# A Model-based Simulation Approach to Error Analysis of IT Services

Long Wang
Center for Reliable High-Performance Comp.
University of Illinois at Urbana-Champaign,
Urbana, IL 61801
longwang@crhc.uiuc.edu

Akhil Sahai, James Pruyne
Hewlett Packard Laboratories
Palo Alto, CA 94304
{akhil.sahai, jim_pruyne}@hp.com

*Abstract*—**Utility computing environments provide on-demand IT services to customers. Such environments are dynamic in nature and continuously adapt to changes in requirements and system state. Errors are an important category of environment state changes as such environments consist of a large number of components, and hence, are subject to errors. In this paper, we design and implement a model-based simulation framework that leverages information about existing service components and their interactions, and provides concrete service behavior in presence of a variety of errors. To evaluate the framework, experiments are conducted on a virtualized blade-server based environment. Results show that the framework is effective and practical in analyzing error impacts on IT services.**

***Keywords- utility computing; model-based; simulation; IT service; error; error analysis; error behavior; error model***

## I. INTRODUCTION

Utility computing environments in form of shared IT services are becoming more prevalent. These environments comprise of large number of components which are inter-dependent on one another and are subject to complex interactions. The environments are prone to unexpected error behavior when underlying components fail, which imposes a major threat in large-scale environments and non-stop services. Therefore, error analysis is necessary for these environments.

An information model-based simulation approach, rather than a static analysis, is applied for the error analysis in the paper for three reasons. First, these environments are dynamic in nature and continuously experience and/or adapt to changes in requirements and system state. For example, components like software processes are continuously created/ destroyed in the runtime. The dynamic aspects of the environments are not captured in static models or analysis (e.g. reachability analysis). Second, a static analysis, as an analytical solution, is hard to use/solve due to state explosion, concerning the large number of components, attributes, and their changes. Finally, our experience of error injection into a variety of real-world systems/applications shows that error behavior is fairly diverse and sometimes unexpected [8] [9]. For example, a flip of the same bit in different runs of an application may cause much different failure behavior because the execution context changes. So a concrete and detailed solution to error analysis (or more general, answering decision support questions) for utility computing environments needs a simulation approach.

Formal information model defined in Common Information Model (CIM) or Unified Modeling Language (UML) is widely used in designing utility computing environments for flexible and general support for various IT services. Our approach leverages information about existing service components and their interactions stored in the information models to achieve exact recreation of the environments. The approach also captures use-case behavior of IT services and introduces various error models for service components. Errors are represented in the same form of the formal information model used for environment design. These are fed into a simulation engine to study error impacts. Moreover, additional components or interactions can be introduced into the environments during simulation, which enables system designers to understand impacts of proposed changes within the context of the existing service, and allows a finer-granularity of analysis on possible system errors.

The information model-based simulation approach can be applied to study of environment changes other than errors, e.g. hardware upgrades, software updates, addition of new features, etc. The approach is a generic methodology and can be seamlessly merged into existing design process of utility computing environments to offer general decision support for designing IT services.

The contributions of the paper are briefly summarized as below: the paper

- proposes a model-based simulation framework to address decision support questions for IT services;
- designs error model representation for a typical utility computing environment (a virtualized blade-server based environment) with no modification to existing information models of the environment;
- implements a prototype of the framework completely and evaluates the effectiveness of the framework for error analysis of the target environment through experiments.

## II. ERROR MODEL REPRESENTATION

A utility computing environment is typically configured and managed through formal information models (in form of CIM or UML). These models accurately specify (a) the composition of the environment, including classes of entities in the environment, instances of the entity classes, and interactions between the instances; and (b) attributes of these entity classes and instances. These models facilitate building of IT services and environment in design phase, as well as management and dynamic reconfiguration in maintenance phase as requirements and/or policies change. As error-present behavior of the



1-4244-0799-0/07/$25.00 ©2007 IEEE

environment is not captured in current models, representation of error models should be designed for error analysis.

