Model-Driven Automated Error Recovery in Cloud Computing

Yu Sun[1], Jules White[2], Jeff Gray[1], Aniruddha Gokhale[2], Douglas C. Schmidt[2]

[1]Dept. of Computer and Information Sciences
University of Alabama at Birmingham
Birmingham, AL 35294, USA
{yusun, gray} @ cis.uab.edu

[2]Institute for Software Integrated Systems
Vanderbilt University
Nashville, TN 37203, USA
{jules, schmidt, gokhale} @ dre.vanderbilt.edu

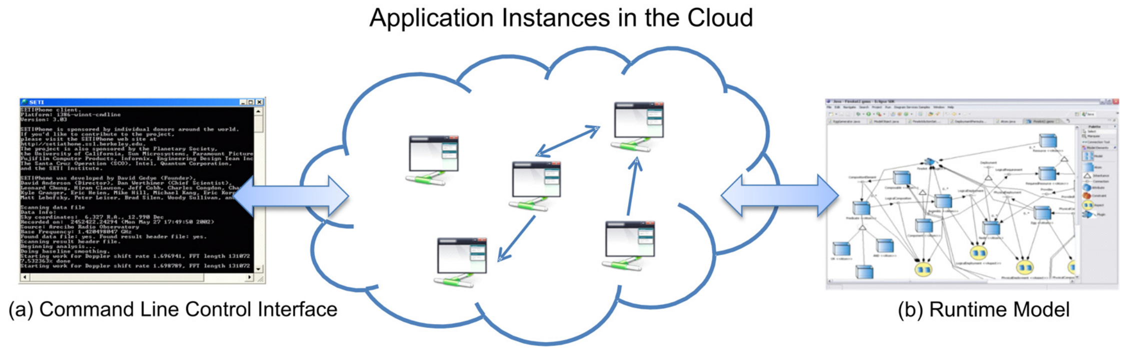
Introduction

With the increasing complexity of software and systems, domain analysis and modeling are becoming more important for software development and system applications. Applying domain-specific modeling languages and transformation engines is an effective approach to address platform complexity and the inability of third-generation languages to express domain concepts clearly [1]. Building correct models for a specific domain can often simplify many complex tasks, particularly for distributed applications based on cloud computing [2] that offer several opportunities for customization and variability.

Cloud computing shifts the computation from local, individual devices to distributed, virtual, and scalable resources, thereby enabling end-users to utilize the computation, storage, and other application resources (which forms the "cloud") on-demand [2]. Amazon EC2 (Elastic Compute Cloud) [5] is an example cloud computing platform that allows users to deploy different customized applications in the cloud. A user can create, execute, and terminate the application instances as needed, and pay for the cost of time and storage that the active instances use based on a utility cost model [3].

Open problems. In the cloud computing paradigm, the large number of running nodes increases the number of potential points of failure and the complexity of recovering from error states. For instance, if an application terminates unexpectedly, it is necessary to search quickly through the large number of running nodes to locate the problematic nodes and states. Moreover, to avoid costly downtime, administrators must quickly remedy the problematic node states to avoid further spread of errors.

Although Amazon EC2 provides a user-friendly and simple interface to manage and control the application instances (Figure 1a), administrators must still be experienced with the administrative commands, the configuration of each application, as well as some domain knowledge about each running instance. Administrators must therefore be highly trained to effectively and efficiently handle error detection and error recovery. The complexity of managing a large cloud of nodes can increase maintenance costs, however, especially when personnel are replaced due to turnover or downsizing.


