Model Driven Quality Assurance Techniques for DRE Applications

Arvind S. Krishna & Emre Turkay
Andy Gokhale, Douglas C. Schmidt
Institute for Software Integrated Systems (ISIS)
Vanderbilt University
Nashville, TN 37203

Adam Porter, Cemal Yilmaz
Atif Memon

University of Maryland
College Park

Work supported by AFRL contract# F33615-03-C-4112 for DARPA PCES Program
**Context: Deployment & Configuration Spec**

**Packaging**
- bundling a suite of software binary modules and metadata representing application components

**Installation**
- populating a repository with the packages required by the application

**Configuration**
- configuring the packages with the appropriate parameters to satisfy the functional and systemic requirements of application without constraining to any physical resources

**Planning**
- making appropriate deployment decisions including identifying the entities, such as CPUs, of the target environment where the packages will be deployed

**Preparation**
- moving the binaries to the identified entities of the target environment

**Launching**
- triggering the installed binaries and bringing the application to a ready state

**Adaptation**
- Runtime reconfiguration & resource management to maintain end-to-end QoS
Various strategies in CIAO can be configured at initialization time.
CIAO ORB is configured with service configuration files.

```
estts/Latency/Thread_Per_Connection/svc.conf
ACE_Svc_Conf>
<!--  -->
<!--  -->
<static id="Advanced_Resource_Factory"
params="-ORBReactorType select_mtu -ORBReactorMaskSignals 0 -ORBFlushStrategy blocking"/>
<static id="Client_Strategy Factory"
params="-ORBTransportMuxStrategy EXCLUSIVE -ORBClientConnectionHandler RW"/>
<static id="Server_Strategy_Factory"
params="-ORBConcurrency thread-per-connection"/>
/ACE_Svc_Conf>
```
Definition

• Several different ways to configure the underlying middleware
• Several domains have common characteristics. For example:
  • Avionics Domain and Enterprise Applications might share applications that have similar QoS requirements
  • Both might share similar concurrency, latency and jitter requirements
• If we can codify these recurring configurations then these are patterns that can be reused across these domains for the same middleware application

Identification

• To reduce the cost for identification of these patterns we have followed a three pronged approach
  1. Developed Configuration tools that allow end-users to model and generate semantically correct configuration options
  2. Developed Empirical evaluation tools to benchmark and run experiments on components configured using the configuration options
  3. Developed Distributed Continuous Quality Assurance techniques to iteratively run these configurations on varied hardware/OS/compiler platforms to identify recurring solutions to resolving QoS requirements.

Representation

• These patterns are represented as set of tuples \{(x, value(x))\} where x is the configuration option and value(x) is the configuration setting for x
Model Based Solution \(\rightarrow\) OCML

- Developed a domain-specific modeling language for TAO/CIAO called Options Configuration Modeling Language (OCML) using GME
- User provides a model of desired options & their values e.g.,
  - Middleware bus resources
  - Concurrency & connection management strategies
- Constraint checker flags incompatible options
- Synthesizes XML descriptors for middleware configuration
- Generates the documentation for the middleware configuration
- Online validation of the configurations
OCML Workflow

- OCML is used by
  - **Middleware developer** To design the Configuration Model
  - **Application Developer** To configure the Middleware for a Specific Application

- OCML metamodel is platform independent
- OCML models are platform specific
Configuration Aspect: Challenge #1

1: Users define the application QoS requirements such as pulse rate for the timer.

2: Design model transformers to synthesize platforms-specific configurations for achieving QoS. Currently done with the OCML modeling paradigm.

How do you ensure this configuration maximizes the QoS?
Planning Aspect: Challenge #2

- How do you determine current resource allocations?
- How do you correlate QoS requirements of packages to resource needs?
- How do you ensure that the selected targets will deliver required QoS?
BGML Motivation
• Provide a model-driven tool-suite to empirically evaluate and refine configurations to maximize application QoS

BGML Workflow
1. End-user composes the scenario in the BGML modeling paradigm
2. Associate QoS properties with this scenario, such as latency, throughput or jitter
3. Synthesize the appropriate test code to run the experiment and measure the QoS
4. Feed-back metrics into models to verify if system meets appropriate QoS at design time

• The tool enables synthesis of all the scaffolding code required to set up, run, and tear-down the experiment.
• Using BGML tool it is possible to synthesize:
  • Benchmarking code
  • Component Implementation code
  • IDL and Component IDL files
Resolving Config & Planning Challenges

Resolving Challenge #1
• How to determine the configuration settings for the individual components to achieve the QoS required for this scenario?

