Model-based Feedback for Software Performance Improvement

Dipartimento di Informatica
Università degli studi di L’Aquila

PhD student
Catia Trubiani
catia.trubiani@univaq.it

Advisor
Vittorio Cortellessa
vittorio.cortellessa@univaq.it

Roadmap

» Motivation
» Software Performance Analysis Process
» Our approaches
» Related work
» Ongoing and future work
**Motivation**

» What to change to improve the software design?

**Software Performance Analysis Process**

1. Model2Model Transformation
2. Performance Model
3. Model Solution
4. Performance Indices
5. Performance Antipatterns
6. Results Interpretation and Feedback Generation

» Performance Indices complexity
  - Numbers to be interpreted
  - Different levels of granularity
  - Cross-checking of software system characteristics
**A corresponding “Process”**

1. **Patient** → Health Transformation → **Sick Patient** → Sickness Analysis → Symptoms
2. **Physician correspondances**
   - Patient → Model
   - Sick → Performance
   - Indices → Symptoms

(Performance) Antipatterns


  - **Look at negative features of a software system:**
    - The definition includes common mistakes (i.e. “Bad practice”) in software development as well as their solutions
    - Conceptually similar to “Design Patterns”: recurring solutions to common design problems

- **What to avoid and how to solve (performance) problems!**
Performance Antipatterns


<table>
<thead>
<tr>
<th>Antipattern</th>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Snapshot-based</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Concurrent Processing Systems</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>&quot;Pipe and Filter&quot; Architectures</td>
<td>The slowest filter in a &quot;pipe and filter&quot; 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><strong>Monitoring-based</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>One-Lone Bridge</td>
<td>At a point in execution where only one, or a few processes may continue to execute.</td>
<td>To alleviate the congestion, use the Shared Resources Principle to minimize conflicts.</td>
</tr>
</tbody>
</table>

Snapshot vs Monitoring

**Snapshot:** average values in a small interval of time

\[
\Delta \quad \Delta \quad \Delta \quad \Delta
\]

time

**Monitoring:** multiple intervals of time (i.e., a set of several snapshots)

\[
\Delta \quad 2\Delta \quad \ldots \quad n\Delta
\]

time
Snapshot-based example

Current Processing Systems

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing cannot make use of available processors.</td>
<td>Restructure software or change scheduling algorithms to enable concurrent execution.</td>
</tr>
</tbody>
</table>

One-Lane Bridge

<table>
<thead>
<tr>
<th>Problem</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>At a point in execution where only one, or a few, processes may continue to execute.</td>
<td>To alleviate the congestion, use the Shared Resources Principle to minimize conflicts.</td>
</tr>
</tbody>
</table>
Our Approach: Conceptual Framework

- **Main activities**

  1. Specifying Antipatterns
  2. Detecting Antipatterns
  3. Solving Antipatterns

Contributions

- **(1) Specifying Antipatterns**
  - Structuring system elements
  - Formalizing antipatterns as logical predicates on these elements
  - Model-driven techniques: defining a metamodel for antipatterns
Problem: software developers’ world (vocabulary) is intrinsically distant from performance analysts’ one.

What are the system elements needed for specifying performance antipatterns?

Different system elements of antipatterns can be organized into “Views”

Projection on system elements

Some proposals

» Key Idea: Neutral structured data

» Parametric notations that can be instantiated with any concrete modeling notation (e.g. UML+Marte, SPA, Aemilia, ADLs).

» Notation-independent approach
  - A first proposal: XML-based approach
  - A second proposal: Model-driven approach
A first proposal: XML-based

Neutral structured data: an XML schema

Static View details
**Client/Supplier Paradigm**

» Flatten relations types

![Diagram of Client/Supplier Paradigm]

- **Class A** (client)
- **Class B** (supplier)
- **Whole** (client)
- **Part** (supplier)
- **Parent**
- **Child**

**A first proposal: XML-based**

» Dynamic View details

![Diagram of Dynamic View details]

- **DynamicView**
- **Attributes**
- **Behavior**
- **Messages**
- **Entity**
- **Operator**

**Progetto PRIN PaCo (Performability-aware Computing)**
Lucca, 25-26 giugno 2009
A first proposal: XML-based

Deployment View details

Formalizing antipatterns

Basic Idea: a performance antipattern is a logical predicate $L_{\text{antipatName}}$

A logical predicate is built with views predicates: StaticView predicate $S_{\text{antipatName}}$, DynamicView predicate $D_{\text{antipatName}}$, DeploymentView $D_{\text{antipatName}}$

$$L_{\text{antipatName}} = S_{\text{antipatName}} \circ D_{\text{antipatName}} \circ D_{\text{antipatName}}$$

Each (static, dynamic, deployment)View predicate is built with a set of basic predicates $B_{\text{p}}$

$$*_{\text{anti patName}} = *_{B_{1}}(\text{null}, \text{null}) \ldots (\text{null}, \text{null}) *_{B_{n}}$$
Formalizing antipatterns

- How to define a basic predicate $BP_i$?