Application Instances in the Cloud
SETI
(a) Command Line Control Interface
(b) Runtime Model

```
SETI@home client.
Platform: i386-winnt-cmdline
Version: 3.03

SETI@home is sponsored by individual donors around the world.
If you'd like to contribute to the project,
please visit the SETI@home web site at
http://setiathome.ssl.berkeley.edu.
The project is also sponsored by the Planetary Society,
the University of California, Sun Microsystems, Paramount Picture
Fujifilm Computer Products, Informix, Engineering Design Team Inc
The Santa Cruz Operation (SCO), Intel, Quantum Corporation,
and the SETI Institute.

SETI@home was developed by David Gedye (Founder),
David Anderson (Director), Dan Werthimer (Chief Scientist),
Leonard Chung, Hiram Clawson, Jeff Cobb, Charles Congdon, Cha
Kyle Granger, Eric Heien, Mike Hill, Michael Kang, Eric Korp
Matt Lebofsky, Peter Leiser, Brad Silen, Woody Sullivan, an

Scanning data file
Data Info:
Sky coordinates:  6.327 R.A., 12.990 Dec
Recorded on:  2452422.24294 (Mon May 27 17:49:50 2002)
Source: Arecibo Radio Observatory
Base Frequency: 1.420498047 GHz
Found data file: yes. Found result header file: yes.
Scanning result header file.
Beginning analysis...
Doing baseline smoothing.
Starting work for Doppler shift rate 1.696941, FFT length 131072
7.532363% done
Starting work for Doppler shift rate 1.698789, FFT length 131072
```

Figure 1. Two Options to Control Application Instances

Even with experienced administrators, the process of error recovery involves the following challenges:

- It is hard to locate errors accurately with a large number of running application instances.
- It may take too much time to detect and locate an error, causing a long period of service termination or further error propagation.
- Error recovery becomes a time-consuming and error-prone task when it involves multiple and/or complex modification actions.

- Without a logging function, it is hard to track past recovery actions, thereby leading to a potentially unreliable recovery.
- Key error recovery knowledge may become concentrated in a few individuals, which becomes problematic when they leave the organization.

The main reason for these challenges is the lack of automated mechanisms to aid error recovery. Relying on manual actions to handle each error does not scale as the number of node instances increases.

Solution approach → Automated Evolution of Runtime Models. This chapter presents a new automated approach to error recovery in cloud computing based on high-level graphical models of distributed application state to present the application status to an administrator. When administrators identify and correct errors in the model, an inference engine is used to identify the specific state pattern in the model to which they were reacting. Moreover, the administrator's recovery actions are recorded. Subsequently, the model is automatically monitored for the previously identified error state and when it is encountered, the recorded recovery actions are automatically replayed or presented to the administrator as a potential course of action.

In this approach, a domain-specific modeling language (DSML) is first developed to define the running status of a specific cloud application. A DSML is usually a graphical domain-specific language to model the various aspects of a system. It tends to support higher level abstractions than general-purposing languages, resulting in less effort to specify a system. A runtime model is then constructed automatically to serve as a graphical monitoring interface and to reflect the running nodes and states of an application. Whenever errors appear in the cloud, they are also reflected in the model (i.e., models are relevant at runtime). A causal connection is established such that correcting errors in the runtime model triggers the same corresponding changes in the cloud. Because models are a high-level abstraction of the application instances, administering changes by editing the models (Figure 1b) is easier and more efficient than using the traditional command-line interface. Moreover, changes to high-level modeling abstractions can trigger the automatic issuance of multiple low-level administrative changes in the cloud.

Recovering from errors by modifying runtime models is still a manual procedure that has similar drawbacks with respect to scalability, productivity, knowledge centralization, and repeatability. Instead of correcting erroneous model states manually for each occurrence of a fault, this chapter focuses on "Recovery by Demonstration," which is derived from the idea of model transformation by example [4]. Recovery by demonstration is a process whereby an administrator manually specifies a series of recovery actions for an error state and then the automation infrastructure automatically replays the same recovery actions when an identical error state is encountered in the future. Recovery by Demonstration is facilitated by the use of domain-specific models and model transformations. In particular, the first time that an error appears in a cloud application, a user must manually discover the problematic model elements associated with the error and manually demonstrate the recovery process by modifying the model to remove the error state.

This chapter describes a plug-in to a domain-specific modeling tool that can record all the recovery operations performed and then infer the error state from patterns in the model. After the first occurrence of the error, the model's state is continuously checked for the error. Whenever the error pattern is matched, the recorded recovery operations are replayed automatically to correct the error (please see Figure 2).