**Target environment.** Our target environment is a commercial utility computing environment for a virtualized desktop service. The environment consists of multiple physical machines (*PMs*), which are blade servers located in enclosures which in turn are hosted in racks. Each PM hosts multiple virtual machines (*VMs*). Customers log on and log off allocated VMs using a remote desktop protocol (RDP) in the virtualized desktop service (via a process called *RDP Client*). A *connection manager* performs allocation/reclamation of VMs with the aid of an agent process on each VM (*VM Agent*). Figure 1 depicts sample part of the class/instance model in CIM. The class of *CIM ManagedElement* is provided by the CIM model skeleton. There are two PM instances, three VM instances (VM1, VM2 on PM2, and VM3 on PM1), and a single instance of connection manager. Interactions between instances (e.g. the *"reside in"* interaction) are represented as CIM associations.

Components in the target environment are subject to a variety of hardware errors (persistent defects, transient fail-stops, performance degradation, and partial failures) as well as software errors (persistent program bugs, transient fail-stops, performance degradation, race conditions, and configuration/ administration errors). In-depth analysis of these errors is not given here due to space limit, and can be found in the technical report [11].

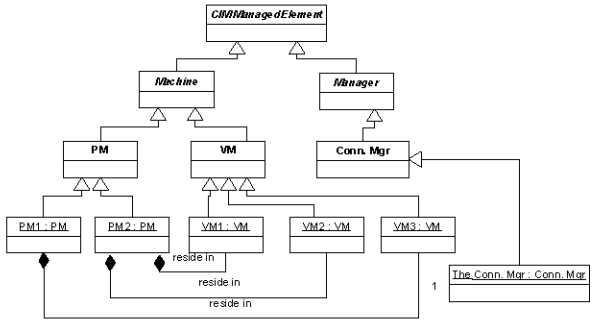


Figure 1: Sample part of information model for the target environment

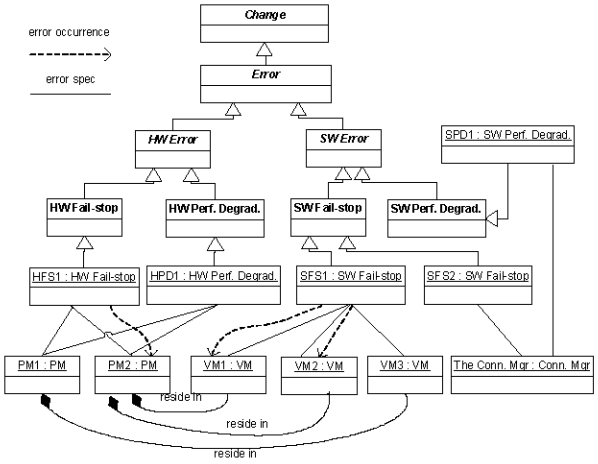


Figure 2: Error model representation for the target environment

**Error model representation.** The error model representation is designed as information model, as illustrated in Figure 2. The *error* is a subclass of the general *change* class. The *error* class has two subclasses, *hardware error* and *software error*, and each of them is partitioned into categories representing different types of error models (only part of error categories are shown in Figure 2).

Each error class has multiple instances and each error instance captures specific error specification and error behavior of a class of environment components. For example, the error instance *HFS1* in Figure 2 contains (as attributes) the MTTF (mean-time-to-failure) for fail-stop errors of physical machine, as well as behavior of the failed physical machine when the machine suffers from a fail-stop error (the behavior is specified in terms of changes to related entity/error instances). Such a relation between *HFS1* and the relevant entity instances (i.e. physical machines) is represented as an interaction of *error specification* (illustrated as a solid line in Figure 2). Another interaction between error instance and entity instance, *error occurrence,* is established when an error of the specific error model happens to the entity instance (illustrated as a dash arrow in Figure 2). The dash arrow from *HFS1* to *PM2* means that a hardware fail-stop error occurs to the *PM2* physical machine.

**Capturing instance-specific behavior.** In the error model representation an error instance captures error characteristics and behavior for all instances of an entity class. It is convenient and practical as entity instances may be dynamically created in the runtime, and instances of the same entity class usually have similar error characteristics/behavior. A rule-based mechanism is introduced to capture instance-specific error behavior. Each entity instance has key attribute values, or *ID*. Error specification and behavior are defined in an error instance using these IDs to designate individual components. Here is an example: upon a power outage *PM1* crashes while *PM2* does not crash because *PM2* has a battery. Then the error behavior defined in *HFS1* may look like:

*if PM.ID = "PM1" then crash;*

*if PM.ID = "PM2" then use_battery;*

Besides IDs, other attribute values of entity instances may be applied for instance recognition. For the example above the behavior can be defined as:

*for PMs with battery, use_battery;*

*for PM2 without battery, crash;*

Such rule-based definitions are also used for specifying complicated use case behavior in our simulation framework.

