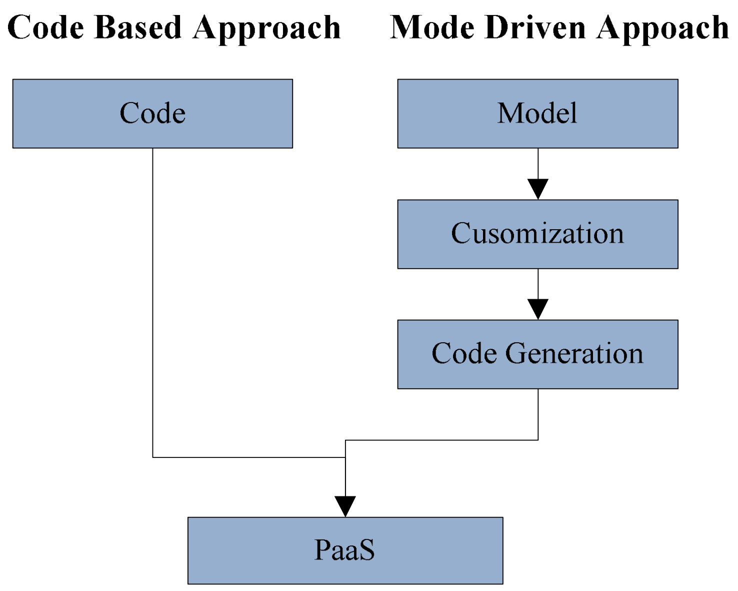


Figure 3.2: Code Based versus Model Driven Approach

con?guration and many other troublesome system administration tasks. GAE handles deploying code to a cluster, monitoring, fail-over, and launching application instances as necessary. GAE currently supports Python, Java, and Go languages. GAE also provides an eclipse plug-in to assist developers to create, develop and deploy applications. The compiled applications are deployed on a read-only file system on GAE. It has an admin console dashboard that allows the application administrators to create applications, monitor their executions, and check their usage/quota information.

PaaS Support for Tenant Customization: Most PaaS systems do not allow direct access of underlying dataStore or database, while common SaaS customization is done by storing configuration data into a database. At runtime, the SaaS controller will retrieve tenant application components from the database, compose the retrieved components into code at runtime, compile the composed code into executable code, deploy the executable code into the platform, execute the deployed code, monitor the execution, and return the results to the user Tsai *et al.* (2014). SaaS customization involves searching software components stored in the SaaS database, creating new components if necessary, and compose the needed component into a code that can be compiled. If a PaaS system does not allow direct access to the underlying database, the customization will be slowed. A PaaS-based SaaS designer has three choices:

- **Limited or no customization:** Essentially, the SaaS offers limited or no customization capabilities for tenant application developers. For example, a limited customization is to allow change of logos in a Web system with identical functionality;

- **Explicit customization points with options:** Using this approach, a SaaS system allows tenant application developers to customize their choices only at specific customization points with limited options, and each customization point also come with its constraints. Tenant application developers can upload their options for each customization point, but these options must pass the constraints specified in the customization point;

- **Virtual customization workspace for tenant application developers:** Using this approach, a SaaS will allow tenant developers to develop their customized applications like traditional SaaS systems such as Salesforce.com or EasySaaS where GUIs, workflows, services, and data components can be selected and composed for specific tenants. However, SaaS must develop this virtual workspace on top of a PaaS where direct database access may not be available.

GAE provides indirect support for customization by offering the configuration mechanism. A configuration of an application specifies the way the application is uploaded, the construction of dataStore indexes, and task scheduling priorities. GAE configurations include

- Deployment descriptor: This is used to determine URLs mapping to servlets and authentication requirement for URLs;

- App configuration: This specifies the app's registered application ID, the version identifier of the latest code, and the location of the resource files used by the application;
- Backend configuration: This declares the name and desired properties of each backend server;
- Index configuration: This specifies the indexes needed for the application;
- Scheduled tasks: defines the job execution times or regular intervals;
- Task queue configuration: This specifies both push queues and pull queues;
- DOS (Denial of Service) protection configuration: This defines the quota to prevent DOS attacks. These configurations are at the platform level, rather than at the application level. Thus, the SaaS developer needs to design its own customization features in GAE.

PaaS Support for MTA: Each PaaS system supplies its own MTA support mechanisms. For example, GAE provides the namespace, and each tenant is uniquely identi?ed by an entry in the namespace, and that information is used to distinguish one tenant from another. This namespace supports is at the code level as it is provided as namespace APIs. Each entry in the namespace essentially is an address within GAE, and the system uses the address to look for items related to a tenant for processing. However, the programmer must be cautious when dealing with namespaces, because it is possible that it can inadvertently cause data leaks. This is caused as all tenants share the same namespace mechanisms, and isolation among tenants or access control by tenants must be designed by the SaaS designers.

Namespace is currently supported in DataStore that is a schemaless object dataStore, Memcache that is in-memory data cache for high performance web application, Task Queue that is designed for executing background work, Blobstore that is used to serve data objects are much larger than the size allowed in the DataStore service, DataStore viewer that is the console for queries, and bulkloader that is designed for loading bulk data. The following example shows how to use NameSpaceManager to retrieve and switch the Namespace in code.

```
String oldNamespace = NamespaceManager.get();
NamespaceManager.set (”asu”);
try {
 dosomething(); // do something under namespace ”asu”
} finally {
NamespaceManager.set (oldNamespace);
}
```

PaaS Support for SaaS Scalability: Each PaaS system has its unique scalability mechanisms. For instance, GAE, Amazon EC2 and Azure allow the users to change the numbers of CPUs and RAMs to scale out applications, and a user feels like using a virtual machine with the given resources. However, such scalability mechanisms are often controlled by the PaaS, and such control may not be directly available to the SaaS.

GAE uses instances to scale applications, and instances are basic building units in GAE. It offers the language, runtime, APIs, application code, and the memory. GAE automatically allocates instances to the application as traffic increases to support the increased workloads. Each instance maintains the incoming requests in its own queue. If the load increased to a certain threshold, GAE will automatically create a new instance to handle the increased load. The max number of instances that can be defined is specified in the configuration files, and it can be adjusted at runtime.

PaaS Support for SaaS Reliability: Most PaaS systems offer reliability of the system, e.g., each write in GAE is a triplicate write where three copies are made in different chunks to ensure that data will not be lost easily. In case of a failure, GAE tries to recover the lost data automatically without involving the user. GAE also claims to offer a 99.95% SLA (Service-Level Agreement) of in any calendar month. However, a PaaS system cannot guarantee any reliability for applications that runs on top of the PaaS.

GAE also offers reliability mechanisms for application developers to take advantages the services provided in GAE. Specifically, it offers APIs to detect outage and schedule downtime for maintenance or outrage. Currently, availability of the blog-store, dataStore read, writes, image service, mail service, memcache service, task queue service, URL fetch service, and XMPP service capabilities are supported to enable the application to take actions in case that these services are down. GAE does not support fault tolerant mechanisms at the service/workflow level. The following code illustrates GAE APIs that support reliable computing:

```
CapabilityStatus status =
service.getStatus(Capability.DATASTORE_WRITE.getStatus();)
if (status == CapabilityStatus.DISABLED) {
 // Do something to compensate
}
```

Often a SaaS system keeps redundant data and metadata, as well as redundant load balancers.

3.3 Model-Driven SaaS In PaaS

Comparing to traditional software, a SaaS system has its unique software architecture and operation procedure as it is effective a consortium of tenant applications running in the same software, and each tenant application does not interfere with other applications. Furthermore, it also allows new tenant applications to be added into the consortium while other tenant applications are running. Thus, a SaaS system can be divided into two major functionalities: 1) tenant application development and 2) SaaS control and operation.

Tenant Application Development (TAD): This supports tenant application development via customization. For example, the following steps will be taken for a tenant to compose applications following the Grapevine approach:

1. Login onto the SaaS application;
2. Input the keywords and their relationships to the SaaS recommendation system;
3. After matching the tenant application requirements with the existing applications and components in the SaaS database, the recommendation system returns a collection of similar composed applications for customization with alternative GUI, workflow, service and data components from the SaaS database;
4. The tenant can choose one application from the set , and customize it with replacement components including those supplied by the tenant;
5. Evaluate the newly customized application via constraint analysis and simulation, and if the composed application passes the evaluation, the application, together with any new components that are supplied, will be updated in the SaaS database storage, and various relationships will also be updated so that the recommendation system can use for other tenants in the future.

SaaS Control and Operation (SCO): This supports SaaS executions including scheduling, handling user requests, distributed redundancy management and recovery, scalability including resource allocation and provisioning, and distributed and autonomous data migration for recovery and scalability. For example, a typical SaaS handles a user request in the following steps:

1. Determine the tenant ID of this request;
2. Check if the tenant application is already in the cache. If yes, send the request for the tenant application for execution;
3. If not, retrieve the tenant components from the SaaS database, and compose the tenant application using the retrieved components. After compiling the composed application, send the request to the just compiled code for execution;
4. After the execution, update the SaaS databases, and return the results to the user.

Currently, most of SaaS systems have been developed by software design and coding. This paper proposes to use a model-driven approach to develop the SaaS. Instead of developing the code for SaaS infrastructure and tenant components, a SaaS developer can focus on building the models for TAD and SCO, and used the automated code generation capabilities to generate the SaaS code and tenant components.. The tenant developers can also use the same modeling approach to develop tenant applications using the SaaS infrastructure on top of a PaaS system. Both models can be simulated before deployment. This paper uses the PSML (ACDATER)Tsai *et al.* (2009, 2006a) modeling language to model services and workflows for the SaaS development. This modeling approach has been used to develop large applications including semiconductor manufacturing control software and command-and-control software. One advantage of this approach is that once a model is updated, the code is re-generated to rapid deployment.

3.3.1 Modeling Language

ACDATER Model Tsai *et al.* (2009) includes Actors, Conditions, Data, Actions, Timing, Relations and Events. An Actor is a system with a clear boundary that interacts with other actors. A Condition is a predicate on data elements used to determine the course of a process. A Data element is an information carrier that represents the state of an actor or the status of the entire system. An Action represents an operational process to change state of an actor or perform computation. An Attribute specifies various properties. A timing element specifies the time constrains o Actors, Conditions, Data, Actions, and Relations. An Event represents an observable occurrence with no time duration, and it can be an input to an actor or an output from an actor. ACDATER can be used to model the services and workflows Due to the size limitation, this paper will describe the support for TAD only. Specifically, this paper addresses TAD customization and MTA, and with respect to customization, it will discuss modeling extension points and OIC/Grapevine approaches. One of key support of GAE for SaaS are various configuration files that can be used. For MTA SaaS, these files can be used to store tables needed for customization and MTA as shown in Table 3.1.

3.3.2 Modeling Customization with Extension Points

To support TAD using the extension-point (section 2) approach, the SaaS needs to have the following support:

1. Database support to store, search, and retrieve applications, workflows (with or without extension points) and services, Each extension points will store a list of

Table 3.1: Tables Required for Tenant-Development

Table Names	**Description**
Tenant Information Table (TIT)	This stores tenant c information.
Tenant Component Table (TCT)	This stores the tenant read, write, execution rights for various components.
Component Addressing Table (CAT)	This stores the mapping from components to their physical addresses.
Scalability Management Table (SMT)	This stores the scalability information for components.
Cache Management Table (CMT)	This store recent tenant activities.
Redundancy and Recovery Management Table (R-RMT)	This stores the redundancy and recovery details for the components associated with tenants.

options and each points to a workflow or service that can be used for the specific extension point. The pointed workflow can have its own extension points. The database will store the related information such as workflows, services, extension points, constraints for search and discovery.

2. The system will verify the any user supplied options by checking the constraints associated with the extension point.

3. System support to simulate the customized application.

Table 3.2 shows the modeling approach of using the ACDATER approach:

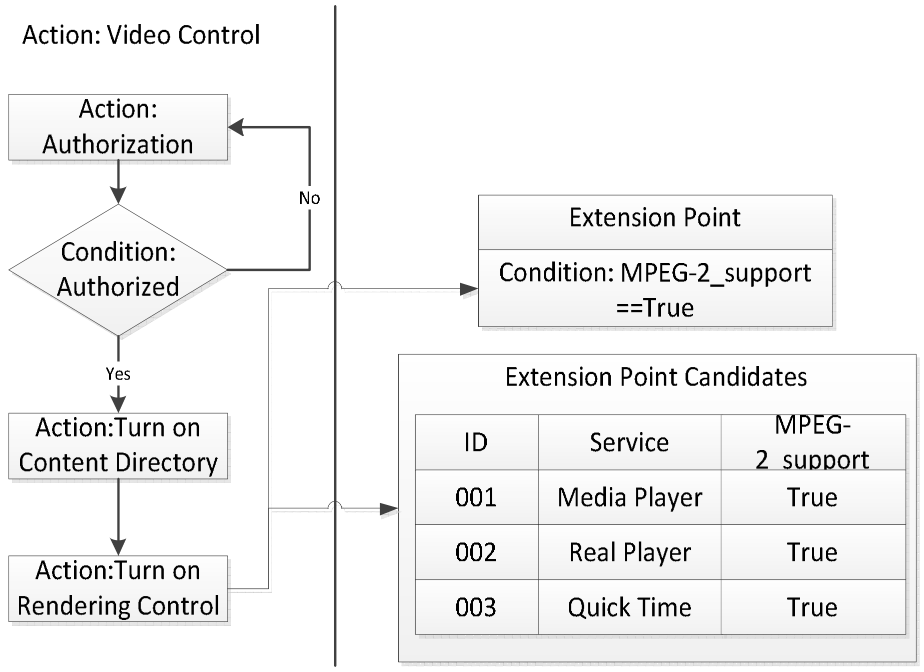
Table 3.2: PSML Modeling for SaaS

Entities to be Modeled	**PSML Modeling**
Workflows	An execution graph using if-then-else, sequence, conditions, and actions
An extension point	It will be a special node with constraints with alternative workflows or services as options
Constraints associated with an extension points	Conditions associated with an extension node
Events such as user request arrival	Events
GUI components	An actor with GUI attributes
Service components	An actor with IOPE attributes
Data components	Data.

The following diagram illustrates an example of this approach. The workflow is an Action that describes a user authorized to a video control system, turn on a content directory, and then turn on a video control. An extension point was specified with links to the video control (as alternative options). Each extension point can have multiple constraints, for instance in the example, it requires the MPEG-2_support proprieties to be true. And these constrains can be specified for selection the extension candidates. Each newly supplied option needs to be verified by the TAD system by executing the constraints associated with the extension point.

3.3.3 Modeling OIC/Grapevine Customization

To support OIC/Grapevine approach Tsai *et al.* (2010c), in addition to the database and simulation support described in Section 3.2, the system needs to have


Action: Video Control
Action: Authorization
Condition: Authorized
No
Yes
Action:Turn on Content Directory
Action:Turn on Rendering Control
Extension Point
Condition: MPEG-2_support ==True
Extension Point Candidates

ID	Service	MPEG-2_support
001	Media Player	True
002	Real Player	True
003	Quick Time	True

Figure 3.3: A PSML Action Describes a Video Control

1. A recommendation system, but this system does not need to be specified by tenant developers, but by the SaaS developer;

2. A tenant application development system to allow tenant developers to choose and compose applications using workflows using either retrieved and/or uploaded components;

3. A system to update various relationships among tenant components once a new tenant application is developed.

3.3.4 Modeling Multi-Tenancy Architecture

MTA needs to address the following features Tsai *et al.* (2011d):

- **Access right for each component:** Each tenant can access its own components used in its tenant application and data area;

Table 3.3: Tenant Information Table (TIT)

TenantName	**ID**	**Status**	**CPU**
MediaCtrlTenant1	Tenant1	off	4 CPUs
MediaCtrlTenant2	Tenant2	off	...
...	...	...	...

- **Component sharing constraints for each component:** Each component needs to specify those tenants that can have access;

- **Tenant resource addressing:** This stores either physical or virtual addresses for various resources for each tenant. The resources can be components in the SaaS database, cache, memory, data storage, and their constraints such as quota, upper limits and lower limits to support sandbox (for security reasons). This table will be dynamically updated during the SaaS execution. While the TAD will not involve in execution of these tables but the TAD is responsible to create this infrastructure.

Tenant Information Table (TIT): When a new tenant configuration arrives, the specified resource requirements such as CPU, memory, and storage will be stored in the TIT.

Each tenant has its own components and this information will be stored in a Tenant-Component Table (TCT). As a tenant may use selected components, all cross-product of tenants components will be a large but sparse table. Read(R), Write(W), and Execution(E) access rights can be specified in the TCT. As shown in the following table authorization, content directory and rendering control are the three components used in the workflow in the Video Control example.

The CAT (Component-Address Table) will store the virtual address provided by the PaaS for different components.

Table 3.4: Tenant-Component Table (TCT)

TenantID	**Authorization**	**Content Directory**	**...**	**Rendering Control**
Tenant1	RWE	RE	...	RE
Tenant2	RWE	RE	...	RE
...	...	...	...	...

Table 3.5: Component-Address Table (CAT)

Component	**ID**	**Address**
Authorization	Authorization1	Server0
ContentDirectory	ContentDirectory1	Server0
RenderingControl	MediaPlayer	Server1
RenderingControl	RealPlayer	Server2
RenderingControl	QuickTime	Server3

3.3.5 Scalability Management

To support the scalability, the SaaS system needs to know the location of each tenant application such as processor, cluster, and site information. The following table shows that for tenant1, the content directory and rendering control components can be scaled to 2 and 3 times.

3.3.6 Modeling Cache Management

The SaaS can manage its cache management using MemCache API provided by GAE using the information in the CMT). The next table shows the recent used tenant and the information related to this tenant in the cache.

Table 3.6: Scalability Management Table (SMT)

TenantID	**Authorization**	**Content Directory**	**...**	**Rendering Control**
Tenant1	1	2	...	3
Tenant2	1	1	...	1
...	...	...	...	...

Table 3.7: Cache-Management Table (CMT)

GUI component (x)	Authorization
Location	Server1
Copy	1
Which Tenant use	Tenant1
Recent visit	Timestamp of visit
Weight	3
...	...

3.3.7 Modeling Redundancy and Recovery

A SaaS system can store two or more copies of data and metadata of each tenant and various tables mentioned in the PaaS as shown in Table 8. Note that this will create six copies of data and metadata in the PaaS as GAE automatically triplicate the storage by default. Also, it will define recovery is required or not for the component specified.

3.4 Code Generation and Case Study

The PSML models including modeling, analysis, simulation, and code generation tools have been under development since 2000.

After modeling and customization, the tenant application model can be converted into executable code to be deployed in a PaaS system such as GAE.

Table 3.8: Redundancy & Recovery Management Table (RRMT)

Tenant ID	**Authorization**		**Content Directory**		**Rendering Control**	
	Redundancy ♯	**Recovery**	**Redundancy ♯**	**Recovery**	**Redundancy ♯**	**Recovery**
Tenant1	1	N	2	Y	2	Y
Tenant2	1	N	2	Y	1	Y
...	...	...	...	...	...	...

The first step is the translation process, where first Tenant Information Retrieval Algorithm. Tenant component details, access rights, physical addresses, scalability, R&R information are collected from the table-look-up interface in this step. Then, PSML model is serialized as XML file after service discovery. A BFS (breadth-first search) graph traversal algorithm and uses a queue to record traversed nodes (including the PSML Assign, Execute and Execute steps) Li *et al.* (2012a); Lee *et al.* (2009). When the node is pushed out of the queue, the corresponding XML generation function will use the collected tenant information and write the content to a XML file. In the meanwhile, it also calculates the subsequent unprocessed nodes and pushes them into the queue for further processing. The output of the algorithm is an XML specification of the workflow.

The second step is code generation, where each element is first put to code generator and the code generator generates the target code based on the CodeSmith templates. The templates are executable and they convert the XML descriptions of the models to executable code. Functions are built for each step, and workflow is retained through function invocations. Figure 3.4 specified the detail regarding the mechanism of the code generation template. It is essentially a series of converters mapping from the customized model to the target code.

Algorithm 2 Tenant Information Retrieval Algorithm

```
Input:
1: Component s, String tenantname
Output:
2: Details of the tenant information
3: Tenant t = LookupinCMT(tenantname);
4: if t == NULL then
5:     t = LookupinCMT (t);
6: end if
7: AccessRights rights = LookupinTCT(t, s);
8: Address addresses = lookupinCAT(s);
9: int scalability = LookupinSMT(t,s);
10: int redundancy = LookupInRRMT(t, REDUNDANCY);
11: bool recovery = LookupInRRMT(t, RECOVERY);
12: setTenantInformation(t, rights, address, scalability, redundancy, recovery);
```

Algorithm 3 Model XMLGeneration Algorithm

Input: XmlWriter xmlwriter, ExcutableElement start

Output: An XML file with all the workflow information

```
1: if start = null then
2:    Queue queue = new Queue();
3:    queue.Add(start);
4:    Set processedSteps = new Set();
5:    ExcutableElement next; setTenantInformation(invokeTenantRetrieval(next));
6:    while queue.Count > 0 do
7:       next = (Step)queue.Dequeue();
8:       if processedSteps.Contains(next.Id) then
9:          continue;
10:      end if
11:      Iterator iter = next.next();
12:      while iter.hasNext() do
13:         queue.Enqueue(iter.next());
14:      end while
15:      if next is StepAssign then
16:         GenerateStepAssign((StepAssign)next,xmlwriter);
17:      else if next is StepSelect then
18:         GenerateStepSelect((StepSelect)next,xmlwriter);
19:      else if next is StepExecute then
20:         GenerateStepExecute((StepExecute)next,xmlwriter);
21:      end if
22:      processedSteps.Add(next);
23:   end while
24: end if
```

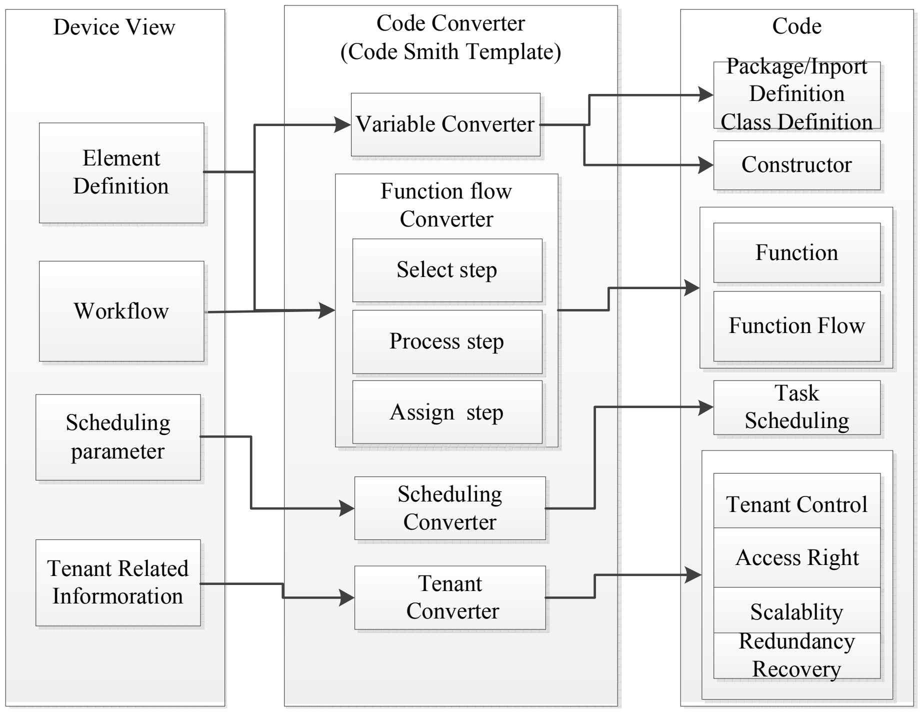


Figure 3.4: Mapping Process in Code Generation

Figure 3.6 shows a code generation example of the VideoControl. The left-hand side is the PSML element view, the middle is the XML model for this PSML model, the right-hand side is the generated Java code. Figure 3.5 is the runtime status of two tenants after deployed to GAE. With the generated code, GAE runs two different binaries for each tenant. Only one model is defined, however, with different tenant customizations, each tenant now has its own isolated execution environment. Figure 3.7 shows in GAE console with the two deployed tenants.

In addition to isolated execution binary, we can also define in the modeling phase to have two or more tenants share the same execution binary. To add more tenants, we simply need to go through Table 3.2 to Table 3.8 for the necessary customization information for the new tenants in the modeling/customization phase.

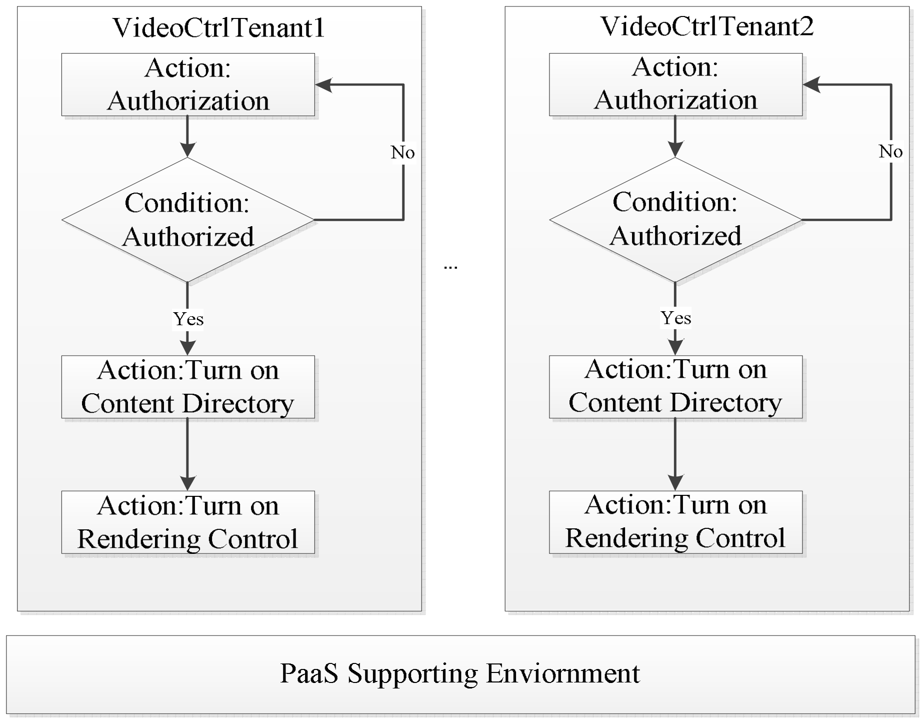
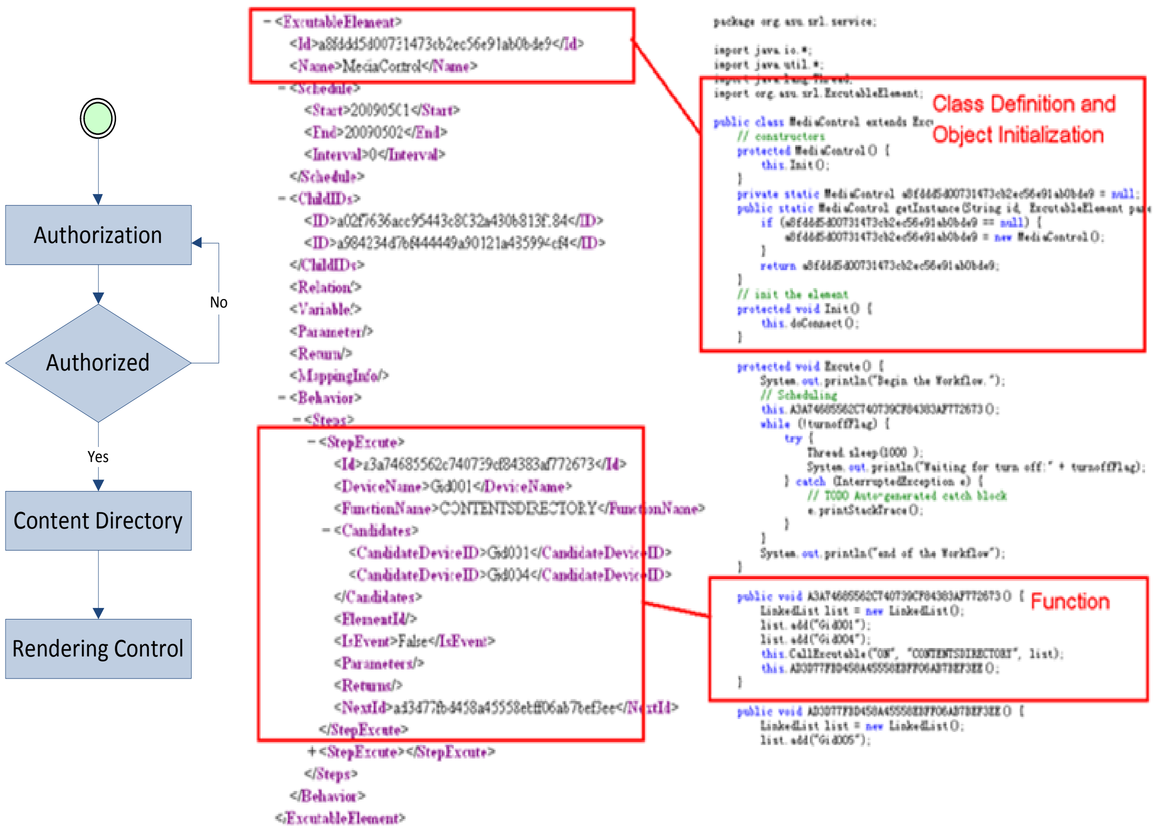

VideoCtrlTenant1
Action: Authorization
No
Condition: Authorized
Yes
Action:Turn on Content Directory
Action:Turn on Rendering Control
...
VideoCtrlTenant2
Action: Authorization
No
Condition: Authorized
Yes
Action:Turn on Content Directory
Action:Turn on Rendering Control
PaaS Supporting Enviornment

Figure 3.5: Two Tenants with Isolated Execution Instances

3.5 Conclusion

This paper described a model-driven development to develop tenant applications for a GAE-based SaaS system. The modeling language used in the PSML and it has associated modeling, analysis, simulation, and code generation tools. This paper demonstrated several tenant applications specified using PSML and generate code for GAE, and execute the code in GAE. Currently, developing a SaaS system is expensive and requires multi-year development with many engineers involved. However, the proposed approach of using the model-driven approach and use a commercially available PaaS system can save significant effort and time in developing a large, scalable and fault-tolerant SaaS system.



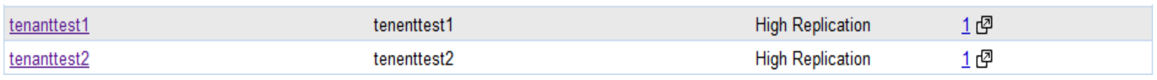
```xml
- <ExcutableElement>
    <Id>a8fddd5d00731473cb2ec56e91ab0bde9</Id>
    <Name>MediaControl</Name>
  - <Schedule>
      <Start>20090501</Start>
      <End>20090502</End>
      <Interval>0</Interval>
    </Schedule>
  - <ChildIDs>
      <ID>a02f7636acc95443c8032a430b813f.84</ID>
      <ID>a984234d7b6444449a90121a43599<cf4</ID>
    </ChildIDs>
    <Relation/>
    <Variable/>
    <Parameter/>
    <Return/>
    <MappingInfo/>
  - <Behavior>
    - <Steps>
      - <StepExcute>
          <Id>a3a74685562c740739cf84383af772673</Id>
          <DeviceName>Gid001</DeviceName>
          <FunctionName>CONTENTSDIRECTORY</FunctionName>
        - <Candidates>
            <CandidateDeviceID>Gid001</CandidateDeviceID>
            <CandidateDeviceID>Gid004</CandidateDeviceID>
          </Candidates>
          <ElementId/>
          <IsEvent>False</IsEvent>
          <Parameters/>
          <Returns/>
          <NextId>ad3d77fb4458a45558ebff06ab7bef3ee</NextId>
        </StepExcute>
      + <StepExcute></StepExcute>
      </Steps>
    </Behavior>
  </ExcutableElement>
```

```java
package org.asu.srl.service;

import java.io.*;
import java.util.*;
import java.lang.Thread;
import org.asu.srl.ExcutableElement;

public class MediaControl extends Exc
    // constructors
    protected MediaControl() {
        this.Init();
    }
    private static MediaControl a8fddd5d00731473cb2ec56e91ab0bde9 = null;
    public static MediaControl getInstance(String id, ExcutableElement pa
        if (a8fddd5d00731473cb2ec56e91ab0bde9 == null) {
            a8fddd5d00731473cb2ec56e91ab0bde9 = new MediaControl();
        }
        return a8fddd5d00731473cb2ec56e91ab0bde9;
    }
    // init the element
    protected void Init() {
        this.doConnect();
    }
    protected void Excute() {
        System.out.println("Begin the Workflow.");
        // Scheduling
        this.A3A74685562C740739CF84383AF772673();
        while (!turnoffFlag) {
            try {
                Thread.sleep(1000 );
                System.out.println("Waiting for turn off:" + turnoffFlag);
            } catch (InterruptedException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
            }
        }
        System.out.println("end of the Workflow");
    }
    public void A3A74685562C740739CF84383AF772673() {
        LinkedList list = new LinkedList();
        list.add("Gid001");
        list.add("Gid004");
        this.CallExcutable("ON", "CONTENTSDIRECTORY", list);
        this.AD3D77FBD4458A45558EBFF06AB7BEF3EE();
    }
    public void AD3D77FBD4458A45558EBFF06AB7BEF3EE() {
        LinkedList list = new LinkedList();
        list.add("Gid005");
```

Figure 3.6: Example Code Generation from PSML Model to Java Code



tenanttest1	tenenttest1	High Replication	1
tenanttest2	tenenttest2	High Replication	1

Figure 3.7: Two Tenants Deployed at GAE

Chapter 4

P4-SIMSAAS: POLICY SPECIFICATION FOR MULTI-TENDENCY SIMULATION SOFTWARE-AS-A-SERVICE MODEL

4.1 Introduction

Cloud computing receives significant attentions as it can enable the rapid delivery of computing resources as utilities in a dynamic, scalable, and virtualized manner. A lot efforts have been devoted to build cloud platform and cloud-based enterprise solutions, from both industry and research communities. Typical cloud products include Amazon EC2 Palankar *et al.* (2008b), Google GAE Google (2014), Microsoft Azure Microsoft (2014), Salesforce.com Salesforce (2014), VMware VMWare (2014), Eucalyptus Eucalyptus (2014), Citrix Citrix (2014).