To showcase Recovery by Demonstration, this chapter presents a case study involving the detection and recovery of errors in a 3-tiered Enterprise Java Beans (EJB) application that is executing within the Amazon EC2 platform. By recovering from some common errors, the case study demonstrates the potential for dynamic cloud application reconfiguration through model-driven engineering. The goal of our approach is to manually correct once and then automatically recover anytime and anywhere from the same error.

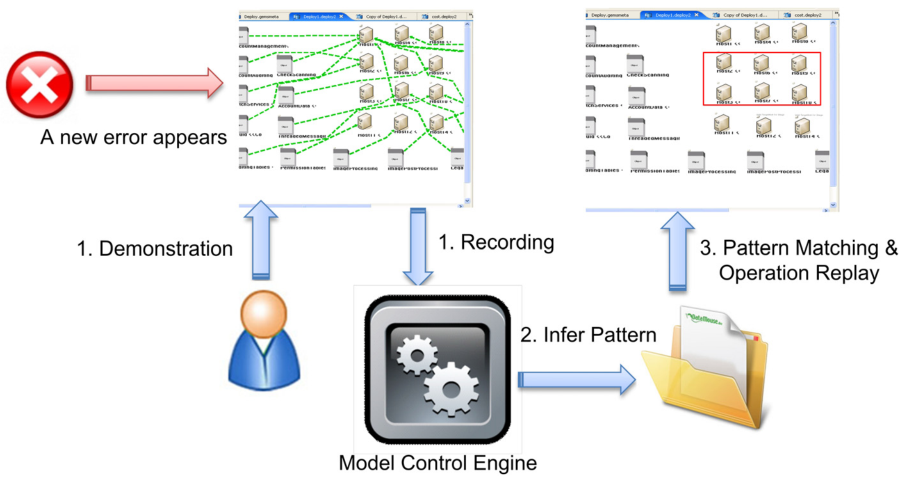


Figure 2. Recovery by Demonstration

Outline of the Proposed Chapter

Section 2 (*Error Recovery in Cloud Computing*) will present a detailed introduction to cloud computing with several successful cloud computing examples. Our experiment is based on the Amazon EC2, so this section will focus more on basic usage scenarios provided in EC2. A distributed version of the Java Pet Store (which is Sun Microsystems' reference implementation of a 3-tiered EJB application) will be used as a case study throughout the chapter. The configuration and execution of this application in Amazon EC2 will be shown to provide a tutorial about how an application instance operates in Amazon EC2. From this background information, the chapter will describe the challenges of error recovery in cloud computing.

The chapter discussion will focus around common errors that may occur when running the Java Pet Store, and then describe the solutions to recover from them. Based on this example, the chapter will summarize and highlight the problems of manual error recovery. These problems motivate the automated solution that is elaborated in the following sections of the chapter.

Section 3 (*Building Runtime Cloud Computing Models*) establishes the first step towards automated error recovery. Although Amazon EC2 offers a set of commands to control and monitor application instances, it has a steep learning curve that can result in increased training costs. Moreover, the static textual information and command line control mode is more challenging to realize automated management due to its form of representation. Amazon EC2's management infrastructure is also not designed to capture application-specific state information or perform application-specific recovery actions. One approach to address the application state representation problem is to build corresponding domain-specific models to represent all the instances, their states and connections. The models can be built using a DSML that is graphical and can express domain concepts at a higher level of abstraction.

DSMLs are defined in domain-specific modeling tool environments. This chapter uses the Generic Eclipse Modeling System (GEMS) [6] tool environment. This section of the chapter first defines the metamodel of the distributed version of Java Pet Store, which describes how each component can be configured and connected. An information exchange module is then developed to communicate with the Amazon EC2 controller and retrieve information about application instances. Based on the information received, a runtime model describing each application and its states is constructed automatically within GEMS.

