Automated Extraction of Palladio Component Models from Running Enterprise Java Applications

Fabian Brosig, Samuel Kounev, Klaus Krogmann
Karlsruhe Institute of Technology (KIT), {firstname.lastname}@kit.edu

October 19, 2009
Performance Prediction during Operation

Application + Increasing Service usage = Utilization, (Resp. Times, Throughput)
Performance Prediction during Operation

Application → Monitoring Data → Performance Model

Utilization, (Resp. Times, Throughput)
Approach and Contributions

- Automatically extract
  - a Palladio Component Model (PCM) instance
  - from a *running* enterprise Java (Java EE) application
  - by means of *online monitoring data*
  - collected by *state-of-the-art monitoring tools*

- Model extraction method, gap between performance models and run-time information can be closed

- Validate method with a representative system
Background – Palladio Component Model

- Palladio Component Model (PCM) [BKR09]
  - Meta Model: Performance-relevant aspects of component-based software architectures

- Model Solving:
  Queueing Network based simulation (amongst others)
Background

- Oracle WebLogic Server (WLS)
- WebLogic Diagnostics Framework (WLDF)
  - Instrumentation engine
  - Monitor runtime information
    (e.g., size of database connection pool)
Model Extraction Method

Extracting a PCM Instance from a Running EJB Application
Model Extraction Method - Overview

- **Scope:** Enterprise JavaBeans (EJB) 3.0, Java Persistence API (JPA)

- Identifying Component Boundaries

- Extracting Inter-Component Control Flow
  - Call Path Tracing

- Extracting Intra-Component Control Flow
  - Call Path Tracing

- Extracting Resource Demands
  - Resource Consumption Monitoring
Call Path Tracing

```
<table>
<thead>
<tr>
<th>Request Id</th>
<th>Record Id</th>
<th>Service</th>
<th>Location</th>
</tr>
</thead>
<tbody>
<tr>
<td>x</td>
<td>1</td>
<td>serviceA1</td>
<td>entry</td>
</tr>
<tr>
<td>x</td>
<td>2</td>
<td>serviceC1</td>
<td>entry</td>
</tr>
<tr>
<td>x</td>
<td>3</td>
<td>serviceD1</td>
<td>entry</td>
</tr>
<tr>
<td>x</td>
<td>4</td>
<td>serviceD1</td>
<td>exit</td>
</tr>
<tr>
<td>x</td>
<td>5</td>
<td>serviceD1</td>
<td>entry</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>x</td>
<td>10</td>
<td>serviceA1</td>
<td>Exit</td>
</tr>
</tbody>
</table>
```
Extracting Control Flow

- Extracting Inter-Component Control Flow
- Extracting Intra-Component Control Flow

```java
class LargeOrderSession {
    ...
    public LargeOrder createLargeOrder(String assemblyId, ...) {
        // extracted external call
        Assembly assembly = createLargeOrder_1_externalcallbackaction_1(assemblyId);
        // extracted internal action
        LargeOrder largeOrder = createLargeOrder_1_inernalaction_2(assembly, ...);
        return largeOrder;
    }

    private LargeOrder createLargeOrder_1_inernalaction_2(Assembly assembly, ...)
    {
        /* compute something */
        LargeOrder largeOrder = new LargeOrder(assembly, ...);
        entityManager.persist(largeOrder);
        return largeOrder;
    }

    private Assembly createLargeOrder_1_externalcallbackaction_1(String assemblyId) {
        Assembly assembly = mfgSession.findAssembly(assemblyId);
        return assembly;
    }
}
```
Example: Extracted Service Behavior

Extracted resource demanding behavior (exemplary service)

Loop iteration number:
- $P(\text{"9 times"}) = 20\%$
- $P(\text{"10 times"}) = 50\%$
- $P(\text{"11 times"}) = 30\%$
Extracting Resource Demands

- CPU resource demands

- Approach 1
  - Approximate resource demands with measured response times

- Approach 2
  - Estimate resource demands based on utilization and throughput data

- Partitioning between
  - app. server CPU (WLS CPU)
  - and database server CPU (DBS CPU)?
Extracting Resource Demands - Example

- Measurement Interval
- CPU Utilization
- Action1 executed 8 times, Action2 executed 4 times

Apply Service Demand Law:
How to partition processing time?

