Software Performance Antipatterns: Modeling and Analysis

Vittorio Cortellessa, Antinisca Di Marco, Catia Trubiani
Computer Science Department, University of L’Aquila, Italy
Credits for this course go to...

Catia Trubiani
Antinisca Di Marco

Romina Eramo
Alfonso Pierantonio
Davide Arcelli

All my undergraduate and graduate students who have lost some of their health on antipatterns
Performance problems

The site has been experiencing some backend performance issues and is currently down for maintenance. We will bring the site back online soon when we can resolve those issues.

OSUOSL Staff
Performance problems

» NASA was forced to delay the launch of a satellite for more than eight months

» The delay was caused because the Flight Operations Segment software had unacceptable response times for developing satellite schedules, and poor performance in analyzing satellite status and telemetry data
Performance problems - how tackled?!

**Modeling**

1970-1990

- **System performance** modeling
  
  (QN, PN, ...)

1990-2000

- **Software performance** since early lifecycle phases

2000-2008

- **Model transformations** for performance model generation

2008-today

- **Performance model evolution** to support **adaptive software** to changes: requirements, context, ...

**Analysis**

1970-today

- **Solve performance models** with (ever more sophisticated!) tools

Bottleneck analysis

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
Software models

UML Use Case Diagram

UML Deployment Diagram

Software Performance Antipatterns: Modeling and Analysis, SFM-12: MDE

ARCHI_TYPE boa (const integer ma_num := 5,
const rate download_rate := 2441.40625,
const rate upload_rate := 305.17578125,
const rate balancer_rate_a := 20000000,
const rate balancer_rate_b := 10000000,
const rate server_reg_rate := 70000000,
const rate server_result_rate := 85995,
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_req_best_path, inf>

INPUT_INTERACTIONS

UNI receive_best_path;
genenerate_best_path_req

OUTPUT_INTERACTIONS

UNI transmit_req_best_path

Architectural specification
Performance

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
Software models vs Performance indices

The interpretation of performance indices is not a trivial task!
What to change in order to improve the software performance?

Requirements:
responseTime(service) \leq 8\text{sec}

Software Architectural Model

Software models vs Performance indices
Software models vs Performance indices

What to change in order to improve the software performance?

- **Structure** (e.g. splitting components)
- **Behavior** (e.g. reducing number of messages)
- **Deployment** (e.g. moving components across hosts)

Models are here intended as instruments to support decisions along the software lifecycle.
Outline

» Problem statement
» (Logic-based) Reasoning on Performance Antipatterns
» Performance Antipatterns in Modeling Languages
  – Unified Modeling Language (UML)
  – A Domain Specific Language (PCM)
  – An Architecture Description Language (AEmilia)
» Advanced MDE techniques
### Problem Statement...

... and different ways to approach it
Software performance process

Modeling

(Annotated) Software Model → Model2Model Transformation → Performance Model

e.g. UML+Marte, Automata, Process Algebras, PCM, ...

e.g. Queueing Networks, Simulation Models, ...

Analysis

Model Solution → Performance Results

Refactoring

PERFORMANCE RESULTS

- Numbers to be interpreted
- Different levels of granularity
- Cross-checking of software system characteristics

Results Interpretation & Feedback Generation

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
## Software Performance Feedback: state-of-the-art

<table>
<thead>
<tr>
<th></th>
<th>Approach</th>
<th>(Annotated) Software Architectural Model</th>
<th>Performance Model</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Antipattern-based</strong></td>
<td>Williams et al. 2002</td>
<td>Software Execution Model</td>
<td>System Execution Model</td>
<td>SPE-ED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parsons et al. 2008</td>
<td>JEE systems</td>
<td>Reconstructed runtime model</td>
<td>PAD</td>
</tr>
<tr>
<td><strong>Rule-based</strong></td>
<td>Barber et al. 2002</td>
<td>Domain Reference Arch.</td>
<td>Simulation Model</td>
<td>RARE / ARCADE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Xu 2010</td>
<td>UML</td>
<td>Layered QN</td>
<td>Perf. Booster</td>
</tr>
<tr>
<td><strong>Design Space Exploration</strong></td>
<td>Simple Criteria</td>
<td>Zheng et al. 2003</td>
<td>Simulation Model</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Ipek et al. 2008</td>
<td>Artificial Neural Network</td>
<td>Simulation Model</td>
<td>-</td>
</tr>
<tr>
<td><strong>Meta-heuristics</strong></td>
<td>Canfora et al. 2005</td>
<td>Workflow Model</td>
<td>Workflow QoS Model</td>
<td>-</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Martens et al. 2010</td>
<td>PCM</td>
<td>Simulation Model</td>
<td>PerOpteryx</td>
</tr>
</tbody>
</table>

