Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

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

Wei-Tek Tsai
Wu Li

School of Computing, Informatics and Decision Systems Engineering, Arizona State University
Tempe Arizona 85281 USA

Xiaoying Bai

Department of Computer Science and Technology, Tsinghua University
Beijing, CHINA

Jay Elston
School of Computing, Informatics and Decision Systems Engineering, Arizona State University
Tempe Arizona 85281 USA

ABSTRACT

Simulation can benefit from cloud computing that often comes with thousands of processors and its software is structured as Software-as-a-Service (SaaS) with its Multi-Tenancy Architecture (MTA). To support multiple tenants, simulation SaaS models need be modeled and customized to fulfill the various functional and quality requirements of individual tenants. The multitude options of tenant-specific data have made the simulation models and execution processes rather complicated. This paper presents P4-SimSaaS that comes with an new ontology system and an innovative tenant related policy specification for Simulation SaaS. P4-SimSaaS can reduce the complexity in the MTA simulation models and consequently increase the flexibility in MTA simulation execution environment. A case study is offered to demonstrate the entire framework.

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, Iamnitchi, Ripeanu, and Garfinkel 2008), Google'GAE (Google 2010), Microsoft's Azure (Microsoft 2010), Salesforce.com's Force.com (Salesforce 2010), VMware (VMWare 2010), Eucalyptus (Eucalyptus 2010), Citrix (Citrix 2010).

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 (LanerGroup 2010) designed a simulator L-SIM 2.0 to simulate the business process management systems through RESTful Web Services deployed in the cloud platform. Thomas (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, Park, and Fujimoto 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, Li, Sarjoughian, and Shao 2011) was proposed as a new approach to simulate service-oriented software in a cloud infrastructure. It can simulate a family of application using the same code base running in a PaaS (Platform as a Service) such as Google's GAE (Google App Engine), Amazon's EC2, and Microsoft's 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 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.

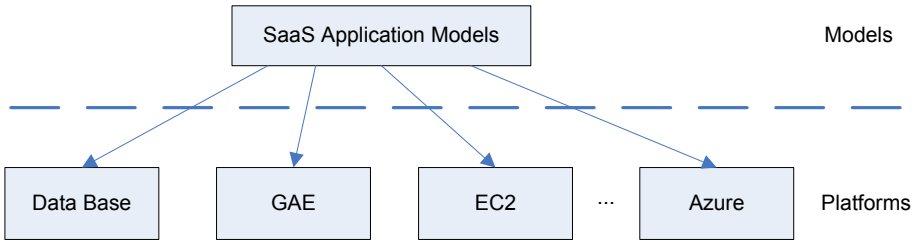

Figure 1: PaaS Independent SaaS Application Development

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 (Kagal 2002) as a formal and executable language to enforce constraints over allowable and obligated actions on resources. Pi4-SOA (Zhou, Tsai, Wei, Chen, and Xiaoying 2006) is a policy 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 2 shows an MTA simulation architecture. In the architecture, PaaS is the supporting environment for the SimSaaS Execution Envi-

ronment, which is built at SaaS layer. SimSaaS supports the MTA simulations which are built from the MTA-Based Models.