Cloud-based simulation is an active research area. Simulation can benefit from cloud computing with its vast computing resources, scaling abilities, and an infrastructure support for MTA (Multi-tenancy Architecture). For examples, Lanner group Group (2014) designed a simulator L-SIM 2.0 to simulate the business process management systems through RESTful Web Services deployed in the cloud platform. Thomas Paviot (2010) used open-source software to develop a CAD simulator in a cloud platform to reduce the needs for expensive hardware and costly licensing scheme. Malik, Park and Fujimoto Malik *et al.* (2009) discusses about executing parallel and distributed simulation using a master/worker design for parallel discrete event simulation.

Simulation Software-as-a-Service (SimSaaS) Tsai *et al.* (2011d) was proposed as a new approach to simulate service-oriented software in a cloud infrastructure. It can

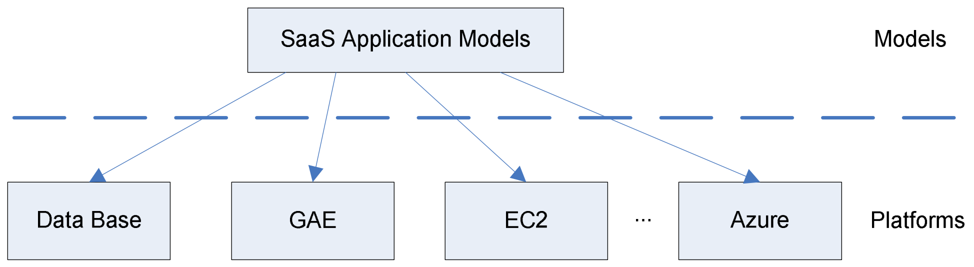


Figure 4.1: PaaS Independent SaaS Application Development

simulate a family of application using the same code base running in a PaaS (Platform as a Service) such as Googles GAE (Google App Engine), Amazons EC2, and Microsofts Azure. These PaaSs often supports tenant identification, isolation, addressing, and resource sharing. Sometimes a SaaS platform can also use a distributed database running on a clusters of processors. Often the SaaS infrastructure is fully integrated with the PaaS, for example, Salesforce.com CRM SaaS runs in a fully integrated environment Force.com. While full integration has significant advantages of performance, separating the SaaS from a PaaS has other advantages. Specifically, a platform-independent SaaS infrastructure can generate code to be run in different PaaS environments as shown in Figure 4.1. As each PaaS has its own unique design structure, dividing the SaaS application into this platform-independent SaaS development, followed by platform-dependent PaaS code generation and execution support reusability, portability, and vendor-neutrality. In this way, simulation also plays a critical role, as SaaS applications are developed, they can be simulated in a virtualized system, rather the target PaaS for testing and evaluation before deployment and execution in a PaaS. For example, one can develop a SaaS application, perform simulation, and once the application is considered successful, it will then be deployed to GAE for execution. Deriving from PaaS dependent to PaaS independent simulation models requires the model to be designed in a succinct and configurable way for the different cloud PaaSs.

The MTA SimSaaS needs to meet several goals: First, SaaS providers need to support tenants with a configurable code base, such that each tenant can configure its own version based on the same code base; Second, SaaS providers need to support various of tenants with a multitude of options of tenant-specific data, domain data and application metadata. SimSaaS provides two-level multi-tenancy support for SaaS simulation. At the data level, tenants can share domain data and service metadata while maintain tenant-specific data. At the application level, tenants can create their over versions based on the same configurable code base. However, MTA simulation is complicated due to diversified tenant-specific requirements and discrepant runtime behavior. The design of reconfigurable database and code base is usually hard.

This paper proposes P4-SimSaaS in the SimSaaS project. P4-SimSaaS introduces an ontology system for tenant specification with a policy system for tenants regulation. As each tenant may have its unique configuration, it is difficult to check all the issues at the design time, and thus some constraints will be enforced at runtime by the policy mechanism. In this way, the SaaS maintainers allow tenant to have freedom in composing their applications using services in the SaaS infrastructure, but enforce various constraints including security constraints at runtime or simulation time. This approach will simplify the design process for tenants while maintaining high integrity of SaaS applications. The ontology system can be used to specify the vocabulary, classification and relationships in different simulation domains, thus to facilitate simulation modeling including domain knowledge sharing, logical reasoning, policy specification, and simulation model discovery.

Tenants policies are designed and associated to tenants execution. A Policy system often consists a set of rules to regulate the related actions. Many policy languages are available, such as Rei as a formal and executable language to enforce constraints over allowable and obligated actions on resources. Pi4-SOA Zhou *et al.* (2006) is a policy

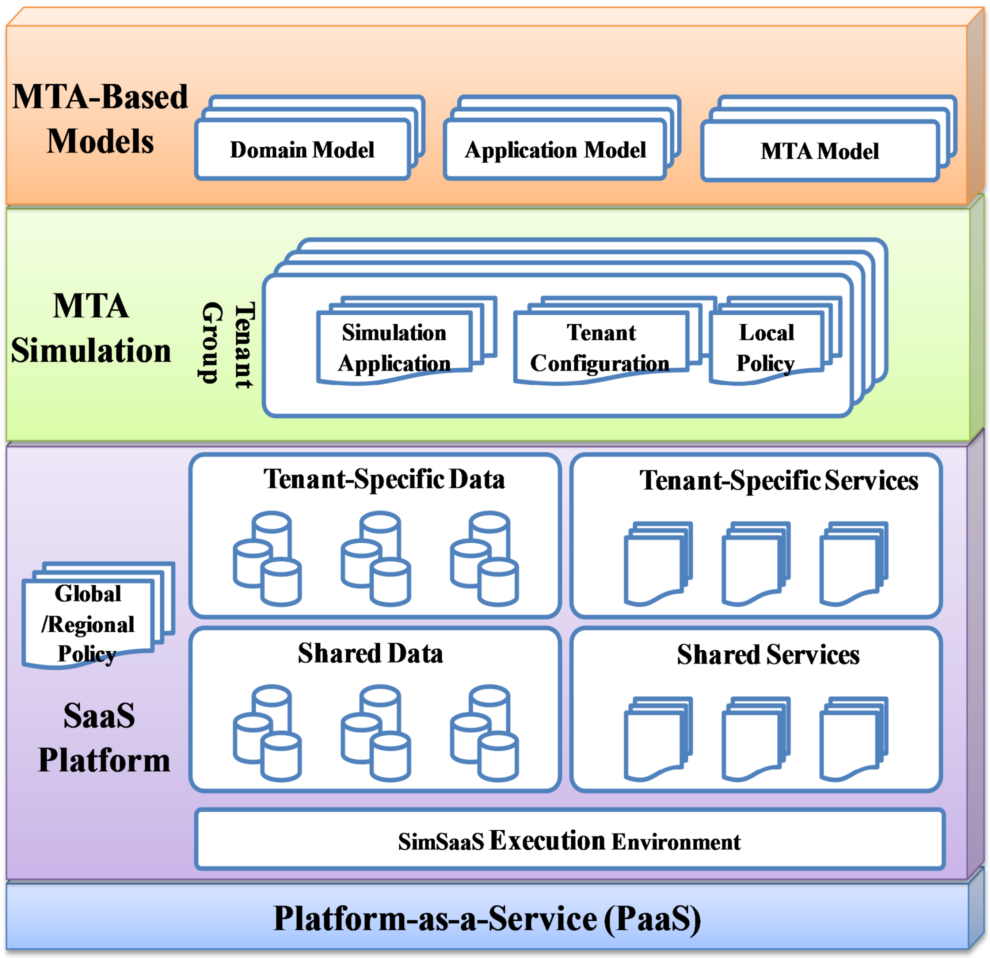


Figure 4.2: MTA Simulation Architecture

infrastructure for verification and control of service collaboration based on service metadata and collaboration patterns stored in the service registry. In P4-SimSaaS, MTA policies are constructed for the shared tenant model as well as tenant specific requirements. They are defined at three levels: global policies for shared code base, regional policies for groups of tenants, and local policies for individual tenant. Figure 4.2 shows an MTA simulation architecture. In the architecture, PaaS is the supporting environment for the SimSaaS Execution Environment, which is built at SaaS layer. SimSaaS supports the MTA simulations which are built from the MTA-Based Models.

This contributions of this paper are as follows:

- An ontology system for MTA simulation SaaS modeling and policy definition.
- A MTA policy specification for the shared tenant model and the multitude of tenant requirements.
- A modeling process that designs the information from the ontology system and associates the global, regional and local policies for the desired MTA simulation model.

This paper is structured as follows: Section 2 discusses ontology system design for MTA SaaS simulation. Section 3 describes the policy specifications related to the MTA SaaS simulation. Section 4 describes the consumer centric ontology based modeling process guided by the policies specified. Section 5 illustrates a case study for the proposed solution. Section 6 concludes the paper.

4.2 Ontology Systems

4.2.1 Ontology

The SimSaaS modeling and policy definition process discovers and reuses the data from the ontology system. The ontology system for SimSaaS has three major ontologies: Domain, MTA and Application ontology. Domain ontology stores the information related to the different categorization of the domains; MTA ontology contains tenant-related information, e.g., identities, sharing strategies, and access control strategies; Application ontology stores the metadata for composing the simulation model, e.g., participants, GUIs, services, workflows, and data. Domain ontology Tsai *et al.* (2010c) defines the knowledge in a specific domain; MTA ontology helps to configure the existing model with the tenant-specific information; Application ontology can be used to facilitate code generation for simulation.

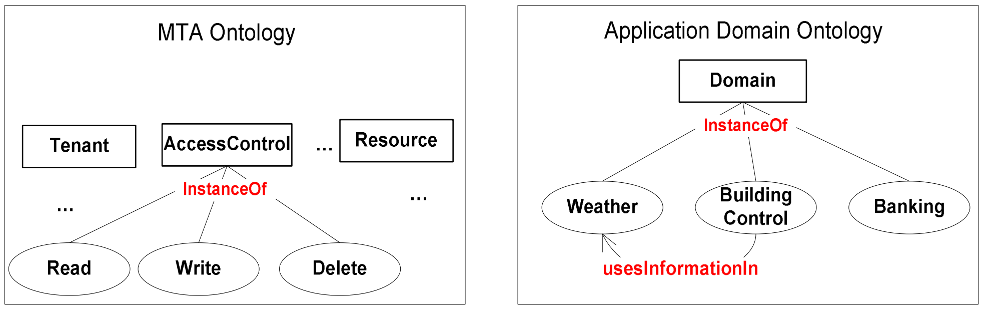


Figure 4.3: Sample MTA Ontology and Domain Ontology

Table 4.1: Classes in the MTA Ontology

Class	**Properties**	**Description**
Tenant	hasID, isClientOf, usesResource, usesCapability, providesCapability	Concept that represents individual tenant of the SaaS
Resource	hasID, hasCapacity, hasCost	Represents the resources that are available to tenants
Access Control		Type of access a tenant has for a resource. Instances include actions such as *read, create, write, modify,* and *delete.*
Capability	hasID, usesResource	A specific function or feature that a tenant can perform.

MTA Ontology contains the related information for supporting MTA. Earlier paper Tsai *et al.* (2011d) explained in details about the essential elements in supporting MTA, e.g., the identity, accessibility, sharing strategies of MTA. Other than that, scalability, security issues can be also included if required by the MTA design.

Table 4.1 shows some classes that are defined in the MTA ontology, and Table 4.2 shows some key properties and relationships.

Table 4.2: Relationships and Properties in the MTA Ontology

Properties and Relationships	**Relationship type**	**Description**
hasID, hasName	Identity	Provides identification for objects
isClientOf	business	Indicate a business relationship between tenants
usesCapability, usesResource, providesCapability	Usage	Relationships that define when resources are.
hasCapacity, hasCost	Resource	Relationships that define the magnitude or cost of using a resource or capability.

Domain Ontology: A collection of information about application domain ontologies. For example, the *Building Control* domain has *Building, Device, Service,* and *Room*; the bank domain has *Accounts* and *Securities.* Table 4.3 shows key classes in the domain ontology, and Table 4.4 shows key properties and relationships.

Figure 3 shows a portion of an MTA ontology and an Application domain ontology. In the MTA ontology, the classes *Tenant, AccessControl,* and *Resource* are shown. Some instances of *AccessControl* are also shown. For the Application Domain Ontology, three application domains are presented, *Building Control Weather,* and *Banking.* A dependency between the *Building Control* domain and the *Weather* domain is shown to represent the fact that some simulations of building control may rely on simulations of weather.

Application Ontology contains the metadata for the application model construction. The structure of Application ontology depends on domain content.

Table 4.3: Relationships and Properties in the MTA Ontology

Class	**Properties**	**Description**
Domain	hasID, hasNamespace	Concept that represents an application domain ontology

Table 4.4: Properties and Relationships in the Application Domain Ontology

Properties and Relationships	**Relationship type**	**Description**
hasID, hasName	Identity	Provides identification for domains
usesInformationIn	Usage	Relationships that define which domains use other domains.

Table 4.5 shows some classes in the Building Control Ontology, and Table 4.6 shows some key properties and relationships.

Figure 4.4 shows a sample building control application domain ontology for a smart home.

Boxes represent classes, ovals represent properties that can be assigned to objects of a given class, and arrows show the relationships that objects in different classes can have. For example, an object of type **Room** *hasID* (which defines its identity), *isType* (which would give the room some usage information), and is *ownedBy* someone. It isLocatedIn a object of type **Building**.

4.2.2 Relationships

Relationships express different types of associations among classes and items. Relationships exist within an ontology and cross different ontology systems. According to the relationships specified, a service provider can identify the related objects quickly for modeling. Properties are used identify and further classify individual objects.

Table 4.5: Classes in the Building Control Application Domain Ontology

Class	**Properties**	**Description**
Device	hasID, isType, ownedBy	Concept that represents a device or other equipment. Examples include a light switch, or an A/C control.
Service	hasID, isType, hasSpecification, hasQoS	Represents a computer-accessible service that provides either a user service, or access to a device.
Building	hasID, isType, ownedBy	Represents a building.
Room	hasID, isType	Represents a room
AccessControl	hasID, hasPolicy	Represents the access policies for a device or other object within the domain.
RoomUser	hasID, hasRole	Represents people that use a building.

Figure 4.5 shows three intra-ontology relationships (**categories, properties,** and **part-whole**), and two inter-ontology relationship (**use, typeof**). MTA ontology and Domain ontology are cross-referenced by **use** relationship. Application Ontology are cross-referenced by **typeof** relationships. The cross-referencing from Domain Ontology to MTA ontology is unique for MTA simulation since the simulations desires for the tenant information other than just for application metadata out of Application ontology.

These relationships can help the tenants to discover and reuse the data for designing and configuring their desired simulation applications and policies.

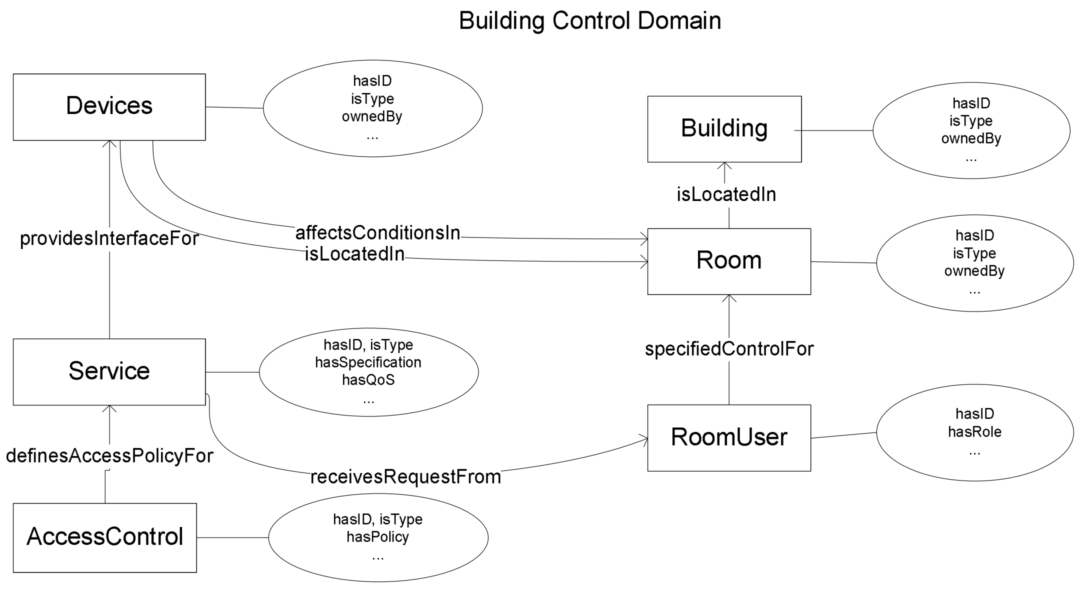


Figure 4.4: Sample Application Ontology in Building Control Domain

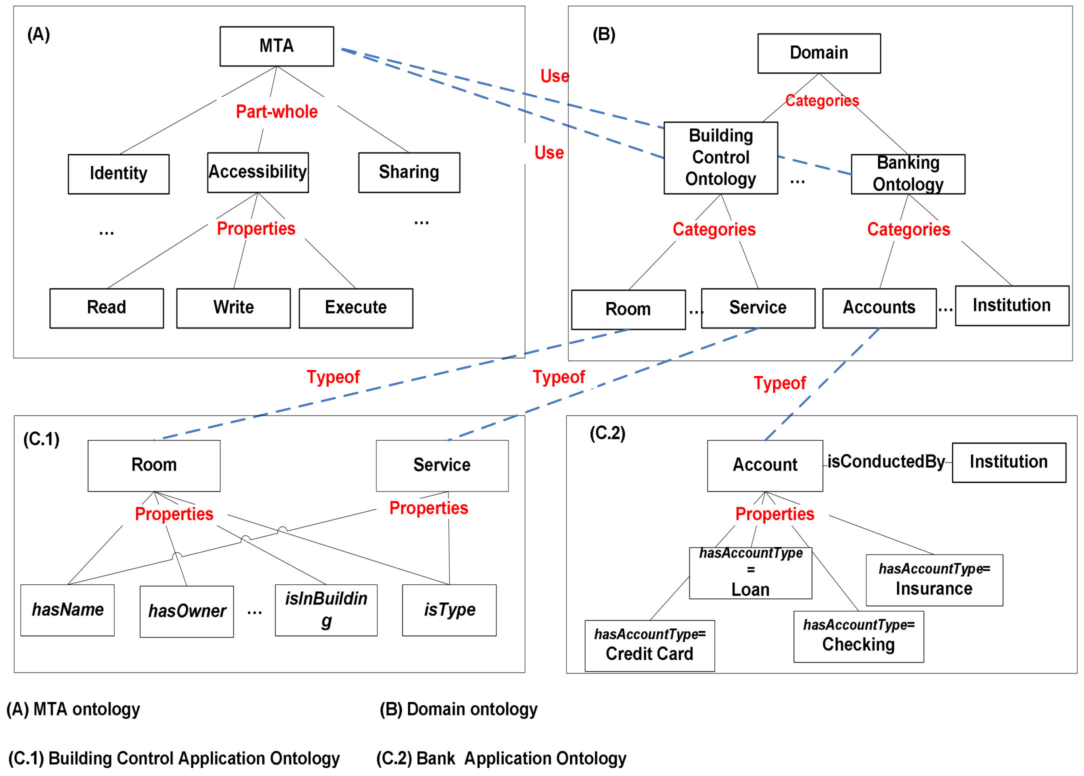


Figure 4.5: Cross Referencing among Ontologies

Table 4.6: Properties and Relationships in the Building Control Ontology

Relationships	**Relationship type**	**Description**
hasID, hasType	Identity	Provides identification for objects in the domain.
isLocatedIn	Location	Defines a locational relationship
ownedBy	Ownership	Defines object ownership.
providesInterfaceFor	Control	Defines what device a service provides an interface to.
affectsConditionsIn	Control	Specifies that a device can affect the conditions of a room or building.
hasSpecification	Usage	Defines the protocol for using a service.
specifiesControlFor	Control	Users can control buildings and rooms.
receivedRequestFrom	Control	Defines where requests are received from
hasPolicy	Security	Defines conditions for access and permission
hasRole	People	Defines information about people.

4.3 MTA Policy Specification

Policies can be used to enforce various constraints at simulation time, several policy languages are available already such as Rei , XACML Godik *et al.* (2002), Appel Turner *et al.* (2007), and PSML-P Zhou *et al.* (2006). As these languages have been developed before the development of MTA SaaS, thus they have not addressed MTA SaaS specific issues.

MTA simulation SaaS modeling process identifies the required model elements from the ontology system. This modeling process has two steps: tenant-independent modeling, and then configured this model with tenant-specific modeling. This approach handles the access control in the MTA simulation model. However, if the application is complex, it is difficult to incorporate all the requirements into the model, and it is also difficult for the SaaS maintainer to enforce various behaviors of all the tenants at runtime.

This paper propose using a policy approach to model constraints, and various policies can be specified to be enforced at runtime. Policies will be hierarchical such as global, regional, and local policies where a global policy will be enforced for all SaaS applications, a local policy will be enforced for a specific tenants, and a regional policy will be enforced for a group of tenants. In this way, each tenant just needs to model its unique features, while allowing various policies to enforce global, regional and local constraints. For example, a SaaS infrastructure administrator will maintain those global policies, such as security and privacy policies that must be enforced for all SaaS applications and tenants, a tenant administrator will maintain local policies to ensure that its customer applications running the SaaS applications are executed in a secure and fair manner; another tenant administrator may maintain a set of regional policies as it is a part of a business consortium with other tenants. In this way, the simulation model can be lightweight, flexible and easy to maintain while sophisticated as constraints will be specified and maintained by different groups of people for different purposes, making the SaaS application easier as it does not need to address all these issues at the same time.

A sample global policy is that the SaaS infrastructure may request additional processors if the current workload of all the tenants has exceeded 50% CPU usage for elastic computing, a key feature of cloud computing. A sample regional policy can be as shutting down TVs after 9 PM for tenant applications related to school dormitories, and a local policy can be lowering TV volume by 20% after 10 PM for tenant related to TV control in a given area.

A policy usually includes three components: a target, a condition set, and an action. In a MTA SaaS environment, a policy needs to consider two more elements: the application domain and the tenant.

Definition: Range

Let R be a finite set of Ranges, R = $\{r_1, r_2, ..., r_n\}$, a range $r_i \in$ R for a policy is a tuple $r_i = < d_i, te_i, ta_i >$, where

- d_i is the domain ID for the policy, and $d_i \in$ *DomainOntology.*
- te_i is the tenant ID for the policy, and $te_i \in$ *MTAOntology.*
- ta_i is the target, and it represents the target of the policy, and $ta_i \in$ *MetadataOntology.*

Definition: Condition

Let C be a finite set of Conditions, C = $\{c_1, c_2, ..., c_n\}$, a condition $c_i \in$ C where each c_i can be any of the following three possibilities:

- c_i is an crispy condition expression that the value of $c_i \in \{true, false\}$;
- c_i is an fuzzy condition (Kosko 1991)that the value of $c_i \in \{Fuzzy\_values\}$;
- c_i is an adaptive condition that the value of $c_i \in \{adaptive\_datastructure\}$, that the adaptive data structure can be a condition tree in implementation.

In a crispy condition, the value of the condition expression can only be true or false. The bound of the truth value is firm under this scenario. For instance, the temperature is over 80. And the policy action is triggered provided that the range and condition factors are both meet.

In a fuzzy condition, the condition expression is a fuzzy value. For instance, the condition can be defined as temperature around 80. Under this case, the fuzzy value treats both temperature 79 and 81 as around 80. Thus the policy value could be triggered by both 79 , 80 and 81. More detailed about fuzzy theories can be followed in Biacino and Gerla (2002).

In an adaptive condition, the condition value can be changed by different contexts. For instance, in the spring, when the temperature is over 82, trigger the action turn on AC. In the summer, when the temperature is over 78, trigger the action turn on AC.

Definition: MTA Policy

Let P be a finite set of policies, P = $\{p_1, p_2, ..., p_n\}$, a MTA policy $p_i \in$ P is a tuple $p_i = < pid_i, R_i, C_i, A_i >$, where

- pid_i is id for the policy, that is unique to identify each policy and $\forall p_i$, $p_j \in$ P, if i $\neq$ j, $pid_i \neq pid_j$;
- R_i is the range id for a policy, that $R_i \in \mathit{Range}$ that defined above;
- $C_i = \{c^i_1, c^i_2, ..., c^i_m\}$ is the set of conditions for p_i, that $C_i \in \mathit{Condition}$ that defined above;
- A_i is the action associated with the policy. It denotes that A_i will be executed provided R_i AND $C_i ==$ true.

Definition: Global Policy

Let P be a finite set of policies, P = $\{p_1, p_2, ..., p_n\}$, a global policy $pg_i \in$ P where $te_i ==$ null. A global policy applies to all tenants.

Definition: Regional Policy

Let P be a finite set of policies, P = $\{p_1, p_2, ..., p_n\}$, a regional policy $pg_i \in$ P where $te_i = \{te_{i1}, te_{i2}, ..., te_{in}\}$. A global regional applies to a group of tenants.

Definition: Local Policy

Let P be a finite set of policies, P = $\{p_1, p_2, ..., p_n\}$, a local policy $pl_i \in$ P where ta_i != null. A local policy is applicable to a specific tenant.

Definition: Policy Set

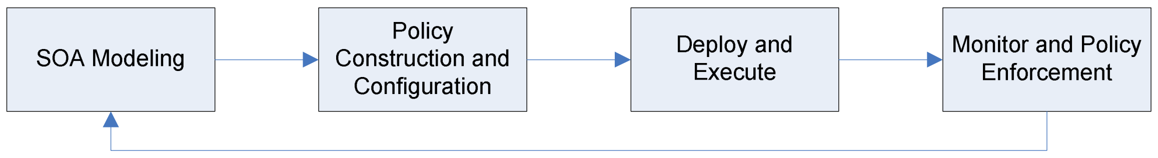
Let Pset be a finite set of policy sets, Pset = $\{pset_1, pset_2, ..., pset_n\}$, a policy set $pset_i \in$ Pset where each Pset = $\{p_1, p_2, ..., p_n\}$, where $p_1, p_2, ..., p_n$ are policies.

The case study part will demonstrate policies stored in the XML format. Also, it will demonstrate the usage of global and local policies, and policy sets.

4.4 Modeling and Simulation Process

This section demonstrates the constructing of the simulation model by identifying and reusing the data from the associated ontology systems, and specifying policy for different tenants including global, regional, and local policies to be enforced at simulation time.

To make the modeling process efficient, approaches such as Consumer-Centric SOA (CCSOA) Tsai *et al.* (2006a) can be used. In CCSOA, in addition to publishing simulation services, simulation requirements and simulation workflows can also published, discovered and used. Once these requirements are published, a service provider can submit their software or services to meet the application requirements. A service consumer can compose the existing service using composition languages such as PSML and BPEL.


SOA Modeling
Policy Construction and Configuration
Deploy and Execute
Monitor and Policy Enforcement
Figure 4.6: Four-Step Cyclical SimSaaS Modeling and Simulation Process

The modeling and simulation process follows a service-oriented approach as characterized in the following cyclical four-step process(Figure 4.6):

1) **SOA Modeling:** In the context of MTA SaaS, each simulation model can be treated as a service and it can be published, discovered and composed.

1. Publishing: a service can be published so that it can be discovered and used by others, and as policies can be published and reused by others as well;

2. Discovery: a service can be discovered using various search strategies and algorithms can be used to select the best services to be used among discovered services, and policies can also be discovered by others including tenants, end users, and SaaS maintainers;

3. Composition: Selected services can be connected by a workflow to form a new application, and policies can be composed by reusing several policies.

As a MTA simulation model, abundant simulation model candidates from the ontology system can offer abundant options for the modeling process. To solve this issue, different service selection and ranking algorithm can be applied, including reputation based ranking, usage based ranking, QoS-based service selection and ranking with trust and reputation management Vu *et al.* (2005) by EPFL, context-sensitive ranking algorithm Haveliwala (2003), social closeness ranking and even correlation ranking based on a topology model.

2) **Policy construction and tenant configuration:** The constructed simulation model can be configured with the MTA properties and also be associated with

the MTA policies for the tenant constrains. Policy and tenant information should be specified once the tenant-independent models are created. Meanwhile, these constructed models can be configured directly using the Tenant Configuration model specified in SimSaaS. This configuration model includes the information specified in MTA ontology, including identity, accessibility, and sharing strategies.

In composing the policy, the ontology system is used. The cross referencing among the ontology can match the Application ontology from Domain Ontology, then query the associated Application ontology for the desired conditions and actions. For instance, in a building control definition, we can construct a global policy P0 for tenant A, if the CPU usage is over 80%, allocates one more CPU for the application.

One can also define local policies, P1 for tenantA, for instance, if temperature is over 80, then turn on the AC. However, another policy p2 for tenant B, if the temperature is over 80, turn on the fan.

Global policies are categorized to the PolicySet regarding the associate application. Regional policies are categorized to the PolicySet regarding the associate tenants. Local Policies are categorized to the PolicySet regarding to the associate tenant.

Algorithm 4 Monitoring and Policy Enforcement

```
Input:
1: Ranked policy set, Simulation model
Output:
2: Simulation result
3: StartMonitoringSimulation();
4: while more simulations do
5:    Log();
6:    Type type = SelectPolicyExecutionAlgorithm();
7:    if type == FYI then
8:       while hasPolicy() do
9:          executePolicy();
10:         executeSimulation();
11:      end while
12:      Log();
13:   else if type == conservative then
14:      while !checkPolicy() do
15:         Log();
16:      end while
17:      executePolicy();
18:      executeSimulation();
19:      Log();
20:   else if type == greedy then
21:      Rollbackpoint roolbackPoint = setRollBackPoint();
```

```
        executePolicy();
        executeSimulation();
        Log();
        if !checkPolicy() & PolicyNoneViolable() then
            rollback(rollbackPoint);
        end if
        Log();
        ReRankPolicies();
    end if
end while
```

3) **Deploy and execute:** The composed application can be deployed in a cloud environment to be executed; Similar to DDSOS Tsai *et al.* (2006c), simulation deployment in SimSaaS can support both the on demand and automated way.

Various issues requires to be considered for MTA simulation execution. Including runtime tenant management, tenant physical and logical addressing, tenant resource allocation, and capability control. The details about simulation deployment and execution can be followed in earlier paper SimSaaS.

Before the policies are deployed, the policies are configured in different policy sets and with each police sets, the policies are ranked by the policy editors.

4) **Monitor and policy enforcement:** The execution of the simulation can be monitored and various runtime policies can be enforced.

As shown in the Algorithm 4.4, the service execution and policy enforcement processes are monitored by SimSaaS infrastructure. While monitoring, the execution processes of simulation services and the polices are stored through the distributed tracing infrastructure in PaaS. Frameworks such as Dapper from Google Sigelman *et al.* (2010), Magpie Barham *et al.* (2003) from Microsoft, and X-trace Fonseca *et al.* (2007) from UC Berkeley can be used as the distributed tracing infrastructure. Later on, data provenance mechanism can be applied to these tracing data for simulation analysis, simulation tuning, and policy ranking.

Policy enforcement can be categorized into three ways Tsai *et al.* (2006b) including:

- For your information (FYI) algorithm: the policy engine does not perform policy checking at runtime. It simply do the static checking before policy enforcement. In runtime, the policy engine only logs the relevant information into log file for analysis. It is not suiable for safety Ccritical processes.
- Conservative algorithm: the simulation engine stops whenever it needs to check the policy enforcement point (PEP). It holds the simulation step until the next step can be confirmed safe. If the PEP needs to check lots of polices in simulation, this algorithm will consume a large amount of time.
- Greedy algorithm: similar to the conservative algorithm but that the execution and condition checking are separated to paralleled two threads. The simulation execution thread will execute as far as it can go however, the condition checking thread will wait for the PEP for the result. A rollback point is saved before the simulation execution thread. And if the PEP returns out as false, the simulation execution thread can choose to roll back or do an compensation or proceed depending on the extend of the systems compromise for the violation of the policies.

Moreover, policies may conflict with each other. For instance, a local policy can conflict with a global policy, or two local policies conflict with each other within a tenants scope. These cases can be solved by doing the consistent and completeness checking over the policies Zhou *et al.* (2006), or applying policy combination and integration algorithms such as Li *et al.* (2009); Mazzoleni *et al.* (2006).

A rerank of the policies can be done based on the traced date from the tracking framework. These ranking can be updated to the existing policy set and it will replace the original policy ranking results.

4.5 Case Study

The case study involves two tenants ABCDorm and XYZHome in Building control domain. Both models are derived from the building control simulation, whereas, configured by different tenant configurations specified by SimSaaS. The requirements and constrains of both tenants are listed in the following Table.

4.5.1 SOA Modeling

Following the SOA modeling process(publish, discovery and composition), Dormitory Control Simulation and Home Control Simulation are constructed separately for the two tenants ABCDorm and XYZHome. The discovery process uses the ontology system. The cross referencing among the ontologies can be beneficial to find the desired service in an easier way. For instance, we can first query Domain ontology to narrow down the Building Control Application Ontology for designing the simulation model. In that, it is easier to discover the desired services for composing the functions.

To simplify the scenario, this case study only involves a few services including TV Control, Temperature Control and Kitchen Control. The overview constructed model can be seen in Table 4.8. It specified the related services and required involved data for the two different simulations.