**Error propagation.** Errors propagate across the environment through interactions between components, and the interactions through which an error propagates are registered in the corresponding error instance. For example, when a fail-stop error occurs to *PM2* (i.e. an *error occurrence* interaction is established between *HFS1* and *PM2* in Figure 2), the simulation engine finds that the *"reside in"* interaction is registered for error propagation in *HFS1*, and *VM1* and *VM2* are residing in *PM2* through the *"reside in"* interactions. As a result, *error occurrence* interactions are established between *SFS1* and *VM1/VM2* according to the propagation behavior specified in *HFS1*. Such a procedure continues until there is no interaction found for victim components to propagate the error.

In summary, the error model representation:

- captures various types of errors in category and hierarchy as classes and instances in information model, without any modification to existing model components;
- captures instance-specific error behavior as well as error propagation.

## III. MODEL-BASED SIMULATION FRAMEWORK

Our model-based simulation framework employs event-driven simulation to mimic environment behavior under errors. Figure 3 illustrates the architecture of the framework. A simulation engine reads in input information from outside, performs simulation experiments, and generates results for analysis. There are four kinds of input information: (a) information models of entity classes/instances and error representation stored in model repositories (e.g. CIM model database); (b) definition of additional entity classes and instances which are not present in existing models; (c) behavior of entity instances in target use-cases; and (d) experiment setup and parameters (e.g. parameters of stochastic distributions, synthetic workload, and number of machines).

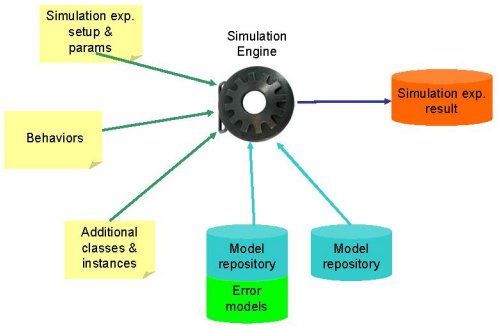


Figure 3: Architecture of the model-based simulation framework

Each entity instance in the target environment is represented as a *component* in simulation. Information about entity classes and instances is leveraged as *component attributes* and *component interactions*. Furthermore, a concept of *state* is introduced to all components for event-driven simulation.

An *event* is a message with a target component and an event ID. An event may have parameters with it. Each event is scheduled according to a predefined stochastic distribution to trigger actions of the target component at a particular occasion. The stochastic distribution characterizes the semantic of the event, e.g. exponential distribution is used to characterize arrival of customer requests to log on a virtualized desktop.

Component behavior is specified as a state machine defined by an *action matrix* in the entity class of the component (see Figure 4). Upon delivering an event to the target component in a specific state, the simulation engine locates the corresponding item in the action matrix, and triggers specified *actions* accordingly (an action can be state transition, attribute update, or interaction setup/teardown). Update of component attributes during simulation mimics real-world execution. Simulation of rule-based behavior is performed by checking a condition (a first-order predicate) before triggering actions. So *cond1* must be satisfied for *act11, act12 ...* to be triggered in Figure 4.

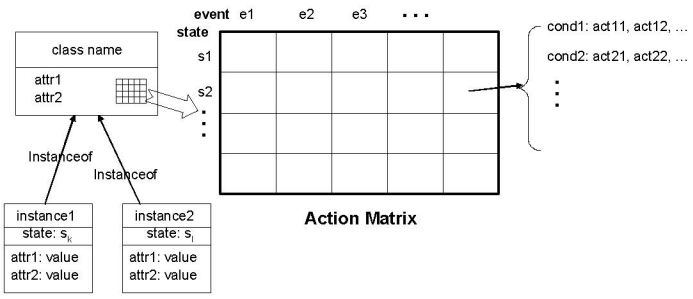


Figure 4: Structure of the action matrix in an entity class

Performance measurement and result analysis are conducted via a) statistical calculation for state occupation and event delivery, and b) analysis of simulation event traces by scripts.

Due to space limit, details of the framework are not given here. The features of the framework are summarized below:

1. By employing real-world data from information models present in a real computing environment, the framework provides more accurate results for the target environment compared to a best-effort recreation of the environment, or a different hypothetical environment.