Response Times: Action1 = 1 sec, Action2 = 2 sec

8 * 1 sec = 4 * 2 sec \(\Rightarrow\) 1:1 Relation

Resource Demands
WLS CPU demands and DBS CPU demands

```java
transaction {

} commit {

}
```

- Working phase ➔ WLS CPU
- Commit phase ➔ DBS CPU
- DB read vs. DB write?
Related Work I

- Trace-based approaches
  - Hrischuk [HWRI99], Israr [IWF07]
    - Extraction of Layered Queueing Network (LQNs) structures
    - Not component-based, no explicit control flow in components
  - Dynatrace Tool [DYNT]
    - Transaction call tree in distributed JavaEE / .NET environments
  - Briand [BLL06]
    - UML sequence diagrams

- Java application monitoring at run-time
  - Carrera [CGT+03]: hotspot and bottleneck analysis
  - Compas [MM02]: generates interaction models
Related Work II

- PCM extraction
  - Java2PCM [KKKR08]
    - Component-level control-flow based on static code analysis
  - Krogmann [KKR08a, KKR08b]
    - Static analysis (code) and dynamic analysis (run-time)
    - Not focused on extraction during system operation
    - Abstracts from concrete timing values (Java bytecode operations)
    - Extraction by means of machine learning techniques
  - ArchiRec [CKK08]
    - Provides component boundaries based on static code analysis
Case Study

With a prototype of SPECjEnterprise2010
SPECjEnterprise2010

- State-of-the-art, industry-standard benchmark
- Realistic, three-tier, Java EE application
- Representative workload
System Environment

- Benchmark deployed on three machines

![Diagram of system environment with Oracle WebLogic Server and Oracle Database 11g](image-url)
Results – Scenario 1: „Schedule Work Order“

- Model A: Resource demands approximated with measured response times
- Model B: Resource demands estimated based on utilization and throughput data

Model A: $U_{WLS\_CPU} = 0.12$, Model B: $U_{WLS\_CPU} = 0.81$, Steady State Time: 1020 sec
Results – Scenario 2a: „Create Vehicle“

- Model A: Resource demands approximated with measured response times
- Model B: Resource demands estimated based on utilization and throughput data

Model A: $U_{WLS\_CPU} = 0.09$, Model B: $U_{WLS\_CPU} = 0.75$, Steady State Time: 1140 sec
Results – Scenario 2a: „Create Vehicle“

Model A: U_{WLS\_CPU} = 0.09, Model B: U_{WLS\_CPU} = 0.75, Steady State Time: 1140 sec
Results – Scenario 2b: „Schedule Work Order“

- Model A: Resource demands approximated with measured response times
- Model B: Resource demands estimated based on utilization and throughput data
Conclusion
Summary and Future Work
Concluding Remarks - Summary

- Automated end-to-end model extraction
  - Extracts PCM instance from EJB application
    - Components, Component connections
    - Component internals
    - Resource demands
  - During application run-time
- Used WLDF to implement tool prototype

- Case study with prototype of SPECjEnterprise2010
  - Feasibility, Proof-of-concept
Concluding Remarks – Future Work

- Extend extraction method
  - Asynchronous Messaging
  - Extraction of component internals [KKR08a, KKR08b]
  - Resource demand estimation (e.g., statistical regression [Pac08, Rol95], JRockit method sampling)

- Extend case study
  - SPECjEnterprise2010
  - Application Server Cluster

- Descartes vision
Questions?
Using WLDF to Enable Performance Prediction for Java EE Applications