Table 4.7: Requirements for ABCDorm and XYXHome Control Simulation

		ABCDorm	XYZHome
Simulation Model		Dormitory Control Simulation	Home Control Simulation
Tenant Specific Requirements	Logo	Yes	Yes
	♯ of Users	30	3000
	Bandwidth Usage	0.1 Gbytes	1 Gbytes
	CPU time	0.5 CPU hours	10 CPU hours
	Sample Constrains	1. Shut down TV after 10 PM to TV control in living room. 2. If temperature is over 80, then turn on the AC.	1. Lowering TV volume by 20% after 10 PM to TV control in living room. 2. If the temperature is over 80, turn on the fan.

4.5.2 *Configuration and Policy Construction*

Once the simulation model is done, we can start configure these models with the tenant information. The two models come with their own simulation models.

According to SimSaaS tenant configuration. The following configurations can be done to the two different models.

$$\text{XYZHome} \rightarrow \text{Home Control Simulation}$$
$$\text{ABCDorm} \rightarrow \text{Dormitory Control Simulation}$$

For the different tenants, different policies are constructed by the tenant constrains. These policies associate with the MTA SimSaaS model in the cloud infrastructure.

Table 4.8: Overview of Sample MTA Simulation Models

	Dormitory Control Simulation	Home Control Simulation
Services	TV Control, Temperature Control	TV Control, Temperature Control, Kitchen Control
Data	Samsung TV, GE Air Conditioning	LG TV, Vornado Electricity Fan, Haier Microwave, GE Refrigerator

Note that as a global policy p5 applies to all the simulation runs at the SimSaaS run-time including Dormitory Control Simulation and Home Control Simulation. P1 and P2 only belong to tenant ABCDorm. P3, P4 only apply to XYZHome. According to PThus when a CoolDown functions are invoked in both simulations, ABCDorm will turn on the AC, while in XYZHome it will turn on the electricity fan.

4.5.3 Deployment and Execution

The Figure 4.7 shows an deployment overview of the whole case study. P1 and P2 are configured for ABCDorm, and P3 and P4 are configured with XYZHome. P5 is configured at global level. The Figure also tells that the global and regional policies stay at the global level compare to local policies that stay with the individual tenants.

Policy offers more flexibility for the simulation details of different tenants although they share the same service. Moreover, the overhead for the service designs can be dispersed to the different policies of the tenants. Thus it can both benefit to the service provider and the simulation execution engine.

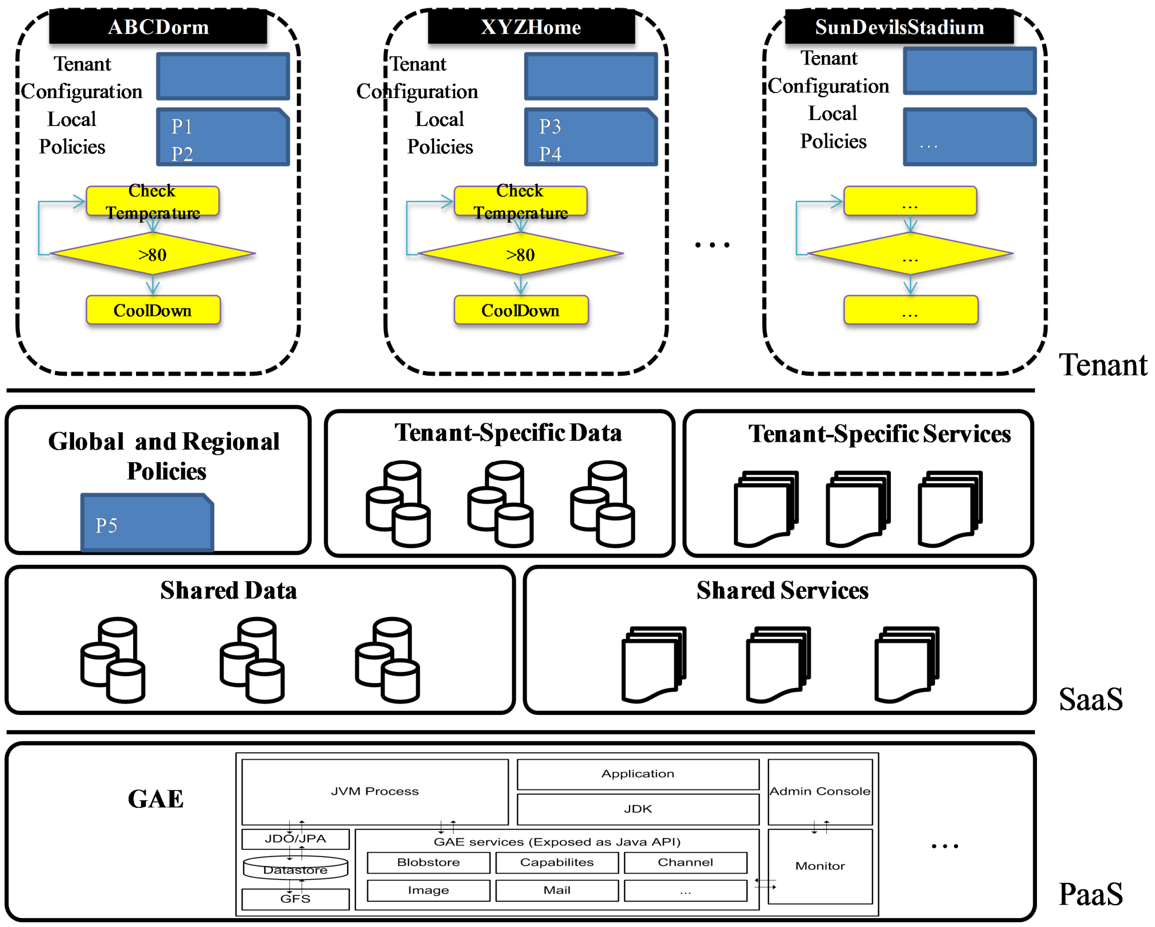


Figure 4.7: Deployment OverView of Global, Regional and Local Policies

4.5.4 Monitoring and Policy Enforcement

The monitoring and execution of the policies follows algorithm in section 4.4. For instance, when ABCDorm Simulation is put to execution. The SimSaaS simulation engine starts to monitor the whole process.

The ABCDorm has a simulation temperature control. Before the execution of the temperature control, it first checks which simulation algorithms it desires to take. For instance taking the greedy algorithm for temperature control, the simulation engine starts the execution of temperature control service before the P2 is evaluated. Taking this way, the simulation can save the time in setting up the environment such as starting the AC. If the policy condition does not meet, a compensation operation of warming up can be done.

4.6 Conclusion

This paper presented an ontology based framework and the tenant related policies, to support building an flexible simulation models that can meet the variability of tenant-specific requirements in SimSaaS. MTA simulation model construction can benefit from the MTA ontology system provided, and this ontology can also be used for the policy design. The specified policies can be used to meet the different requirements of the tenants. The innovative solution can greatly save the time in tenant modeling and execution of SimSaaS. A case study is offered to demonstrate the entire framework.

Table 4.9: Sample Policies for MTA Simulation Models

Policy ID	Policy Description	Policy Type	Associated Tenant
P1	Within a range ABCDorm in Building control domain; If condition time after 10PM is meet; Trigger the action shutting down TV in living room.	Local Policy	ABCDorm
P2	Within a range ABCDorm in Building control domain; If condition temperature over 80 is meet; Trigger the action turning on the AC.	Local Policy	ABCDorm
P3	Within a range XYZHome in Building control domain; If condition time after 10PM is meet; Trigger the action of lowing down the volume of TV to 20% in living room.	Local Policy	XYZHome
P4	Within a range ABCDorm in Building control domain; If condition temperature over 80 is meet; Trigger the action turning on the Electricity Fan.	Local Policy	XYZHome
P5	Within a range Building control domain; If the CPU usage is over 80%; Trigger the action of allocating two more CPUs.	Global Policy	All tenants in Building Control Domain

Chapter 5

TIMING SPECIFICATION AND ANALYSIS FOR SERVICE-ORIENTED SIMULATION

5.1 Introduction

This paper deals with time-based service-oriented simulation. Modeling time is important for service-based software systems and necessary when services must complete their tasks within specified timing constraints. Service-oriented models such as WSDL and BEPL do not support specifying time yet even though the timing constraints can be added. Thus, services with timing information stands to significantly strengthen design, implementation, and testing of complex, real-time atomic and composite services.

In this paper, the PSML (Process Specification and Modeling Language) Tsai *et al.* (2007a) that supports development of service-based software system is extended to quantify timing of services. DEVS (Discrete Event System Specification) ASU (2014) is a component-based modeling and simulation approach and it supports specification of timing.

5.2 Related Work

Service-oriented simulation has been an active research area recently, and some existing projects include XMSF (Extensible Modeling and Simulation Framework) XMSF (2007), the simulation grid system Cosim-Grid Li *et al.* (2005) and GridSim Buyya (2008), Interface Simulation and Testing Framework (ISTF), Dynamic Distributed Service-Oriented Simulation (DDSOS) Tsai *et al.* (2006c), Dynamic Service-

Oriented Collaboration Simulation (DSOCS) Tsai *et al.* (2007c), simulation framework with Microsoft Robotic Studio Tsai *et al.* (2008d), and SOA DEVS (SOAD) Sarjoughian *et al.* (2008). XMSF creates a modeling and simulation framework that utilizes a set of web-enabled technologies to facilitate modeling and simulation applications. XMSF involves Web/XML, Web services, and internet/networking to improve the interoperability. Cosim-Grid is a service-oriented simulation grid based on HLA, PLM (Product Lifecycle Management) IBM (2007), and grid/web services. It applies OGSA (Open Grid Services Architecture) Globus (2007) to modeling and simulation to improve HLA in terms of dynamic sharing, autonomy, fault tolerance, collaboration, and security mechanisms. GridSim is an open-source simulation framework that allows users to model and simulation the characteristics of Grid resources and network. It provides intensive support for grid simulation, such as workload trace simulation, jobs allocation, and network traffic simulation Buyya (2008). ISTF INOA (2014) is an extensible SOA simulation tool, and it simulates the end-to-end distributed application scenarios and demonstrates how individual components will interface with each other in production.

DDSOS is an SOA simulation framework that provides simulation runtime services and supports. DDSOS has runtime infrastructure (RTI) like the one in HLA. Within this framework, simulation code can be dynamically generated and configured whenever demanded by the users. The DDSOS framework provides the two layers of modeling support by PSML, an on-demand automated dynamic code generator (service) can generate executable code for simulation and for real applications directly from the model (specification) written in PSML, an on-demand automated dynamic code deployment service, an simulation engine engines, and an extended RTI.

As SOA development is mainly model-driven, and it is different from traditional software development, thus SOA simulation will be model-driven and the modeling languages used will have a great impact on the lifecycle of service-oriented application development. Each component in the simulation framework, including the simulation engines as well as application components and tasks, are modeled as services or workflows using various modeling languages such as BPEL or PSML-S Tsai *et al.* (2007a) , and DEVSML Mittal *et al.* (2007). The activities that can be simulated include service publishing, service discovery, service composition, dynamic architecture, reconfiguration, dynamic collaboration, policy enforcement. For example, PSML can model service-oriented applications, simulation code can be automatically generated, and various kinds of analyses can be performed on the model specified.

SOA allows services to be discovered and composed at design or runtime, and for those systems composed at runtime, its behavior is determined at runtime. The DSOCS framework Tsai *et al.* (2008d) addresses this issue by integrating the SOA dynamic collaboration and simulation concepts, and it supports modeling and simulating systems with the distributed, interactive, and discreteCevent driven focuses.

5.2.1 Service-Oriented Simulation Approaches

One approach to service-oriented simulation views simulation as an integrated part of application development. Simulation is used to verify a service-oriented model, and also used at runtime to validate that the application execution produces the same results as simulation Tsai *et al.* (2008d, 2007a). Another important feature is the policy enforcement which often needs to be simulated Tsai *et al.* (2008b), as policy software cannot be executed alone, before policies can be deployed to enforce constraints. In this way, service-oriented simulation carefully follows the service-oriented models and lifecycles. In this approach, the goal is not to simulate the support function of SOA, but instead use these services as a part of simulation infrastructure.

Another approach for service-oriented simulating outgrows from the DEVS framework. The SOA DEVS (SOAD) approach is developed to support representing and simulating service-oriented applications Sarjoughian *et al.* (2008). In this approach, DEVS formalism and the SOA principles together enable simulating basic aspects of service-oriented applications such as discovery, composition, publication, and subscription. Currently, the DEVS-Suite is an object-oriented simulator implemented in Java ASU (2014). Unlike DEVS/SOA which enables execution of models as services Mittal *et al.* (2007), the DEVS-Suite simulator is not service-based.

It is important for SOA DEVS models to be simulated as services and take advantage of dynamic publication, discovery, composition, and policy enforcement offered by SOA. This is similar to an object-oriented simulation, which was mainly based on object-oriented framework and designed to take advantage of polymorphism, dynamic binding offered by object orientation.

This paper provides a way to unify these two approaches by incorporating the timing information into the service-oriented modeling language PSML.

5.3 Specifying Timing in Service-Oriented Model

This paper proposes several timing specification techniques to the service-oriented model language PSML (Process Specification and Modeling Language) based on ACDATER (Actors, Conditions, Data, Actions, Timing, and Events, Relations). Specifically, this paper uses Delay, Min (for minimum time needed), Max (for maximum time needed), Deadline, and Distribution to Actors, Conditions, and Actions.

5.3.1 Timing Specifications

The timing information is specified by various guards including uni-guards, bi-guards, and n-guards. As Data elements do not have timing information associated with them, only Actor, Condition, Action elements have timing specifications. An event is considered as an epoch. Each model can have zero or more guards, referred as the guard set, to specify timing and other relationships among ACDATER elements.

Uni-Guards: These express timing constraints involving a *single* element and they can be applied to Actors, Conditions and Actions. There are four different types of uni-guards:

- Delay specifies the minimum time that the concerned elements must wait before computing. Delay is often used for real-time processing to establish the environment before execution. An example is a heart defibrillator where its capacitor needs to be charged before applying therapy.

- Deadline specifies the time within which the computation or communication must be completed.

- Min specifies the minimum amount of time that computation or communication will take.

- Max specifies the maximum amount of time the computation or communication will take.

- Distribution specifies the time distribution function for computation or communication. The distribution largely depends on the characteristics of the input such as size. A distribution function can be deterministic or stochastic such as Poisson distribution.

A deadline associated with an element can be from three sources following the three-party structure of SOA: providers, clients, or brokers.

- A deadline imposed by a provider can be a QoS guarantee that the provider is willing to provide for all of its clients.

- A broker may act as independent agent to verify the deadline provided by a provider satisfies the stated specification. With the brokers assurance, a client has more confidence of using the timing specification provided by the provider.
- A deadline imposed by a client is the requirements that the client desires for a specific application, and the provider may or may not be able to provide.

In ACDATER specification, timing specifications provided by providers are associated with elements, but timing requirements for clients are carried by the current active process during simulation. As a computation is being carried out, the timing requirements by clients will be updated (e.g., at some regular time intervals) to see if the client requirements are being met. Furthermore, based on the current status, the active process may make runtime decision to select different services for execution based on the timing specification associated with elements supplied by providers. This will be further discussed in Section 7.

Note that a Deadline should be equal or smaller than the sum of Max and Delay. Otherwise the Deadline is meaningless.

Figure 5.2 shows a simple workflow for an account update process to illustrate the above timing concepts. It consists of three services login, update, and logout. The workflow is modeled as an Actor; login, update, and logout services are modeled as Actions; IsBlank is a condition. The update service needs to verify the IsBlank condition is satisfied before the update is performed. The login service can have timing constraints in Figure 5.1:

$$\text{Delay (login)} = 0;$$

$$\text{Deadline (login)} = 0.09;$$

$$\text{Min (login)} = 0.01;$$

$$\text{Max (login)} = 0.1;$$

$$f(x)=\left\{\begin{array}{c}0.01, 0 < x \leq 10 \\ 0.05, 10 < x \leq 100 \\ 0.08, 100 < x \leq 1000 \\ 0.1, x > 1000\end{array}\right\}, x \in N^{+}$$

Figure 5.1: Timing Constraints

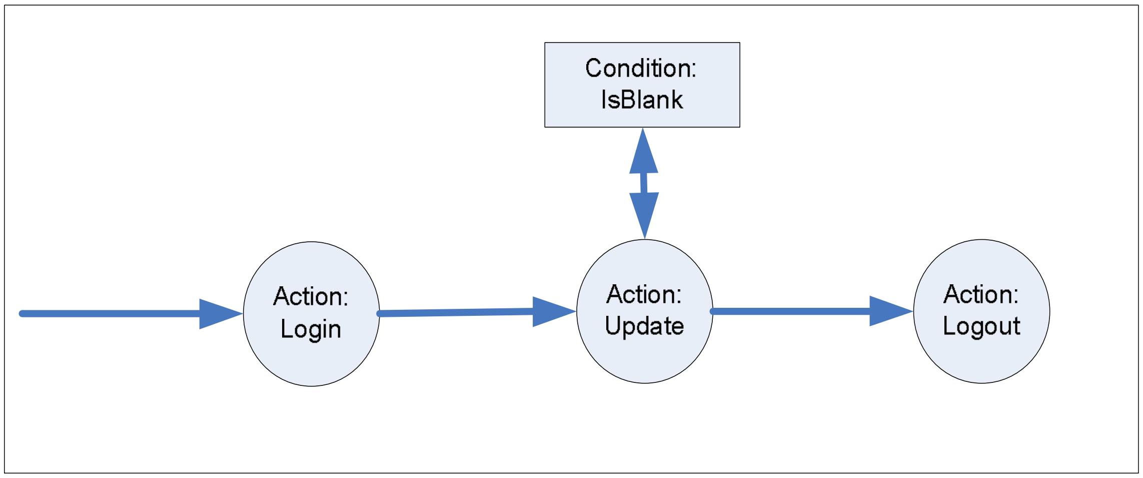


Figure 5.2: A Simple Account Update Process

This means that the login should act immediately without delay to minimize customer waiting. The minimum processing time is 0.01 second, and the maximum processing time is 0.1 second to reduce customer waiting. f(x) defines the timing distribution for the login service in terms of the number of concurrent users x. The timing requirement will be relaxed with a specified value according to the distribution function. **Bi-Guards:** They express timing constraints with respect to two elements in the ACDATER model, and these timing constraints are also specified using Delay, Deadline, Min, Max, and Distribution.

- Delay specifies the minimum time that the second element must wait after completing the first element.

- Deadline specifies the time that the two elements must be completed.

- Min specifies the minimum amount of time to complete the two elements.
- Max specifies the maximum amount of time to complete the two elements.
- Distribution specifies the distribution function.

The relation between the deadline and max is similar to uni-guards. The following examples show the bi-guards between the login, update, and logout services. The first bi-guard shows that after finishing the login service, the update service must wait for 0.1 second to ensure that the database is properly activated. The second bi-guard shows that these two operations must be completed within 0.2 second.

Delay (login, update) = 0.1;

Deadline (update, logout) = 0.2;

N-Guards: They express timing constraints with respect to n elements in the ACDATER model, and these timing constraints are also specified using Delay, Deadline, Min, Max, and Distribution where n can be any positive integer number. Thus one can have tri-guards and quad-guards. For example, an if-then-else construct requires a tri-guard, and a sequence of four steps requires a quad-guard.

The following descriptions show more uni-guards and bi-guards examples.

Actor Administrator:

Delay (administrator)=0;

Deadline (administrator)= 0.4;

This means that to start the workflow in the Actor, the delay for the system should be zero, and the entire workflow should be completed within 0.4 seconds.

Action Login:

Delay (login) = 0;

Min (login) = 0.01;

Max (login) = 0.1;

Deadline (login) = 0.09;

Deadline (login, update) = 0.1;

Action Update:

Delay (update) = 0;

Min (update) = 0.01;

Max (update) = 0.1;

Deadline (update) = 0.09;

Deadline (login, update) = 0.1;

Condition IsBlank:

Delay (IsBlank) = 0;

Deadline (IsBlank) = 0.1;

Action Logout:

Delay (logout) = 0;

Min (logout) = 0.01;

Max (logout) = 0.1;

Deadline (logout) = 0.09;

Deadline (update, logout) = 0.1;

5.3.2 Consistency of Timing Constraints

Timing constraints specified should be consistent with each other. For example, a bi-guard must be consistent with those uni-guards involved in the bi-guard. Figure

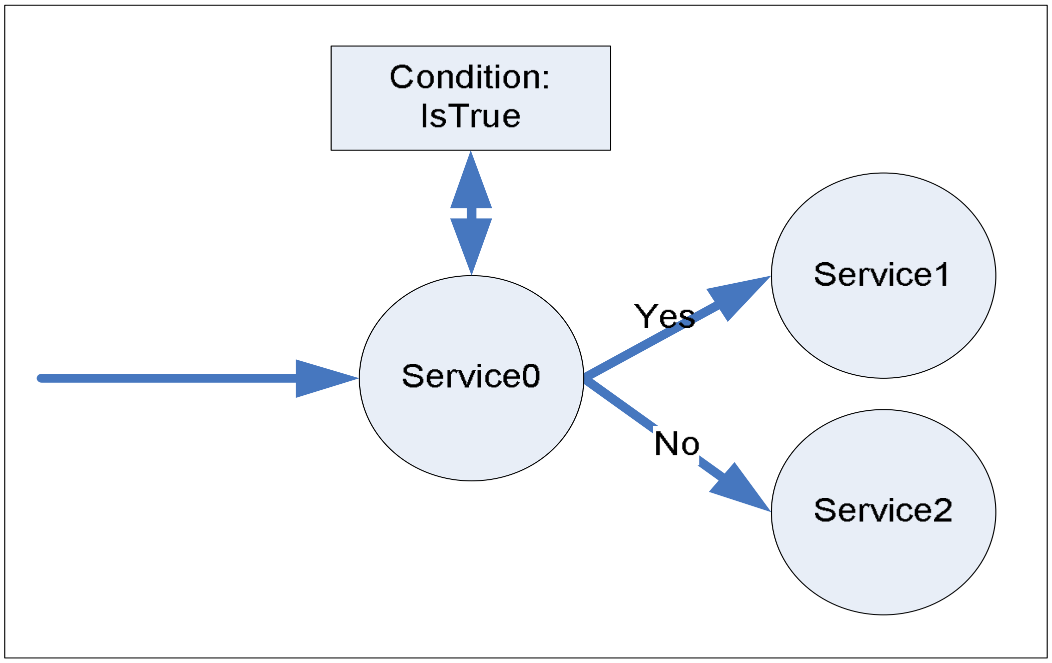


Figure 5.3: An If-Then-Else Process

5.3 shows an if-then-else process. After the services0 verifies the condition, it can either choose to execute service1 or services2 depending on the evaluation result. The tri-guard, as it involves three elements and thus tri-guard, must be consistent with individual uni-guards and two bi-guards. Bi-guards should be consistent with the uni-guards:

$$\text{Delay1} = \text{Delay (Service0, Service1)} = \text{Delay (Service0)} + \text{Delay (Service1)}$$

$$\text{Delay2} = \text{Delay (Service0, Service2)} = \text{Delay (Service0)} + \text{Delay (Service2)}$$

$$\text{Min1} = \text{Min(Service0, Service1)} = \text{Min(Service0)} + \text{Min(Service1)}$$

$$\text{Min2} = \text{Min(Service0, Service2)} = \text{Min(Service0)} + \text{Min(Service2)}$$

$$\text{Max1} = \text{Max (Service0, Service1)} = \text{Max(Service0)} + \text{Max(Service1)}$$

$$\text{Max2} = \text{Max (Service0, Service2)} = \text{Max(Service0)} + \text{Max(Service2)}$$

Tri-guards should be consistent with bi-guards:

$$\text{Delay (Service0, Service1, Service2)} = \text{Delay (Service0)} + \text{or(Delay1, Delay 2);}$$

Min (Service0, Service1, Service2) = Min(Service0) + minimum (Min 1, Min 2);

Max (Service0, Service1, Service2) = Max(Service0) + maximum (Max 1, Max 2);

Deadlines are different from Min, Max and Delay. Bi-guards Deadline (service0, service1) might not be equal to the Deadline (service0) + Deadline (service1). The same also applies for the Deadlines for tri-guards.

5.3.3 States in ACDATER

Actor, Condition and Action elements in ACDATER have states. Their states can either be active or inactive. An active state of Actor, Condition and Action means the element is under execution. Similarly, the inactive state means the element is idle.

For a workflow, it can either have only one element at the active states or multiple elements at the active states simultaneously. When multiple elements are in the active state simultaneously, the workflow might need synchronization among the elements.

Guards could be applied to the elements at the active states to help the synchronization. Figure 5.4 shows Service1 and Service2 are in the active state at the same time. The requirement for this flow is that Service1 and Service2 need to start at the same time and the whole flow needs to finish in 10 seconds. Deadline of the tri-guards can be applied to guarantee a sub-flow can be finished within a certain time, and the delay for Service1 and Service2 need to be synchronized to make sure the two services start at the same time.

Delay(Service1) = Delay(Service2)

Deadline (Service0, Service1, Service2) = Deadline(Service0) +

maximum(Deadline(Service1), Deadline(Service2)) < 10

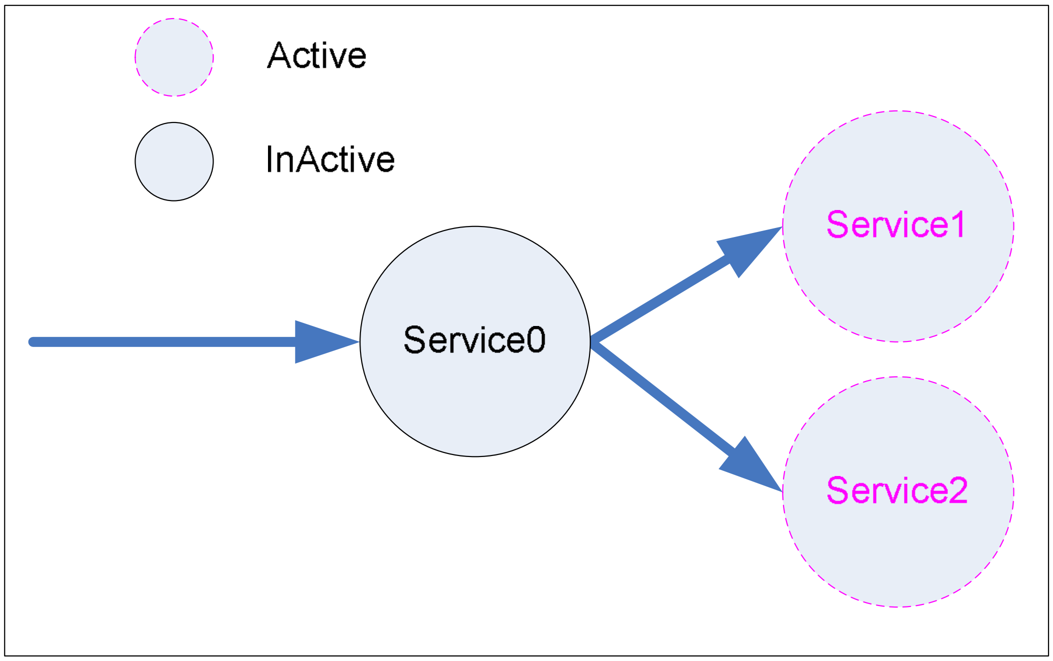


Figure 5.4: Two Active Services in One Process

Min(Service0, Service1, Service2) = Min(Service0) + maximum(Min(Service1), Min(Service2)

Max(Service0, Service1, Service2) = Max(Service0) + maximum(Max(Service1), Max(Service2)

5.4 Simulation Architecture

Models that are developed according to the extended PSML approach can be simulated as shown in Figure 5.5. The simulation environment consists of Event Generator, Simulation Engine, and Simulation Analysis Engine. The Event Generator generates events to drive the simulation. The Simulation Engine executes simulation code given simulation configuration files and generated events. The results of the simulations are stored in log files. The Simulation Analysis Engine uses the log files at runtime to generate data depicting how the model dynamics are changing under timing constrains. These results can also be visualized and evaluated by modelers.

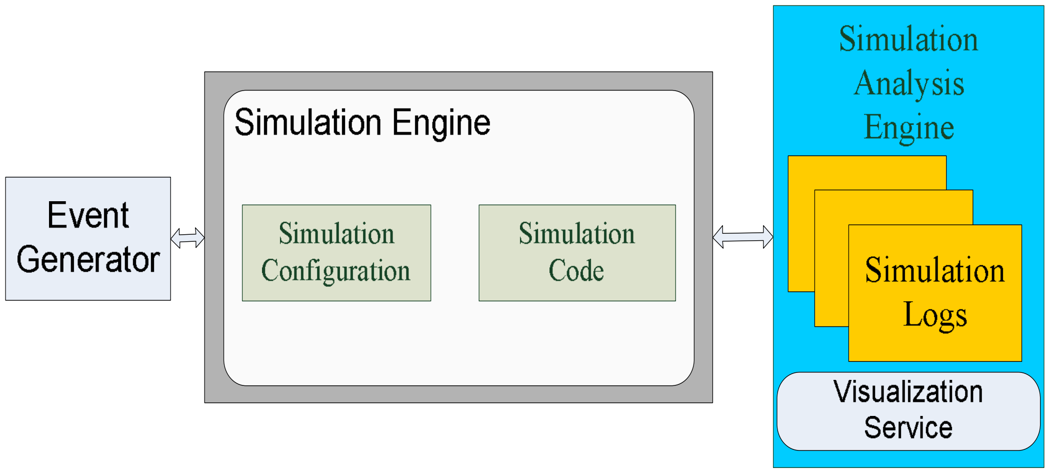


Figure 5.5: Simulation Environment

5.5 Simulation Lifecycle

Once the service-oriented model has the timing information, it can follow a service-oriented life cycle where application templates, collaboration templates, workflows, and services can be published and discovered for reuse to rapidly develop the simulation model.

Figure 5.6 shows the simulation lifecycle as the following phases:

- Service-oriented requirement and design phase: The model takes the system requirements and then discovers the needed services and workflows from the existing repositories. Note that not only do the services and workflows need to match the functionality requirement; they also need to meet the timing constraints specified by the timing specifications. If the needed services or workflows are not available, it will be necessary to either develop them or modify the existing services or workflows if their source code or design is available. The modified or new items will be published in various repositories for publishing so that they can be reused.

- Model evaluation phase: This phase evaluates the application model developed in the previous phase. Specifically, all the timing inconsistencies will be detected and eliminated at this phase.

- Code generation phase: This phase generates code from the application model developed in the previous phase with identified or developed services and workflows.

- Simulation execution and monitoring phase: This phase executes the application code including sending messages to participating services at remote sites. Furthermore, the execution is monitored and data are collected for analysis.

- Data analysis and evaluation phase: This phase will analyze the data collected in the previous phase, and the evaluation results will be used to guide the modification of the application model in the next cycle. After completing this step, the process goes back to the service-oriented requirement and design phase. This phase will be further explained in Section 6.

Note that this simulation lifecycle can be an integrated part of SOA application lifecycle where simulation plays a key role in verifying and validating system requirements and performance.

5.6 Data Collection and Model Tuning

Following the SOSE (Service-Oriented System Engineering) Tsai (2005) approach, timing information can be collected for each Actor, Condition, Action, and Events in a model. The collected information can be used to tune the model so that it can better reflect the system.

Figure 5.7 shows an SOA communication bus architecture with time tracking services. As one can see, each service and workflow is bundled with a tracking service

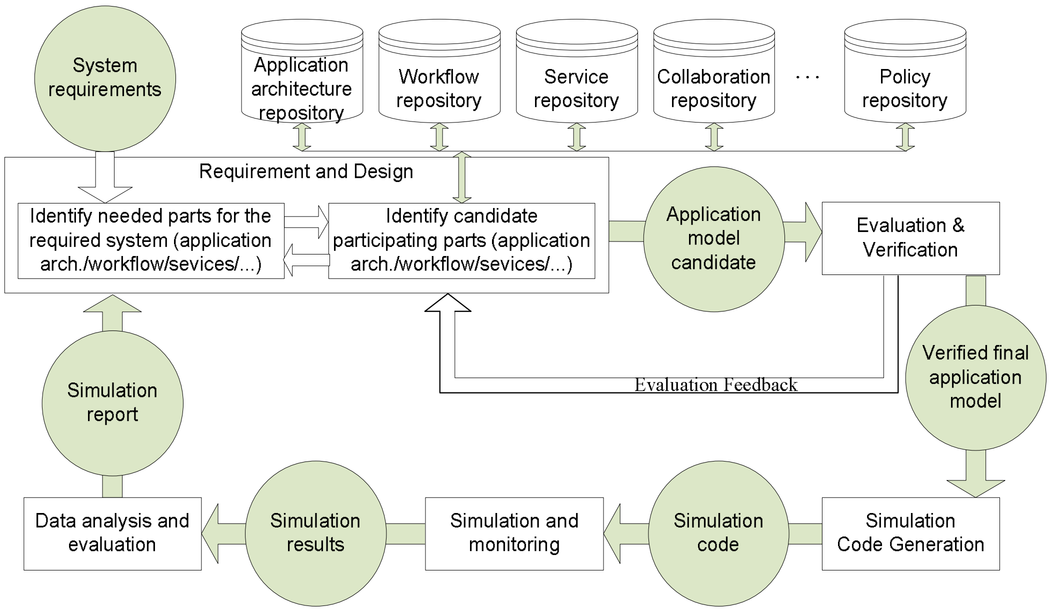

System requirements
Application architecture repository
Workflow repository
Service repository
Collaboration repository
...
Policy repository
Requirement and Design
Identify needed parts for the required system (application arch./workflow/sevices/...)
Identify candidate participating parts (application arch./workflow/sevices/...)
Application model candidate
Evaluation & Verification
Evaluation Feedback
Verified final application model
Simulation report
Data analysis and evaluation
Simulation results
Simulation and monitoring
Simulation code
Simulation Code Generation

Figure 5.6: Model-Driven Service-Oriented Simulation Process

that keeps tracking all the information associated with timing and other information. The information tracked can also be used to evaluate system security, reliability, and integrity. Note that this is a simplified version of general SOA data provenance architecture.