"Software Performance Antipatterns: Modeling and Analysis", SFM-12: MDE
» Antipattern-based approaches
   - They make use of antipatterns knowledge to cope with performance issues


... 

Rule-based approaches

- They encapsulate general knowledge on how to improve system performance into executable rules


Design space exploration - simple criteria

- They explore the design space by examining alternatives that can cope with performance flaws


... 

Design space exploration - Metaheuristics

- They make use of evolutionary algorithms that look for design alternatives aimed at improving the system performance


... 

<table>
<thead>
<tr>
<th>Approach</th>
<th>(Annotated) Software Architectural Model</th>
<th>Performance Model</th>
<th>Framework</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Antipatterns-based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Williams et al. 2002</td>
<td>Software Execution Model</td>
<td>System Execution Model</td>
<td>SPE-ED</td>
</tr>
<tr>
<td>Parsons et al. 2008</td>
<td>JEE systems</td>
<td>Reconstructed runtime model</td>
<td>PAD</td>
</tr>
<tr>
<td><strong>Focus of this course</strong></td>
<td><strong>Unified Modeling Language (UML), Palladio Component Model (PCM), AEmilia ADL</strong></td>
<td><strong>Queueing Network,Simulation Model, Markov Chain</strong></td>
<td><strong>Performance Antipatterns and feedback on software Architectures (PANDA)</strong></td>
</tr>
<tr>
<td>Barber et al. 2002</td>
<td>Domain Reference Arch.</td>
<td>Simulation Model</td>
<td>RARE / ARCADE</td>
</tr>
<tr>
<td>Xu 2010</td>
<td>UML</td>
<td>Layered QN</td>
<td>Perf. Booster</td>
</tr>
<tr>
<td><strong>Rule-based</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
Adding antipatterns in the software performance process

Modeling
(Annotated) Software Architectural Model → Model2Model Transformation → Performance Model

Analysis
Model Solution → Performance Results

Refactoring
Performance Antipatterns

Results Interpretation & Feedback Generation

"Software Performance Antipatterns: Modeling and Analysis", SFM-12: MDE
» **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

» **What to avoid and how to solve (performance) problems!**

Performance Antipatterns: why are they complex?


<table>
<thead>
<tr>
<th>Antipattern</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>Occurs when a single class or component either 1) performs all of the work of an application or 2) holds all of the applications data. Either manifestation results in excessive message traffic that can degrade performance.</td>
<td>Refactor the design to distribute intelligence uniformly over the applications top-level classes, and to keep related data and behavior together.</td>
</tr>
</tbody>
</table>
## Performance Antipatterns classification

<table>
<thead>
<tr>
<th>Antipattern</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Single-value</strong></td>
<td></td>
<td></td>
</tr>
<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”</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>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td><strong>Multiple-values</strong></td>
<td></td>
<td></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>

**Single-value vs Multiple-values**

» **Single-value**: performance indices are evaluated in a certain interval, i.e. the mean, max or min values.

![Single-value example](image)

» **Multiple-values**: performance indices are evaluated along the time, i.e. the values trend (or evolution).