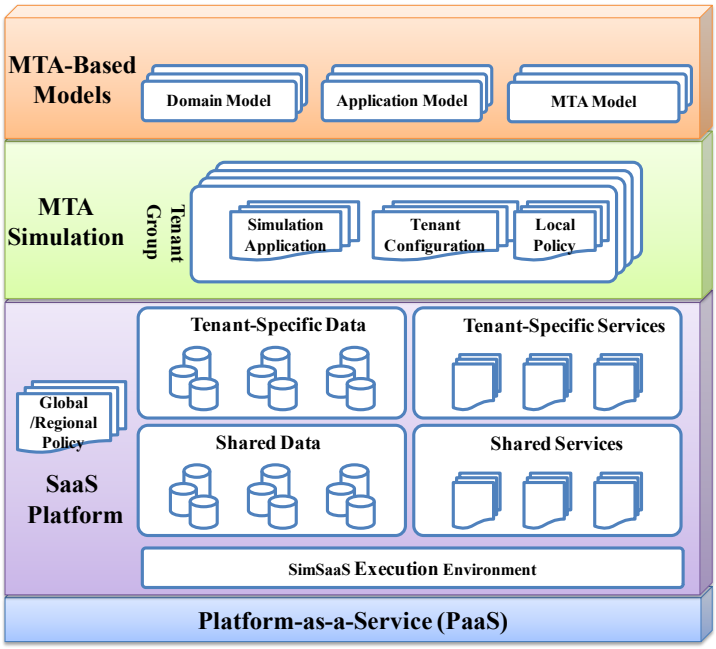

Figure 2: MTA Simulation Architecture

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.

2 ONTOLOGY SYSTEMS

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.(Tsai, Shao, and Li) Domain ontology 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. See Figure 3.

MTA Ontology: contains the related information for supporting MTA. Earlier paper (Tsai, Li, Sarjoughian, and Shao 2011) 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.

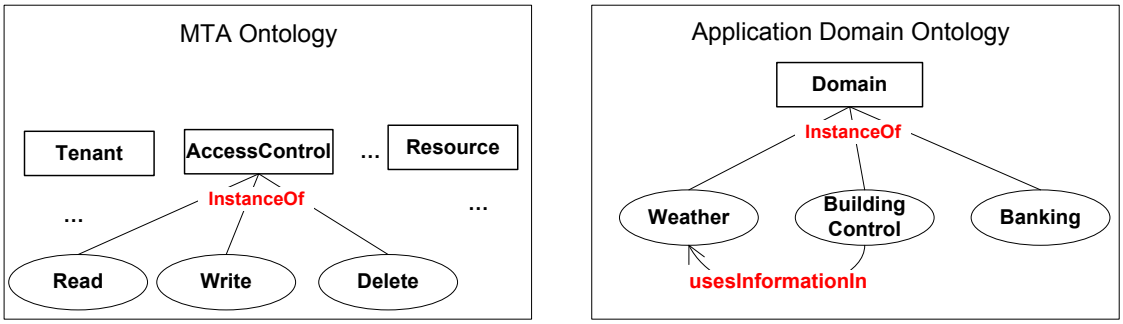

Figure 3: Sample MTA Ontology and Domain Ontology

Table 1 shows some classes that are defined in the MTA ontology, and Table 2 shows some key properties and relationships.

Table 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
AccessControl		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.

Table 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 3 shows key classes in the domain ontology, and Table 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.

Table 3: Classes in the Application Domain Ontology

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

Table 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.

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

Table 5 shows some classes in the Building Control Ontology, and Table 6 shows some key properties and relationships. Figure 4 shows a sample building control application domain ontology for a smart home.

Table 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.

Table 1 -- Properties and Relationships in the Building Control Ontology

Relationships	Relationship type	Description
hasID, has-Type	Identity	Provides identification for objects in the domain.
isLocatedIn	Location	Defines a locational relationship
ownedBy	Ownership	Defines object ownership.
providesInter-faceFor	Control	Defines what device a service provides an interface to.
affectsCondi-tionsIn	Control	Specifies that a device can affect the conditions of a room or building.
hasSpecifica-tion	Usage	Defines the protocol for using a service.
specifiesCon-trolFor	Control	Users can control buildings and rooms.
receivedRe-questFrom	Control	Defines where requests are received from
hasPolicy	Security	Defines conditions for access and permission
hasRole	People	Defines information about people.