When the services used are from third parties and their source code is not available, the tracking service can only work in a non-intrusive way by intercepting messages sent between services and communication bus. For example, computation time, tcomputation, can be computed from tresponse - trequest. When a number of request/response pairs were recorded, the average, min, max computation time can be calculated. When an input classifier is added to the tracking service, the timing distribution function can be obtained.

If the service code is available, the tracking service can work in an intrusive way by adding code to services recording the time when interested events happen, therefore more information, such as delay, can be obtained.

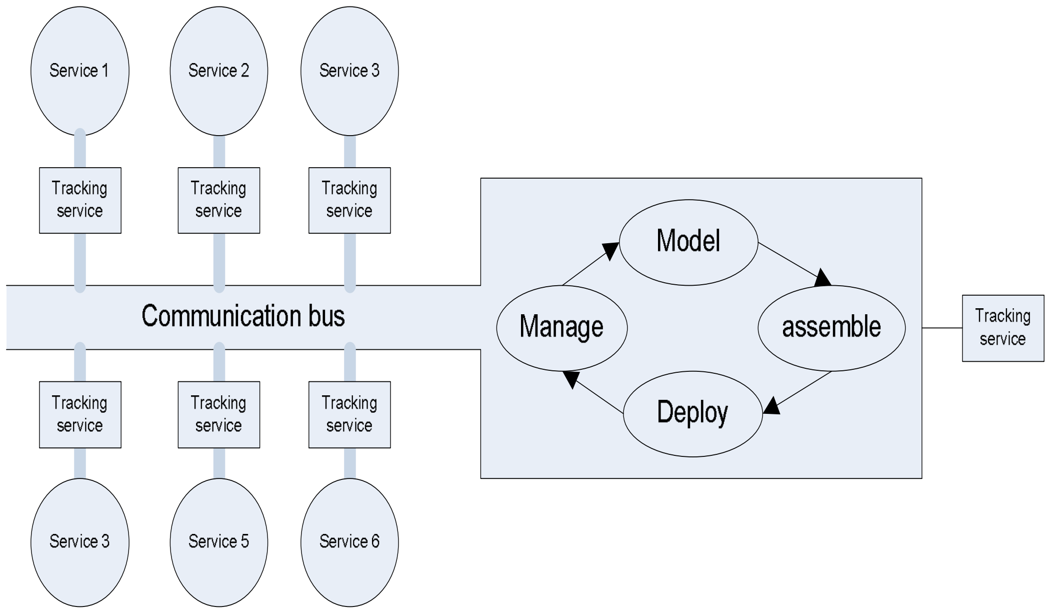


Figure 5.7: Time Tracking SOA Architecture

Similarly, as a full specification is available for workflow tracking, more interesting timing information can be collected, such N-guards. Figure 5.8 shows how to collect the timing information for a tri-guard. The small circles represent points of interest in a workflow. In this example, the time when Service0 receives a request, and the time Servcie1 and Service2 send out response are the points of interest. With the information collected, Min, Max, Delay can be easily computed for this tri-guard.

Data collection and model tuning can happen in three different stages:

1. Simulated system and simulated environment: At the initial stage of development, a model is specified for the system that needs to be produced. Some initial assumptions, such as the value for Min, Max for primitive parameter set, can be made for each model element. The values for more complex input set can be obtained by running simulation with different simulated environments.

2. Real system and simulated environment: At this stage, a system based on the model in stage 1 has been built, so data can be collected by running the system

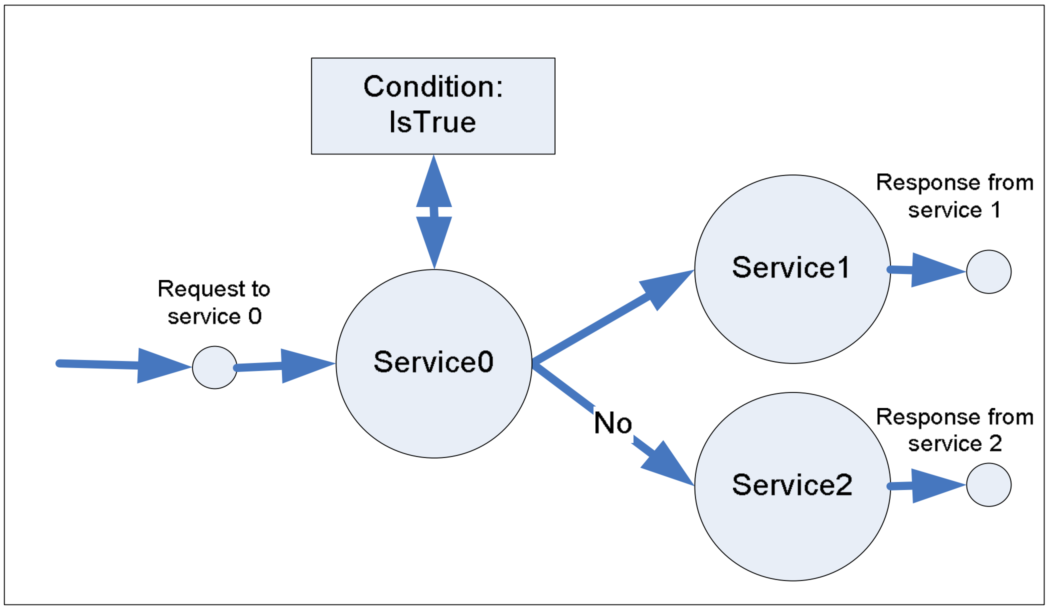


Figure 5.8: Time Data Collection for Workflow

with the simulated environment. Some incorrect assumptions made in stage 1 may be found and corrected at this stage.

3. Real system and real environment: This stage is similar to stage 2. The only difference is at this stage, the data collected is no longer from simulations, but real execution of the system.

Figure 5.9 shows the relationship among the three stages. After the 3-stage tuning, the model will reflect the behaviors of the real system.

5.7 Dynamic Simulation with Aid from Static Analysis

Essentially, timing analysis for a service-oriented application can be obtained by aggregating the time needed for the entire applications from the participating services, workflows, application templates and collaboration templates in a bottom-up manner. The bottom-up process starts from analyzing timing requirement for atomic services, which usually involves gathering timing constraints using data collection technique.

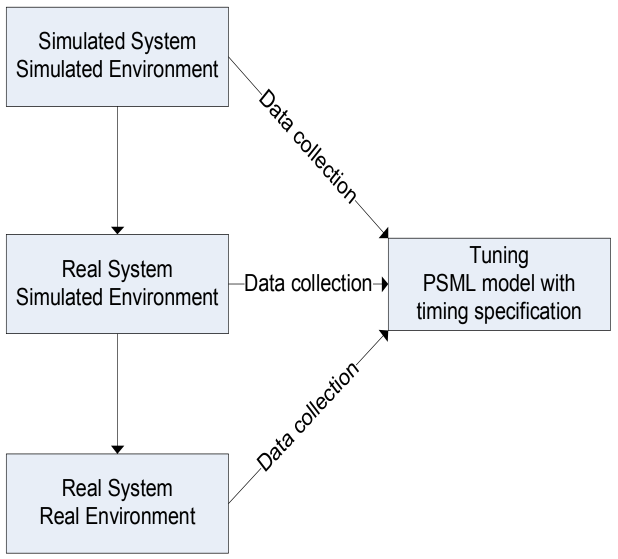


Figure 5.9: Data Collection and Model Tuning

Once the timing information about atomic services is known, the timing information about composite services or workflows that call only atomic services can now be computed. During simulation, a process that carries out the computation carries its own timing requirements. During the simulation execution, whenever a service is called, the calling process needs to provide its timing requirements to match with the timing information of the called services to verify timing requirements against timing capabilities.

Figure 5.10 shows the process for the static analysis and dynamic simulation.

Simulation modeling: Simulation modeling is done in the PSML modeling tool. The tool can model the services, workflows and the timing constraints for all services and workflows.

Code generation: Code generation includes element generation, path generation and timing constraint derivation. Element generation will generate the simulation code for all the atomic elements. Timing constraints can be attached to the generated code. Path generation will generate the workflow among the elements.

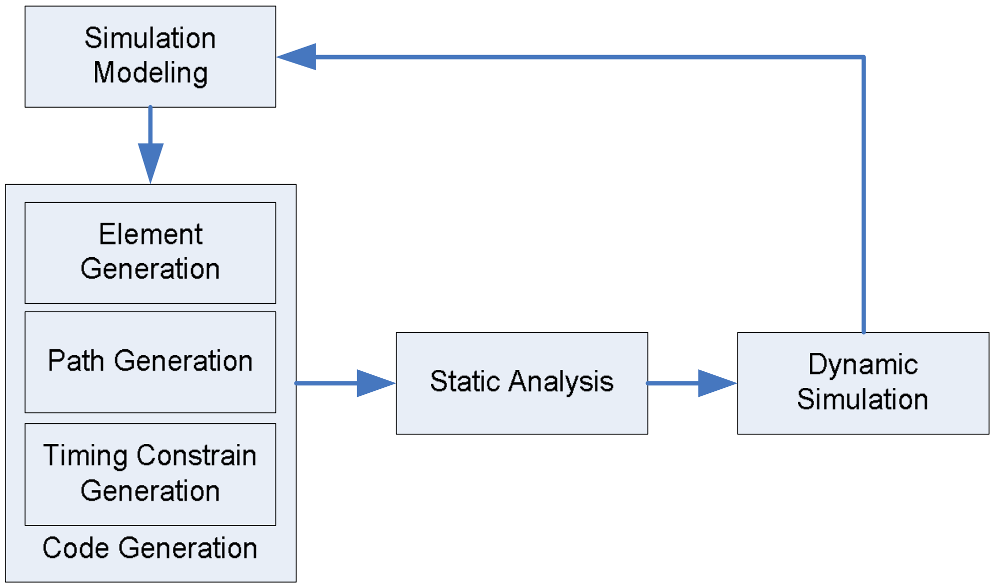


Figure 5.10: Dynamic Simulation with Aid from Static Analysis

Timing constraint derivation will derive the timing constraints based on the generated workflow. The timing constraints generation algorithm is similar to the N-Guard timing constraint generation algorithm.

Static simulation analysis: Static simulation analysis will need at least one set of simulation result and all the existing timing constraints. The analysis will verify the correctness and consistency of the current timing model and evaluate the timing performance of the system.

For example, if a uni-guard specification for service1 is defined as Deadline (service1) =5 and the simulation result to run this service1 is 10 second, then the simulation result is inconsistent with the timing constraints. This means either the existing timing specifications need to be updated or a different service or workflow should be used.

Dynamic simulation: Dynamic simulation can either refer to runtime path selection or runtime timing constraint configuration. Runtime path selection means

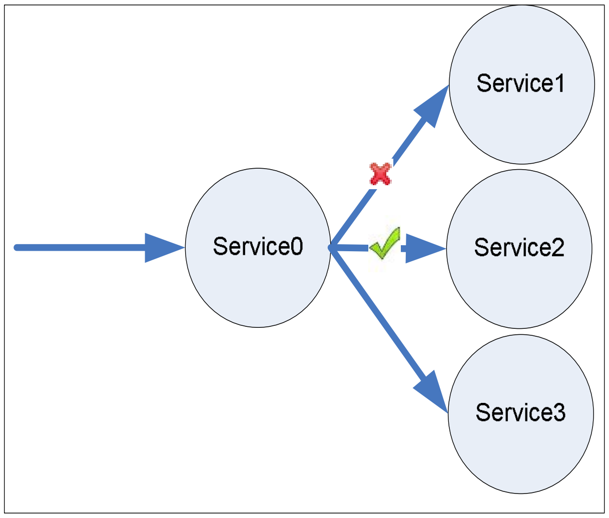


Figure 5.11: Switch Flow Sample

given different timing constraints, the simulation engine would dynamically choose path to satisfy the timing constraints.

For example, suppose there is a 3-way switch.

In Figure 5.11, the bi-guard between service0 and service1 is defined as Delay (service0, service1) = 5. Now, suppose the runtime simulation reports that when choosing the path service0 to service1, the result is 10 seconds. In this way, runtime path selection will choose the second path service0 to service2. In this manner, the simulation program dynamically decide which service to run based on timing constraints attached to the concerned services.

Runtime timing constraint configuration means the simulation engine would configure the timing constrains at runtime. The constraints come from the static analysis step.

5.8 Case Study

In this section, a case study based on a popular robotic game, sumoBot competition, is provided to illustrate the enhanced timing information for PSML. The main purpose of sumoBot is to have the robot stay inside of an arena while trying to push the opponent out. Timing plays an important role in order to win this competition.

The strategy used is to actively search for the opponent (Search), and once a target is located, the robot speeds up and tries to push it off the arena (Chase). However, at any time during the search or chase process a BorderDetected event is generated and sent to the system, the robot must preempt its current action, and start to perform a Backup & Turn action to avoid going off the arena. A PSML model for the sumoBot competition has the following elements:

Actor: sumoBot

Action: Search, Chase, Backup & Turn

Event: BorderDetected

Condition: TargetFound, TargetLost

Figure 5.12 shows an event-based workflow for the competition using PSML model. Some important timing constraints are listed as follows:

- Within 0.1 second after a target is found, a Chase action must be followed.
- With 0.5 second after a target is lost, a Search action must be followed.
- Within 0.05 second after a BorderDetected event is received, a Backup & Turn action must be followed. This constraint has the top priority.
- A Backup & Turn action should not last more than 2 seconds.

The timing information and constraints can be imported into the PSML using the schema explained in Section 3 as follows:

Action Search:

$$\text{Delay (Search)} = 0;$$

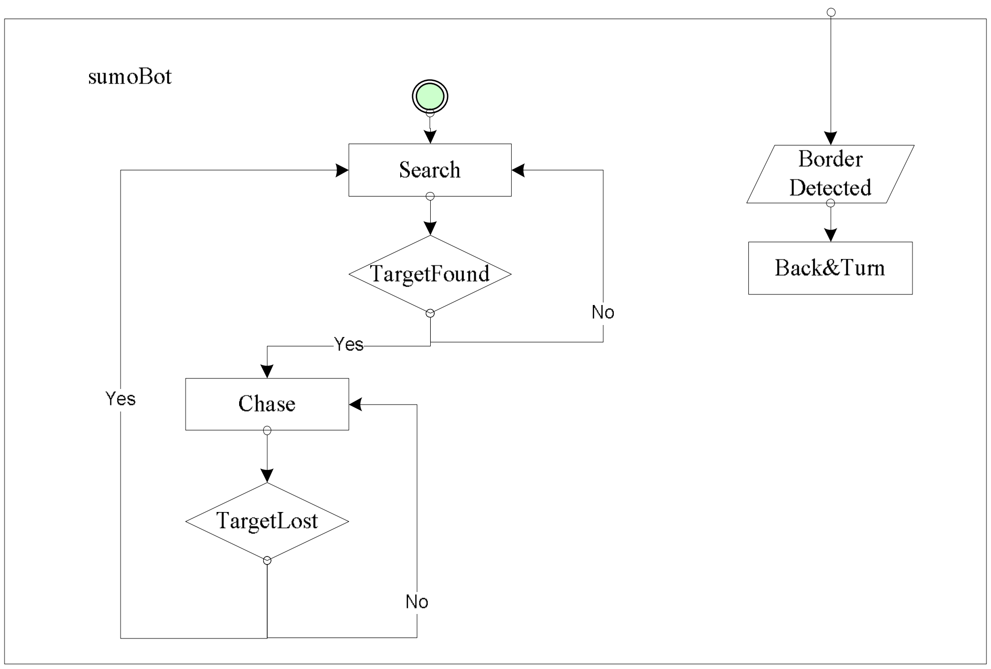


Figure 5.12: SumoBot Workflow

$$\text{Min (Search)} = 1;$$

$$\text{Max (Search)} = \infty;$$

$$\text{Deadline (Search)} = \infty;$$

$$\text{Deadline (TargetLost, Search)} = 0.5;$$

Action Chase:

$$\text{Delay (Chase)} = 0;$$

$$\text{Min (Chase)} = 1;$$

$$\text{Max (Chase)} = \infty;$$

$$\text{Deadline (Search)} = \infty;$$

$$\text{Deadline (TargetFound, Chase)} = 0.1;$$

Action Backup & Turn:

$$\text{Delay (Backup \& Turn)} = 0;$$

$$\text{Min (Backup \& Turn)} = 0.5;$$

Max (Backup & Turn) = 2;

Deadline (Backup & Turn) = 2;

Deadline (BorderDetected, Backup & Turn) = 0.05;

Condition TargetFound:

Delay (TargetFound) = 0;

Deadline (TargetFound) = 0.1;

Deadline (TargetFound, Chase) = 0.1;

Condition TargetLost:

Delay (TargetLost) = 0;

Deadline (TargetLost) = 0.1;

Deadline (TargetLost, Search) = 0.5;

According to the data collected during the first simulation, a time trajectory, as shown in Figure 12, can be drawn. Static timing analysis reports that there are 3 violations of time constraints: the time slot between Search and Chase is 0.15 sec, longer than the specified 0.10 sec; it took 0.30 sec for the robot to respond to the BorderDetected event, while the constraint is 0.05 sec; Backup & Turn lasted 2.5 sec, exceeding the time constraint by 0.5 sec.

After improving the services and workflow, and tuning the model, the simulation was rerun. Figure 5.14 shows a new timing trajectory.

If there are multiple Backup & Turn service implementations, dynamic service selection technique can be applied. After the workflow is deployed on the robot, system keeps tracking whether if any service exceeds the required execution time and tries to switch a different implementation to eliminate the violations. For example, the Backup & Turn service exceeded the required execution deadline, therefore, the

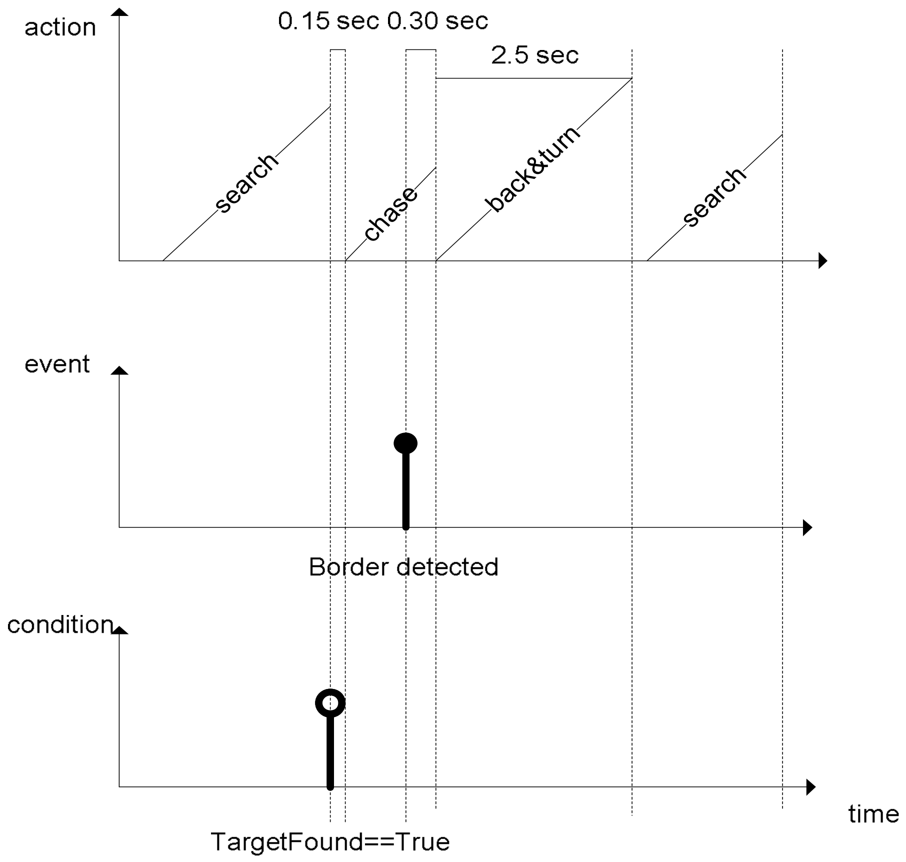


Figure 5.13: Action, Event and Condition Time Trajectories

system will select another service that has the shortest execution time guarantee out of the available implementations.

5.9 Conclusion

This paper proposes timing specifications into a service-oriented model and applies timing specification in analysis and simulation. Timing specification can be analyzed to ensure their completeness and consistency by static analysis as well as by simulation. Timing information can also be obtained by simulation. As service-oriented simulation can be an integrated part of service-oriented application development, timing specification and analysis will enable simulation to obtain more accurate results. A distinct feature of the timing specification is that timing diagrams can be automat-

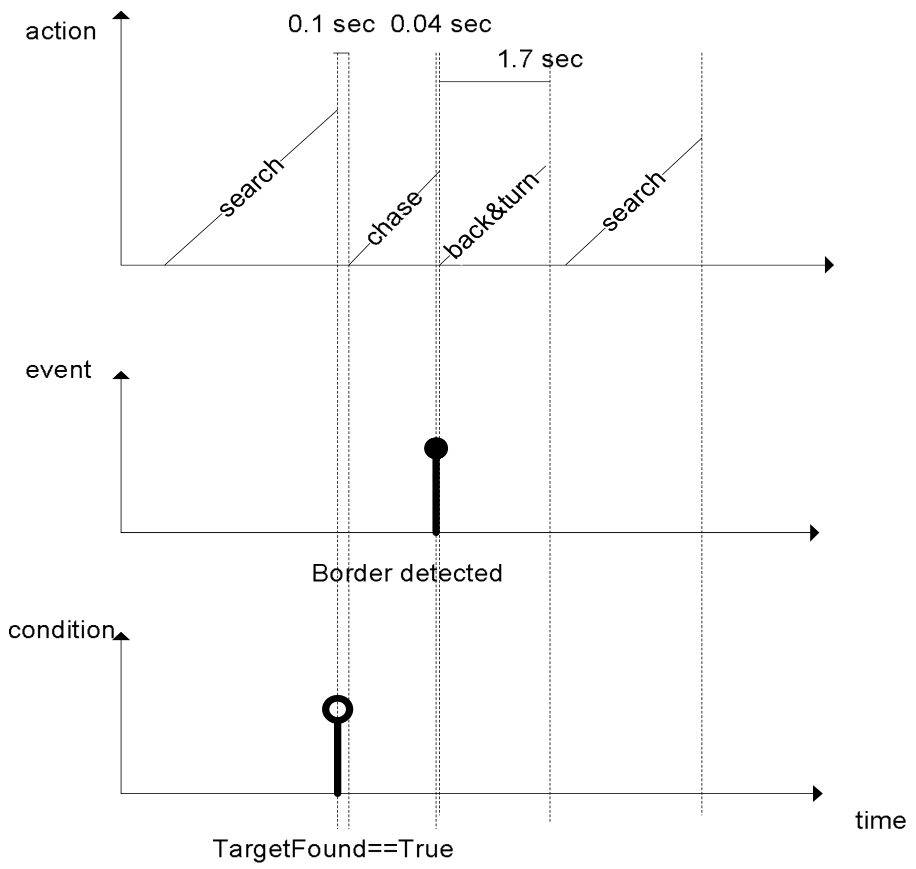


Figure 5.14: Action, Event and Condition Time Trajectories

ically generated once the simulation is performed, and the simulation is performed using the simulated code that is generated from a service-oriented model.

Chapter 6

EVENT-DRIVEN SERVICE-ORIENTED SIMULATION FRAMEWORK

6.1 Introduction

Service-oriented architecture (SOA) and event-driven architecture (EDA) have received significant attention recently. Leading IT organizations support the combination of SOA with EDA and label it as SOA 2.0 Wikipedia (2014a), IBM Marechaux (2010), Oracle Oracle (2010), and Tibco Tibco (2010) all have their own event-driven SOA systems. In Marechaux (2010), IBM suggests using enterprise service bus (ESB) to pass events around the system. An ESB service needs to have event transport services, event detection/triggering services, and protocol mediation services to facilitate event processing. In Oracle (2010), Oracle defines simple events as state changes of data throughout its lifecycle, while a complex event is an aggregation of simple events, and need to be detected outside a process. In Chandy and Schulte (2009), Chandy and Schulte presented event processing architecture in business applications. Various formalisms have been introduced to model composite event detection, including event detection tree Vaculin and Sycara (2007), Petri nets Zang and Fan (2007), finite state automata Pietzuch *et al.* (2004), and event detection graphs Akdere *et al.* (2008).

Event-driven SOA system has several advantages Wikipedia (2014a); Hanson (2005) over traditional SOA systems:

1. Decoupling: event publishers need not be aware of the existence of event subscribers, and this further reduces the coupling among system components

2. Consumer-based/event-based triggering: Services can be triggered by flow of control and event messages allowing system to mimic business processes.

3. Synchronous/asynchronous operations: Event-driven SOA supports both asynchronous and asynchronous operations, and these reduce blockings and improve system responsiveness.

Service-oriented simulation is an active research area Buyya (2008) such as XMS-F, ISTF INOA (2014), Cosim-Grid Li *et al.* (2005), DDSOA Tsai *et al.* (2006c), DCP simulation Tsai *et al.* (2007c, 2006a), Ontology-based simulation Tsai *et al.* (2007b), SOA simulation framework Tsai *et al.* (2007a) and SOAD Sarjoughian *et al.* (2008); Mittal *et al.* (2007). However, few consider the combination of SOA and EDA. In Tsai *et al.* (2008c,a), a BPEL engine is used as a simulation engine to simulate SOA applications, and an event-driven policy enforcement framework is proposed to verify the SOA applications. In Tsai *et al.* (2009), it proposes a timing specification and the associated analysis techniques for PSML, a service-oriented modeling language. Specifically, timing constraints such as delays, processing time, deadlines among elements can be specified, and consistency among timing constraints can be verified. The timing information can be used for static analysis to estimate time needed as well as dynamically to verify the runtime behaviors of service-oriented applications.

This paper proposes a simulation framework with the following features:

1. Share and reuse events, event generators, composite event specifications, event handlers, and event handler templates through an ontology system that facilitates publishing, discovery, and composition.

2. Specifications of composite events in PSML and patterns for event generation.

3. Static analysis with event triggering graphs and dynamic simulation with event generation, composite event specification and event handling.

This paper is organized as follows. Section 2 presents the simulation framework. Section 3 shows the PSML event specification. Section 4 discusses the publishing and sharing of events, event handlers through the ontology systems. Section 5 presents the static analysis and dynamic tuning for the simulated systems. Section 6 presents a case study to show the effectiveness of the framework. Section 7 concludes this paper.

6.2 Simulation Framework

6.2.1 Events and Their Attributes

Events have the following attributes:

Event Type: atomic events and composite events. An atomic event is a single, low-level occurrence generated from data changes, service changes or any arbitrarily triggered events. A composite event is a high-level event formed from other events.

Event Occurrence: onetime events, periodic events, sporadic events, and continuous events. A onetime event occurs once only. A periodic event occurs repetitively after a fixed interval. A sporadic event occurs repetitively after arbitrary intervals. A continuous event means that the type of event happens continuously.

Event Content: command events, data carrier events, and change notification events. A command event happens when a certain action order is given with no additional data. A data-carrier event is generated by the agent services responsible for collecting or sensing data from the outside environment or another system and injecting it into the system. A change notification event takes place when there is a certain change in the environment state.

Event Scope: internal events, environmental events, and mixed events. An internal event is generated on a change of the internal state of the system. An environmental event is generated when there is a change with environment state variables. A mixed event is an event that involves both internal and environmental state changes.

Event Priority: events can have priorities from low to high.

Event Deployment: centralized events and distributed events. A centralized event is generated from a centralized local broker in the system. A distributed event is usually emitted from a remote agent.

Event Vertical Causality Hanson (2005): Lifecycle events, execution events and management events. A lifecycle event specifies for a change in the lifecycle of a service, such as create, destroy, start, stop, and resume. An execution event specifies for the runtime status change of a service, such as idle, busy. A management event signifies that the defined limits or ranges in the service management have been exceeded, such as bandwidth overloaded, storage overuse, security breached.

Event Horizontal Causality Hanson (2005): System-layer events, platform-layer events, service-layer events, business-layer events and application layer events. For example, a system-layer event can be the closing of a port in the hardware. A platform-layer event can be the modification of a data source. A service-layer event can be an addition of a new service.

6.2.2 Framework Architecture

Figure 6.1 shows the main components of the simulation framework. It consists of a list of repositories and services. The repositories include event repository, composite event specification repository, event generator repository, event handler repository and event handler template repository. The publishing and searching are guided by the ontology systems in these repositories. The services are categorized through three service groups: simulation modeling services, simulation runtime services and simulation analyzing services.

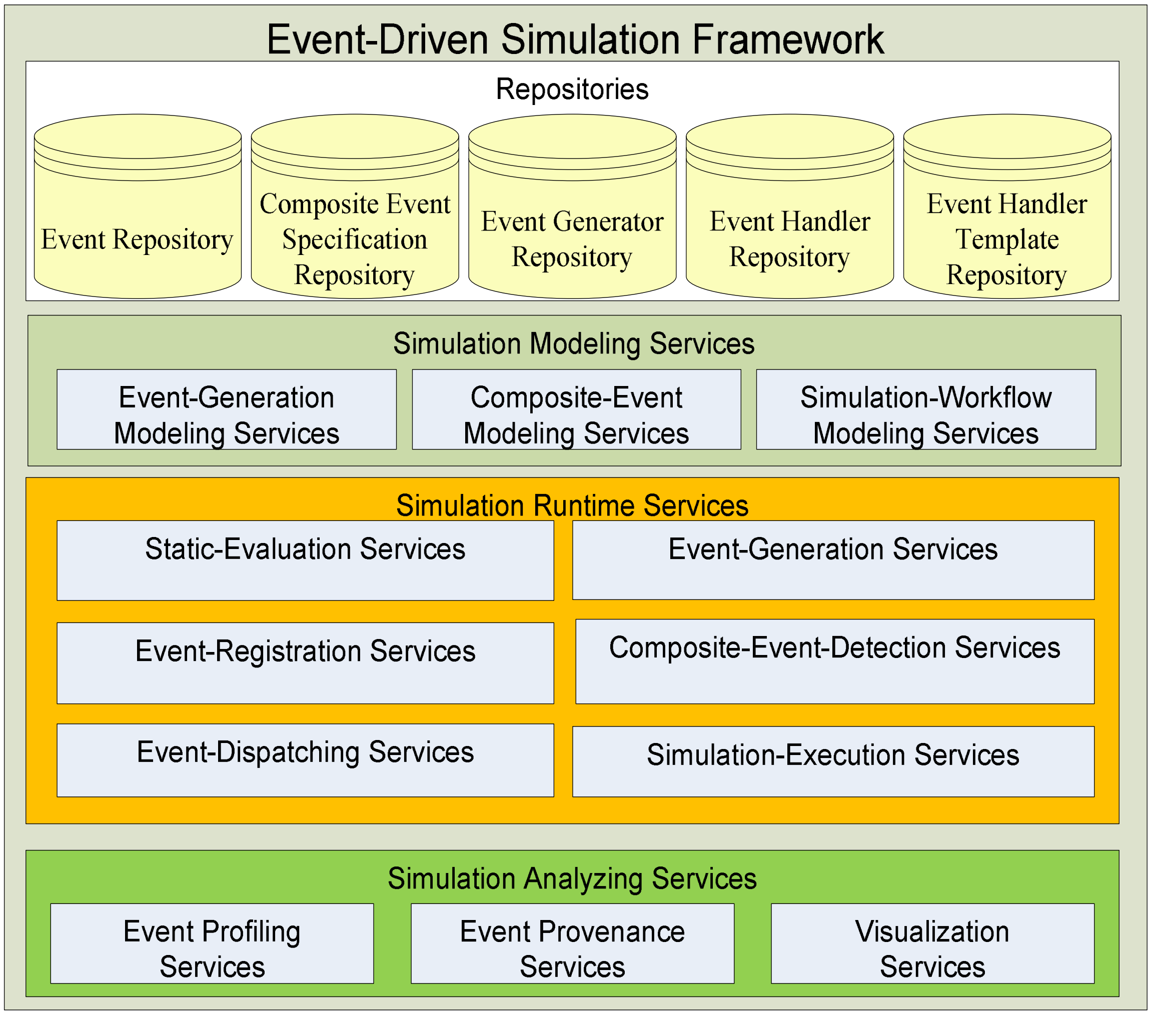

Event-Driven Simulation Framework
Repositories
Event Repository
Composite Event Specification Repository
Event Generator Repository
Event Handler Repository
Event Handler Template Repository
Simulation Modeling Services
Event-Generation Modeling Services
Composite-Event Modeling Services
Simulation-Workflow Modeling Services
Simulation Runtime Services
Static-Evaluation Services
Event-Generation Services
Event-Registration Services
Composite-Event-Detection Services
Event-Dispatching Services
Simulation-Execution Services
Simulation Analyzing Services
Event Profiling Services
Event Provenance Services
Visualization Services

Figure 6.1: Simulation Framework Architecture

Simulation modeling services include event-generation-modeling services, composite-event-modeling services and simulation-workflow composite services. Application developers can use them to search and reuse events, event generators, composite event specifications, and event handlers from the respective repositories, and build the simulation model for a specific application/service. After the modeling phase, they deploy the event generators, simulation workflow and the composite event specifications to the event-generation services, simulation-execution services and the composite-event-detection services.

Static-evaluation services evaluate the simulation model statically after it is produced from the modeling services. The evaluation may use the event graphs to be discussed in section 5.

Event-generation services generate atomic events according to user-specified patterns to drive the simulation.

Composite-event-detection services detect the composite events by following the composite event specifications.

Event-registration services manage the event registration information through the publishing and subscribing APIs.

Event-dispatching services dispatch events to the corresponding event handlers by looking up in the event registration services.

Simulation-execution services execute the simulated workflows driven by the events.

Event-provenance services log and track the simulation data from the simulation execution services.

Event-profiling services analyze the runtime execution information of the events and aid the dynamic tuning for the simulation.

Visualization services display the data from the event profiling and provenance services.

6.2.3 Sample Simulation Flow

The simulation process has four phases: preparation, deployment, execution, and tuning. Figure 6.2 shows a sample simulation process.

