Enabling Performance Antipatterns to arise from an ADL-based Software Architecture

Vittorio Cortellessa, Martina De Sanctis, Antinisca Di Marco, Catia Trubiani
University of L’Aquila
L’Aquila, Italy

Joint IEEE/IFIP Conference on Software Architecture & European Conference on Software Architecture
Helsinki, Finland, 20-24 August 2012
What to change (at the architectural level) in order to improve the software performance?

Performance requirements:
responseTime(service_x) < 8sec

responseTime(service_x) = 10sec
responseTime(service_x) = 7sec
Problem statement

Modelling

Architectural specification

Performance specification

Analysis

Performance Evaluation

Performance results

Refactoring

Performance Antipatterns

Results Interpretation & Feedback Generation

**Antipatterns**: negative features of a software system

- Conceptually similar to design patterns: recurring solutions to common design problems
- The definition includes common mistakes (i.e. bad practices) in software development as well as their solutions

### Performance Antipatterns: what to avoid and how to solve performance problems

<table>
<thead>
<tr>
<th>Antipattern</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Unbalanced Processing</td>
<td>Processing cannot make use of available processors.</td>
<td>Restructure software or change scheduling algorithms to enable concurrent execution.</td>
</tr>
<tr>
<td>“Pipe and Filter” Architectures</td>
<td>The slowest filter in a “pipe and filter” architecture causes the system to have unacceptable throughput.</td>
<td>Break large filters into more stages and combine very small ones to reduce overhead.</td>
</tr>
<tr>
<td>Extensive Processing</td>
<td>Extensive processing in general impedes overall response time.</td>
<td>Move extensive processing so that it doesn't impede high traffic or more important work.</td>
</tr>
<tr>
<td>The Ramp</td>
<td>Occurs when processing time increases as the system is used.</td>
<td>Select algorithms or data structures based on maximum size or use algorithms that adapt to the size.</td>
</tr>
</tbody>
</table>


Round-trip performance process in the AEmilia ADL

Text-to-Model Transformation

AEmilia Grammar

conform to

AEmilia Architectural specification (file.aem)

transformation rules

rule 'mapElemTypes'
  from elem_type et
to ElemType
queries
  name : /#et;
  elemHeader : /et/#et_header;
  inputInt : /et//interaction_list_input
            //#interactionInput;
  outputInt : /et//interaction_list_output
            //#interactionOutput;
  behavior : /et/#behavior_equation_list;
Mappings
  etName = name.WORD;
  elemHeader = elemHeader;
  iDecl = inputInt;
  oDecl = outputInt;
  behaviorDec = behavior;
end_rule

PAML[AEmilia]
Metamodel

AEmilia Model

Antipatterns Detection

OCL code for the detection of the Extensive Processing Antipattern

```oclnotation
--function for the detection of the Extensive Processing antipattern
def: checkExtensiveProcessingCond(element: ElemType, maxOpResDemand: Real, minOpResDemand: Real) : Boolean =
let opWithHighResDemand : Sequence(Behavior::Action) =
  findOpWithHighResDemand(element, maxOpResDemand) in
let opWithLowResDemand : Sequence(Behavior::Action) =
  findOpWithLowResDemand(element, minOpResDemand) in
if (opWithHighResDemand -> size() <> 0 and opWithLowResDemand ->size() <> 0) then
  opWithHighResDemand -> exists(act1: Behavior::Action |
  opWithLowResDemand -> exists(act2: Behavior::Action |
    belongToTheSameChoice(act1, act2)))
else
  false
endif

def: findOpWithHighResDemand (elemType: ElemType, bound: Real) : Sequence(Behavior::Action) =
  Behavior::Action.allInstances() ->
    select(act: Behavior::Action |
      act.belongs.etName = elemType.etName and
      act.rate.oclsIsTypeOf(Behavior::RateExp) and
      getActionRate(act) >= bound) -> asSequence()
```