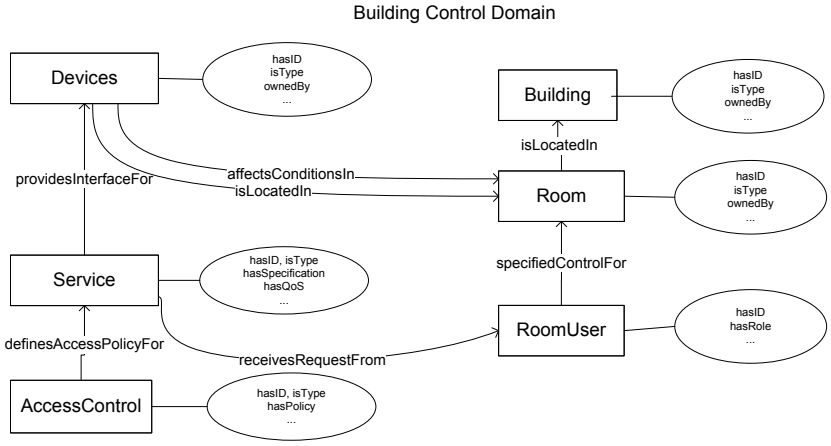

Figure 1 Sample Application Ontology in Building Control Domain

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***,

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.

Figure 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.

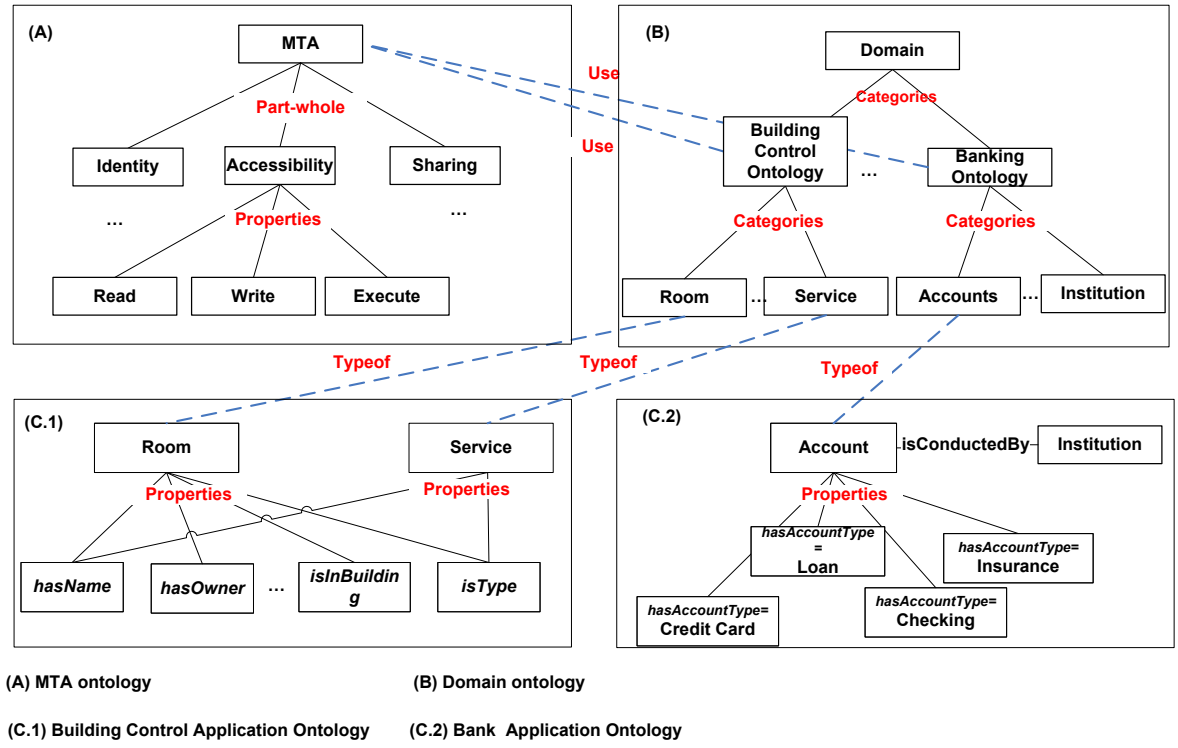

Figure 5: Cross Referencing Among Ontologies

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

3 MTA POLICY SPECIFICATION

Policies can be used to enforce various constraints at simulation time, several policy languages are available already such as Rei (Kagal 2002), XACML (Godik, Anderson, Parducci, Humenn, and Vajjhala 2002), Appel (Turner, Reiff-Marganiec, Blair, Cambpell, and Wang 2007), and PSML-P (Zhou, Tsai, Wei, Chen, Xiao 2009). 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 proposes 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 = $\{r_1, r_2, \dots, r_n\}$, a range $r_i \epsilon R$ for a policy is a tuple $r_i = < d_i, te_i, ta_{i,i} >$, where