Components

Component Interfaces

Background – Palladio Component Model
Service provided by a component described by: Resource Demanding ServiceEffectSpecification (RDSEFF)

Performance relevant behavior
- Abstract control flow
- Resource demands
Challenges

- Apportioning resource demands among different resources
- Stable system environment (repeatable experiments)
- Instrumentation Overhead
- Instrumentation vs. JRockit Optimization
- Benchmark driver measured incorrect response times
  - Usage of `System.currentTimeMillis (Windows)`
- Dynamic frequency/voltage scaling
  - EIST (SpeedStep), load-dependent resource
Scenario 1: Resource & Allocation Model

- PCM Resource Environment Model

```
Resource Environment <ResourceEnvironment>
  ` WLS <ResourceContainer> [ID: _yy0krP0Ed61z8f-NidkpQ]
      Processing Resource CPU: Rate: 1000 Scheduling: PROCESSOR_SHARING <ProcessingResourceSpecification>
      ProcessingRate: 1000 <PCM Random Variable>
  ` OracleDBS <ResourceContainer> [ID: _00jfox0P0Ed61z8f-NidkpQ]
      Processing Resource CPU: Rate: 1000 Scheduling: PROCESSOR_SHARING <ProcessingResourceSpecification>
      ProcessingRate: 1000 <PCM Random Variable>
```

- PCM Allocation Model

```
<table>
<thead>
<tr>
<th>WLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation_DelegateWorkOrderSession &lt;DelegateWorkOrderSession&gt;</td>
</tr>
<tr>
<td>DelegateWorkOrderSession &lt;DelegateWorkOrderSession&gt;</td>
</tr>
<tr>
<td>Allocation_WorkOrderSession &lt;WorkOrderSession&gt;</td>
</tr>
<tr>
<td>WorkOrderSession &lt;WorkOrderSession&gt;</td>
</tr>
<tr>
<td>Allocation_MessageSenderSession &lt;MessageSenderSession&gt;</td>
</tr>
<tr>
<td>MessageSenderSession &lt;MessageSenderSession&gt;</td>
</tr>
<tr>
<td>Allocation_MfgSession &lt;MfgSession&gt;</td>
</tr>
<tr>
<td>MfgSession &lt;MfgSession&gt;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OracleDBS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Allocation_JDBCClient &lt;JDBCClient&gt;</td>
</tr>
<tr>
<td>JDBCClient &lt;JDBCClient&gt;</td>
</tr>
</tbody>
</table>
```
Identifying Component Boundaries

- Options to specify component boundaries
  - Every EJB is considered as a separate component.
  - EJBs contained in the same package are considered as a single component.
  - The grouping of EJBs into components is specified explicitly.

Tool Prototype: Specification provided as XML doc conforming to an XML schema
Tool Prototype - Extraction Steps

- Extraction Steps
  - Configure WLDF to
    - extract the application's structure
    - extract performance-relevant control flow
  - - Monitor running application -
  - Configure WLDF to
    - extract resource demands
  - - Monitor running application -
  - Extract PCM repository model and PCM system model
Extracting Intra-Component Control Flow I

- Naming of methods, where performance-relevant actions are moved to

```
<methodname> = <pname> ' ' <idA> ' ' <action> ' ' <idZ>
<pname> = The name of the parent method.
<action> = 'internalaction' | 'externalcallaction' |
            'loopaction' | 'loopbody' | 'branchaction' | 'branchtransition'
:idA> = A number that, together with <pname>, uniquely identifies the parent method. In case of method overloading, the parent method’s name alone could not be used as identifier.
:idZ> = A number that uniquely identifies the extracted method amongst those extracted methods with the same parent method. Note that the number only serves as identifier, it is not an index of a sequence.
```
References


http://www.q-impress.eu/Q-ImPrESS/CMS/Overview/index.html.
References


References