![Multiple-values example](image)
Software performance process: introducing **automation**

**Modeling**

(Annotated) Software Architectural Model → Model2Model Transformation

**Analysis**

Performance Model → Model Solution → Performance Results

**Refactoring**

Detection & Solution of Performance Antipatterns

Antipattern-based Rules and Actions

Results Interpretation & Feedback Generation

Problem

Reasoning on PA

PA in Modeling Languages

MDE techniques

"Software Performance Antipatterns: Modeling and Analysis", SFM-12: MDE
» A bird’s-eye look to the problem

1- Representing Antipatterns

2- Detecting Antipatterns

3- Solving Antipatterns
Representing Antipatterns:
What are the software architectural model elements we need for representing antipatterns?
Detecting Antipatterns: How to explore the architectural models to recognize antipattern occurrences?
Solving Antipatterns:
What are the *refactoring actions* that lead the architectural model to remove performance flaws?
(LOGIC-BASED) REASONING ON PERFORMANCE ANTIPATTERNS
Graphical representation of the “Blob” Antipattern

**PROBLEM**: “occurs when a single class or component either 1) performs all of the work of an application or 2) holds all of the applications data. Either manifestation results in excessive message traffic that can degrade performance”
Graphical representation of the "Blob" Antipattern

**SOLUTION:** "Refactor the design to distribute intelligence uniformly over the applications top-level classes, and to keep related data and behavior together"
Graphical representation of the "Concurrent Processing Systems" Antipattern

**PROBLEM:** "Occurs when processing cannot make use of available processors"
Graphical representation of the “Concurrent Processing Systems” Antipattern

**SOLUTION**: “Restructure software or change scheduling algorithms to enable concurrent execution”
Graphical representation of the “Empty Semi Trucks” Antipattern

**PROBLEM:** “Occurs when an excessive number of requests is required to perform a task. It may be due to inefficient interface that, in some cases, implies an inefficient use of available bandwidth (low size messages)”
**SOLUTION**: “The Batching performance pattern combines items into messages to make better use of available bandwidth (in case of low size messages). The Coupling performance pattern, Session Facade design pattern, and Aggregate Entity design pattern provide more efficient interfaces.”

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
First Order Logic-based representation of antipatterns

How to make more “formal” (i.e. machine-processable) the specifications of antipatterns???
Different model elements of antipatterns can be organized into "Views"

**STATIC VIEW**: software resources, relationships among them, ... to model static aspects

**DYNAMIC VIEW**: interactions such as messages between sw resources, ... to model dynamic aspects

**DEPLOYMENT VIEW**: hardware resources, ... to model deployment aspects
» **Basic Idea**: a performance antipattern can be formalized as a logical predicate $LP_{\text{antipatName}}$

» A logical predicate is made of (Static, Dynamic, Deployment) basic predicates, $BP_i$

$$LP_{\text{antipatName}} = BP_1 (\land, \lor) \ldots (\land, \lor) BP_n$$
The Empty Semi Trucks antipattern occurs when the following predicate is true:

\[
\exists s\! w\! E_x \in s\! w\! E, \ S \in S \mid \\
F_{\text{numRemMsgs}}(s\! w\! E_x, S) \geq Th_{\text{maxRemMsgs}} \wedge \\
(F_{\text{maxNetUtil}}(P_{s\! w\! E_x}, s\! w\! E_x) < Th_{\text{minNetUtil}} \vee \\
F_{\text{numRemInst}}(s\! w\! E_x, S) \geq Th_{\text{maxRemInst}})
\]

where $s\! w\! E$ represents the set of $\text{SoftwareEntityInstance}$($s$), and $S$ represents the set of $\text{Service}$($s$).

All the $(s\! w\! E_x, S)$ instances satisfying the predicate must be pointed out to the designer for a deeper analysis.
Auxiliary elements in the formalization process

» **Functions**

> $F_{functionName}$ elaborates information of the model  

(e.g. $F_{numMsgs}$ is a function counting the number of messages sent by an instance of the class/component model element)

» **Thresholds**

> $Th_{thresholdName}$ is a value (*estimated by the software architects*) used to establish the *acceptable range* of values for system features

(e.g. $Th_{maxMsgs}$ is a threshold value representing the upper bound for an acceptable number of messages exchanged among two software instances. It can be estimated, for example, as the average number of all messages sent by all software entities, plus the corresponding variance)
The Concurrent Processing Systems antipattern occurs when the following predicate is true:

\[
\exists P_x, P_y \in \mathbb{P} \mid
\begin{align*}
F_{\text{max}QL}(P_x) & \geq T_{\text{max}QL} \land \\
( F_{\text{max}HWUtil}(P_x, \text{cpu}) & \geq T_{\text{max}CpuUtil} \land \\
F_{\text{max}HWUtil}(P_x, \text{disk}) & \geq T_{\text{max}DiskUtil} ) \lor \\
( F_{\text{max}HWUtil}(P_y, \text{cpu}) & < T_{\text{min}CpuUtil} \land \\
F_{\text{max}HWUtil}(P_y, \text{disk}) & < T_{\text{min}DiskUtil} )
\end{align*}
\]

where \( \mathbb{P} \) represents the set of \textit{ProcesNode}(s).