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

Definition: Condition

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

- c_i is an crispy condition expression that the value of $c_i \epsilon \{true, false\}$;
- c_i is an fuzzy condition (Kosko 1991)that the value of $c_i \epsilon \{Fuzzy\ values\}$;
- c_i is an adaptive condition that the value of $c_i \epsilon \{adaptive\ datastructure\}$, 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; Zadeh 1965) .

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 = $\{p_1, p_2, \dots, p_n\}$, a MTA policy $p_i \epsilon P$ 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 $\forall p_i, p_i \epsilon P$,if $i \neq j$, $pid_i \neq pid_i$
- R_i is the range id for a policy, that $R_i \epsilon Range$ that defined above;
- $C_i = \{c_1^i, c_2^i, \dots c_m^i\}$is the set of conditions for p_i, that $C_i \epsilon Condition$ 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 = $\{p_1, p_2, \dots, p_n\}$, a global policy $pg_i \epsilon P$ where $te_i == null$. A global policy applies to all tenants.

Definition: Regional Policy

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

Definition: Local Policy

Let P be a finite set of policies, P = $\{p_1, p_2, \dots, p_n\}$, a local policy $pl_i \epsilon P$ 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 = $\{pset_1, pset_2, \dots, pset_n\}$, a policy set $pset_i \epsilon Pset$ where each Pset = $\{p_1, p_2, \dots, p_n\}$, where $p_1, p_2, \dots, p_n$ 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 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, Xiao, Paul, and Chen 2006) 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.

The modeling and simulation process follows a service-oriented approach as characterized in the following cyclical four-step process, as shown in Figure 6:

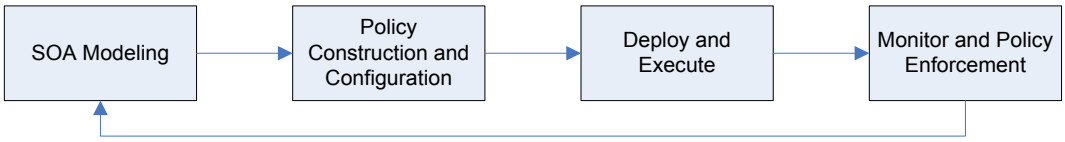

Figure 2 Four-Step Cyclical SimSaaS Modeling and Simulation Process

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.
   a. 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.
   b. 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;
   c. 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, Hauswirth, and Aberer 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.

3) **Deploy and execute**: The composed application can be deployed in a cloud environment to be executed; Similar to DDSOS (Tsai, Fan, Chen, and Paul 2006), 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 Algorithm 1, 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, Barroso, Burrows, Stephenson, Plakal, Beaver, Jaspan, and Shanbhag 2010), Magpie (Barham, Isaacs, Mortier, and Narayanan 2003) from Microsoft, and X-trace (Fonseca, Porter, Katz, Shenker, and Stoica 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.