Preparation phase:

1. Application developers use simulation-modeling services to search and reuse the assets from various repositories, and develop a simulation model. The model includes three types of elements: event generators, composite event specifications,

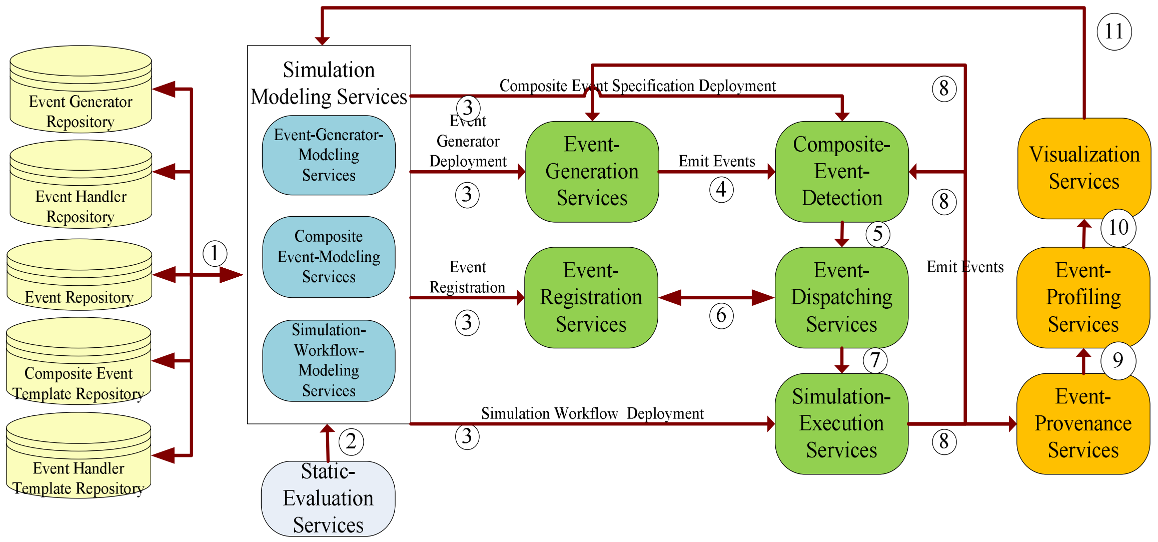

Event Generator Repository
Event Handler Repository
Event Repository
Composite Event Template Repository
Event Handler Template Repository
1
Simulation Modeling Services
Event-Generator-Modeling Services
Composite Event-Modeling Services
Simulation-Workflow-Modeling Services
2
Static-Evaluation Services
Composite Event Specification Deployment
Event Generator Deployment
3
Event Registration
Simulation Workflow Deployment
Event-Generation Services
Emit Events
4
Composite-Event-Detection
5
Event-Registration Services
6
Event-Dispatching Services
7
Simulation-Execution Services
8
Emit Events
Visualization Services
10
Event-Profiling Services
9
Event-Provenance Services
11

Figure 6.2: Sample Simulation Flow

and simulation workflows. The simulation workflows include event handlers and other application logics.

2. Static evaluation services evaluate the model developed earlier. They also verify if event specifications and simulation workflows are consistent with each other.

Deployment phase: 3. In the phase, the simulation model including event generators, composite event specifications and simulation workflow is deployed. After this, the simulation runtime services will run the simulation with the current model.

Execution phase: 4. Event-generation services generate atomic events and distribute them to the composite-event-detection services.

5. Event-detection services relay all atomic events to event dispatching service. In addition, they also detect composite events according to composite event specifications and pass them to the event dispatching services.

6. Event-dispatching services check the event-registration service, and find the event handlers registered for the detected events.

7. Event-dispatching services dispatch events to appropriate event handlers.

8. Simulate execution services execute the simulation workflows according to the events received.

Tuning phase: 9. Event-provenance services log simulation statuses through the process and meanwhile, they may send back request to simulation modeling service to change the model with events, event generators and event handlers.

10. Event-profiling services compare the results and analyze the runtime execution information. The analyzed results can be displayed through the visualization services.

11. Based on the simulation results, application developers can modify the model with the modeling services. They can also change the event generation patterns, composite event specifications, and continue tuning the simulation until it reaches a satisfactory result.

6.3 PSML Event Specification

A composite event may consist of atomic and/or other composite events. Composite-event-detection services can detect a composite event according to event specifications and send it to the event-dispatching services. This paper extends the Event in PSML Tsai *et al.* (2007d) based on ACDATER (Actors, Conditions, Data, Actions, Timing, and Events, Relations) to specify the events. The event model is extended by offering the basic event operations as shown in the Table 6.1.

The following example shows a composite event with three sub events event1, event2 and event3.

Seq (event1, event_composite)

event_composite = conj (event2, event3)

Table 6.1: Event Specification in PSML

Sequential	Seq (e1, e2)	A composite event occurs when second sub event happens after the first;
Conjunction	Conj (e1, e2)	A composite event occurs when all of sub events occur at the same time;
Disjunction	Disj (e1 , e2)	A composite event occurs when at least one of its sub events happens;
Exclusive	Exc (e1 , e2)	A composite event occurs when only one of the sub events happens;
Negation	Neg(e1, e2)	A composite event occurs when the first sub event happens while the second does not happen.

And these specifications can be abbreviated as Seq (event1, conj (event2, event3)), which means event1 happens first, and then it will be followed with a composite event which requires event2 and event3 occur at the same time.

Another example:

Exc (event1, event_composite)

event_composite = Disj (event2, event3)

And these specifications can be abbreviated as Exc (event1, Disj(event2, event3)) which means only one of its sub events will happen. Disj (event2, event3) occurs when at least one of event2 or event3 happens.

Moreover, timing constraints in PSML including Delay, Deadline, Min, Max, and Distribution discussed in [25] can be applied to events.

For example, a composite event can be expressed as:

Seq (event1, event2)

Delay (event1, event2) = 0.5

Deadline (event1, event2) =10

These specifications can be abbreviated as

Seq (event1, event2, Delay (0.5), Deadline (10))

which means event2 happens after event1 after a delay of 0.5 seconds and with a deadline of 10 seconds.

Another example:

Neg (event1, event2)

Delay (event1, event2) = 0.9

Max (event1, event2) = 5

These specifications can be abbreviated as

Neg (event1, event2, Delay (0.9), Max (5))

which means event1 happens while the event2 does not happen after a delay of 0.9 seconds and within a max of 5 seconds.

6.4 Publishing and Sharing via Ontology System

Events, event handlers and event generators ontology systems provide classification and reasoning to guide the search and publishing. This section uses the robotics domain as an example to demonstrate how ontology systems can improve the publishing and sharing of events. Figure 6.3 shows the robotics domain event ontology. The ontology is only for demonstration purpose, it is by no means exhaustive.

Sensors: Events triggered by the sensors that detect the surrounding environment. The events can be further categorized by the different types of the sensors. As shown

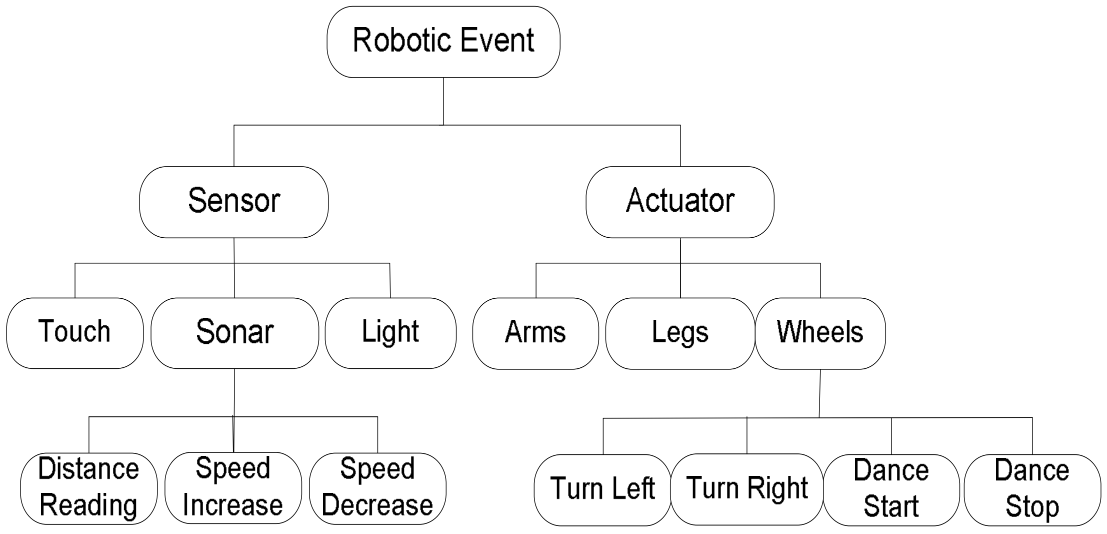


Figure 6.3: Sample Robot Domain Event Ontology

in the figure, sonar-triggered events may include, Distance Reading Event, Speed Increase (SI) Event, and Speed Decrease Event. Speed Increase/Decrease events are composite events, which can be detected through Event Detection Service by analyzing the atomic Distance Reading Event.

Actuators: Events triggered by the actuators of a robot, such as arms, legs, or wheels. Events triggered by the actuators can often be seen as internal events, since they are usually generated by application workflows or event handlers.

Each event has various attributes; the ontology system provides a reasoning mechanism to assist searching and publishing for events. For example, an event triggered by Sonar may also have the following attributes: Type: Atomic; Time: Continuous; Content: Shipping Domain; Scope: Environmental; Priority: High; Deployment: Distributed. An event-discovery service can search for events according to these attributes.

Event-handler ontology is similar to the event ontology. For each event, at least one event hander should be in the event-handler ontology; otherwise the system will have an event without an appropriate event handler. An event handler may handle

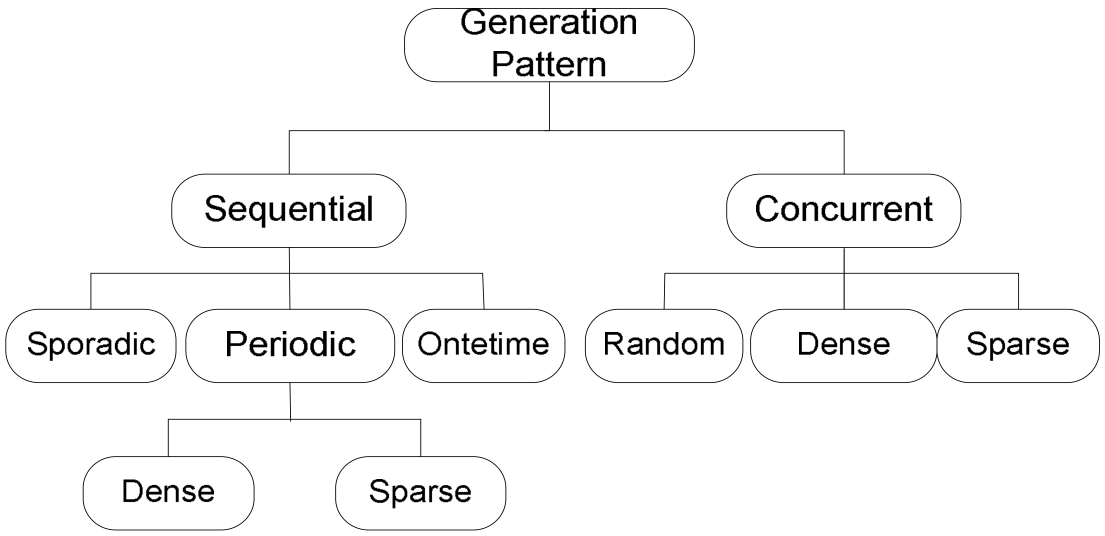


Figure 6.4: Sample Event Generation Patterns Ontology

multiple events, and an event may have multiple event handlers, and an appropriate event handler will be selected based on the workload of these event handlers at runtime.

Both event and event handlers are stored in the repositories. They can be shared and reused by the other consumers. When a new type of event or a new type of event handler is developed, they need to be registered in the ontology to facilitate future searching and reuse. Note that they need to verified and validated before they can be used, and they can be verified by associated verification mechanism associated with the repositories.

There are basic event generation patterns as well as complex event generation patterns. The complex event generation patterns may be composed by basic event generation patterns. Event generators can also be classified using event generation pattern ontology. Figure 6.4 shows a sample event generation pattern ontology.

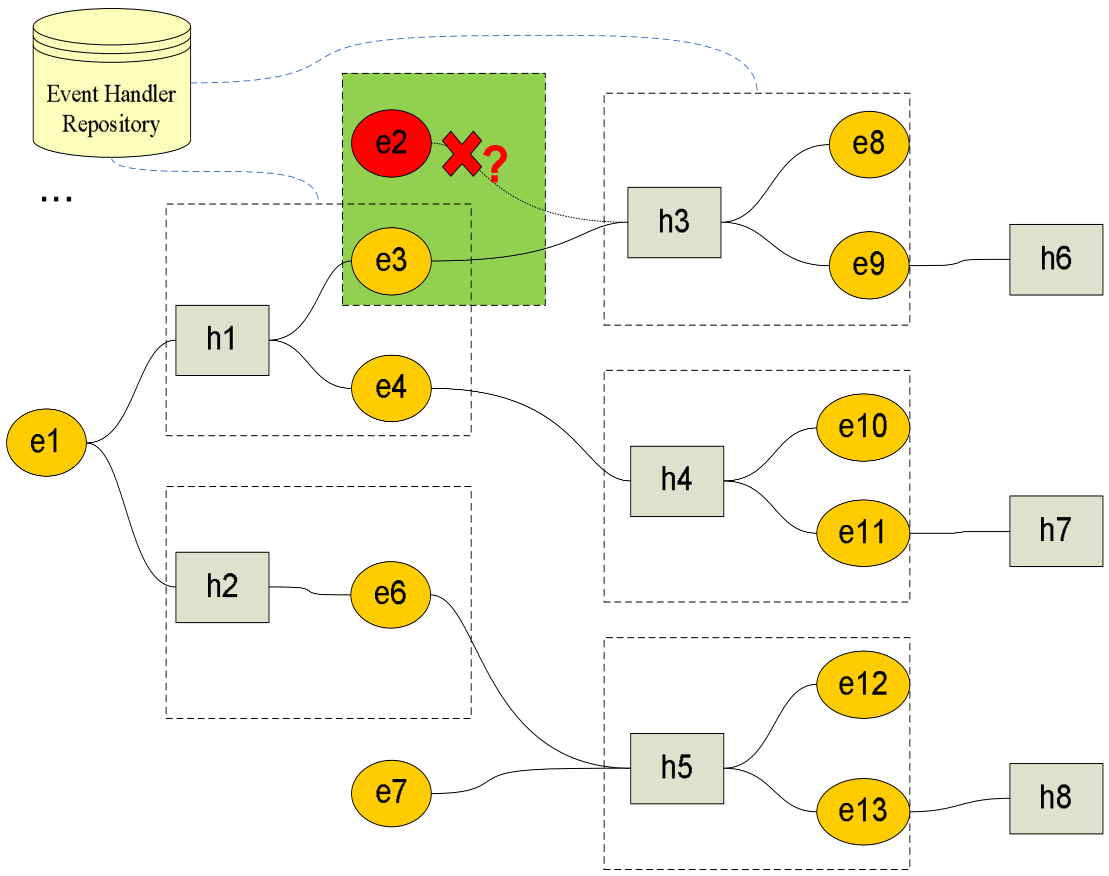


Figure 6.5: Sample Event Graph

6.5 Static Analysis and Dynamic Tuning

Static Analysis can be performed prior to simulation to verify various properties. For example, both event generators and event handlers can emit events, and linking these together will generate an event graph as illustrated in Figure 6.5 where $\{e1, e2, e3, ...\}$ is a set of events and set $\{h1, h2, h3, ...\}$ is a set of event handlers. For example, an external event e1 may be handled by two event handlers h1 and h2, each will trigger new sets of events, and these new events will be handled by their corresponding event handlers. In this way, an event graph can be generated from the ontology prior to simulation. The event graph can be useful to formal analysis.

A user may specify various relations among events including conjunction, disjunction, sequence, negation, and timing constraints such as Delay and Deadline.

For example, the composite event required by h3 in the graph is marked as green. Its event specification is:

$$\textit{Seq (e2, e3)}$$

$$\textit{Delay (e2, e3) = 0.5}$$

$$\textit{Deadline (e2, e3) = 15}$$

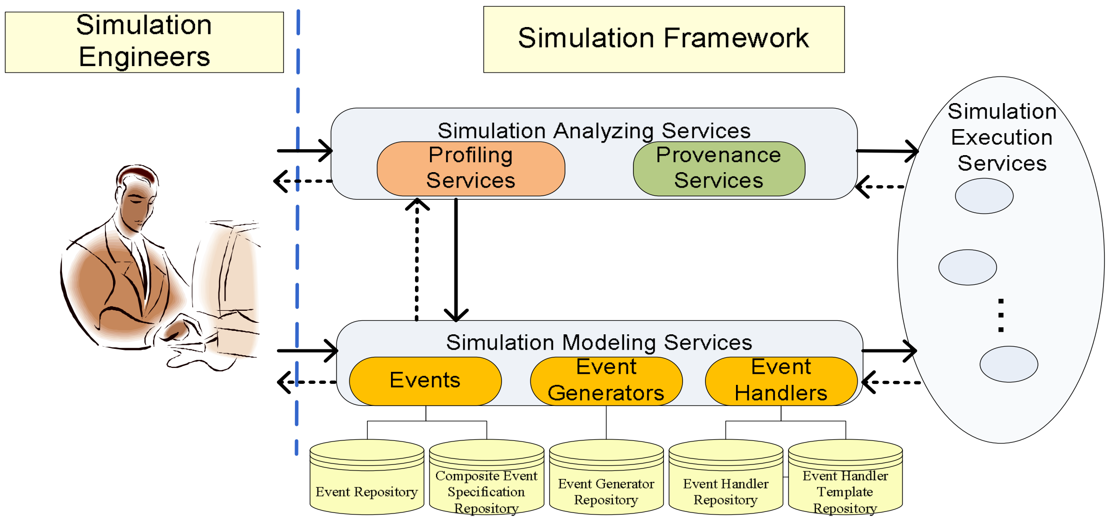
While e3 can be emitted from the event handler h1, e2 should be generated from an event generator. Static analysis can detect the missing event and alert the simulation designer for this fault.

Dynamic Tuning of the event-driven simulation includes the interferences of simulation users, simulation analyzing services, simulation modeling services and the simulation execution services. And it is done by through the dynamic cyclic process as shown in Figure 6.6.

While static analysis provides useful data, many analyses can be done only via simulation. Facts like e3 have to happen with a 0.5 second delay after e2 and e3 has to occur within 15 seconds after e2 can be verified during simulation execution.

Simulation modeling services are useful to make the process dynamic. By monitoring the execution data from the simulation analyzing services, a simulation engineer can change the model.

Event-profiling services in the simulation analyzing services are also useful to make the tuning process dynamic. Based on the information from the event provenances services, event-profiling services can control the selection of events, event generators, and event handlers by following the predefined algorithms in the services. A variety of algorithms can be used to decide, depending on the factors of interest. Some of the



Simulation Engineers
Simulation Framework
Simulation Analyzing Services
Profiling Services
Provenance Services
Simulation Execution Services
Simulation Modeling Services
Events
Event Generators
Event Handlers
Event Repository
Composite Event Specification Repository
Event Generator Repository
Event Handler Repository
Event Handler Template Repository

Figure 6.6: Dynamic Tuning Process

factors of interests can be functionality, minimal risk, minimal cost, minimal recovery time, minimal execution time, and priority. A combination of these factors may also be used in the dynamic tuning of the simulation model.

6.6 Case Study

This section illustrates the simulation framework using a robotic example with the following services:

Robot Master: It is a laptop equipped with a keyboard, a speaker, an IR sensor, a bluetooth adapter and a camera.

Robot Dog: It is a robot with two wheels, an arm, a touch sensor, a bluetooth adapter, a sound sensor and a speaker.

IR Ball: It is a ball with an IR emitter. commands to the robot dog using the keyboard, and the commands are sent to the robot dog via Bluetooth. Commands from the keyboard have a higher priority than voice commands. When robot dog grabs the ball, the touch sensor will notify the robot master that the ball is in possession. If the robot dog drops the ball at any time, the touch sensor will notify the robot master. To make things interesting, a secret command will be issued for the robot dog to perform a robot dance if the operator taps the direction keys three times fast.

Preparation Phase:

An application model is developed by reusing the existing services, events, event handlers in the according repositories. The search is guided by ontology systems. An example of event ontology in robotics is shown in Figure 6.3. Before building the real robots, a simulation needs to be conducted to verify if the application logic for this scenario is correct. The following atomic events are needed:

- Voice command events (E_v),
- Action completed command events (E_a),
- Keyboard command events (E_k),
- Touch sensor event (E_t),
- IR sensor reading events (E_i).
- Composite event E_{secret}

To achieve the secret command, a composite event needs to be detected: E_{secret}= Seq(Seq(E_k, E_k, Deadline(t)), E_k, Deadline(t)), t = 0.05 sec. Therefore, after modeling phase, the three outputs are produced as shown in Figure 6.6:

- Simulation Workflow.
- Event Generators: IR reading event generator, keyboard command generator, Voice Command Event generator, Touch Sensor Event Generator.
- Composite Event Detection Specifications: E_{secret}= Seq(Seq(E_k, E_k, Deadline(t)), E_k, Deadline(t)), t = 0.05 sec.

At this stage, event graphs are also generated. A sample partial event graph is also given in Figure 8. A static analysis of the event graph is performed according to the simulation model.

Establishment phase:

The outputs from the preparation phase are deployed into the simulation framework runtime environment. Proper parameters for the external event generators are set to mimic different real world scenarios.

Execution phase:

This phase is a coordination of all the components in the simulation framework to complete the simulation task. For this specific example, different generation patterns can be passed to the IR reading event generator and keyboard command event generator. For example, for the keyboard command event generator, one can have the following patterns:

- Random Pattern: This generator emits a random keyboard command after an arbitrary interval.
- Dense Pattern: This generator emits random keyboard commands at a very fast pace.
- Spare Pattern: This generator emits a random keyboard commands at a slow pace.
- Dense Secret Command Pattern: This generator emits 3 consecutive keyboard commands at a fast pace.

Tuning phase:

Through the provenance and profiling services, one found out that when using dense secret command pattern, the robot will try to start to dance over and over

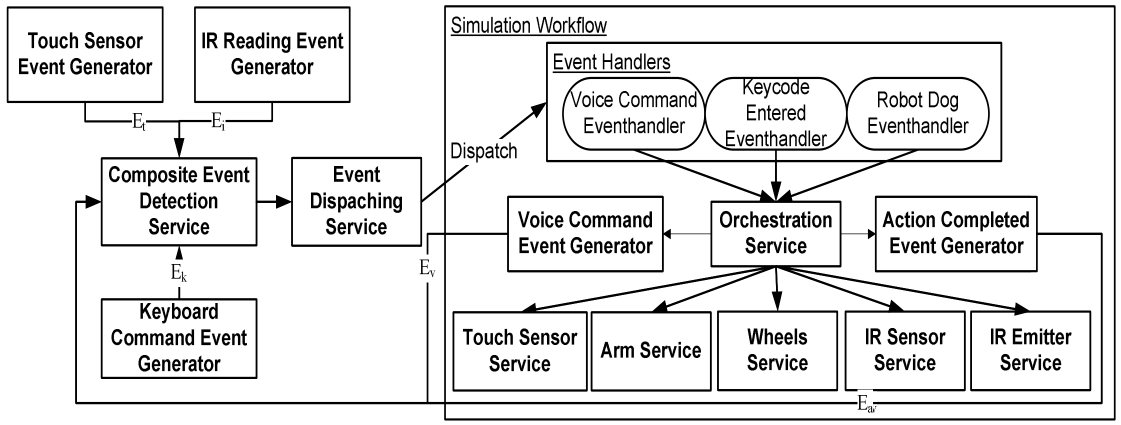

Touch Sensor Event Generator
IR Reading Event Generator
E_t
E_i
Composite Event Detection Service
Event Dispaching Service
E_k
Keyboard Command Event Generator
Simulation Workflow
Event Handlers
Voice Command Eventhandler
Keycode Entered Eventhandler
Robot Dog Eventhandler
Dispatch
E_v
Voice Command Event Generator
Orchestration Service
Action Completed Event Generator
Touch Sensor Service
Arm Service
Wheels Service
IR Sensor Service
IR Emitter Service
E_{av}

Figure 6.7: Simulation Case Study

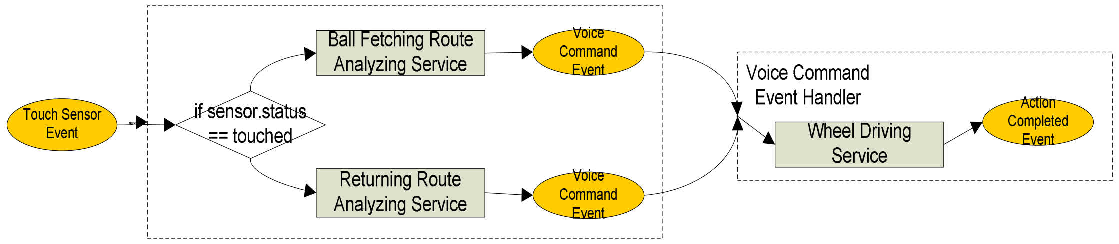

Touch Sensor Event
if sensor.status == touched
Ball Fetching Route Analyzing Service
Returning Route Analyzing Service
Voice Command Event
Voice Command Event
Voice Command Event Handler
Wheel Driving Service
Action Completed Event

Figure 6.8: Sample Event Graph

again. This is not a desired behavior because one wants the robot dog finishes its dance routine before it can start another. The model was adjusted accordingly to correct this fault.

One interested in the performance of processors in robotic applications. The number of sensors is an important factor in this evaluation; however, due to the limited number of sensors, one needs to perform simulation to complete the evaluation. This simulation used various numbers of touch sensor event generators to simulate different number of touch sensors. Each of them is a thread in the simulation. The simulations are based on the Intel Core 2 Duo processor for various combinations of processor frequencies and number of threads that are run in parallel. Figures 6.9 to 6.11 show the simulation results.

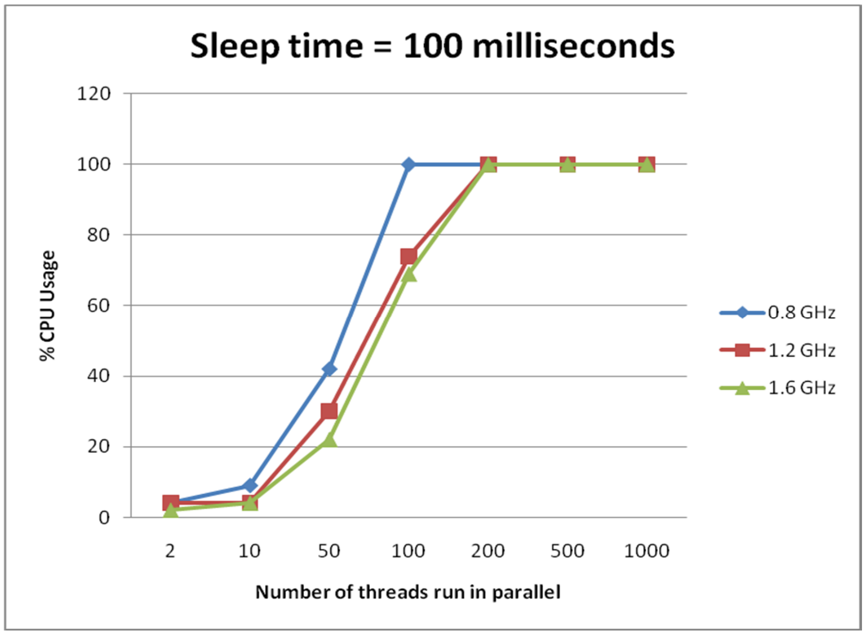


Figure 6.9: Sleep Time = 100 Milliseconds

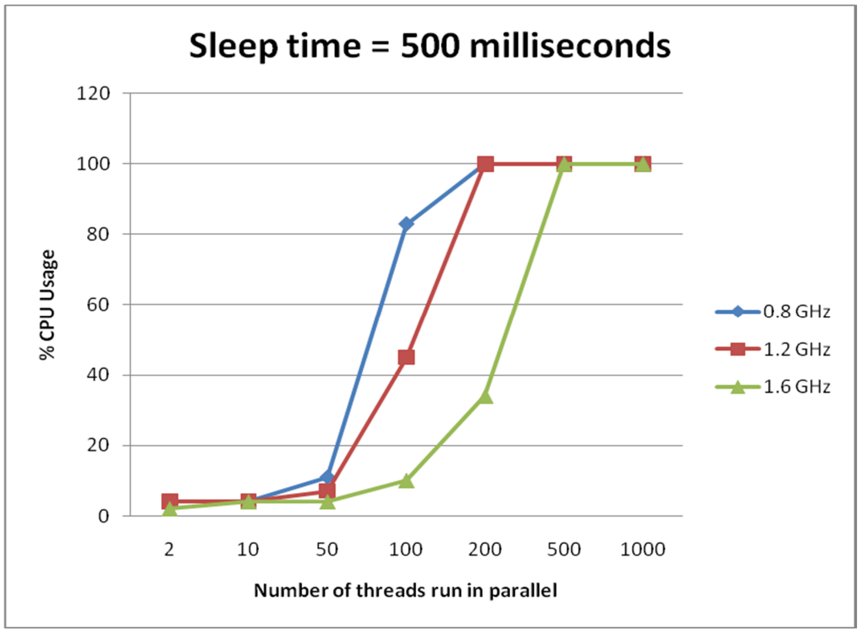


Figure 6.10: Sleep Time = 500 Milliseconds

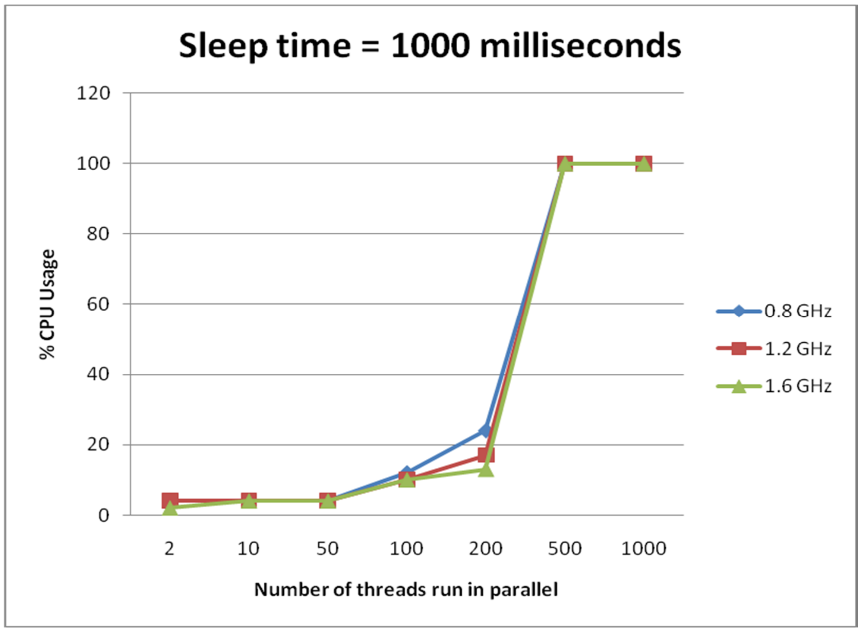


Figure 6.11: Sleep Time = 1,000 Milliseconds

6.7 Conclusion

This paper presents a service-oriented event-driven simulation framework. It is designed to allow the sharing and reusing events, composite event specifications, event generators, event handlers and event handler templates through an ontology system. Also, it offers the event specification of the service-oriented language PSML to allow static analysis. Furthermore, dynamic tuning can be achieved through the changing the event generators, composite event specifications and event handlers.

Chapter 7

LTBD: A TRIAGE SOLUTION FOR SAAS

7.1 Introduction

Originally ”triage” is a medical term to separate, sort, sift or select medical conditions. In software engineering, this assigns a priority and severity to failures, and this is done peliodioedes, say every hour or every day, or as necessary. Triage **?** is important for software quality assurance especially in a cloud environment. Specifically, in a modern SaaS (Software-as-a-Service), multiple tenants may share the same SaaS platform, and tenants share service components stored in the database. Each transaction for a tenant is composed by service components stored, and is compiled and deployed at runtime to execute application. If there is a failure, it is necessary to know the cause of the failure, but as each transaction software consists of shared service components, thus engineers need to examine the log data to identify the failed transactions, and then the service components involved that caused the failure.

For a large transaction system, engineers need to examine a large number of logs to identify the failures. Currently, this process is mostly manual, expensive, and slow. Furthermore, for a modern cloud-based transaction system, millions of transactions may arrive, and for each transaction log data will be produced resulting in millions of log data are produced and recorded. Furthermore, due to concurrent processing, log data will arrive in an asynchronous manner, thus log analysis is a challenging task.

Many frameworks have been addressing the logging issues from different aspects. For example Splunk Splunk.com (2015), LogStash Logstash.net (2015), uilog Zou *et al.* (2014), IBM SmartCloud Analytics Analysis (2015), Fluentd Project (2015), Google Analytics Google.com (2015), XpoLogXpolog.com (2015) and Nagios Nagios.org (2015). Depending on the approach, they all have different usages within their own domains. To handle the unstructured, big volume of data mixed with untraceable failures, none of the existing systems have a complete solution. This paper proposes a new logging solution for effective and efficient triaging of the production failures with these challenges.

Logs in cloud are characterized as volume, unstructured formats and large amount of untraceable failures. Failures and exceptions happen frequently on production unexpectedly as the developers continue putting new changes to existing or new systems. Often in commercial log processing, logs will be kept for one month, and then new data will erase 30-day old data. Even with trustworthy cloud platforms such as Amazon and Google, their services still experience constant downtime according to their status reports Status.aws.amazon.com (2015) Code.google.com (2015). Service downtime on production system become frequent, and every minute of failures on production can result in huge loss. Programmers normally put least priority at coding phase for logging, thus causing later on unstructured formats along with the heterogeneous deployment of the different components in cloud. As log data accumulate at the rate of hundreds millions per hour on commercial production sites, they become complicate to manage.