V. Cortellessa, M. De Sanctis, A. Di Marco, C. Trubiani: ”Enabling Performance Antipatterns to arise from an ADL-based Software Architecture”, WICSA/ECSA @ Helsinki, Finland, 20-24 August 2012
Bus on Air (BoA)

http://www.busonair.eu
Modeling

AEmilia Architectural specification (boa.aem)

```
ARCHI_TYPE boa{
    const integer ma_num := 5,
    const rate download_rate := 2441.40525,
    const rate upload_rate := 303.17578125,
    const rate balancer_rate_a := 200000000,
    const rate balancer_rate_b := 100000000,
    const rate server_reg_rate := 700000000,
    const rate server_result_rate := 55995,
    const rate data_fetch_rate := 36.585,
    const integer buffer_size := 10
}

ARCHI_ELEM_TYPES

ELEM_TYPE MA_Type(void)

BEHAVIOR

MobileApp(void; void) =
    <generate_best_path_req, inf> .
    <transmit_reg_best_path, inf> .
    <receive_best_path, _> . MobileApp()

INPUT_INTERACTIONS

UNI receive_best_path;
generate_best_path_req

OUTPUT_INTERACTIONS

UNI transmit_reg_best_path
```

V. Cortellessa, M. De Sanctis, A. Di Marco, C. Trubiani: “Enabling Performance Antipatterns to arise from an
ADL-based Software Architecture”, WICSA/ECSA @ Helsinki, Finland, 20-24 August 2012
Analysis

<table>
<thead>
<tr>
<th>Performance Requirements</th>
<th>BoA</th>
</tr>
</thead>
<tbody>
<tr>
<td>$U(DB) &lt; 0.6$</td>
<td>0.99</td>
</tr>
<tr>
<td>$Th(receiveBestPath) &gt; 200$ reqs/sec</td>
<td>36.58 reqs/sec</td>
</tr>
<tr>
<td>$Th(deliverReqBestPath_A) &gt; 100$ reqs/sec</td>
<td>24.39 reqs/sec</td>
</tr>
<tr>
<td>$Th(deliverReqBestPath_B) &gt; 100$ reqs/sec</td>
<td>12.19 reqs/sec</td>
</tr>
<tr>
<td>$RT(receiveBestPath) &lt; 2$ sec</td>
<td>2.73 sec</td>
</tr>
</tbody>
</table>

e.g. the user has to receive the best path with a **RESPONSE TIME** not larger than 2 seconds whereas the performance analysis predicts that the response time is equal to **2.73 seconds**
Refactoring - detecting antipatterns

"Extensive Processing" (EP) 
Antipattern occurrence

Refactoring - solving antipatterns

"Extensive Processing" Antipattern removal

V. Cortellessa, M. De Sanctis, A. Di Marco, C. Trubiani: "Enabling Performance Antipatterns to arise from an ADL-based Software Architecture", WICSA/ECSA @ Helsinki, Finland, 20-24 August 2012
Analysis of the refactored architectures

<table>
<thead>
<tr>
<th>Performance Requirements</th>
<th>Performance Analysis</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>BoA</td>
</tr>
<tr>
<td>U(DB) &lt; 0.6</td>
<td>0.99</td>
</tr>
<tr>
<td>Th(receiveBestPath) &gt; 200 reqs/sec</td>
<td>36.58 reqs/sec</td>
</tr>
<tr>
<td>Th(deliverReqBestPath_A) &gt; 100 reqs/sec</td>
<td>24.39 reqs/sec</td>
</tr>
<tr>
<td>Th(deliverReqBestPath_B) &gt; 100 reqs/sec</td>
<td>12.19 reqs/sec</td>
</tr>
<tr>
<td>RT(receiveBestPath) &lt; 2 sec</td>
<td>2.73 sec</td>
</tr>
</tbody>
</table>

e.g. the removal of the “Pipe and Filter” (P&F) Antipattern allows to fulfill all the requirements
Contributions

- A model-driven approach to detect performance antipatterns in ADL-based software architectures
- Implementation of a tool that automatically detects antipatterns in AEmilia specifications

Future works

- Experimenting the approach on other ADLs (e.g. AADL, EAST-ADL, etc.)
- Validation of the approach on industrial case studies
- Introducing automation for antipatterns solution
Thank you!

For further information please refer to:
http://code.google.com/p/panda-aemilia

Questions

{vittorio.cortellessa, antinisca.dimarco, catia.trubiani}@univaq.it, martinadesanctis@yahoo.it