Reflecting the state of running application instances is just one goal. What is more important, however, is to enable changes to the real instances by modifying models, i.e., creating a causal connection between models and executable instances. The exchange module should therefore also be capable of passing the same model editing actions to the real instances. To demonstrate the effect, one simple editing example will be given to show the causal connection. This high-level graphical representation and the causal connection still require manual intervention and suffers from the same challenges mentioned in the introduction (e.g., scalability issues that may affect productivity and correctness).

Section 4 (*Recovery by Demonstration*) will present how our approach automates error detection and recovery. Editing the models to implement recovery is actually a model transformation process, or more precisely, an endogenous model transformation [7], where both the source and target model conform to the same metamodel. The most direct way to automate this process is to use model transformation languages, such as ATL [8] or C-SAW [9]. By specifying the precondition and transformation rules, an error condition in a model could be captured and then transformed to a correct application state. Generalizing the precondition and rules from the view of a metamodel is hard, however, and using transformation languages gives an extra learning burden to the administrators, which could increase the cost of training and maintenance. To avoid learning new languages, the transformation process can be simplified by the idea of "Recovery by Demonstration" using the three steps shown in Figure 2 and described below:

1. **Demonstration and recording**. In this step cloud administrators directly edit the problematic model elements and connections, and make necessary changes to recover from the errors. At the same time, a rule inference engine (a plug-in written for GEMS) can record all the edit operations performed by the administrator.
2. **Inference and summary**. In this step the error pattern is inferred, based on the recorded operations. This pattern describes the specific characteristics of the error: where are the problematic model elements (e.g., a model element connected with two other same kind of model element), and what are the signs of errors (e.g., the value of a certain attribute exceeds the normal value). The recorded operations are then generalized so that their operands can be mapped to this pattern. The combination of the pattern and operations is an example of an error recovery case that can be reused automatically.
3. **Pattern matching and operation replay**. In this step the inferred patterns are checked whenever changes occur in the models. If a part of the model matches a certain error pattern, it is recognized automatically and the corresponding recovery operations are replayed at this location to simulate the manual recovery process.

The error recovery case study from Section 2 will be used to illustrate these steps. This section will show the recorded operations, inferred pattern structure, and the automated recovery process of implementing this example by our approach. At the end of this section, a summary of the results of the approach will accompany an evaluation that compares the time and difficulty to realize one error recovery task using three different approaches: purely manual correction, model transformation, and recovery by demonstration.

Section 5 (*Related Work*) will provide a survey of related work and associate newly developed technologies with the existing ones to support automated error recovery in cloud computing. Related research within the areas of autonomic computing, models at runtime, and model transformation techniques will be compared and reviewed within the context of the proposed chapter.

Section 6 (*Lessons Learned and Future Work*) points out the potential extensions and the current limitations of our approach. For instance, the recording mechanism could be used to realize logging so that the reason of the failure can be tracked; it is also possible to record more generic actions rather than model editing operations only, such as running a model transformation, and invoking certain applications. The summarized error recovery cases in the repository can be applied in the automated recovery process, and also serve as a valuable source of error recovery training experience. In addition, the metamodel of the application could be made more generic so that it can describe many kinds of applications running in Amazon EC2 or even in all cloud computing platforms. This idea could also be useful in some other scenarios where runtime models or model transformation are needed.

One limitation with our current approach is that the metamodel defined as the case study of this chapter is for a specific application, the result being that different metamodels for different applications must be created first to enable the construction of runtime models, which is not generic enough to specify a common solution. In future work, we will analyze more applications and try to capture their common concepts and properties in order to build a generic metamodel capable of capturing the state of a wide variety of cloud computing applications.

Section 7 (*Conclusion*) will present concluding remarks and project demo links.

References

1. Schmidt, D. "Model-Driven Engineering," IEEE Computer, vol. 39 no. 2, pp. 25-32 (2006).
2. Hayes, B. "Cloud Computing," Communications of the ACM, 51(7):9–11 (2008).
3. Rappa, M. "The Utility Business Model and the Future of Computing Services," IBM Systems Journal, vol. 43, no. 1, pp. 32-42 (2004).