All the \((P_x, P_y)\) instances satisfying the predicate must be pointed out to the designer for a deeper analysis.

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
The Blob antipattern occurs when the following predicate is true:

\[
\exists \text{swE}_x, \text{swE}_y \in \text{swIE}, S \in \mathcal{S} \mid
\begin{align*}
( \text{F}_{\text{numClientConnects}}(\text{swE}_x) &\geq T_{\text{maxConnects}} ) \lor \\
( \text{F}_{\text{numSupplierConnects}}(\text{swE}_x) &\geq T_{\text{maxConnects}} ) \lor \\
( \text{F}_{\text{numMsgs}}(\text{swE}_x, \text{swE}_y, S) &\geq T_{\text{maxMsgs}} ) \lor \\
( \text{F}_{\text{numMsgs}}(\text{swE}_y, \text{swE}_x, S) &\geq T_{\text{maxMsgs}} ) \\
&\lor
\end{align*}
\]

where \( \text{swIE} \) represents the set of \textit{SoftwareEntityInstance(s)}, and \( \mathcal{S} \) represents the set of \textit{Service(s)}.

All the \((\text{swE}_x, \text{swE}_y, S)\) instances satisfying the predicate must be pointed out to the designer for a deeper analysis.
Each antipattern can be expressed by means of first-order logics

But this is only OUR interpretation of their textual description


<table>
<thead>
<tr>
<th>Antipattern</th>
<th>Formula</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob (or god class/component)</td>
<td>$\exists x, y, s \mid (F_{\text{numClientConnect}}(s,x,y) \lor F_{\text{numConnect}}(x,y) \lor (F_{\text{numMMax}}(s,x,y) \geq \text{Th}<em>{\text{mMMax}}) \lor F</em>{\text{numMMax}}(s,x,y,\text{all}) \lor F_{\text{numMMin}}(s,x,y)))$</td>
</tr>
<tr>
<td>Concurrent Processing Systems</td>
<td>$\exists P, P \in P \mid (F_{\text{mMMax}}(P_x) \geq \text{Th}<em>{\text{mMMax}}) \land (F</em>{\text{mMMin}}(P_x, \text{cpu}) \geq \text{Th}<em>{\text{mMMin}}) \land (F</em>{\text{mMMin}}(P_x, \text{disk}) \geq \text{Th}<em>{\text{mMMin}}) \land (F</em>{\text{mMMin}}(P_x, \text{all}) \geq \text{Th}_{\text{mMMin}})$</td>
</tr>
<tr>
<td>Ubiquitous Processing</td>
<td>$\exists P, P \in P \mid (F_{\text{mMMax}}(P_x) \geq \text{Th}<em>{\text{mMMax}}) \land (F</em>{\text{mMMin}}(P_x, \text{cpu}) \geq \text{Th}<em>{\text{mMMin}}) \land (F</em>{\text{mMMin}}(P_x, \text{disk}) \geq \text{Th}<em>{\text{mMMin}}) \land (F</em>{\text{mMMin}}(P_x, \text{all}) \geq \text{Th}_{\text{mMMin}})$</td>
</tr>
<tr>
<td>Excessive Processing</td>
<td>$\exists P, P \in P \mid (F_{\text{mMMax}}(P_x) &lt; \text{Th}<em>{\text{mMMax}}) \land (F</em>{\text{mMMin}}(P_x, \text{cpu}) &lt; \text{Th}<em>{\text{mMMin}}) \land (F</em>{\text{mMMin}}(P_x, \text{disk}) &lt; \text{Th}<em>{\text{mMMin}}) \land (F</em>{\text{mMMin}}(P_x, \text{all}) &lt; \text{Th}_{\text{mMMin}})$</td>
</tr>
</tbody>
</table>
Key-Question:

Once we have (somehow) “represented” antipatterns, how can we detect them in a software model?
First rough approach

A Java rule-engine application able to parse any XML document compliant to a Schema that we have defined as antipattern vocabulary.