Most transaction processing systems use service-oriented structure. See Figure 7.1 for illustration. For example, a transaction system may have X services, and each transaction often involve say on average Y services, thus not only the transaction information is unique for each transaction, the software services involved are also different. Each transaction software consist of a number of services selected from the database, and they are composed in real-time to serve each transaction. Supporse $X = 200$ and $Y = 50$ Then $\mathrm{C}\binom{50}{200}$ combinations of transactions are formed. Moreover, with each transaction having a minimum of 10 services, and a maximum of 180 services, thus the number of transaction will be a extremely large number.

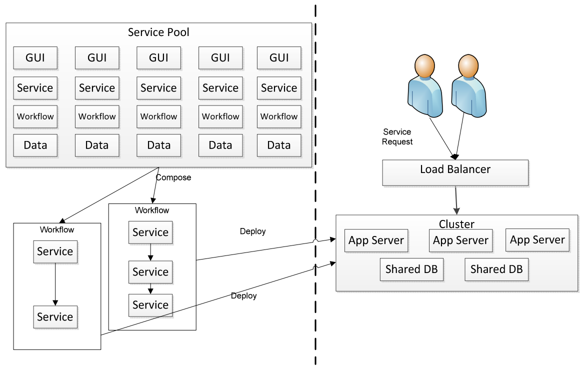
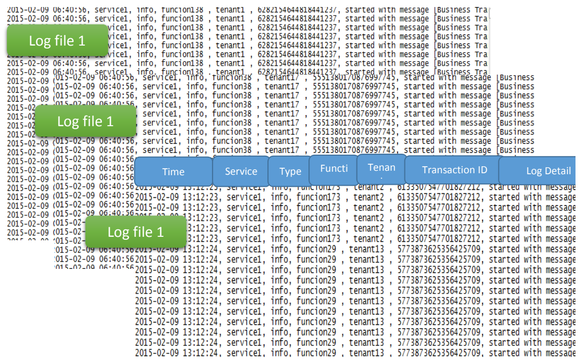


Figure 7.1: SOA Structure for Cloud Services



```
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
2015-02-09 06:40:56, service1, info, funcion138 , tenant1 , 6282154644818441237, started with message [Business Tra
```

```
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
015-02-09 06:40:56, service1, info, funcion38 , tenant17 , 5551380170876997745, started with message [Business
```

```
2015-02-09 13:12:23, service1, info, funcion173 , tenant2 , 6133507547701827212, started with message
2015-02-09 13:12:23, service1, info, funcion173 , tenant2 , 6133507547701827212, started with message
2015-02-09 13:12:23, service1, info, funcion173 , tenant2 , 6133507547701827212, started with message
2015-02-09 13:12:23, service1, info, funcion173 , tenant2 , 6133507547701827212, started with message
2015-02-09 13:12:23, service1, info, funcion173 , tenant2 , 6133507547701827212, started with message
2015-02-09 13:12:23, service1, info, funcion173 , tenant2 , 6133507547701827212, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
2015-02-09 13:12:24, service1, info, funcion29 , tenant13 , 5773873625356425709, started with message
```

Figure 7.2: Sample Log Files in Cloud

Log data are done at the best effort only, thus production of log data is not guaranteed. For example, Figure 7.2 showed a typical log data. Currently, most companies address this problem by having engineers on site to examine the log after software processing, and manually determine if any transaction has failed. This process is cumbersome, expensive, and time consuming.

Log data are unstructured as each transaction will be different. For example, the buyer (name, address, contact information, ID), seller (name, address, contact information, ID), items traded, dollar amount, time and place of transactions. As each transaction unique, log data for different transactions will be different. Specifically, each transaction log can have variations in

- Buyer;
- Seller;
- Items involved in a transaction;
- payment methods;
- Discount or promotion code;
- Currency;
- Time and place of transactions;
- Risk; and
- Compliance.

The same person engaged in two different transactions may result in different risks and compliance issues.

There can be multiple reasons for failures:

1. Code exception: Exception happened in the workflow, log information is still logged.
2. Service outrage: Entire or partial service is out thus no details is logged.
3. Media failure: The physical media of the server is malfunctioned thus no log or only partial logs are logged.

The result can always be categorized as below

1. Partial log missing.
2. Complete log missing.

Most of the time, if it is a pure workflow exception and the details can be logged, the log is traceable as the interaction log between services are still there. However, if there is a partial or complete missing log, it is hard to diagnose further. In the case of partial log missing, it complete the case as it depends on the available information in the log details and the key activity log entry availability.

As for complete service missing, it can be:

- Service out completely and no subsequent service was invoked.
- Only log service is out but service was not interrupted, thus the subsequent service call is still invoked.

In the later case we still need to continue trace in the other service for the workflow. This approach is consistent with the approach of using data to identify data **?** or smart data approach.

Contributions of this paper:

1. Proposed a failure detection and correction mechanisms into a real-time transaction management. The authors are not aware that any prior work has addressed this problem;
2. Proposed a workflow indexing method by using MapReduce.
3. The proposed system has been extensively simulated and evaluated using large amount of data, data size large enough for large online e-business systems;

The paper is organized as below. Section 2 describes the RTRM structure. Section 3 describes the related work. Section 4 designs the solution for a fast triage; section 5 evaluates the performance of the proposed solution.

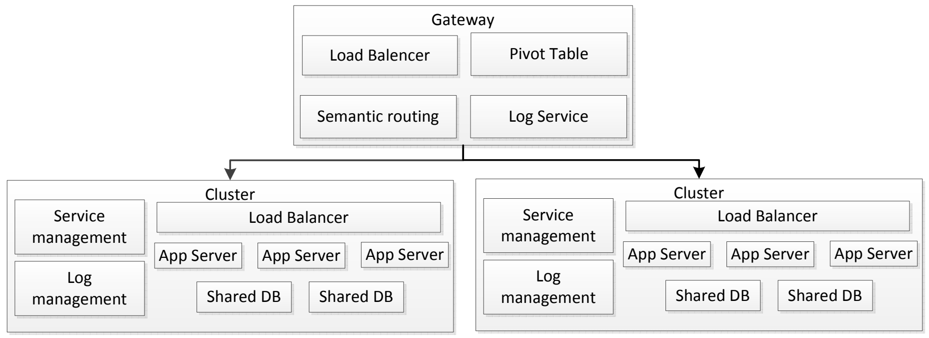


Figure 7.3: RTRM Architecture Design

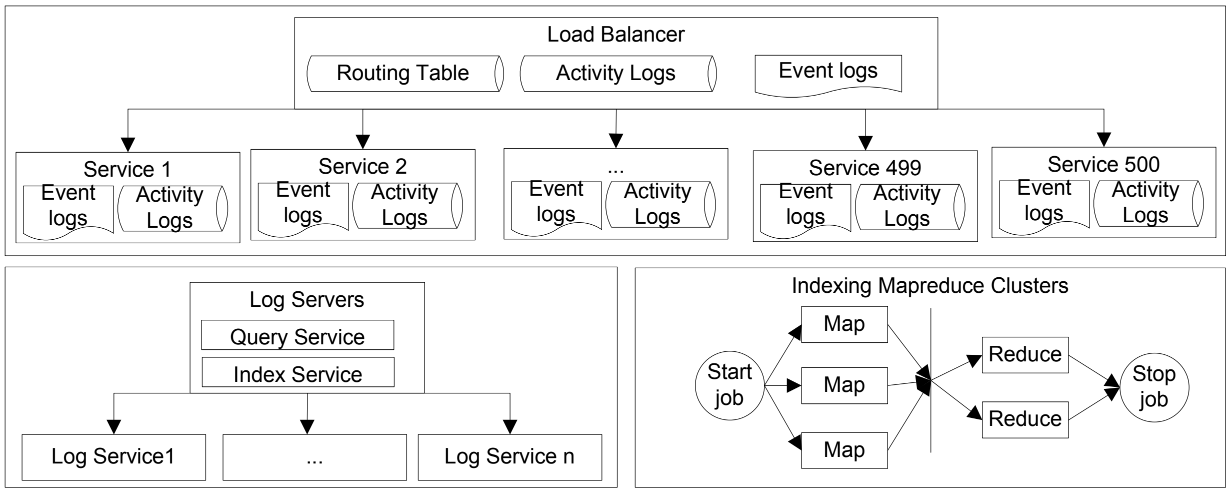


Figure 7.4: Log Architecture on a Cloud

7.2 RTRM Architecture/Tradditional Triage

7.2.1 RTRM Architecture

A typical online transaction system has the following architectures:

1. A real-time transaction management (RTRM) as shown in Figure 7.3 that takes user inputs and process the transaction using a pool of cloud resources. A RTR-M often has a number of participating systems such as credit card verification, payment methods, risk analysis, fraud prevention. And some of these systems are not under control of the RTRM;

2. Usually RTRM is structured in a service-oriented manner with tens of thousands of software services that can be called to provide services. As each user request is unique as the user will select the needed services just before submitting a transaction, thus numerous combinations of services will be called. For example, a client at Amazon.com may select two books and three CDs for Christmas gifts, and the transaction will select these books and CDs with Christmas gift wrap with expedited delivery before Christmas. Another client may buy one vacuum cleaner instead choose normal shipping method, and these two transactions are different and cannot be predicted before hand.

3. The RTRM produces significant log data that need to be analyzed. However, the RTRM cannot release much resources to handle logs because each server must be ready to receive user requests. As a variety of transactions will be created, the data created are not structured;

4. The log data are written into another cloud system LDAS (Log Data Analyzer System) at the end of each hour as shown in Figure 7.4, the processors must process the log data as soon as possible because more log data will be coming within a hour;

5. Another complication is that the RTRM may fail or other participating systems may fail, but either way the log data will either contains failure or miss important data, and the RTRM may not have time or resource to detect and correct these failures;

6. Instead LDAS must takes over the failure detection and correction after analyzing the data. Currently, failure detection is handled by engineers and thus caused significant effort and time.

The RTRM architecture can be seen on Figure 7.3, that is divided into two layers: Gateway layer and Cluster layer.A typical SaaS handles a user request in Algorithm 5 . The RTRM including scheduling, handling user requests, logging, distributed redundancy management and recovery, scalability including resource allocation and provisioning, and distributed and autonomous data migration for recovery and scalability.

Gateway does the high-level routing of the traffic for tenants. It is composed by the load balancer , semantic routing service,and pivot table.

1. Load balancer does the routing based on the semantic routing service, consistent hashing service and pivot table.

2. *Semantic routing* routes the traffic for a specific tenant to the same cluster or close by cluster. When traffic to one tenant comes in, it determines the transaction based on the tenant and routes directly to the corresponding cluster.

3. *Pivot table* documents the routing logic for the transactions in the gateway for the load balancer. It dumps the data to a replica after fixed intervals in a distributed system. The data of the pivot table will later on be used in the log retrieving logic.

Cluster is composed by the platform services and execution services.

1. Platform service serves the services with the management actions including deploy, scheduling, redundancy&recovery management etc.

2. Execution components includes the stateless app servers used to serve different services, stated DB cross app servers and cache service that to share the in memory data crass the stateless app servers.

Algorithm 5 Service Request Management

```
Input:
 1:
 2: Determine the tenant ID of this request;
 3: Check if the tenant application is already in the load balancer cache;
 4: if Application in cache then
 5:     Send the request for the tenant application for execution;
 6: else
 7:     Retrieve the tenant components from the SaaS database;
 8:     Compose the tenant application using the retrieved components;
 9:     Compiling the composed application;
10:     Send the request to the binary for execution;
11: end if
```

7.2.2 Traditional Triage

This subsection describes the current triage in detail to illustrate the difficulties encountered. Data size, large numbers of services and workflows, inconsistent coding, and inconsistent logging behaviors made log analysis difficult. For example, Figure 1 shows two isolated files stores the same workflow logs one fo LoginService the other for AuthenticationService.

According to the RTRM architecture, workflows, representing transactions, do not enter log data, but those participating services do. Thus, if a workflow failed, it is necessary to trace those logs entered by participating services. The issue becomes involved because

1. Each service may be used by many transactions;

2. The system may execute many transactions at the same time resulting in interleaving log data for these transactions;

3. Log data may not be entered because the transaction failed to complete;

4. Log data use service and workflow IDs in an inconsistent manner. For example, in the same workflow, some services will use service IDs, but other services use the workflow ID. In this case, searching using the workflow ID will produce incomplete log data. Figure 7.5 illustrates two services, the loginService uses ServiceID whereas the AuthenticationService uses the WorkflowID;

5. Service calls may not be explicit among services, thus one needs to scan the 36 GB log file to identify those missing find out those missing services. For example, LoginService called AuthenticationService but this information is not available in the log. Scanning and querying the log data will be expensive, and engineers need to identify the items to be searched first, and this needs an interactive process. As multiple transactions with shared services may fail with the same failure code, care must be exercise to distinguish those services that involved in multiple transactions that failed;

6. Missing log data causes troubles. For example in the number 3 box (Figure 7.5) at the AuthenticaonService, the next service call from AuthenticationService to AuthDBService cannot be found in the log. There are two alternative choices for engineers:

    - assuming there is no call missed as no log can indicate anything missing (but in this case, the engineer erred);

    - assuming there are something missing, but the engineer needs to determine which ones are missed.

In this case, AuthDBService is a good choice as it is involved in authentication, but what if multiple similar services are available or maybe other unrelated services are missed?

The conventional triage process is shown in Figure 7.6 with the following steps:

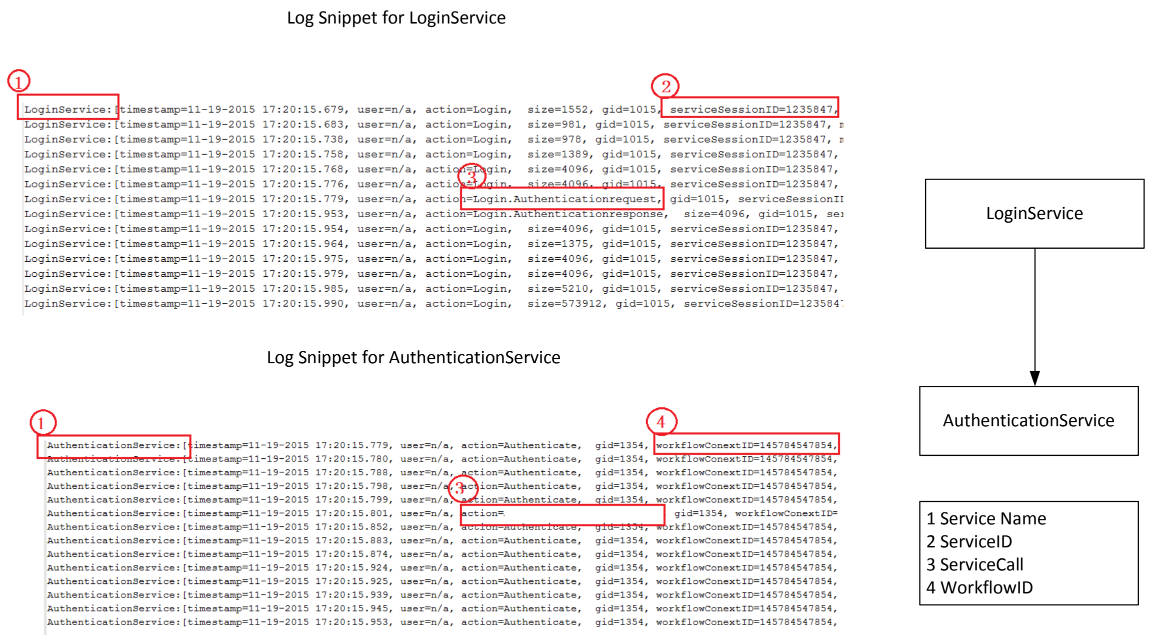
- Migrate the log from production to off-production servers;
- Identify related services in the workflows;
- Reconstruction the workflow from the log;
- Query the database and check code to understand the workflow; and
- Manually generate reports for the concerned workflows.

After the examination into the logs, database and code base, one can identify cause of the failure. The next step is to prioritize the issue based on business impacts. A most important business impact is how many similar transactions happened. The engineer needs to collect the number of impacted workflow types and the number of incidences to answer this. Existing queries are mostly based on preset queries or Spluk like systems that only report at failure code level. As a failure code can have multiple workflow types related, a more fine-grained understanding of workflows is necessary.

7.3 Related Work

This section first introduces the four metrics that based on the previous discussion for a online transaction processing system. Then, these metrics will be used to evaluate the existing logging solutions.

OLTP and OLAP: OLTP (Online transaction processing) and OLAP(Online Analytical Processing)Curino *et al.* (2011) can provide different functions.



```
LoginService:[timestamp=11-19-2015 17:20:15.679, user=n/a, action=Login,  size=1552, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.683, user=n/a, action=Login,  size=981, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.738, user=n/a, action=Login,  size=978, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.758, user=n/a, action=Login,  size=1389, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.768, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.776, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.779, user=n/a, action=Login.Authenticationrequest, gid=1015, serviceSessionII
LoginService:[timestamp=11-19-2015 17:20:15.953, user=n/a, action=Login.Authenticationresponse,  size=4096, gid=1015, se
LoginService:[timestamp=11-19-2015 17:20:15.954, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.964, user=n/a, action=Login,  size=1375, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.975, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.979, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.985, user=n/a, action=Login,  size=5210, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.990, user=n/a, action=Login,  size=573912, gid=1015, serviceSessionID=123584
```

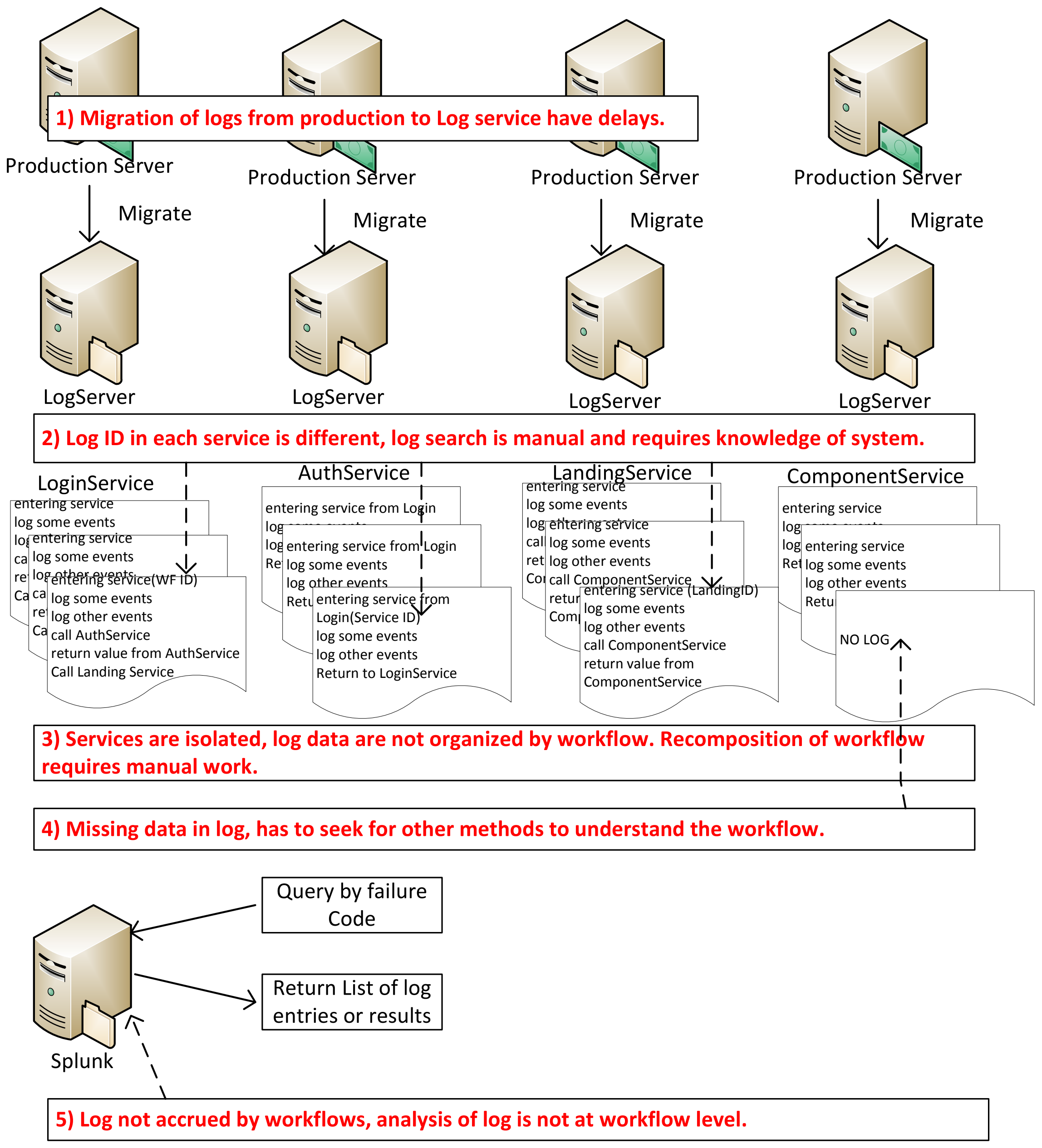
```
AuthenticationService:[timestamp=11-19-2015 17:20:15.779, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.780, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.788, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.798, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.799, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.801, user=n/a, action=                            gid=1354, workflowConextID=
AuthenticationService:[timestamp=11-19-2015 17:20:15.852, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.883, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.874, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.924, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.925, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.939, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.945, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.953, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
```

Figure 7.5: A Sample Workflow with Log Snippets

- OLTP systems provide source data, whereas OLAP systems help to analyze it.
- OLTP systems serve production transaction whereas OLAP queries are often complex and involve aggregations.
- OLTP and OLAP systems are isolated from each other most of the time.

Depending on the urgency for the log analysis, some log solutions will be categorized at the OLAP systems such as Splunk and LogStash. Some log solutions are categorized at the OLTP category such as Nagios that will enable the log processing system to stay along with the online transaction processing system. It is challenging to have a real-time checking mechanism to understand a desired system without impacting the production performance.

Fault-tolerant capabilities: Failure is a major factor impacts the online transaction processing.



1) Migration of logs from production to Log service have delays.
Production Server
Production Server
Production Server
Production Server
Migrate
Migrate
Migrate
Migrate
LogServer
LogServer
LogServer
LogServer
2) Log ID in each service is different, log search is manual and requires knowledge of system.
LoginService
AuthService
LandingService
ComponentService
entering service(WF ID)
log some events
log other events
call AuthService
return value from AuthService
Call Landing Service
entering service from Login(Service ID)
log some events
log other events
Return to LoginService
entering service (LandingID)
log some events
log other events
call ComponentService
return value from ComponentService
NO LOG
3) Services are isolated, log data are not organized by workflow. Recomposition of workflow requires manual work.
4) Missing data in log, has to seek for other methods to understand the workflow.
Query by failure Code
Return List of log entries or results
Splunk
5) Log not accrued by workflows, analysis of log is not at workflow level.

Figure 7.6: Conventional Triage Method and Problems

- Most of the tools can offer statistics details based the accumulated log data for one type of failures such as Splunk, LogStash, IBM smart cloud analytics.

- However, to understand a complete interaction graph of a transaction in a data center, it is necessary to have capability to recover the failure points for understanding the complete transaction workflow.

Failures can be categorized as two types, internal failures and external failures. In both cases, it is important to have the log-analysis system to plot a complete interaction graph based on the topology, previous log statistics, and data mining algorithms. Nagios has some of the fault alerting capability based on its predefined check points for the statistic data; however, tools like Splunk, LogStash neither can recover a failure interaction graph nor give a complete interaction graph for a given workflow.

Flexible data handling capabilities: In the cloud infrastructure, each service serves multiple tenants and transactions at a time in public clouds Tsai *et al.* (2011c)Tsai *et al.* (2011b). As transaction volume grows, log data also increase dramatically.

- The volume of logs to be handled in a cloud pose a challenging task while retrieving data.

- For fast throughput of the service, logs are logged sequentially by time in plain text files. For example, log files can be either saved on the servers and then forwarded to the log servers with delays or rolled hourly/daily. It is necessary for the log-processing system to have the capability to handle the big data in logs.

- Based on the volume of the data, OLTP system normally do not handle the log queries directly unless really urgent or requires minimum delays. Based on the volume of the transaction, big data normally will cause significant impacts at performance of both CPU and memory of production servers.

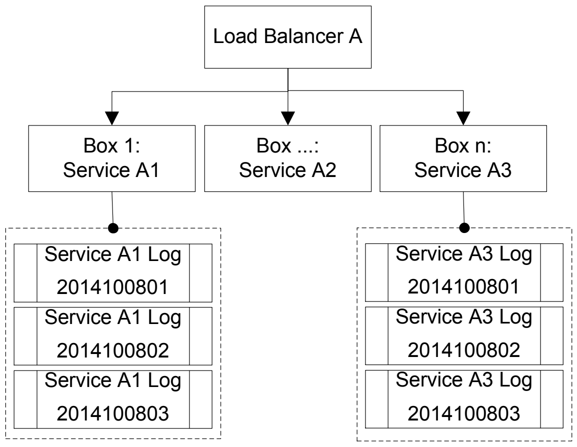

Load Balancer A
Box 1: Service A1
Box ...: Service A2
Box n: Service A3
Service A1 Log 2014100801
Service A1 Log 2014100802
Service A1 Log 2014100803
Service A3 Log 2014100801
Service A3 Log 2014100802
Service A3 Log 2014100803

Figure 7.7: Log File Structure in Cloud

Figure 7.7 describes a deploy structure of the log file in a transaction processing system. Box represents the physical server to host the service in cloud. Each client interact with the services in a transaction. Log files are normally stored based on the service running in each App Servers. The volume of a typical cloud for host online transaction processing can be estimated to be up to 36GB per hour based on the below estimation: Suppose 10 transactions complete per second for each tenant, and each transaction on average interacts with 20 services. Each service logs an average 50 entries for each transaction. And each log entry is estimated with 1kb of data. For each hour, the average log file size for each service is 1.8 GB. And the estimated log size is for every tenant is 36GB of data based on the below calculation. In the case 10 tenants live in this cloud, the logs grow easily to 360GB hourly.

White-box vs black-box

- A log service analysis is considered as black when it exposes the input and out of the services only.

- When the details of the internal workflow of the service is logged, it is considered a white-box log analysis.

Nagios, Google Analytics are black-box logging solution: Nagios, interested in the result or accumulated result of a certain type of predefined service call; Google Analytics, documents the call to a remote service at the stand point of a client. Splunk and LogStash solutions can get inside of the workflow of the service details called white-box analysis.

Based on these information, the paper briefly surveyed the below major logging solutions.

- Splunk Splunk.com (2015) is a commercial product that captures indexes and correlates data in a searchable repository. It uses Hadoop in the backend for indexing the logs and then serves the log searching via its frontend by the search processing language they offered. It gives the companies flexibility in their existing logging environment and enable to integrate the logs via one searching portal. However, its framework focuses on retaining the data and indexing afterwards for better searching purposes. It is an OLAP system, and it does not have the capability to recover the failed logs.

- Cloud Application Logging for Forensics Marty (2011) describes a logging solution to provide a proactive approach to logging to ensure the data required by forensic investigations will be generated and collected. It is an OLAP system.

- Logstash Logstash.net (2015) is an open-source project offers similar functions as Splunk. The ELK: Elasticsearch (search), Logstash (ingestion and processing), and Kibana (reporting and visualization) stack forms the framework and all of them are Apache-licensed projects. It is an OLAP system, and it does not have the capability to recover the failed logs.

- Unilog Zou *et al.* (2014) classifies log into different catalogs by fault types. It also used Keyword matrix to accelerate the speed of classification. Unilog is more of data analysis of the log thus it is an OLAP system.

- IBM SmartCloud Analytics Analysis (2015) is a log analytics tool. It allows the customer to analyze the operational data types in the logs to help identify, isolate and resolve problems. The software can take source of data from multiple sources including logs, events, metrics, support documents and trouble tickets. SmartCloud Analytics is an OLAP system that can handle big data.

- Nagios Nagios.org (2015) Log Server is a log monitoring and management system that allows organizations to configure, view and sort logs on a given network. The data of the log is from real time and it allows the customers to predefine various check points and metrics for the monitoring. A certain type of alert is triggered when threshold is surpassed, and it will alert the customer with the exception details at the failure point. It is with the OLTP system, it can detect the exceptions at real time.

- Google Analytics Google.com (2015) Web log analysis tools offered by Google. It works by the inclusion of a JavaSrcirpt code on the website. When users view a pages, the JavaScript retrieves the data about the page and send back the information to the Google Analytics services.It only documents the input and output of the page thus it has no visibility of the service workflow on the application host websites. It is an OLAP system with black box visibility of the services.

- XpoLog Xpolog.com (2015) is a log-analysis and management platform to access, config, analyze logs, and view information. Logs are exported to databases and SQL queries are used to query against the logs. The XpoLog contains the following components: Log Viewer, Log Analyzer, Reports, Seach, Log Management. It is an OLAP system with black-box visibility of services.

- FluentdProject (2015) is an alternative to Splunk. Its backend is written in C++ and frontend in Ruby. It uses a pluggable architecture makes it compatible with numerous data sources and data outputs. It also supports as the backend for alerting, analysis and archiving of the log data. It is an OLAP system with black-box visibility of services.

- Other tools

  - Deep-log analyzer Deep-software.com (2014) is a local log analysis tool that works on the website without requiring any code on the site. It is similar to Google Analytics.

  - AWStats Destailleur (2015) is a free Web analysis tool evaluates Web logs with different reports. It can be used to analyze the FTP and mail logs as well as Web log files. It also have the capability to export reports to XML, text, and PDF.

  - RealTracker Realtracker.com (2015) places code on a Web pages to track the page. It is similar to Google Analytics, it can also generate reports based on these log data.

The table 7.1 summarized the major logging solutions.

Table 7.1: Comparision of Major Logging Solutions

Tool name	**OLTP /OLAP**	**Fault tolerant capability**	**Big Data handling capability**	**White box /black box**
Splunk	OLAP	N.A.	Y.	White box
Cloud Log for Forensics	OLAP	N.A.	N.A.	White box
Logstash.	OLAP	N.A.	Y.	White box
Unilog	OLAP	N.A.	N.A.	White box
IBM Smart Cloud Analytics	OLAP	N.A.	Y.	White box
Fluentd	OLAP	N.A.	Y.	White box
Nagios	OLTP	Y.	Y.	Black box
Google Analytics	OLAP	N.A.	Y.	Black box
XpoLog	OLAP	N.A.	Y.	White box

7.4 Logging Solutions

A common method for searching the related information in a box is to use grep function in the Unix/Linux environment. However, this only works when one exactly know the log file. It becomes time and resource consuming to grep one transaction ID crossing the data center in seeking for only one set of logs. It is important to model the logs first and then define the operations upon for triage.

Need for real-time log analysis at transaction times: Majority of log analyze solutions such as Splunk, LogStash, and Fluentd are OLAP system, that means that the log processing and analyzing process normally happen off the production servers. As the transition of log data migrate from production machines to log servers, a significant delay incurred between log reports and failure time.

Workflow and failure point recovery capabilities: Majority of these solutions focus on the log reporting part of production data, and most of these such as Splunk, IBM Smart Cloud Analytics are not designed to describe a transaction lifecycle. Hence when a transaction failure happens, the log data are normally detached at the failure point and the engineer cannot recover the failures easily.

Flexibly Data processing capabilities: Table 7.1 surveyed the logging solutions. From the backend, most log processing solutions used Mapreduce/indexing or similar techniques to handle the big volume of data. This in a sense will reflect the potential delay in the indexing part of it. On the other hand, the query keyword might not be in the existing index, it might be necessary to have a re-index mechanism after no result is hit after the initial query.

White-box log analysis: Log data can be served for different purpose and so are the log analyzing tools. Tools such as Nagios and Google Analytics view the target system as a black box and interested in the interaction of the service call only; however some others such as Splunk and Fluentd can dig into the details of the service workflow. For triage the production failure, developers prefer the white box solution to have visibility of the issue. But the more exposure to the data, the more unstructured the data are. To resolve that, a new log structure will be needed to unify log data for efficient searching and log tracking.

7.4.1 Modeling Logs

1. **Behavior logs (BL)** is related with the interactions of services. It includes the activity of entering a service, exiting a service, and interactions with another

services. These details happen at service calls only, the data are structured and can be saved for efficient processing.

2. **Event logs (EL)** is the detailed logging information during the executing of workflow, it contains the print outs during the executions of the service that can be large in size and thus documented in the plain text files on servers.

BL and EL together form the call stack of a workflow in logs. Logging the BL and EL at the code is straight forward, as following:

```
int serviceCall(){
...
// calling serviceB from serviceA.
int retvalue = callserviceB(serviceRequest);
// log behavior call into behavior log.
Log.logtoBehavior(this, "serviceB");
...
int valB;
....
// do some processing. log valB to event log.
Log.logtoEvent(this,valB);
...
return flag;
}
```

BL can be stored in tables in DB, that are a collection of Log Entries as

$\sum BehaviorEntry$

$BehaviorEntry = < Ts, Lt, S, F, Ts, Tx, Ln, S >$where

$< Ts >$ is the time stamp;

$< Lt >$ is the log type that belongs to the enumeration of start, end, break, interaction etc;

$< S >$ is the service name;

$< F >$ is the function name;

$< Ts >$ is the tenant ID;

$< Tx >$ is the transaction ID;

$< Ln >$ is the line number in the log file, and

$< S >$ is the log message, typically plain text file.

EL can be saved as log files, that are a collection of Log Entries at each file $\sum EventEntry$. Each line of the EL is a tuple that contains detail for a transaction, that can be represented as:

$EventEntry = < Ts, Ll, S, F, Te, Tx, E, S >$ where $< Ts >$ is the time stamp;

$< Ll >$ is the log level that belongs to an enumeration of info, warning, and failure;

$< S >$ is the service name;

$< F >$ is the function name;

$< Ts >$ is the tenant ID;

$< Tx >$ is the transaction ID;

$< E >$ is the failure code, and

$< S >$ is the log message, typically plain text or XML file. If this EL a service call, it will be the endpoint to the next service.