Algorithm 1: Monitoring and Policy Enforcement

```
Input  : Ranked Policy Set, Simulation Model
Result : Simulation Result
StartMonitoringSimulation();
while more simulations do
    Log();
    Type type = SelectPolicyExecutionAlgorithm();
    if type == FYI then
        while hasPolicy() do
            executePolicy();
            executeSimulation();
        end
        Log();
    end
    else if type == conservative then
        while !checkPolicy();
        do
            Log();
        end
        executePolicy();
        executeSimulation();
        Log();
    end
    else if type == greedy then
        Rollbackpoint roolbackPoint = setRollBackPoint();
        executePolicy();
        executeSimulation();
        Log();
        if !checkPolicy() AND PolicyNoneViolable() then
            rollback(roolbackPoint);
        end
        Log();
        ReRankPolicies();
    end
end
```

Policy enforcement can be categorized into three ways (Tsai, Chen, Paul, Zhou, and Fan 2006) 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 suitable for safety –critical 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 system's 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 tenant's 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, Wang, Qardaji, Bertino, Rao, Lobo, and Lin 2009; Mazzoleni, Bertino, Crispo, and Sivasubramanian 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.

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 Table 7.

Table 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.

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 8. It specified the related services and required involved data for the two different simulations.

Table 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

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.

XYZHome⟶ Home Control Simulation
ABCDorm⟶ 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. Note that as a global policy p5 applies to all the simulation runs at the SimSaaS runtime 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. (Table 9)

Table 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

5.3 Deployment and Execution

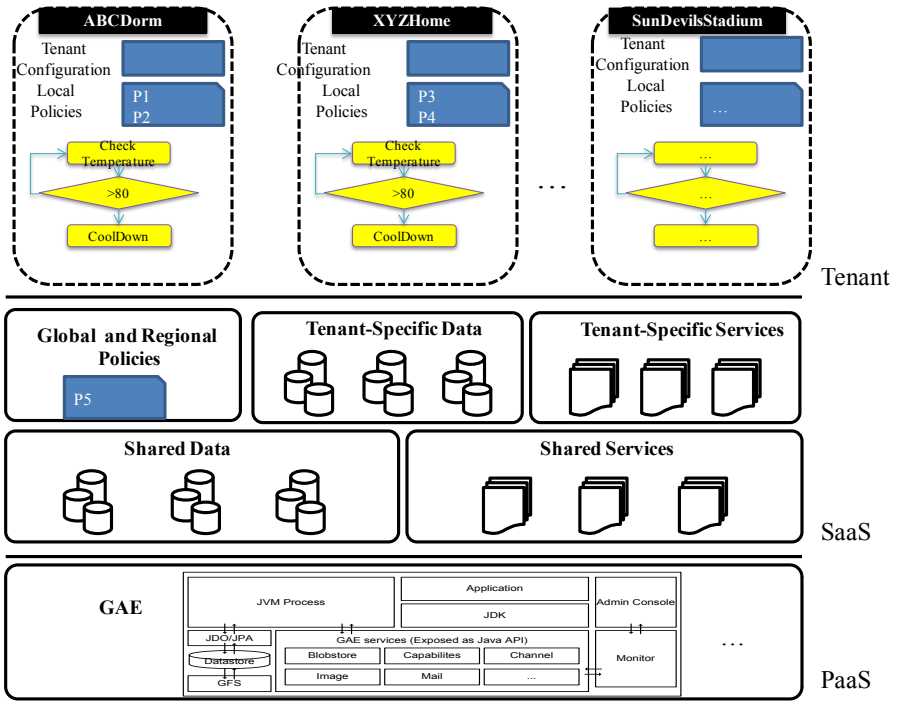

Figure 7: Deployment OverView of Global, Regional and Local Policies

Figure 7 above 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.

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.

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.

7 ACKNOWLEDGMENT

The project is sponsored by U.S. National Science Foundation project DUE 0942453, and the European Regional Development Fund and the Government of Romania under the grant no. 181 of 18.06.2010.

REFERENCES

Barham, P., R. Isaacs, R. Mortier, and D. Narayanan. 2003. "Magpie: Online modelling and performance-aware systems." Pp. 15-15: USENIX Association.

Biacino, L. and G. Gerla. 2002. "Fuzzy logic, continuity and effectiveness." *Archive for Mathematical Logic* 41:643-667.

Citrix. 2010. vol. 2010.

Eucalyptus. 2010. vol. 2010.

Fonseca, R., G. Porter, R.H. Katz, S. Shenker, and I. Stoica. 2007. "X-trace: A pervasive network tracing framework." Pp. 20-20: USENIX Association.