Boeing’s BasicSP Scenario
• **Navigation Display** – displays GPS position updates
• **GPS Component** – generates periodic position updates
• **Airframe Component** – processes input from the GPS component and feeds to Navigation display
• **Trigger Component** – generates periodic refresh rates

**Step 1:** Model Component Interaction using BGML Paradigm

**Step 2:** Configure each Component associating the appropriate IDL interfaces
Modeling Avionics Scenario

### Step 3: Associate Operations with the Component Ports

```cpp
void start_time_probe ()
{
  if (!timer_count)
    test_start = ACE_OS::gethrtime ();

  start_timer_probe = ACE_OS::gethrtime ();
}

void stop_time_probe ()
{
  finish_timer_probe = ACE_OS::gethrtime ();
  history.sample (finish_timer_probe - start_timer_probe);
  if (++ timer_count == niterations)
  {
    test_end = ACE_OS::gethrtime ();
    ACE_DEBUG ((LM_DEBUG, "test finished\n"));
    ACE_Throughput_Stats::dump_throughput ("Total", gsf,
              test_end - test_start,
              stats.samples_count ());
  }
}
```

### Step 4: Associate QoS metrics to measure in the scenario

- BGML Paradigm allows actual composition of the target interaction scenario, auto-generates benchmarking code
- Each configuration option can then be tested to identify the configuration that maximizes the QoS for the scenario
- These empirically refined configurations can be reused across applications that have similar/same application domains
- These configurations are configuration patterns – Configuration and Customization (C&C) patterns
BGML Metrics

Code Generation Metrics
• BGML allows generation of ~ 80% of all the required files to enact the scenario.

Identification of C&C Patterns
• For each component arrive at the configuration space i.e. all possible configuration parameters to tune QoS.
  • Nav Display component plays role of client with no requirements for concurrency
  • Using each option permutation arrive at the option that optimizes QoS, e.g. latency for the application
  • Set of configuration options along with the values represents a reusable configuration pattern

For pure clients the following settings represents a reusable C&C pattern:
{{ORBProfile = null}, (ORBClientConnectionHandler = RW), (ORBTransportMuxStrategy = Exclusive), (ORBConnectStrategy = Reactive)}}
Distributed Continuous QA – Motivation

Persistent Challenges
- Scale
  - Massive configuration & optimization space
- Time to market pressure
  - Rapid updates, frequent releases
- Heterogeneity
  - Scattered resources & distributed developers
- Incomplete information
  - Unobservable usage context

Emerging Opportunities
- Leverage remote computing resources and network ubiquity for distributed, continuous QA

Existing Quality Assurance Processes:
- Auto-build scoreboard systems
  - e.g. Mozilla’s Tinderbox
- Error reporting based on prepackaged installation tests
  - e.g. GNU GCC and ACE & TAO
- Online crash reporting
  - e.g. Netscape’s QFA and Microsoft’s Watson
- Shortcomings: inadequate, opaque, inefficient and inflexible QA processes
  - Scope: Generally restricted to functional testing and often incomplete
  - Documentation: No knowledge of what has or hasn’t undergone QA
  - Control: Developers have no control over the QA processes
  - Adaptation: Can’t learning from the earlier test results
Skoll Project

- Vision: QA processes conducted around-the-world, around-the-clock on powerful, virtual computing grid provided by thousands of user machines during off-peak hours
- Generic Skoll DCQA Process
  - Distributed
    - Identify QA task (e.g., testing, profiling, anomaly detection of program family)
    - Divide goal into subtasks each of which can be performed on a single processing node (i.e., user machines)
  - Opportunistic
    - When a node becomes available allocate one or more subtasks to it
    - Distribute subtasks to node and collect results when available
  - Adaptive
    - Use data-driven feedback to schedule and coordinate subtask allocation
- We are currently building an infrastructure, tools and algorithms for developing and executing thorough, transparent, managed, adaptive DCQA processes
The Skoll System

Visualization
  e.g., scoreboard

Automatic characterization
  e.g., classification trees

Visualization
  e.g., scoreboard

Skoll Server(s)

Skoll Clients

Adaptation strategies
  e.g., nearest neighbor

Intelligent Steering Agent
  (ISA)

Adaptation Strategies
  ...

Subtask Code
  <sub-task>
    <download>
      <cvs>cvs.doc.wustl.edu</cvs>
      <module>ACE+TAO</module>
      <version>v5.2.3</version>
    </download>
    ...
  </sub-task>

Visualization
  e.g., scoreboard

Visualization
  e.g., scoreboard
Discovery & Classification of C&C patterns

A: Model test-application
B: Synthesize required configuration parameters using OCML
C: Synthesize benchmarking code using BGML
D: Feed test code to Skoll framework
E: Run on varied platforms to measure QoS variations
F: Maintain database of results from which C&C patterns can be identified

Context
- Identifying C&C patterns allows reusable best configurations to be readily applied to similar QoS requirements in various domains.

Problem
- Identification hard as we need to explore the entire configuration space. These keep growing at a great pace.
  - Leads to configuration space explosion.
  - No time to run tests for each combination for lack of resources.

Solution → Distributed Continuous Quality Assurance (DCQA)
- Framework for running tasks on user donated machines around the world, around the clock.
  - Uses spare CPU cycles on user machines, like SETI@home project
- Enables identification and validation of C&C patterns on range of hardware, OS and compiler platforms
Downloading the Middleware & Tools

OCML & BGML are part of the CoSMIC MDM tool suite

• Beta and Stable release can be accessed from http://www.dre.vanderbilt.edu/Download.html

• http://www.dre.vanderbilt.edu/cosmic