A tenant is a group of users sharing the same view on cloud services. A tenant consumes same set of shared services, and contains a number of properties and functions as in a tuple $< TenantInfo < TenantID, TenantName >, TenantConfig < Service, AccessControl, Policy >>$

$TenantInfo$ is a tuple $< TenantID, TenantName >$ that contains the tenant Id and name.

$TenantID$ is a unique ID that represents a tenant in cloud.

$TenantName$ is the name for the tenant.

$TenantConfig$ defines the tenant related details in the cloud other than identity. It is a tuple $< Service, AccessControl, Policy >$

$Service$ defines the used atomic and composed service used for a tenant.

$AccessControl$ defines the tenant level access including Read, Write, Execute and delete.

$Policy$ defines the different level of policies to the tenants in a cloud.

Any major database can be used to store the BL, for instance relational database, column-based database or BigTable. Different type of storage backend can be chosen depending on the requirement for the performance in log storing and searching. In fact, separating BL and EL will change the existing logging structure slightly. However, separation of the BL out of the EL will extract the essential information for the call stack of the workflow and thus enable the effective rebuilding of the call stack.

Storing the BL in a database will greatly increase the searching speed. On the other hand, store the event logs in the database is not practical duo to the volume and performance concern. BL will count for 1% to 2% of the logs thus making this solution practical.

7.4.2 Triage Single Workflow

A service call can be rebuilt as a connected directed graph trough the BL. Algorithm 6 states log traversing. When no problem happens to a transaction, the graph is connected and traversable. When a failure occurs or missing log entry happens, the connections between the services can be lost thus the graph becomes disconnected.

This is called Failed interaction call (FIC). It is necessary to recover the FIC and rebuild the call stack.

With the to be proposed solutions one can identify via the transaction ID, tenant ID in the BL, and regardless what edge it is, one can recover the FIC for the workflow by algorithms.

Algorithm 6 Log Matching Algorithm

```
Input: $TimeStamp, TenantId, TransactionId$
Output: $\sum LogEntries$ = ResultSet;
  ResultSet;
  BehaviorEntry;
  while GetNextAccessedService!=NULL do
    if Application in BehaviorLog then
      BehaviorEntry.starteEntry = selectStartFromCache(TenantID, TransactionID);
      BehaviorEntry.endEntry = selectEndFromCache(TenantID, TransactionID);
    else
      BehaviorEntry.starteEntry = selectStartFromBahaviorLogs(TenantID, TransactionID);
      BehaviorEntry.endEntry = selectStartFromBehaviorLogs(TenantID, TransactionID);
    end if
    ResultSet.append(BehaviorEntry);
    BFS(ResultSet);
  end while
```

Definition 7.4.1. (**IN: Interaction Network**) An interaction network is denoted as $G(V, E, Att)$ where

V is the set of vertices, that denotes services;

$E \subseteq V \times V$ is the set of edges, that denotes the service calls;

and $Att = \{a_1, a_2, ...a_n\}$ is the set of attributes for a vertex, n is the total number of the attributes.

Definition 7.4.2. (**IIN: Incomplete Interaction Network**): given an interaction network G = (V, E, Att) and a network G' = (V', E' Att'), network G' is called an Incomplete Interaction Network.

G' can be called IIN has to meet the conditions: iff (1) V' $\subset$ V and E' $\subset$ E (2) $\forall$ v $\in$ V and $\forall$ v' $\in$ V' if v = v' then Att(v) = Att(v'); (3) $\exists$ v $\subset$ V and $\exists$ e $\in$ E(V) , but e $\notin$ E'(V')

There can be different types of IINs such as vertices or edges are missing.

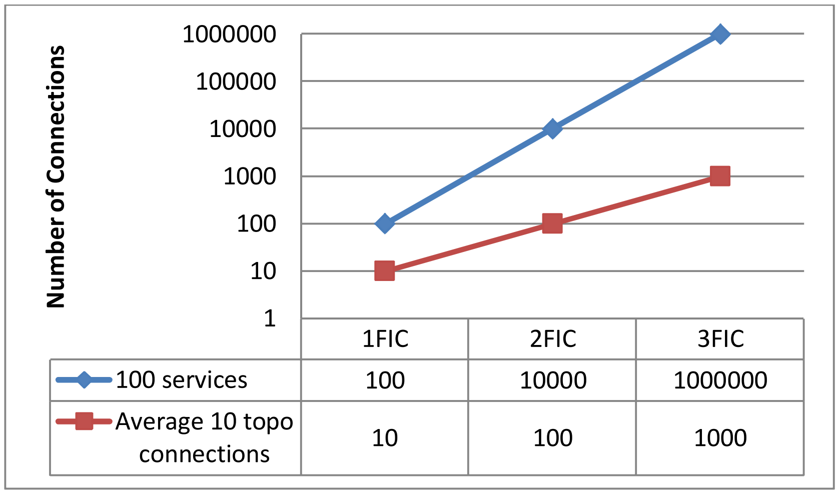
Definition 7.4.3. (INT: Interaction Network with Time dimension): An interaction network is denoted as $GT = G(V, E, Att, T)$ where V is the set of vertices, that denotes the services;

$E \subseteq V \times V$ is the set of edges, that denotes service calls;

and $Att = \{a_1, a_2, ...a_n\}$ is the set of attributes for vertices, n is the total number of attributes. Att denotes the attributes to describe services.

and $T = \{t_1, t_2, ...t_n\}$ is the timestamp for the G. In implementation, we normally use t_i to denote the time span between t_i and ti_{-1} .

Topology is the static interaction graph between services that can be retrieved from source code, configuration and production firewall rules. It describes an overall interactions between different services.



	1FIC	2FIC	3FIC
100 services	100	10000	1000000
Average 10 topo connections	10	100	1000

Figure 7.8: Number Of Workflow Combinations With And Without Topology

For example, in a data center with N services, theoretically, any service can have N possible connecting destinations. For a workflow involves 50 services, in a data center with 100 services deployed, with one FIC happened in the workflow, the possible services to be investigated can be up to 100. If there are N FICs in the workflow, the combination of the service will soon make it impossible to investigate.

With the topology involvement of production site, it dramatically reduces the candidate service in Figure 7.8 for the investigation and on average, the total outbound connection can be eliminated to a very reasonable number. Having the below Complete Topology before doing the service rebuild thus will effectively help the workflow rebuild with FICs.

Definition 7.4.4. (**CT: Complete Topology**): is denoted as $GT(V, E, Att, T)$ where V is $\forall T$ the set of completed vertices used defined in the code and configuration. E is $\forall T$ the set of completed Edge used defined in the code and configuration. Att is $\forall T$ the set of completed attributes related.

T denote the time span between t_1 and t_n that t_n denotes the current time and t_1 denotes the start time of the service.

For example, a workflow A executed at t_i, v1 represents the service 1 and v2 represents the service 2 and v3 represents the service 3. There is a service call from v1 to v2 and then from v2 to v3.

At a different t_{i+1} , the same workflow was executed, but it only has the connection from service v1 to v2, and service v2 is missing the connection to v3 at the time stamp t_{i-1}.

This can be

- A missing log entry that means no issue,
- A one time failure such as a timeout,
- Code failure that caused a service outrage.

Definition 7.4.5. **(SC: Structural Rebuild)** Structural rebuild defines the mechanism to recover the FIC for an IIN. It can be implemented in different ways. Two typical ways can be empirical finding and data mining. The Algorithm 7 describes the method for the FIC completion on production.

- Empirical finding: If there is a FIC from $v_i \to v_j$ at t_i , and at t_{i+1} there is no call from from $v_i \to v_j$. One can derive that there is a potential FIC from from $v_i \to v_j$.

  Empirical finding can be based on statistics Chuvakin *et al.* (2012), calculating the weight of the service calls by the statistics.

  $W(E_{i \to j}) = \frac{\sum V_i \to Vj}{\sum_{k=0}^{n} V_i \to Vk}$

  $E_{i \to j}$ denotes the service call from service V_i to V_j Weight W is calculated based on the percentage of all the outbound service calls from V_i to V_j divided by the total outbound n types of service calls from V_i .

  $W(E_{i \to j})$ is used to rank the candidate edges for the FIC.

Algorithm 7 FIC Rebuild Algorithm

```
Input: $IN : G, V : start$
Output: Result;
  if start != null then
    Queue queue = new Queue();
    queue.Add(start);
    Set processedServices = new Set();
    V next = null;
    while queue.Count > 0 do
      next = (V)queue.Dequeue();
      if processedServices.Contains(next.Id) then
        continue;
      end if
      Iterator iter = next.next();
      while iter.hasNext() do
        if iter.next().isInTopology() is true then
          queue.Enqueue(iter.next());
        else if iter.next().isTraversableInHistory(pasthour) then
          LogStep(next,"recovered based on statistics");
          queue.Enqueue(iter.next());
        else if iter.possibleNextPR() then
          LogStep(next,"recovered based on PR");
          queue.Enqueue(iter.next());
        else if iter.possibleNextDatamining() then
            LogStep(next,"recovered based on Data Mining");
            queue.Enqueue(iter.next());
        end if
    end while
    LogStep(next);
    processedSteps.Add(next);
  end while
end if
```

- Page Rank Page *et al.* (1999): Page rank used to weight the edges among the vertices for an IIN G. For example, for a service U the numerical weight can be denoted as PR(U). PR value of an outbound service call is equal to the service's own PageRank score divided by the number of outbound links L(V). The below example shows the PR(U) is equal to the PR of its outbound service V_i, V_{i+1}, V_{i+2}

  $PR(U) = \frac{P(V_i)}{L(V_i)} + \frac{PR(V_{i+1})}{L(V_{i+1})} + \frac{PR(V_{i+2})}{L(V_{i+2})}$

  In the general case, the PageRank Page *et al.* (1999) value for any vertices (services) U can be expressed as:

  $PR(U) = \sum_{v \subseteq B_u} \frac{PR(V)}{L(V)}$

  PageRank value for a service U depends on the PageRank values for each page V contained in the set Bu (the set containing all vertices connect to vertices U), divided by the number L(V) of calls from page V. $PR(U)$ is used to rank the candidate edges for the FIC.

- KNN: The intuition here is that similar transactions can result in similar calling structures among the different services. Hence if one transaction with an existing calling structure has a FIC, a potential prediction can be made for the target service by using the different data mining algorithms. Several memory-based algorithm can be used, including k-Nearest Neighborhood(kNN) Wu *et al.* (2008). Using kNN, the most commonly used calculation is Euclidean distance formula.

  $distance = \sqrt{\sum_{i=1}^{n}(p_i - q_i)^2}$

  distance is used to rank the candidate edges for the FIC. It makes use of two vectors of data: p and q. p is the set of known patterns and the q is the set of pattern we are trying to detect. One goes through each set of vectors for the workflow,k closest neighbors can be get by using a majority voting scheme.

7.4.3 Macro Log Analysis of Workflows

The previous section discussed **micro log analysis** for individual transactions. As it deals with individual transactions only, it can be done at real-time where the resource is limited.

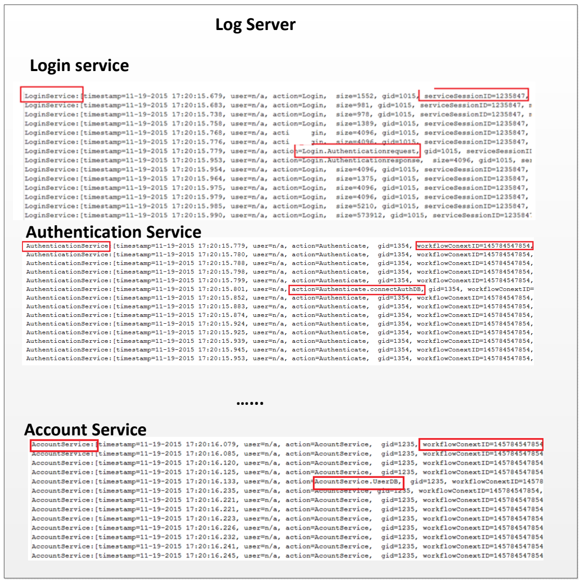
Macro log analysis deals with multiple transactions at the same time. It includes statistics analysis, workflow identification and clustering. It is computationally expensive and thus it is suitable mainly for off-time analysis. The log data can be preprocessed to produce the indexes to speed up the query processing. For example, this can be done by Splunk. Specifically, queries from Splunk return the frequency of each transaction with the list of services. An example from Splunk Query is in Figure 7.11.

Traditional triage performs the following two steps:

- **Step 1:** Log data are migrated from production to backend servers on a regular basis. Log data are stored in the format of raw text files and DB entries as shown in 7.9. Log data are organized chronologically by services instead of by workflows. Each workflow contains various services and each service log is stored in a corresponding file and database.

- **Step 2:** Log-processing tool such as Splunk indexes log data as shown in Figure 7.10, and one can query the log entry indexes for frequency analysis. Query for log entries can be improved since log entries are indexed. But workflows are not indexed at this phase. It is possible to analyze multiple transactions at this time, but it requires extensive work and significant experience.

- **Step 3:** Step 3 named workflow indexing can be added to analyze a collection of workflows. Figure 7.12 shows the process for log triage including the three steps.



```
LoginService:[timestamp=11-19-2015 17:20:15.679, user=n/a, action=Login,  size=1552, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.683, user=n/a, action=Login,  size=981, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.738, user=n/a, action=Login,  size=978, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.758, user=n/a, action=Login,  size=1389, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.768, user=n/a, acti     gin,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.776, user=n/a, acti     gin,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.779, user=n/a, action=Login.Authenticationrequest, gid=1015, serviceSessionI
LoginService:[timestamp=11-19-2015 17:20:15.953, user=n/a, action=Login.Authenticationresponse,  size=4096, gid=1015, se
LoginService:[timestamp=11-19-2015 17:20:15.954, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.964, user=n/a, action=Login,  size=1375, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.975, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.979, user=n/a, action=Login,  size=4096, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.985, user=n/a, action=Login,  size=5210, gid=1015, serviceSessionID=1235847,
LoginService:[timestamp=11-19-2015 17:20:15.990, user=n/a, action=Login,  size=573912, gid=1015, serviceSessionID=123584
```

```
AuthenticationService:[timestamp=11-19-2015 17:20:15.779, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.780, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.788, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.798, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.799, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.801, user=n/a, action=Authenticate.connectAuthDB, gid=1354, workflowConextID=
AuthenticationService:[timestamp=11-19-2015 17:20:15.852, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.883, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.874, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.924, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.925, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.939, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.945, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
AuthenticationService:[timestamp=11-19-2015 17:20:15.953, user=n/a, action=Authenticate,  gid=1354, workflowConextID=145784547854,
```

```
AccountService:[timestamp=11-19-2015 17:20:16.079, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.085, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.120, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.125, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.133, user=n/a, action=AcountService.UserDB,  gid=1235, workflowConextID=14578
AccountService:[timestamp=11-19-2015 17:20:16.235, user=n/a, action=AcountService, gid=1235, workflowConextID=145784547854,
AccountService:[timestamp=11-19-2015 17:20:16.221, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.221, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.223, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.226, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.232, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.241, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
AccountService:[timestamp=11-19-2015 17:20:16.245, user=n/a, action=AcountService,  gid=1235, workflowConextID=145784547854
```

Figure 7.9: Raw Log Files Stored on Log Server

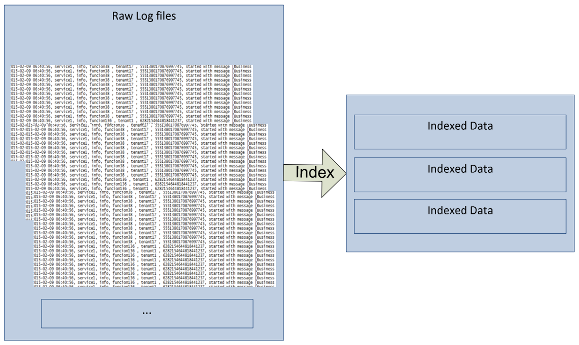
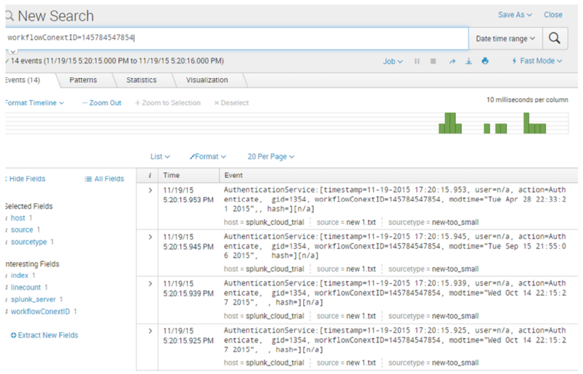


Figure 7.10: Index the Raw Log Files for Log Query



```
workflowConextID=145784547854
14 events (11/19/15 5:20:15.000 PM to 11/19/15 5:20:16.000 PM)

i  Time                    Event
>  11/19/15 5:20:15.953 PM AuthenticationService:[timestamp=11-19-2015 17:20:15.953, user=n/a, action=Auth
                           enticate,  gid=1354, workflowConextID=145784547854, modtime="Tue Apr 28 22:33:2
                           1 2015",, hash=][n/a]
                           host = splunk_cloud_trial   source = new 1.txt   sourcetype = new-too_small
>  11/19/15 5:20:15.945 PM AuthenticationService:[timestamp=11-19-2015 17:20:15.945, user=n/a, action=Auth
                           enticate,  gid=1354, workflowConextID=145784547854, modtime="Tue Sep 15 21:55:0
                           6 2015",   hash=][n/a]
                           host = splunk_cloud_trial   source = new 1.txt   sourcetype = new-too_small
>  11/19/15 5:20:15.939 PM AuthenticationService:[timestamp=11-19-2015 17:20:15.939, user=n/a, action=Auth
                           enticate,  gid=1354, workflowConextID=145784547854, modtime="Wed Oct 14 22:15:2
                           7 2015",  , hash=][n/a]
                           host = splunk_cloud_trial   source = new 1.txt   sourcetype = new-too_small
>  11/19/15 5:20:15.925 PM AuthenticationService:[timestamp=11-19-2015 17:20:15.925, user=n/a, action=Auth
                           enticate,  gid=1354, workflowConextID=145784547854, modtime="Wed Oct 14 22:15:2
                           7 2015",  , hash=][n/a]
                           host = splunk_cloud_trial   source = new 1.txt   sourcetype = new-too_small
```

Figure 7.11: A Splunk Sample Query Result Represented in a List

There are three challenges for workflow indexing: workflow construction, processing storage/time requirements and processing data size.

Workflow construction can be resolved by the algorithms in section 4.2. Each individual transaction data are first identified, traversed and constructed into the workflow.

Processing storage/time requirements are challenging as each service contains numerous log entries and most of them are irrelevant to the workflow structure, the workflow structure needs to be extracted and stored in a data structure requires less memory for clustering the workflows by types.

There are three ways to store an individual workflow:

1. Nodes as objects and edges as pointers;

2. A list of edges between numbered nodes;

3. A matrix containing all edges between numbered node x and node y.

Indexing the workflows requires comparing two workflows by types, the first and second data structure both need to traverse the workflow for the comparison, thus the time complexity will be the BFS or DFS time complexity. The third, matrix is enabled with its math operation, and if it is a bit array, it can be O(1).

Bit array, known as bit map, is an array data structure that compactly stores bits. Bit array is effective at bit-level parallelism to perform operations quickly Wikipedia (2015). Comparing if two bit array is the same can be as simple as do an OR operation between two arrays and this is O(1). The below definition defines the data structure WorkflowBitArray. As it stores in bits, it can efficiently solve the processing storage and time issue.

Definition 7.4.6. (**WorkflowBitArray**): For a bit array $WorkflowBitArray[N][M]$, $WorkflowBitArray$ denotes a workflow, index N and M at row and column denotes to a service and it ranges from 0 to N-1. The corresponding bit is can be true(1) or false(0). If service n calls service m, $WorkflowBitArray[n][m]$ is marked as true(1); Otherwise, marked as false(0).

OR operation of bit array is used to examine if two workflows have the same structure.

- If WorkflowBitArrayA OR WorkflowBitArrayB is true(1) then two workflows are different;
- If WorkflowBitArrayA OR WorkflowBitArrayB is false(0), then two workflows have the same structure.

For example, in a three-service workflow, service A calls service B, and then service B calls service C. It is represented as:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

.

As for comparison, another workflow B stored as:

$$\begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{pmatrix}$$

.

One can do $(workflowA)\ OR\ (workflowB)$, and the following result shows that the two workflows are the same type.

$$(010001000)\ OR\ (010001000) = 000000000 \tag{7.1}$$

For a third workflowC structured as bit array 010000000. The following $(workflowA)$ OR $(workflowC)$ shows they are identical type.

$$(010001000)\ OR\ (010001000) = 000000000 \tag{7.2}$$

For a third workflow C structured as bit array 010000000. The following $(workflowA)$ OR $(workflowC)$ shows they are different types.

$$(010001000)\ OR\ (010000000) = 000001000 \tag{7.3}$$

Processing data size can be resolved by using MapReduce or other parallel algorithms. Algorithm 8 and 9 illustrate the process to use Mapreduce to scale out the computation.

- At the mapper phase 7.13 of the Mapreduce, individual workflows are constructed by different mappers, the results are stored in WorkflowBitArrays.

- At the reducer phase 7.14, bit array OR operation is executed on each WorkflowBitArray to categorize the workflows. Workflow indexes are generated as the output of Mapreduce.

- Various queries can be done based on workflow indexes. A report can be generated from the workflow indexes to reflect the workflow statistics.

For instance, one can use Mapreduce to calculate all LoginService types out of one hour RTRM logs. Lots of login flows are related but with variants by countries, including USLoginFlow, CandadaLoginFlow, and ChinaLoginFlow. As shown in 7.15, the following steps can be done for the indexing of login workflows.

1. Query for the log in related services, identified around 30,000 transactions.

2. Distribute the transaction logs to 20 different mappers to construct the workflows, each mapper handles around 1500 transactions. As the log data are pre-indexed by IDs, query for log entries is faster than grepping the log files. Each workflow takes around 1 second to complete. This step takes around 20-30 minutes to complete.

3. Count the workflow types at 10 different reducers for workflow indexes. Since it is stored in the WorkflowBitArray, computation of the bit array is at O(1) time. Including the overhead for retrieving the data. The entire process takes 3 to 5 minutes to complete.

4. Workflow indexes generated.

Analysis based on workflow indexes can be done into two major parts:

- **Fine-grained business impact estimation:**

  Figure 7.16 shows a report generated from the workflow indexing example. It shows the shares of each workflow type. As shown in Figure 7.16, the outputs from workflow indexing enable the decision makers to determine an accurate number of a given workflow type. As the average transaction value can be estimated based on history data, the business impact can be calculated as $BV = N * V$ where BV is the business impact value, N is the number of the impacted workflows and V is the number of the estimated per transaction value.

  For instance, the CanandaLoginFlow has be identified to be defective, the total of impacted value can be calculated as 1108*56=62048 USD, where 1108 is the total related workflows and 56 is the estimated per transaction value.

  Estimate business impacts calculated helps decision makers accurately make business decisions by workflow types.

- **Decision tree based failure detection:**

  A decision tree can be built based on different workflows to fast triage the failures. For instance, workflow type BrazilLoginFlow is not enabled yet at production. According to the log, there are 10 occurrences of this workflow type in the log. The decision tree for this is straightforward as seen in Figure 7.17. If logs are seen for BrazilLoginFlow, the configuration file for BrazilLoginFlow is wrongfully configured. Another example can be seen on the CanandaLoginFlow in Figure 7.17. According to its decision tree, when error type 10225 is seen on this CanadaLoginFlow, the error is in CanadaAuthenticationService, in the other case, if error type 10227 is met on this flow, the error type is on CanadaAccountService.

The decision tree can be a community based knowledge base maintained by the triage engineers and the developers. Other than the query function described above, they are also updated constantly to make it consistent with the desired behaviors. For instance, if the transaction is launched in Brazil, the BrazilLoginFlow should be enabled and the decision tree should be updated since the business behavior has changed.

On top of the decision tree, some tasks can be automated in the system. For example, when 10227 error code is identified on production for CanadaLoginFlow, the database connections on the server has been used up and the solution is to restart the server. These types of processes can be automated when having a decision tree.

Algorithm 8 WorkflowIndexMapper

```
Input: $LogData, WorkflowType$
Output: $WorkflowBitArray$
  while GetNextWorkflow!=NULL do
    WorkflowBitArray = ReconstructWorkflow();
  end while
```

Algorithm 9 WorkflowIndexReducer

```
Input: $WorkflowBitArray$, $WorkflowBitArrayCandidate$
Output: Sum of Workflows in every WorkflowType
  if wfBitArray isTypeof WorkflowBitArrayCandidate then
    Count++;
  else
    AddNewWorkflowType(wfBitArray);
  end if
```

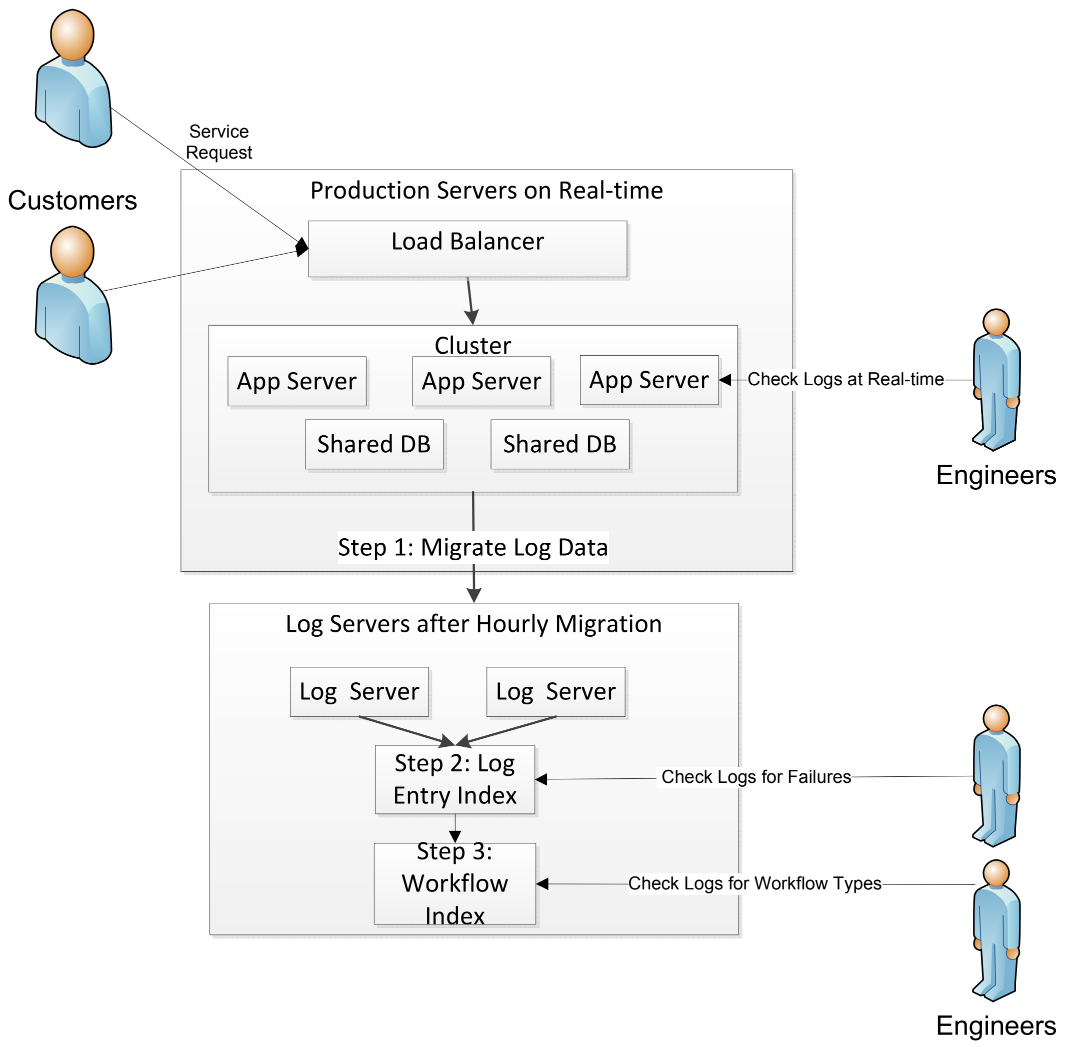


Figure 7.12: Overall Process for Log Triage

7.5 Experiment and Evaluation

7.5.1 Experiment

The experiment runs a large scale simulation with over a million BL to demonstrate the performance of the solution for retrieving the logs and completing the FICs with the transactions (details described in Table 7.2). At the end, different scenarios of the FIC distributions are discussed and workflow level exception rates are discussion. Finally, the rebuild algorithm are evaluated based on the inputs.

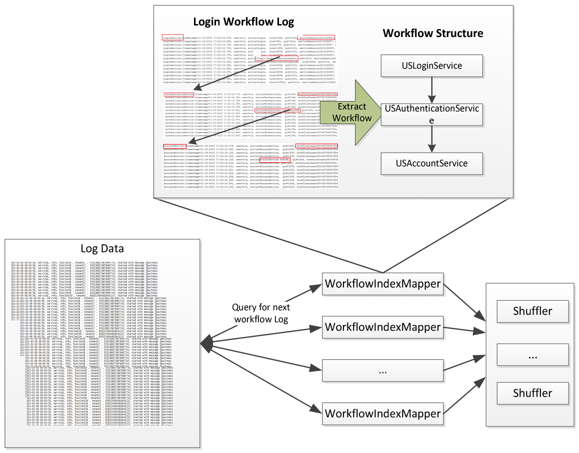


Figure 7.13: Workflow Indexing: Mapper Phase

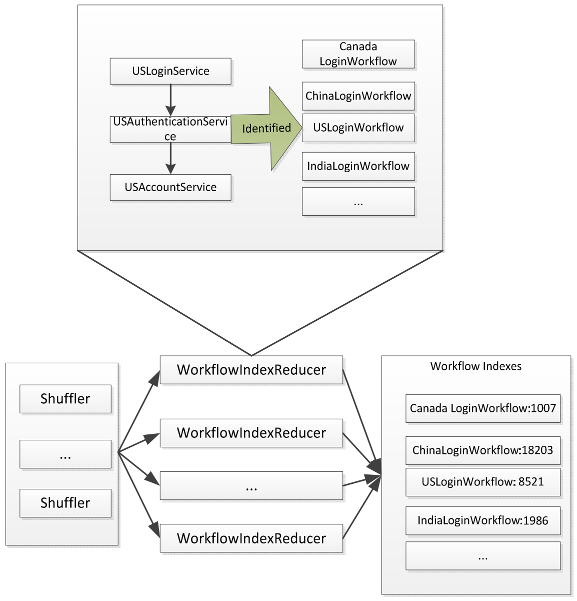


Figure 7.14: Workflow Indexing: Reducer Phase

Figure 7.4 shows an architecture view about the logs deployed on cloud. Giving the shopping transaction Figure 7.19 , with a unique transaction ID, that has several FICs in the workflow. In total,twelve services involved in the test workflow. It is unknown whether the transaction is complete or not as the interaction with the credit card authorization in workflow has been lost, and the charge service is also disconnected at the workflow. Given a transaction ID, we can search in the BL from the start

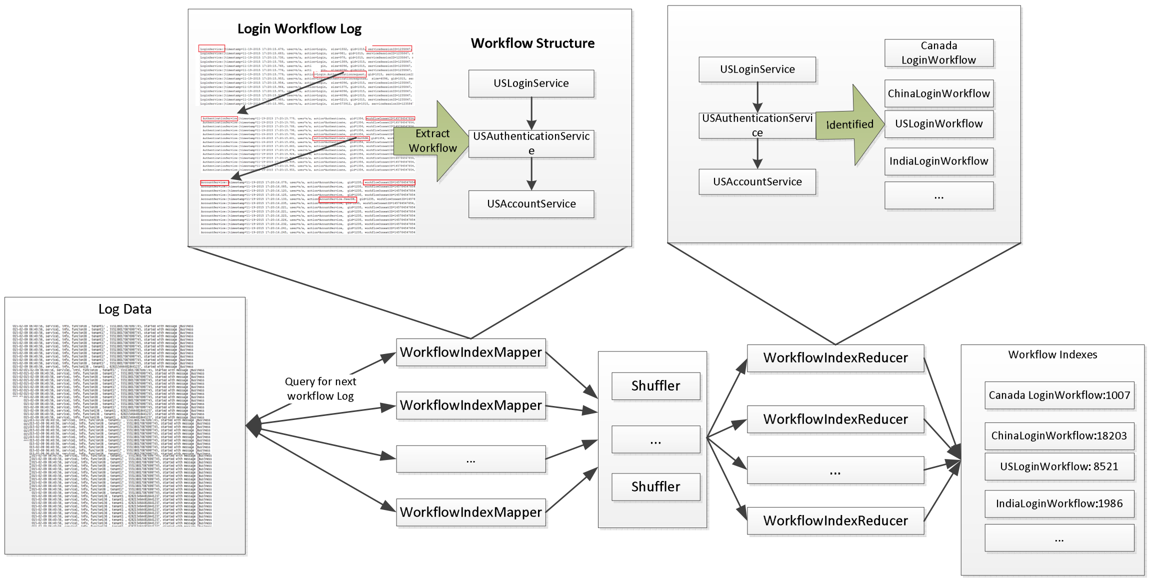


Figure 7.15: An Example of Workflow Indexing

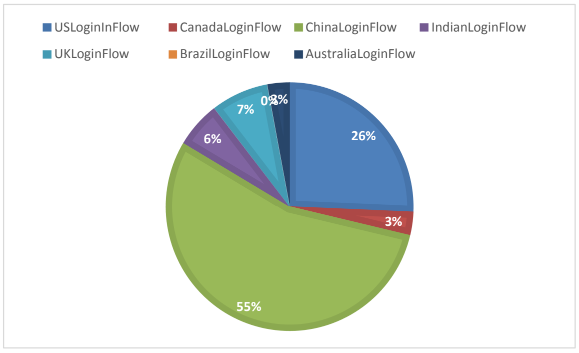


Figure 7.16: A Sample Report Generated from Workflow Indexes

service *checkout*. Via the BFS searching in the BL, we can get all the interactions before the service *compliance analysis* and build a connected IN.