An excerpt of the Java application for the “Concurrent Processing Systems” antipattern
A case study: the “E-commerce System”

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Required Value</th>
<th>Predicted Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT(browseCatalog)</td>
<td>1.2 sec</td>
<td>1.5 sec</td>
</tr>
<tr>
<td>RT(makePurchase)</td>
<td>2 sec</td>
<td>2.77 sec</td>
</tr>
</tbody>
</table>
A preliminary complex step: setting thresholds

<table>
<thead>
<tr>
<th>antipattern</th>
<th>parameter</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>$T_{h_{\text{maxConnect}}}$</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{maxMsgs}}}$</td>
<td>18</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{maxHwUtil}}}$</td>
<td>0.75</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{maxNetUtil}}}$</td>
<td>0.85</td>
</tr>
<tr>
<td>CPS</td>
<td>$T_{h_{\text{maxQueue}}}$</td>
<td>40</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{cpMaxUtil}}}$</td>
<td>0.8</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{diskMaxUtil}}}$</td>
<td>0.7</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{cpuMinUtil}}}$</td>
<td>0.3</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{diskMinUtil}}}$</td>
<td>0.4</td>
</tr>
<tr>
<td>EST</td>
<td>$T_{h_{\text{remMsgs}}}$</td>
<td>12</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{remInst}}}$</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>$T_{h_{\text{minNetUtil}}}$</td>
<td>0.3</td>
</tr>
</tbody>
</table>

- **Design properties (e.g., excessive message traffic)**
- **Performance results (e.g., high network utilization)**

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
Workload: a trouble-making factor
## Performance Antipattern instances in ECS:

<table>
<thead>
<tr>
<th>Antipattern</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blob</td>
<td>libraryController performs most of the work, it generates excessive message traffic towards bookLibrary</td>
<td>Refactor the design to keep related data and behavior together, i.e. delegate some work from libraryController to bookLibrary</td>
</tr>
<tr>
<td>Concurrent Processing Systems</td>
<td>Processing cannot make use of the processor webServerNode</td>
<td>Restructure software or changing scheduling algorithms between processors libraryNode and webServerNode</td>
</tr>
<tr>
<td>Empty Semi Trucks</td>
<td>An excessive number of requests are sent by the userController to perform the task of registering users</td>
<td>Refactor the design combining items into messages to make better use of available bandwidth</td>
</tr>
</tbody>
</table>
An example: detecting the Blob antipattern instance

The \texttt{(libraryController, bookLibrary, browseCatalog)} instance satisfies the \textbf{Blob} predicate, hence it must be pointed out to the designer for a deeper analysis.
And another Key-Question

Once we have (somehow) “detected” antipatterns, which refactoring actions must be taken to remove (some of) them?

First rough approach

First-Order-Logic representation of antipatterns can help: refactoring actions can be (automatically) obtained from negating predicates!
» An example: solving the Blob antipattern instance

The removal of the Blob antipattern gives rise to a new software architectural model (called here as “ECS \ {blob}”)
Performance Analysis of some refactored models

Modeling

- ECS \{est\}
- ECS \{cps\}
- ECS \{blob\}

Unified Modeling Language (UML)

Analysis

Prima-UML → Queueing Network → MVA → Response Time, Utilisation, ...

<table>
<thead>
<tr>
<th>(Annotated) Software Architectural Model</th>
<th>RT(browseCatalog) ≤ 1.2 sec</th>
<th>RT(makePurchase) ≤ 2 sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>ECS</td>
<td>1.5 sec</td>
<td>2.77 sec</td>
</tr>
<tr>
<td>ECS {blob}</td>
<td>1.14 sec</td>
<td>2.18 sec</td>
</tr>
<tr>
<td>ECS {cps}</td>
<td>1.15 sec</td>
<td>1.6 sec</td>
</tr>
<tr>
<td>ECS {est}</td>
<td>1.5 sec</td>
<td>2.24 sec</td>
</tr>
</tbody>
</table>

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
Yet another Key-Question??? Yes!

