Abstract. Continuous integration and quality-of-service (QoS) provisioning is becoming increasingly important in the software development lifecycle of large-scale distributed systems. This poster describes an approach that uses model-driven engineering and generative programming techniques to provide continuous QoS provisioning of large-scale distributed systems throughout the entire development lifecycle, instead of waiting until integration time to analyze and rectify performance problems, which may be too late.

1 Introduction

Traditional methods of development for large-scale distributed systems created systems that were hard to maintain, keep up-to-date with technology transitions, and required constant reinvention of key aspects in system innovations. These development processes are gradually transitioning to Service-Oriented Architecture (SOA) middleware to address many of the complexities of developing these systems. SOA middleware allows developers to capture the “business-logic” of their system in (re)configurable components that can be (re)used in multiple application domains. Moreover, SOA middleware provides out-of-the-box solutions for many quality-of-service (QoS) concerns, such as security and reliability. The technologies that make this possible include middleware solutions, such as the CORBA Component Model (CCM) [3], J2EE [5], and Microsoft .NET [2].

Although this transition is raising the level of abstraction and requiring developers to focus only on the “business-logic” of the system, many design flaws are not discovered much later until integration time. For example, either some components are not available or QoS of components may not be ascertained until years into development. System developers and engineers, therefore, cannot perform any preliminary QoS provisioning of the entire system until much later in the development lifecycle. Waiting too long to locate and rectify such problems, however, can incur substantial time and cost than resolving them in the early stages of development. It is evident that new techniques are needed to assist system developers in continuously understanding QoS provisioning throughout the entire development lifecycle.

Proposed Solution. We are developing a system execution modeling (SEM) tool suite called the Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS) [1].
CUTS uses model-driven engineering (MDE) techniques to assist system developers and engineers in testing large-scale component-based systems over the entire lifecycle of the entire system, i.e., even when all the business logic may not be available by emulating the behavior of the unavailable components. It also provides real-time data collection solutions for capturing QoS metrics and presenting them in a graphical format.

Our work extends the current implementation of CUTS to support the instrumentation of real components so developers can replace the emulated components with the “real” components as they become available. This will allow experiments performed with CUTS to obtain a higher fidelity in results. These features are required to realize the goals of continuous integration and QoS provisioning throughout the entire development lifecycle of the system.

This poster will illustrate our continuous QoS provisioning and integration framework described in Section 2. It will also be accompanied by a demo described in Section 3. Lastly, it will highlight future work described in Section 4.

2 Continuous QoS Provisioning

This section describes the approach that overcomes the key challenges of continuous QoS provisioning of large-scale, distributed systems throughout the entire development lifecycle. Our poster outlines the challenges and the approach we have taken to overcome these challenges.

Emulation of Faux Components. While real components are being developed, CUTS generates faux components that represent their “real” counterparts, which are temporarily used to provision QoS. Developers capture the behavior and resource usage of the real components at a high-level of abstraction using the Component Behavior Modeling Language (CBML) and Workload Modeling Language (WML) [4] domain-specific modeling languages (DSMLs). This is then synthesized to the faux components and emulated on the target architecture to capture QoS performance metrics at the early stages of development.

Instrumentation of “Real” Components. As the real components become available, it is necessary to substitute them for the faux components to gain higher confidence in provisioning system QoS and move towards a more realistic system. However, real components are often not tailored to QoS instrumentation. We, therefore, use a proxy-based approach to instrument the real components. The proxies look and feel like the real components, but contain no application-level functionality. The proxies are only responsible for receiving events and forwarding them to the real hosted component. The proxies also capture various QoS metrics such as execution time using methods similar the those we used for the emulated components.

Generation of Realistic Workloads. Proxy components can replace faux components at either the beginning, middle, or end of the assembled system. Faux components, therefore, must generate valid outputs just in case it is communicating, e.g., sending events, to a real component. Faux components at the beginning of the system are only concerned with valid outputs, whereas, faux components in the middle must know how to correlate valid/invalid input with valid/invalid output. Faux components at the end
of a component assembly can neglect generating valid outputs since they do not exist. Our work, therefore, extends our existing WML DSML to allow the capturing of valid/invalid inputs and outputs. This will then be synthesized into workload data generators that can be inserted into their respective faux component so it can transmit meaningful data, if necessary.

3 Demonstration

In this section we describe our demo that will accompany the poster. The demo will comprise current capabilities of CUTS. We will show a representative system from the domain of computing environments on battle ships. Our demo will illustrate the following key abilities of CUTS:

1. Capture the behavior and QoS characteristics of system applications using DSMLs.
2. Synthesize, emulate and capture execution times of the application components.
3. Evaluate whether application components are meeting the required execution time, and if not, then evaluate other alternatives that can meet the required metrics.

Moreover, this demo will illustrate how CUTS can help system developers and engineers exhaustively narrow an otherwise large solution space that consists of hundreds of deployment and configuration alternatives to a set that will possibly meet the execution time QoS requirements when the system is deployed using the “real” components.

4 Future Work

SOA middleware provides out-of-the-box solutions for many QoS concerns such as ability, reliability, and security. Our future work, therefore, involves extending CUTS to help system developers and engineers understand how these solutions affect overall QoS provisioning, which includes multi dimensional QoS properties, for individual components, and the overall system.

References