As we can see in Figure 7.20, the interactions for the current transaction has been disconnected into three INs and obviously, to understand the complete picture of the transaction, we need to correlate the three INs. The first IN is what we can normally explore by the traverse algorithms in log searching, however, the other two INs are unknown to the program as the interactions to them are lost. We will need a structure rebuild for the interaction network. Before doing that, we will have an pre-generated topology of services at the data center helps to filter out the candidate

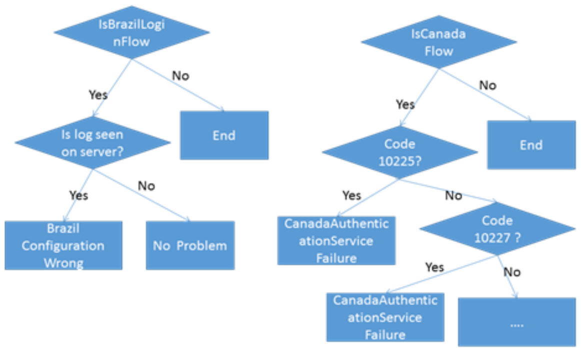


Figure 7.17: Decision Tree Example Based on Workflow Types

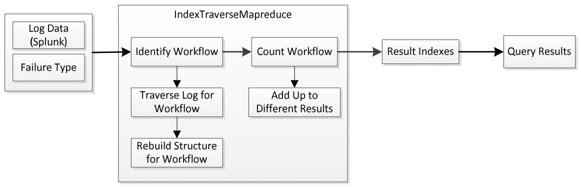


Figure 7.18: Mapreduce to Calculate Workflow Indexes

services in the structure rebuild. The most efficient method method is the empirical finding defined for structure rebuild. By this method, we can compare the previous interactions for the same service at different time as seen at the t_{i-1} time in Figure 7.20. In this example, we can easily recover the workflow by the empirical finding. In rare cases, the empirical finding can match nothing in the current disconnected INs; in that we can use other techniques such as data mining, combinational algorithm for attempting the recovery of the structure further.

7.5.2 Performance for Real-Time Log-Search

The below Figure 7.21 is based on 200GB of transaction logs as mentioned on Table 7.2. And the FICs are evenly distributed at each single services. Using the conventional triage methods, when a customer perform a search at the data center, the average search time will decrease as the search might share the same CPU/database/file systems when they hit the same services for searching. In the example, we search the log of the same type of workflow by the transaction ids, as we can see, because they are

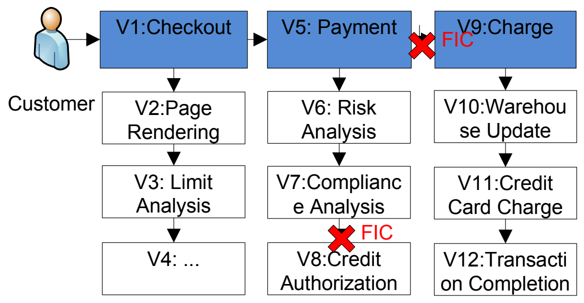


Figure 7.19: Experiment: Failed Interaction Call in a Purchase Transaction

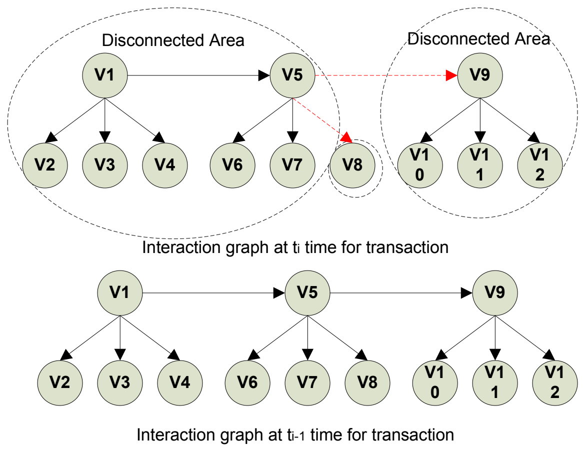
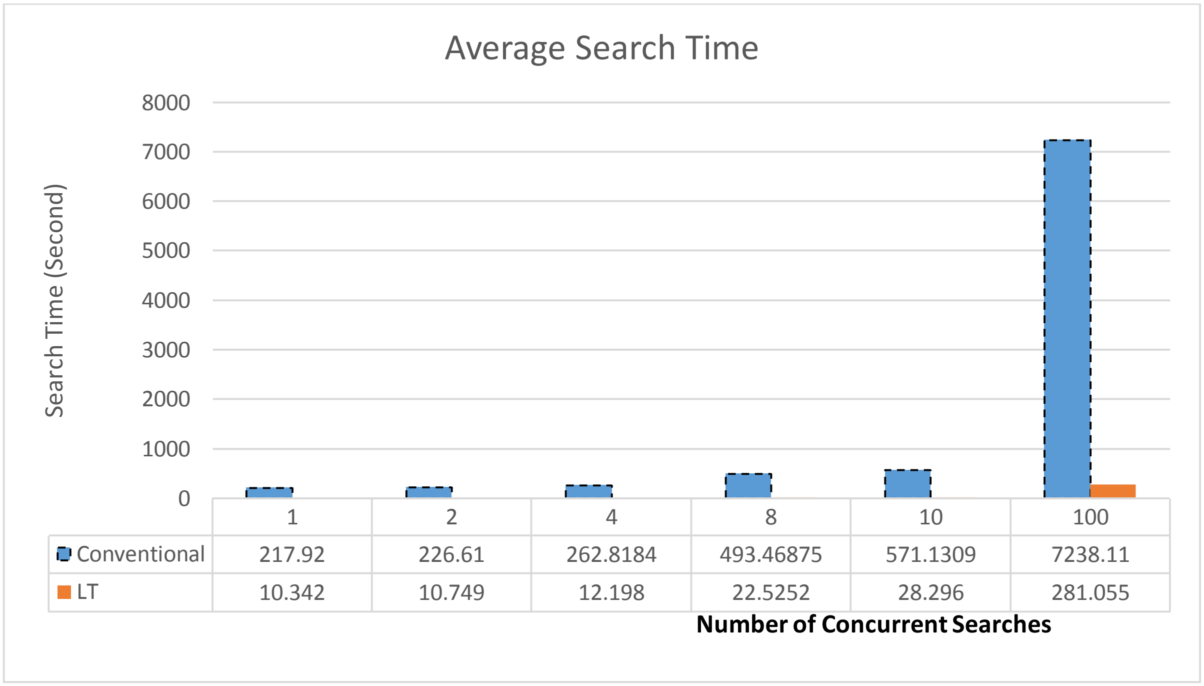

Disconnected Area
Disconnected Area
V1
V5
V9
V2
V3
V4
V6
V7
V8
V10
V11
V12
Interaction graph at ti time for transaction
Interaction graph at ti-1 time for transaction

Figure 7.20: Experiment: Empirical Completion for a FIC

competing the resource while hitting the similar services, the search response time will increase.

Use the traditional triage method search increases to 10 ways, we can see the response time will increase from around 217 seconds of one way to 493 seconds; in extreme cases, when there is 100 concurrent searches were filed, the response time can be upto 7238 seconds. It will take over two hours of response time thus it will be not acceptable if we use this method to triage the live issue. Using the LTBD and log structure in this paper, when number of search increases to 10, we can see the response time will increase from around 10 seconds to 28 seconds, in extreme cases, when there is 100 concurrent searches were filed, the response time can be upto 281 seconds. The reason for the performance improvement in the method in this paper are result from the hybrid structure of the BL and EL. As we extracted the core



	1	2	4	8	10	100
Conventional	217.92	226.61	262.8184	493.46875	571.1309	7238.11
LT	10.342	10.749	12.198	22.5252	28.296	281.055

Figure 7.21: Experiment: Test Evaluation for The Triage Method

Table 7.2: Data Size for One Hour of Simulated Transaction Logs

Total number of services	200
Log data time range	1hour
DB log entries	2millions
Total log file size	200GB
Total number of log files	200

information of the logs and stored them in DB, we can quickly traverse the call stack and hit the logs in a much faster way than traversing all related logs individually.

The second diagram in Figure 7.21 shows the average CPU usage time for both traditional and the LTBD method. Using the tradition method, the average CPU usage surge to more than 50% at four-ways of searching, and LTBD method the average CPU usage still kept at 32% at the eight-ways searching. However, in both cases, when the queries increases to more than 10-ways, the CPU usage will get worsen and the production CPU will be occupied compeletely by the log query. This is not acceptable in production as it needs to serve the transactions. Thus we can see that the best applicable of the LTBD method will be limited to less than 10-ways of concurrent searching.

7.5.3 FIC Detection Rate

At the beginning of the paper we have put the estimation of the typical large scale transaction processing system: a typical data center will have thousands of services ; in each workflow it will contain around 50 of services and we define this number as Sn; out of these services, when a FIC happen, we define the service involved to be Sf.

Failure rate can be defined as

$Frate(Total) = \sum_{n=1}^{Total} \frac{totalfailures}{totalserviceoccurences}$

Based on this, we carries out an experiment of with even distributions.

Even distribution: Suppose FIC can happen at any service at the same possibility, for a service workflow with Sn services involved, the accumulated rate of the workflow can be:

$Frate(workflow) = 1 - (1 - Frate(Total))^{Sn}$

Sporadic failure rate distribution: FIC happens to different type of services at different rate, and in most cases, most of the rate in every service is very low, however, the failure rate at one of the service is very high.

For any give S_n the failure rate can be defined as $Frate(S_n)$, under the condition of every service of having a different failure rate, the accummulated rate can be calculated as :

$Frate(workflow) = 1 - (1 - Frate(S_1)) * (1 - Frate(S_2)... * (1 - Frate(S_n))$

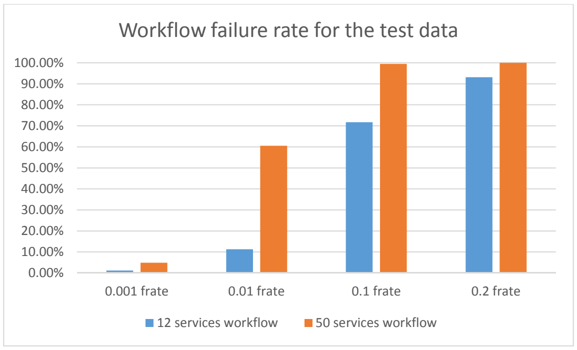


Figure 7.22: Workflow Failure Rate Based on Even Service Failure Rates

According to Figure 7.22 that is based on even distribution of failure rate, one can see that when the failure rate of the workflow failure rate will go up to 60% with the 50 average services in a workflow or 11.30% with the 12 services example used earlier. Thus there is no need to go above 10%(0.1) as the workflow will already be down.

With this set of 0.1, 0.01 and 0.001 average failure rate, we further tested in the simulated log data with the recovery of them. The rate has been proven effective. It is normal that when failure rate increases, the recovery rate will drop. The reason is that when more failure happens, the more chances of a consequence of services will result in FICs in a roll. Under this case, it is difficulty to recover the log as if more than three services in a roll has no log data, it is hard to recover a workflow of that nature.

This set of data can also be proved by conditional possibility. For all the *node* in a graph, given a failure rate of $Frate(S_node)$, the possible two and more consequence node to have failures can be calculated.

Table 7.3: FIC Recovery Rate with Even Distribution

Method used	0.1% Frate	1%Frate	10%Frate
Empirical	97.982%	95.85%	95.2%
Empirical + Page Rank	98.984%	97.95%	96.99%

7.5.4 Workflow Query Evaluation

To evaluate the performance of the workflow query in macro log analysis, we ran the real experiment on the one-hour simulation data that has 0.25 million transactions.

Transaction identification rate is defined first to estimate how many workflows of each transaction type will have.

$TIrate(workflow) = \frac{TotalNumberOfTargetWorkflows}{TotalNumberOfWorkflows}$

A hypothesis that the similar transactions to the target transaction type can hit a $TIrate$ 0.1% of the workflow, 0.01%, 0.001% and 0.0001% of all workflows. The paper assumes that the conventional manual process also designs a dedicated script for calculating the workflow by the program. Suppose each server has full capacity for query, each single query of conventional manual method takes around 10 seconds to reconstruct a workflow. The time used of the conventional method will decrease when the $TIrate$ rate decreases as shown in Figure 7.23. The data of manual process is consistent with the transaction volume as this process needs to be done sequentially at query time. However, the proposed solution to query workflow indexes, the time consumed remains the same linear time because the workflow is pre-calculated and categorized.

The time required to index workflows is not an issue. At off-production mode, numerous servers can be allocated, and as this is a divide and conquer issue, it can be solved by putting in more servers. Also, there is no hard deadline for workflow type computing, as it is more used as a subsidiary for real-time analysis. It starts automatically after log migration and is queried only when is needed.

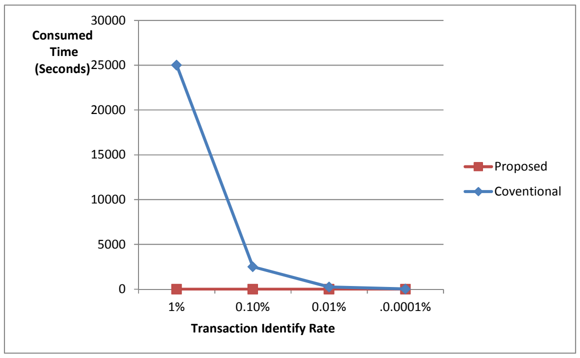


Figure 7.23: Comparison of Conventional and Proposed Workflow Query Time

7.6 Conclusion

The paper addressed the challenges for the log analysis in a cloud infrastructure.

- It presented a modeling framework for logs by proposing BL in addition to EL.
- It demonstrated that using the log model, workflow can be recontructured from failures in real-time production servers.
- It presented a workflow analysis technique to further assist the workflow recoveries.

With all these features, the complexity of triaging production log can be greatly decreased.

Chapter 8

CONCLUSION

This thesis covers the modeling, simulation and analysis on cloud computing.

SimSaaS introduced the overall infrastructure for cloud based simulation. It proposed a MTA simulation framework SimSaaS in cloud computing.

P4-SimSaaS designed the policy specification for the SimSaaS. It presented an ontology based framework and tenant related policies, to support building an flexible simulation model that can meet the variability of tenant-specific requirements in SimSaaS. The specified policies can be used to meet the different requirements of the tenants.

Model-driven development is to develop tenant applications for a GAE-based SaaS system. The modeling language used in the PSML and it has associated modeling, analysis, simulation, and code generation tools. It demonstrated several tenant applications specified using PSML and generate code for GAE, and execute the code in GAE. The proposed approach of using the model-driven approach and use a commercially available PaaS system can save significant effort and time in developing a large, scalable and fault-tolerant SaaS system.

Timing specification can be analyzed to ensure their completeness and consistency by static analysis as well as dynamic tuning by simulation. Timing information can also be obtained by simulation. As service-oriented simulation can be an integrated part of service-oriented application development, timing specification and analysis will enable simulation to obtain more accurate results. A distinct feature of the timing specification is that timing diagrams can be automatically generated once the simulation is performed, and the simulation is performed using the simulated code that is generated from a service-oriented model.

The service-oriented event-driven simulation framework allows the sharing and reusing events, composite event specifications, event generators, event handlers and event handler templates through an ontology system. Also, it offers the event specification of the service-oriented language PSML to allow static analysis.

Log Triage proposed a new framework to handle the unstructured, large volume of data mixed with untraceable failures in cloud. It offers the capability to identify the workflows from the log files at both real-time and non-realtime. It enables the engineers to have a high-level overview of the system and further efficiently and effectively triage production failures.

Further work can be done on expanding the modeling, simulation and analysis along with testing and verification on cloud computing.

REFERENCES

Akdere, M., U. Çetintemel and N. Tatbul, "Plan-based Complex Event Detection Across Distributed Sources", Proc. VLDB Endow. **1**, 1, 66–77 (2008).

Amazon, "EC2", `http://aws.amazon.com/ec2/` (2014).

Analysis, I., "Ibm smartcloud analytics - log analysis", URL `http://www-03.ibm.com/software/products/en/ibm-smartcloud-analytics---log-analysis` (2015).

ASU, "DEVS-Suite", `http://acims1.eas.asu.edu/WebStarts/` (2014).

Barham, P., R. Isaacs, R. Mortier and D. Narayanan, "Magpie: Online Modelling and Performance-aware Systems", in "HotOS", pp. 85–90 (2003).

Biacino, L. and G. Gerla, "Fuzzy Logic, Continuity and Effectiveness", Archive for Mathematical Logic **41**, 7, 643–667 (2002).

Buyya, R., "Service and Utility Oriented Distributed Computing Systems: Challenges and Opportunities for Modeling and Simulation Cmmunities", in "Proceedings of 41st Annual Simulation Symposium (ANSS2008)", pp. 3–3 (2008).

Calheiros, R. N., R. Ranjan, A. Beloglazov, C. A. F. De Rose and R. Buyya, "Cloudsim: A Toolkit for Modeling and Simulation of Cloud Computing Environments and Evaluation of Resource Provisioning Algorithms", Softw. Pract. Exper. **41**, 1, 23–50 (2011).

Chandy, K. and W. R. Schulte, *Event Processing: Designing IT Systems for Agile Companies* (McGraw-Hill Osborne Media, 2009).

Chate., S., "Convert Your Web Application to A Multi-tenant SaaS Solution", `http://www.ibm.com/developerworks/cloud/library/cl-multitenantsaas/` (2010).

Chen, Y. and W.-T. Tsai, *Service-Oriented Computing and Web Data Management* (Kendall/Hunt Publishing, 2010).

Chong, F. and G. Carraro, "Architecture Strategies for Catching the Long Tail", (2006).

Chuvakin, A., K. Schmidt and C. Phillips, *Logging and log management: The authoritative guide to understanding the concepts surrounding logging and log management* (Newnes, 2012).

Citrix, "Citrix", `http://www.citrix.com/` (2014).

Code.google.com, "Google app engine system status", URL `https://code.google.com/status/appengine` (2015).

Cons, L. and P. Poznanski, “Pan: A High-Level Configuration Language”, in “Proceedings of the 16th Conference on Systems Administration (LISA 2002)”, pp. 83–98 (2002).

Curino, C., E. P. Jones, R. A. Popa, N. Malviya, E. Wu, S. Madden, H. Balakrishnan and N. Zeldovich, “Relational cloud: A database-as-a-service for the cloud”, (2011).

Deep-software.com, “Deep log analyzer - iis and apache log analyzer — web analytics software”, URL `http://www.deep-software.com/` (2014).

Destailleur, L., “Awstats - free log file analyzer for advanced statistics (gnu gpl).”, URL `http://www.awstats.org/` (2015).

Eucalyptus, “Eucalyptus”, `http://eucalyptus.cs.ucsb.edu/` (2014).

Fonseca, R., G. Porter, R. H. Katz, S. Shenker and I. Stoica, “X-trace: A Pervasive Network Tracing Framework”, in “Proceedings of the 4th USENIX conference on Networked systems design & implementation”, pp. 20–20 (USENIX Association, 2007).

Globus, “Towards Open Grid Services Architecture”, `http://www.globus.org/ogsa/` (2007).

Godik, S., A. Anderson, B. Parducci, P. Humenn and S. Vajjhala, “Oasis eXtensible Access Control 2 Markup Language (XACML) 3”, Tech. rep., Tech. rep., OASIS (2002).

Google, “Google App Engine”, `http://code.google.com/appengine/` (2014).

Google.com, “Google analytics official website”, URL `http://www.google.com/analytics/` (2015).

Group, L., “Simulation as a Service to Business Process Management (BPM)”, `http://www.lanner.com/comms/090924/L-SIM_September.pdf/` (2014).

Hanson, J., “Javaworld.com, event-driven services in soa”, `http://www.javaworld.com/javaworld/jw-01-2005/jw-0131-soa.html?page=1/` (2005).

Haveliwala, T. H., “Topic-ensitive Pagerank: A Context-sensitive Ranking Algorithm for Web Search”, Knowledge and Data Engineering, IEEE Transactions on **15**, 4, 784–796 (2003).

IBM, “PLM: Product Lifecycle Management”, `http://www-03.ibm.com/solutions/plm/index.jsp/` (2007).

INOA, “Istf”, `http://www.iona.com/solutions/it_solutions/istf.htm/` (2014).

Lee, Y.-H., W. Li, W.-T. Tsai, Y.-S. Son and K.-D. Moon, “A Code Generation and Execution Environment for Service-oriented Smart Home Solutions”, in “Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA2009)”, pp. 1–8 (2009).

Levy, H. M., *Capability-Based Computer Systems* (Butterworth-Heinemann, Newton, MA, USA, 1984).

Li, B. H., X. Chai, Y. Di, H. Yu, Z. Du and P. Xiaoyuan, "Research on Service Oriented Simulation Grid", in "Proceedings of Autonomous Decentralized Systems (ISADS2005)", pp. 7–14 (2005).

Li, N., Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo and D. Lin, "Access Control Policy Combining: Theory Meets Practice", in "Proceedings of the 14th ACM symposium on Access control models and technologies", pp. 135–144 (ACM, 2009).

Li, W., Y.-H. Lee, W.-T. Tsai, J. Xu, Y.-S. Son, J.-H. Park and K.-D. Moon, "Service-oriented Smart Home Applications: Composition, Code Generation, Deployment, and Execution", Service Oriented Computing and Applications **6**, 1, 65–79 (2012a).

Li, W., W. Wu, W.-T. Tsai and B. Esmaeili, "Model-driven tenant development for paas-based saas", in "Proceedings of the 2012 IEEE 4th International Conference on Cloud Computing Technology and Science (CloudCom)", pp. 821–826 (IEEE Computer Society, 2012b).

Logstash.net, "logstash - open source log management", URL `http://logstash.net/` (2015).

Malik, A. W., A. Park and R. M. Fujimoto, "Optimistic Synchronization of Parallel Simulations in Cloud Computing Environments", in "Proceedings of IEEE International Conference on Cloud Computing (CLOUD'09)", pp. 49–56 (2009).

Marechaux, J.-L., "IBM Combining Service-Oriented Architecture and Event-Driven Architecture using an Enterprise Service Bus", `http://www.ibm.com/developerworks/library/ws-soa-eda-esb/` (2010).

Marty, R., "Cloud application logging for forensics", in "Proceedings of the 2011 ACM Symposium on Applied Computing", pp. 178–184 (ACM, 2011).

Mazzoleni, P., E. Bertino, B. Crispo and S. Sivasubramanian, "XACML Policy Integration Algorithms: Not to Be Confused with XACML Policy Combination Algorithms!", in "Proceedings of the eleventh ACM symposium on Access control models and technologies", pp. 219–227 (ACM, 2006).

Microsoft, "Windows Azure", `http://www.microsoft.com/windowsazure/` (2014).

Mittal, S., J. L. Risco and B. P. Zeigler, "Devs-based simulation web services for net-centric t&e", in "Proceedings of the 2007 Summer Computer Simulation Conference", SCSC '07, pp. 357–366 (2007).

Nagios.org, "Nagios - the industry standard in it infrastructure monitoring", URL `http://www.nagios.org/` (2015).

Oracle, "Architecting Event-Driven SOA: A Primer", `http://www.oracle.com/technology/pub/articles/oraclesoa_eventarch.html/` (2010).

Page, L., S. Brin, R. Motwani and T. Winograd, “The pagerank citation ranking: Bringing order to the web.”, (1999).

Palankar, M., A. Iamnitchi, M. Ripeanu and S. Garfinkel, “Amazon S3 for Science Grids: A Viable Solution?”, in “Proceedings of Data-Aware Distributed Computing Workshop (DADC)”, (2008a).

Palankar, M. R., A. Iamnitchi, M. Ripeanu and S. Garfinkel, “Amazon s3 for Science Grids: A Viable Solution?”, in “Proceedings of the 2008 international workshop on Data-aware distributed computing”, pp. 55–64 (2008b).

Paviot, J. F. T., “Implementation of a Saas Based Simulation Platform Using Open Standards and Open Source Software”, in “Proceedings of 12th NASA-ESA Workshop on Product Data Exchange (PDE2010)”, (2010).

Pietzuch, P., B. Shand and J. Bacon, “Composite Event Detection as a Generic Middleware Extension”, Network, IEEE **18**, 1, 44–55 (2004).

Project, F., “Fluentd — open source data collector”, URL `http://www.fluentd.org/` (2015).

Realtracker.com, “Realtracker”, URL `http://www.realtracker.com/` (2015).

Salesforce, “Salesforce”, `http://www.Salesforce.com/` (2014).

Sarjoughian, H., S. Kim, M. Ramaswamy and S. Yau, “A Simulation Framework for Service-oriented Computing Systems”, in “Proceedings of Simulation Conference (WSC2008)”, pp. 845–853 (2008).

Sigelman, B. H., L. A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan and C. Shanbhag, “Dapper, A Large-scale Distributed Systems Tracing Infrastructure”, Google research (2010).

Splunk.com, “Operational intelligence, log management, application management, enterprise security and compliance — splunk”, URL `http://www.splunk.com` (2015).

Status.aws.amazon.com, “Aws service health dashboard - jan 3, 2015 pst”, URL `http://status.aws.amazon.com/` (2015).

Sun, W., X. Zhang, C. J. Guo, P. Sun and H. Su, “Software as a Service: Configuration and Customization Perspectives”, in “Proceedings of IEEE Congress on Services Part II, 2008, SERVICES-2”, pp. 18–25 (2008).

Tibco, “Event-Driven SOA: A Better Way to SOA”, `http://www.tibco.com/multimedia/wp-event-driven-soa_tcm8-803.pdf/` (2010).

Tsai, W., “Service-oriented System Engineering: A New Paradigm”, in “Proceedings of IEEE International Workshop on Service-Oriented System Engineering (SOSE2005)”, pp. 3–6 (2005).

Tsai, W., X. Bai and Y. Huang, "Software-as-a-Service (SaaS): Perspectives and Challenges", Science China Information Sciences **57**, 5, 1–15 (2014).

Tsai, W., Z. Cao, X. Wei, R. Paul, Q. Huang and X. Sun, "Modeling and Simulation in Service-Oriented Software Development", Simulation **83**, 1, 7–32 (2007a).

Tsai, W., Q. Huang, X. Sun and Y. Chen, "Dynamic Collaboration Simulation in Service-Oriented Computing Paradigm", in "Proceedings of 40th Annual Simulation Symposium (ANSS07)", pp. 41–48 (2007b).

Tsai, W., Q. Huang, J. Xu, Y. Chen and R. Paul, "Ontology-based Dynamic Process Collaboration in Service-Oriented Architecture", in "Proceedings of IEEE International Conference on Service-Oriented Computing and Applications (SOCA07)", pp. 39–46 (2007c).

Tsai, W., X. Wei, Z. Cao, R. Paul, Y. Chen and J. Xu, "Process Specification and Modeling Language for Service-Oriented Software Development", in "Proceedings of 11th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS '07)", pp. 181–188 (2007d).

Tsai, W., B. Xiao, R. Paul and Y. Chen, "Consumer-centric Service-oriented Architecture: A New Approach", in "Proceedings of the Second International Workshop on Collaborative Computing, Integration, and Assurance and the Fourth IEEE Workshop on Software Technologies for Future Embedded and Ubiquitous Systems (SEUS 2006/WCCIA2006)", pp. 6 pp.– (2006a).

Tsai, W., X. Zhou and Y. Chen, "SOA Simulation and Verification by Event-Driven Policy Enforcement", in "Simulation Symposium, 2008. ANSS 2008. 41st Annual", pp. 165–172 (2008a).

Tsai, W., X. Zhou and X. Wei, "A Policy Enforcement Framework for Verification and Control of Service Collaboration", Information Systems and e-Business Management **6**, 1, 83–107 (2008b).

Tsai, W., X. Zhou and X. Wei, "A Policy Enforcement Framework for Verification and Control of Service Collaboration", Information Systems and e-Business Management **6**, 1, 83–107 (2008c).

Tsai, W.-T., Y. Chen, R. Paul, X. Zhou and C. Fan, "Simulation Verification and Validation by Dynamic Policy Specification and Enforcement", Simulation **82**, 5, 295–310 (2006b).

Tsai, W. T., C. Fan, Y. Chen and R. Paul, "DDSOS: A Dynamic Distributed Service-Oriented Simulation Framework", in "Proceedings of the 39th Annual Symposium on Simulation", ANSS '06, pp. 160–167 (2006c).

Tsai, W.-T., Y. Huang, X. Bai and J. Gao, "Scalable Architectures for SaaS", in "Proceedings of 2012 15th IEEE International Symposium on Object/Component/Service-Oriented Real-Time Distributed Computing Workshops (ISORCW2012)", pp. 112–117 (2012).

Tsai, W.-T., Y. Huang and Q. Shao, “EasySaaS: A SaaS Development Framework”, in “Proceedings of 2011 IEEE International Conference on Service-Oriented Computing and Applications (SOCA2011)”, pp. 1–4 (2011a).

Tsai, W.-T., Y. Huang, Q. Shao and X. Bai, “Data Partitioning and Redundancy Management for Robust Multi-Tenancy SaaS”, Int. J. Software and Informatics **4**, 3, 437–471 (2010a).

Tsai, W.-T., W. Li, X. Bai and J. Elston, “P4-simsaas: Policy specification for multi-tendency simulation software-as-a-service model”, in “Simulation Conference (WSC), Proceedings of the 2011 Winter”, pp. 3067–3081 (IEEE, 2011b).

Tsai, W.-T., W. Li, H. Sarjoughian and Q. Shao, “Simsaas: simulation software-as-a-service”, in “Proceedings of the 44th Annual Simulation Symposium”, pp. 77–86 (Society for Computer Simulation International, 2011c).

Tsai, W.-T., W. Li, H. Sarjoughian and Q. Shao, “SimSaaS: Simulation Software-as-a-Service”, in “Proceedings of the 44th Annual Simulation Symposium”, ANSS ’11, pp. 77–86 (2011d).

Tsai, W.-T., W. Li, X. Sun, A. Sabnis and Y. Chen, “Event-driven Service-Oriented Simulation Framework”, in “Proceedings of the 2010 Spring Simulation Multiconference”, SpringSim ’10, pp. 176:1–176:9 (2010b).

Tsai, W.-T., G. Qi and X. Bai, “AgileSaaS: An Agile SaaS Development Framework”, in “Proceedings of the Third Asia-Pacific Symposium on Internetware”, Internetware ’11 (2011e).

Tsai, W. T., H. S. Sarjoughian, W. Li and X. Sun, “Timing Specification and Analysis for Service-oriented Simulation”, in “Proceedings of the 2009 Spring Simulation Multiconference”, SpringSim ’09, pp. 51:1–51:9 (2009).

Tsai, W.-T., Q. Shao and W. Li, “OIC: Ontology-based Intelligent Customization Framework for SaaS”, in “Proceedings of 2010 IEEE International Conference on Service-Oriented Computing and Applications (SOCA2010)”, pp. 1–8 (2010c).

Tsai, W.-T., X. Sun, Q. Huang and H. D. Karatza, “An Ontology-based Collaborative Service-oriented Simulation Framework with Microsoft Robotics Studio”, Simulation Modelling Practice and Theory **16**, 9, 1392–1414 (2008d).

Turner, K. J., S. Reiff-Marganiec, L. Blair, G. A. Cambpell and F. Wang, “Appel: An Adaptable and Programmable Policy Environment and Language”, (2007).

Vaculin, R. and K. Sycara, “Specifying and Monitoring Composite Events for Semantic Web Services”, in “Proceedings of Fifth European Conference on Web Services (ECOWS07)”, pp. 87–96 (2007).

VMWare, “VMWare”, `http://www.vmware.com/` (2014).

Vu, L.-H., M. Hauswirth and K. Aberer, “QoS-based Service Selection and Ranking with Trust and Reputation Management”, in “On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE”, pp. 466–483 (Springer, 2005).

Wikipedia, “Event-Driven SOA”, `http://en.wikipedia.org/wiki/Event-driven_SOA/` (2014a).

Wikipedia, “Scalability”, `http://en.wikipedia.org/wiki/Scalability/` (2014b).

Wikipedia, “Bit array”, URL `https://en.wikipedia.org/wiki/Bit_array/` (2015).

Willard, D. E., “New Trie Data Structures Which Support Very Fast Search Operations”, J. Comput. Syst. Sci. **28**, 3, 379–394 (1984).

Wu, X., V. Kumar, J. R. Quinlan, J. Ghosh, Q. Yang, H. Motoda, G. J. McLachlan, A. Ng, B. Liu, S. Y. Philip *et al.*, “Top 10 algorithms in data mining”, Knowledge and Information Systems **14**, 1, 1–37 (2008).

XMSF, “SAIC Web-Enabled RTI”, `http://www.movesinstitute.org/xmsf/projects/WebRTI/XmsfSaicWebEnabledRtiDecember2003.pdf/` (2007).

Xpolog.com, “Xpolog log management - log analysis with log analytic search”, URL `http://www.xpolog.com/` (2015).

Zang, C. and Y. Fan, “Complex Event Processing in Enterprise Information Systems Based on RFID”, Enterp. Inf. Syst. **1**, 1, 3–23 (2007).

Zeng, J., J. Huai, H. Sun, T. Deng and X. Li, “LiveMig: An Approach to Live Instance Migration in Composite Service Evolution”, in “Proceedings of IEEE International Conference on Web Services ICWS2009”, pp. 679–686 (2009).

Zhou, X., W.-T. Tsai, X. Wei, Y. Chen and B. Xiao, “Pi4soa: A Policy Infrastructure for Verification and Control of Service Collaboration”, in “Proceedings of IEEE International Conference on e-Business Engineering (ICEBE’06)”, pp. 307–314 (2006).

Zou, D., H. Qin, H. Jin, W. Qiang, Z. Han and X. Chen, “Improving log-based fault diagnosis by log classification”, in “Network and Parallel Computing”, pp. 446–458 (Springer, 2014).