4. Varro, D. "Model Transformation by Example," in Proceedings 9th International Conference on Model Driven Engineering Languages and Systems, Genova, Italy, vol. 4199 of LNCS, pp. 410 – 424, Springer (2006).
5. Amazon Elastic Compute Cloud (Amazon EC2). http://aws.amazon.com/ec2/
6. Generic Eclipse Modeling System (GEMS). http://www.eclipse.org/gmt/gems/
7. Mens, T., Gorp, P. "A Taxonomy of Model Transformation," Proceedings of the International Workshop on Graph and Model Transformation, vol. 152, pp. 125-142 (2005).
8. Jouault, F., Allilaire, F., Bézivin, J., Kurtev, I. "ATL: A Model Transformation Tool," Science of Computer Programming, vol. 72, nos. 1/2, pp. 31-39 (2008).
9. Gray, J., Lin, Y., Zhang, J. "Automating Change Evolution in Model-Driven Engineering," IEEE Computer, Special Issue on Model-Driven Engineering, vol. 39, no. 2, pp. 51-58 (2006).

Relevance for This Book

This proposed chapter describes a practical framework to improve cloud computing domain analysis and control. All the techniques used in the solution are within the context of model-driven engineering, such as domain-specific modeling, domain-specific modeling languages and tools, runtime model and model transformation. This chapter can serve as an innovative demonstration on how model-driven engineering can be applied to improve domain analysis and its related applications, but also a presentation about how model-driven techniques are used in runtime adaptation.

Author Biographies

Yu Sun is a Ph.D. candidate in the Department of Computer and Information Sciences at the University of Alabama at Birmingham (UAB) and member of the SoftCom Laboratory. His research interests include domain-specific modeling, domain-specific languages and model transformation techniques. He received his BS in Computer Science from Zhengzhou University, China and MS in Computer Science from UAB. He is a student member of the ACM.

Dr. Jules White is a Research Assistant Professor at Vanderbilt University. He received his BA in Computer Science from Brown University, his MS in Computer Science from Vanderbilt University, and his Ph.D. in Computer Science from Vanderbilt University. Dr. White's research focuses on applying a combination of model-driven engineering and constraint-based optimization techniques to the deployment and configuration of complex software systems. Dr. White is the project leader for the Generic Eclipse Modeling System (GEMS), an Eclipse Foundation project.

Dr. Jeff Gray is an Associate Professor in the Computer and Information Sciences Department at UAB, where he co-directs research in the SoftCom Laboratory. His research interests include model-driven engineering, aspect orientation, code clones, and generative programming. Jeff received a Ph.D. in Computer Science from Vanderbilt University and both the BS and MS in Computer Science from West Virginia University. He is a member of the ACM and a Senior Member of the IEEE.

Dr. Aniruddha S. Gokhale is an Assistant Professor of Computer Science and Engineering in the Department of Electrical Engineering and Computer Science at Vanderbilt University. He received his BE (Computer Eng) from Pune University; MS (Computer Science) from Arizona State University; and D.Sc (Computer Science) from Washington University. Prior to joining Vanderbilt, he was a Member of Technical Staff at Bell Labs, Lucent Technologies. Dr. Gokhale is a member of IEEE and ACM. Dr. Gokhale's research combines model-driven engineering and middleware for distributed systems, notably real-time and embedded systems. He is the project leader for the CoSMIC model-driven engineering tool suite at Vanderbilt.

Dr. Douglas C. Schmidt is a Professor of Computer Science and Associate Chair of the Computer Science and Engineering program at Vanderbilt University. He has published 9 books and over 400 papers that cover a range of topics, including patterns, optimization techniques, and empirical analyses of software frameworks and domain-specific modeling environments that facilitate the development of distributed real-time and embedded (DRE) middleware and applications. Dr. Schmidt has over 15 years of experience leading the development of ACE, TAO, CIAO, and CoSMIC, which are open-source middleware frameworks and model-driven tools that implement patterns and product-line architectures for high-performance DRE systems.