2. The designed error model representation allows for capturing sophisticated error models (error characteristics, behavior and propagation).

3. Definition of additional classes/instances enables evaluation of IT services for projected changes/components, or under scenarios incapable or inefficient to be tested on existing testbeds or real workloads (e.g. scalability analysis, sensitivity study, various error models, and boundary/extreme cases).

4. Actions of entity instances can be specified in form of *action statements* which are compiled on the fly before executed in simulation. This provides the flexibility to capture different behaviors of a component when it receives the same event in the same state (especially useful for simulating sophisticated error behavior or changing behavior of components).

## IV. EXPERIMENTATION

The prototype of the model-based simulation framework is completely implemented using JDK 1.5.0 on a real utility computing environment (designed by using CIM). The model repositories of the environment are stored in MySQL 4.1. A simulation tool package, DESMO-J 2.1.1 [10], is employed for basic event scheduling and a library of stochastic distributions.

Two uses cases are targeted for experimentation: user logon and user logoff. Experiments are conducted with fail-stop errors for single-error scenarios, multi-error scenarios, and error-free scalability study. Due to space limit, only the result for single-error scenarios is presented below. (See [11] for more evaluation work and details of the two use cases.)

The goal of the single-error experiments is to answer the question "what happens after a fail-stop error occurs to a component". The experiment setup is summarized below, and experiment outcomes are presented in Table 1.

- There are 5 users.
- A user requests a VM and logs on it at an exponentially distributed interval (the mean is 4 hrs).
- A user logs off a VM after operating it for an exponentially distributed period (the mean is 2 hrs).
- There are 3 PMs. One PM has 7 VMs while the other two have no VMs. (The information is directly obtained from the model repository of the real environment.)
- The simulated period in an experiment is 10 days.
- During an experiment a single fail-stop error is injected into a randomly picked component (PM, VM, RDP client, connection manager, or VM agent) with an MTTF of 2 days. Error is injected by establishing an *error occurrence* interaction between the error instance and the component. Then the corresponding error behavior is triggered and the error is propagated if possible.

100 experiments were conducted and every experiment takes about 8ms after initialization. The sequential mode for simulation is applied in the experiments. As DESMO-J provides both sequential and parallel modes, a better simulation performance (and simulation scalability) can be achieved by applying the parallel mode. So simulation performance is not a big concern for a real-world environment.

Table 1 shows that, not all the experiments have errors injected (e.g. 98 out of 100 experiments are error-injected for physical machines), because there are cases when error injection is scheduled after the simulated period (10 days).

The question "what happens after the error" is answered in the "error behavior" column in Table 1. For example, the simulation results show that (see the *"physical machine"* row in Table 1): in 39 out of 98 experiments with a failed PM, the PM hosting 7 VMs fails and the error propagates to the VMs and VM agents on the PM; two scenarios happen after the error (an experiment may have both occur because multiple VMs fail in one experiment), listed as (a) and (b) in the table row. In both scenarios the user receives no response from the VM and the VM appears hung.

## V. Related Work

Utility computing and Grid computing make IT service popular and techniques on reliability of IT services grow prosperous. Dai et al. [1] simplify real-world Grid services into virtual-tree structures and develop a theoretical analysis of Grid service reliability. Candea et al. [2] propose a methodology to reboot only the components that need to be rebooted for recovering from a failure in computing environments (only failures that can be recovered from reboots are considered).

Some related work exists in error model classification and representation. A thorough and complete classification of error models is given in [3]. A classical approach, fault tree analysis [4], is widely accepted for failure analysis of complicated system. The fault tree analysis targets abstract composition/ specification of system, whereas our error model representation enables studying concrete error-present service behavior through information models in real-world environments.

Dependability modeling and simulation is another area with lots of research work. Stochastic models, e.g. Markov chain, Petri Net, SAN, are widely used. Athanasopoulou et al [5] model a satellite network for dependability evaluation, and Wang et al [6] design a stochastic model for scalability study of deploying coordinated checkpointing protocols in large-scale systems. These models are either best-effort recreation of computing environments or hypothetical ones, and cannot expose concrete behavior of system.