How to drive the process of antipattern solution?

Who guarantees that the removal of antipatterns will lead to “better” performance?

First approach

A guilt-based one, but still many issues to solve!
PERFORMANCE ANTIPATTERNS IN MODELING LANGUAGES
» Antipattern-based process

(Annotated) Software Architectural Model

Model2Model Transformation

Performance Model

Model Solution

Performance Results

Antipatterns-based Rules and Actions

Detecting Antipatterns (i.e. Results Interpretation)

Solving Antipatterns (i.e. Feedback Generation)

(Annotated) Software Architectural Model Candidate₁

…

(Annotated) Software Architectural Model Candidateᵣ

- Generic Modeling Languages (UML)
- Domain Specific Languages (PCM)
- Architecture Description Languages (Aemilia)
- …
Our experience in UML

Unified Modeling Language (UML) → Prima-UML → Queueing Network → Mean Value Analysis → Response Time, Utilisation, ... → OCL code

OCL engine for the Detection of Antipatterns
**UML model**

<table>
<thead>
<tr>
<th>Service Demand (input parameters)</th>
<th>Utilization (output indices)</th>
</tr>
</thead>
<tbody>
<tr>
<td>webServerNode</td>
<td>2.02 msec</td>
</tr>
<tr>
<td>libraryNode</td>
<td>7.05 msec</td>
</tr>
<tr>
<td>controlNode</td>
<td>3 msec</td>
</tr>
<tr>
<td>dbNode_cpu</td>
<td>15 msec</td>
</tr>
<tr>
<td>dbNode_disk</td>
<td>30 msec</td>
</tr>
</tbody>
</table>

**Requirement:** each hardware resource has not to be used more than 80% under the mean workload of 70 requests/second concurrently in execution in the system.
An example: the Blob antipattern as OCL rule

```
context Model ::
Blob() : Set(Component)

def: allComponents : Set(Component) =
    self.allOwnedElements() ->
    select(oclIsTypeOf(Component))
    .oclAsType(Component) -> asSet()

body : allComponents -> select ( 

  usageRule()
)

interactionRule() -> asSet()

context Component ::

usageRule() : Set()

body : cc -> select ( 

  >= getC(

  getComponent(c)

  )

  )

context Component ::

interactionRule() : Set()

body : self -> select ( 

  if singleDeployNode(self)

  then

  getOwningNode().utilization >= thr

  else

  getUsingComponent(self)

  -> select (getSent()

  )

  -> allLifelines

  )

context Component ::

utilizationRule() : Set(Component)

body : self -> select ( 

  if getCommChannelNode(c).attribute.type

  .include(self)

  then getCommChannelNode(c).attribute

  .utilization >= thr

  )
```
Detecting antipatterns in UML

“Blob” Antipattern occurrence
Solving Antipatterns in UML

Notice that these values have changed due to the load re-distribution.

<table>
<thead>
<tr>
<th>Service Demand (input parameters)</th>
<th>Utilization (output indices)</th>
</tr>
</thead>
<tbody>
<tr>
<td>webServerNode</td>
<td>4.07 msec</td>
</tr>
<tr>
<td>libraryNode</td>
<td>5 msec</td>
</tr>
<tr>
<td>controlNode</td>
<td>3 msec</td>
</tr>
<tr>
<td>dbNode_cpu</td>
<td>15 msec</td>
</tr>
<tr>
<td>dbNode_disk</td>
<td>30 msec</td>
</tr>
</tbody>
</table>

**Requirement:** each hardware resource has not to be used more than 80% under the mean workload of 70 requests/second concurrently in execution in the system.
Our experience in PCM