Godik, S., A. Anderson, B. Parducci, P. Humenn, and S. Vajjhala. 2002. "OASIS eXtensible access control 2 markup language (XACML) 3." Tech. rep., OASIS.

Google. 2010. "Google App Engine." vol. 2010.

Haveliwala, T.H. 2003. "Topic-sensitive pagerank: A context-sensitive ranking algorithm for web search." *IEEE Transactions on Knowledge and Data Engineering*:784-796.

Kagal, L. 2002. "Rei: A policy language for the me-centric project." *HP Labs, accessible online http://www. hpl. hp. com/techreports/2002/HPL-2002-270. html*.

Kosko, B. 1991. *Neural networks and fuzzy systems: a dynamical systems approach to machine intelli-*

gence: Prentice-Hall, Inc.
LanerGroup. 2010. "Simulation as a service to business process management (BPM)."
Li, N., Q. Wang, W. Qardaji, E. Bertino, P. Rao, J. Lobo, and D. Lin. 2009. "Access control policy combining: theory meets practice." Pp. 135-144: ACM.
Malik, A., A. Park, and R. Fujimoto. 2009. "Optimistic Synchronization of Parallel Simulations in Cloud Computing Environments." Pp. 49-56: IEEE.
Mazzoleni, P., E. Bertino, B. Crispo, and S. Sivasubramanian. 2006. "XACML policy integration algorithms: not to be confused with XACML policy combination algorithms!" Pp. 219-227: ACM.
Microsoft. 2010. "Azure." vol. 2010.
Palankar, MR, A Iamnitchi, M Ripeanu, and S Garfinkel. 2008. "Amazon S3 for science grids: a viable solution?" Pp. 55-64: ACM.
Salesforce. 2010. "Salesforce." vol. 2010.
Sigelman, B.H., L.A. Barroso, M. Burrows, P. Stephenson, M. Plakal, D. Beaver, S. Jaspan, and C. Shanbhag. 2010. "Dapper, a large-scale distributed systems tracing infrastructure." Technical report dapper-2010-1. Google.
Thomas Paviot and Jelle Feringa. 2010. "Implementation of a SaaS Based Simulation Platform Using Open Standards and Open Source Software." in *12th NASA-ESA Workshop on Product Data Exchange (PDE2010)*.
Tsai, W.T., Q. Shao, and W. Li. "OIC: Ontology-based intelligent customization framework for SaaS." Pp. 1-8: IEEE.
Tsai, W.T., B Xiao, RA Paul, and Y Chen. 2006. "Consumer-centric service-oriented architecture: a new approach." Pp. 6: IEEE.
Tsai, Wei-Tek, Wu Li, Hessam Sarjoughian, and Qihong Shao. 2011. "SimSaaS: Simulation Software as a Service." in *the 44st Annual Simulation Symposium (ANSS)*. Boston.
Tsai, Wei Tek, Chun Fan, Yinong Chen, and Ray Paul. 2006. "DDSOS: A Dynamic Distributed Service-Oriented Simulation Framework." Pp. 160-167: IEEE Computer Society.
Tsai, WT, Y. Chen, R. Paul, X. Zhou and C. Fan. 2006. "Simulation verification and validation by dynamic policy specification and enforcement." *Simulation* 82:295.
Turner, K.J., S. Reiff-Marganiec, L. Blair, G.A. Cambpell and F. Wang. 2007. "APPEL: An Adaptable and Programmable Policy Environment and Language."
VMWare. 2010. vol. 2010.
Vu, LH, M Hauswirth and K Aberer. 2005. "QoS-based service selection and ranking with trust and reputation management." *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*:466-483.
Zadeh, L.A. 1965. "Fuzzy sets*." *Information and control* 8:338-353.
Zhou, X., W.-T. Tsai, X. Wei, Y. Chen and B. Xiaoying. 2006. "Pi4soa: A policy infrastructure for verification and control of service collaboration."