There is also work done for evaluating reliability/ availability of real systems/environments. Fault injection is popularly used for this purpose. NFTAPE [7] is a dedicated software toolset for injecting errors into real systems and assessing system dependability. Error behavior of Ensemble, a reliable group communication library, and error behavior of Linux kernel are studied by deploying NFTAPE toolset in [8] and [12], respectively.

## VI. Conclusion

This paper discusses design and implementation of a model-based simulation framework for analyzing behavior of IT services in utility computing environments under different error models. The framework combines error models with component models existing in real-world utility computing environments, and provides a practical way to accurately evaluate error behavior of these environments. Moreover, projected system designs/changes and use-case scenarios can also be tested on this framework. Experiment results demonstrate effectiveness of the framework for error analysis of IT services in utility computing environments.

Table 1: Experiment results for single-error injections *

| Injected entity | Injections | | Error Behavior |
|---|---|---|---|
| Physical machine | 98 | 59 | The PM fails (the PM does not host VMs). |
| | | 39 | The PM fails -> the 7 VMs fail -> the 7 VM agents fail (the PM hosts VMs).<br>(a) A user requests a VM, a failed VM is allocated, but when the user connects the VM by an RDP client, there is no response;<br>(b) A user was working on a failed VM, and does not receive any response from the VM. |
| Virtual machine | 100 | 73 | The VM fails -> the VM agent fails (the VM is not logged on).<br>Same as (a) above. |
| | | 27 | The VM fails -> the VM agent fails (the VM is logged on).<br>Same as (b) above. |
| RDP client | 92 | 92 | The RDP client fails.<br>The connected VM will not be logged off and is unavailable for future allocation. |
| Connection mgr | 99 | 99 | The connection manager fails.<br>(a) A user requests a VM but receives no response from the connection manager;<br>(b) A user logs off a VM successfully, but the VM is not reclaimed, and the config DB stays inconsistent. |
| VM agent | 100 | 63 | The VM agent fails (the associated VM is not logged on).<br>A user requests a VM and the associated VM is allocated. The user logs on it and logs off it successfully. But the user logoff is not detected and the VM is not reclaimed for future use. |
| | | 37 | The VM agent fails (the associated VM is logged on).<br>The user logs off the associated VM successfully, but the user logoff is not detected and the VM is not reclaimed for future use. |

* 100 experiments were conducted for each row in the table.

## References


[1] Yuan-Shun Dai, et al, "Reliability and performance of tree-structured grid services", IEEE Transactions on Reliability, Vol 55, Is. 2, June 2006.

[2] George Candea, et al, "Improving Availability with Recursive Microreboots" A Soft-State System Case Study", Performance Evaluation Journal, vol. 56, nos. 1-3, March 2004.

[3] Avizienis, A., et al, "Basic concepts and taxonomy of dependable and secure computing", IEEE Transactions on Dependable and Secure Computing, Vol 1, Issue 1, 2004.

[4] Richard E. Barlow, "Reliability and Fault Tree Analysis", published by Society for Industrial and Applied Mathematic, 2nd Ed., 1982.

[5] E. Athanasopoulou, et al, "Evaluating the Dependability of a LEO Satellite Network for Scientific Applications", Proceedings of the 2nd International Conference on the Quantitative Evaluation of Systems (QEST), Torino, Italy, September 19-22, 2005, pp. 95-104.

[6] Long Wang, et al., "Modeling Coordinated Checkpointing for Large-Scale Supercomputers", DSN 2005.

[7] D. Stott, et al, "Dependability assessment in distributed systems with lightweight fault injectors in NFTAPE", Proc. Of the Dependable Computing for Critical Applications Conf., 1998.

[8] Claudio Basile, et al, "Group Communication Protocols under Errors," Proc. 22nd Symposium on Reliable Distributed Systems, SRDS'03, Florence, Italy, 2003.

[9] Long Wang, et al, "An OS-level Framework for Providing Application Aware Reliability", PRDC 2006.

[10] B. Page, et al, "The Java Simulation Handbook. Simulating Discrete Event Systems with UML and Java", Shaker Publ., Aachen, Germany 2005.

[11] Long Wang, et al, "A Model-based Simulation Approach to Error Analysis of IT Services", Technical report HPL-2006-181, HP Laboratories, California, 2006.

[12] Weining Gu, et al, "Characterization of Linux Kernel Behavior under Errors", DSN'03, San Francisco, CA, June 2003.