Modeling

Palladio Component Model (PCM) → PCM2SimuCom

Analysis

Simulation Models → Simulation

Refactoring

Java code

Java engine for the Detection & Solution of antipatterns

Response Time, Utilisation, …
» PCM model

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Required Value</th>
<th>Predicted Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT(system)</td>
<td>10 sec</td>
<td>18.71 sec</td>
</tr>
</tbody>
</table>
An example: the Concurrent Processing Systems antipattern as Java code

```java
public boolean cps(List<ActiveResInfo> list) {
    boolean result = false;
    if (getOverUsedCpu(list).size() != 0 && getUnderUsedCpu(list).size() != 0)
        || (getOverUsedHDD(list).size() != 0 && getUnderUsedHDD(list).size() != 0)) {
        logger.info("The antipattern CONCURRENT PROCESSING SYSTEMS has been detected");
        result = true;
    }
    return result;
}

public List<ActiveResInfo> getOverUsedCpu(List<ActiveResInfo> list) {
    List<ActiveResInfo> result = new ArrayList<ActiveResInfo>(0);

    // @author catia: threshold value for
double thresholdMaxCpu = new Threshold();

    // @author catia: threshold value for
double thresholdCpuQL = new Threshold();

    for (ActiveResInfo el : list) {
        if ((el.type.getEntityManager().equals("CPU") &&
            (el.utilisation > thresholdMaxCpu ||
            el.queueLength > thresholdMaxCpu)) {
            result.add(el);
        }
    }
    return result;
}

public List<ActiveResInfo> getUnderUsedCpu(List<ActiveResInfo> list) {
    List<ActiveResInfo> result = new ArrayList<ActiveResInfo>(0);

    // @author catia: threshold value for min CPU utilisation
double thresholdMinCpu = new Threshold().thresholdMinCpu;

    for (ActiveResInfo el : list) {
        if ((el.type.getEntityManager().equals("CPU") &&
            (el.utilisation < thresholdMinCpu)) {
            result.add(el);
        }
    }
    return result;
}
```
Detecting antipatterns in PCM

“Concurrent Processing Systems” Antipattern occurrence

Problem Reasoning on PA PA in Modeling Languages MDE techniques

“Software Performance Antipatterns: Modeling and Analysis”, SFM-12: MDE
Solving antipatterns in PCM

Concurrent Processing Systems
Antipattern removal

Problem on PA
Reasoning
Languages
MDE
Techniques
Conflict
Antipatterns
SFM-12: MDE
Software Performance Antipatterns
The solution process in PCM

![Graph showing the solution process in PCM](image)

- **Problem**: Reasoning on PA
- **PA in Modeling Languages**
- **MDE techniques**

---

"Software Performance Antipatterns: Modeling and Analysis", SFM-12: MDE
» Our experience in AEmilia

Modeling

AEmilia specification → TwoTowers

Analysis

Markov chain → TwoTowers

Refactoring

OCL code

OCL engine for the Detection of Antipatterns

Response Time, Utilisation, …
AEmilia specification

```plaintext
ARCHI_TYPE box( const integer ma_num := 5,
         const rate download_rate := 2441.40625,
         const rate upload_rate := 305.17578125,
         const rate balancer_rate_a := 20000000,
         const rate balancer_rate_b := 10000000,
         const rate server_req_rate := 70000000,
         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_req_best_path, inf> .
   <receive_best_path, _> . MobileApp()

INPUT_INTERACTIONS

UNI receive_best_path;
   generate_best_path_req

OUTPUT_INTERACTIONS

UNI transmit_req_best_path
```

### Requirement Table

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Required Value</th>
<th>Predicted Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput(system)</td>
<td>15 reqs/sec</td>
<td>12.19 reqs/sec</td>
</tr>
</tbody>
</table>
An example: the Extensive Processing antipattern as OCL code (applied on the Aemilia MM!)

```ocl
-- function for the detection of the Extensive Processing antipattern
def: checkExtensiveProcessingCondition(element: ElemenType,
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: ElemenType, bound: Real) : Sequence(Behavior::Action) =

Behavior::Action.allInstances() ->
  select(act: Behavior::Action | act.belongs.etName = elemType.etName and
  act.root.ooclIsTypeOf(Behavior::RateExp) and
  getActionRate(act) >= bound) -> asSequence()
```
Detecting antipatterns in AEmilia

"Extensive Processing" Antipattern occurrence
Solving antipatterns in AEmilia

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Required Value</th>
<th>Predicted Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Throughput(system)</td>
<td>15 reqs/sec</td>
<td>18.29 reqs/sec</td>
</tr>
</tbody>
</table>
Looking at different modeling notations...

...any modeling language has "antipatterns-concepts"
ADVANCED MODEL-DRIVEN TECHNIQUES...

TO TACKLE THE PROBLEM IN A LANGUAGE-INDEPENDENT WAY
MDE support: metamodel and model transformations

Generic Modeling Language (i.e. UML + Marte profile)

Domain Specific Language (i.e. Palladio Component Model)

Architectural Description Language (i.e. AEmilia)

M2: Metamodels

M1: Models

UML model (Marte annotated)

PCM model

AEmilia specification

...any modeling language has “antipatterns-concepts”
PAML: Perf.Antipatterns Modeling Language
The "Model Elements Specification" sub-MM: SML+
The “Blob” antipattern as PAML-based model
Antipatterns in concrete modeling languages

Problem Reasoning on PA PA in Modeling Languages MDE techniques

Performance Antipatterns

SoftwareEntity

ProcesNode

BasicResDemand

SML+

UML+MARTE

Palladio Component Model

Aemilia ADL

UML Component

PCM Basic Component

ARCHI_ELEM_TYPE

UML Node

PCM Resource Container

...
MDE approach for managing antipatterns

(Annotated) Software Architectural Model → Extractor Engine → SML+ representation of the architectural model → Parsing Engine → Performance Antipatterns as Instances in the Architectural Model → Performance Antipatterns as PAML-based Models → PAML → Antipatterns Modeling → Performance Antipatterns → Results Interpretation & Feedback Generation

Problem
Reasoning on PA
PA in Modeling Languages
MDE techniques
Is UML+Marte expressive enough to specify antipatterns?
Is PCM expressive enough to specify antipatterns?
Is Aemilia expressive enough to specify antipatterns?
### Expressiveness of considered modeling languages

<table>
<thead>
<tr>
<th>Antipattern</th>
<th>UML + Marte profile</th>
<th>Palladio Component Model</th>
<th>AEmilia</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Detectable</td>
<td>Solvable</td>
<td>Detectable</td>
</tr>
<tr>
<td>Blob</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Unbalanced Processing</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concurrent Processing Systems</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Pipe and Filter Architectures</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Extensive Processing</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>Tower of Babel</td>
<td>✔️</td>
<td>✖️</td>
<td>✖️</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>The Ramp</td>
<td>✔️</td>
<td>✖️</td>
<td>✔️</td>
</tr>
<tr>
<td>Traffic Jam</td>
<td>✔️</td>
<td>✔️</td>
<td>✔️</td>
</tr>
<tr>
<td>More is Less</td>
<td>✖️</td>
<td>✖️</td>
<td>✖️</td>
</tr>
</tbody>
</table>

✔️ = fully detectable/solvable  
≈ = partially detectable/solvable  
✖️ = not detectable/solvable
Embedding Antipatterns neutral concept in concrete modeling notations

- Antipatterns are based on a dedicated general purpose language (SML+)
- Weaving models can map the concepts of SML+ into concrete modeling notations (parameter passing)
A transformation $T$ is generated, starting from the WM, between the neutral representation of antipatterns and the concrete modeling language.
DETECTION

A transformation $T'$ is generated, starting from the same WM, to provide an OCL-based executable semantics.
Open Issues - Basics

- Gap textual/formal representations
- Threshold setting

- Uncertainty/incompleteness in:
  > Models, AP specifications, workload, op. profile

- Validation on larger model repositories (real case studies)
Open Issues - Advanced

- Evaluating the framework on:
  > Precision (ratio of actual antipatterns found)
  > Recall (ratio of antipatterns found overall)
- Language-specific antipatterns (other languages to experiment?)
- Antipattern at the model level and the code level
- Combining approaches (antipatterns with metaheuristics)
- Tool construction and integration
Open Issues – Process related

- Legacy constraints
- Conflicting solutions
- Overlapping solutions (composition)

- Combination with other quality attributes (maintenability, reliability, etc...)
We are obviously open to collaborations on any mentioned and non-mentioned point in this domain.

Thank you!

Questions

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