- Supporting elements
  - Auxiliary Functions $F_{\text{func}}$
  - Thresholds $Th_{\text{threshold}}$
    - System monitoring
    - Heuristic evaluations

Formalizing “Blob” antipattern

- An example: how to formalize the performance antipattern “Blob”

<table>
<thead>
<tr>
<th>Antipattern</th>
<th>Problem</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>
</tr>
</tbody>
</table>
Formalizing “Blob” antipattern

- “Occurs when a single class or component (i.e., a software entity) either 1) performs all of the work of an application or 2) holds all of the applications data.”

\[ \text{SVP}_{blob} \equiv \text{SBP}_1 \]

Formalizing “Blob” antipattern

- “Either manifestation results in excessive message traffic that can degrade performance.”

\[ \text{DyVP}_{blob} \equiv \text{DyBP}_1 \]
Formalizing "Blob" antipattern

- \( \text{DeBP}_1: \quad F_{\text{maxHwUtil}}(P_E) \geq \text{Th}_{\text{HwUtil}} \)
- \( \text{DeBP}_2: \quad F_{\text{maxNetUtil}}(P_E) \geq \text{Th}_{\text{NetUtil}} \)

\( \text{DeVP}_{\text{blob}} \equiv \text{DeBP}_1 \land \text{DeBP}_2 \)

Formalizing "Blob" antipattern

- Blob problem formalization

\( \text{LP}_{\text{blob}} \equiv \text{SVP}_{\text{blob}} \land \text{DyVP}_{\text{blob}} \land \text{DeVP}_{\text{blob}} \)

\( \text{SVP}_{\text{blob}} \equiv \text{SBP}_1 \)
\( \text{DyVP}_{\text{blob}} \equiv \text{DeBP}_1 \)
\( \text{DeVP}_{\text{blob}} \equiv \text{DeBP}_1 \land \text{DeBP}_2 \)

\( \exists E \mid F_{\text{numConnections}}(E) \geq \text{Th}_{\text{Connect}} \land F_{\text{numMsgs}}(E) \geq \text{Th}_{\text{Msgs}} \land (F_{\text{maxHwUtil}}(P_E) \geq \text{Th}_{\text{HwUtil}} \land F_{\text{maxNetUtil}}(P_E) \geq \text{Th}_{\text{NetUtil}}) \)
A second proposal: Model-driven

» Defining a metamodel for antipatterns

- APML → AntiPattern Modeling Language
  - SML → Software Modeling Language
  - SML* → An enrichment of SML
  - RML → Refactoring Modeling Language

Metamodel for antipatterns

» An excerpt of APML
Metamodel for antipatterns

» Software Modeling Language SML+

- Static View
- Dynamic View
- Deployment View

Progetto PRIN PaCo (Performability-aware Computing)
Lucca, 25-26 giugno 2009
Metamodel for antipatterns

- Dynamic View of SML+

Metamodel for antipatterns

- Deployment View of SML+
Modeling with APML

An example: how to model the performance antipattern "Blob"

Our metamodel in concrete notations

Weaving Models to drive correspondences between our SML+ and a concrete modeling notation (e.g. UML + Marte).
Antipatterns in concrete notations

- High-order Transformations to drive correspondences between the antipattern model in our SML+ and a concrete modeling notation (e.g. UML + Marte).

Contributions

- (2) Detecting Antipatterns
  - Most promising antipatterns
  - Model-driven techniques: translating the occurrence of antipatterns with OCL expressions
Most promising antipatterns

Which ones and in what order?

» How much a Performance Antipattern $PA_i$ is important for the Violated Requirement $R_j$?
**A first proposal: XML-based**

Java Engine for XML parsing

```java
import org.w3c.dom.*;
...
Document doc = newDocumentBuilder().parse(new File("swSystem.xml"));
NodeList listProcesNodes =
    doc.getElementsByTagName("procesNode");
System.out.println("Total number of processors: "+
    listProcesNodes.getLength());
...
```

**A second proposal: Model-driven**

Translating the occurrence of antipatterns with OCL expressions.
Deciding which antipatterns to solve and in what order

- Planning a simultaneous combination of antipatterns solutions

- Model-driven techniques: solving antipatterns with difference models

» Requirement problems

- Functional requirement
  > Legacy components cannot be split or re-deployed
- Non-functional requirement
  > Budget limitations

» Coherency problems

- Incoherences among antipattern solutions
**Discussion**

- Further issues
  - What happens if antipatterns cannot be solved due to (functional/non-functional) requirements?
  - What happens if feedback cannot be applied due to incoherences between antipattern solutions?
  - Which is the sequential order of antipatterns rules and how to better drive the refactoring?
  - An antipattern solution is able to introduce an other antipattern?

**Simultaneous solutions**

- **Move**: application of a simultaneous set of antipattern solutions

  - Observed values on New Candidate
  - New Software System Model
  - New Candidate Software System Model
  - Candidate Moves List
  - Ranked Antipattern List
  - Acquired Knowledge

- [1] proceed with the next move
- [2] learning
- [3] choose the best candidate up to now
A first proposal: XML-based

» **Basic Idea:** solving a performance antipattern by the negation of the logical predicate \( \neg LP_{\text{antipatName}} \)

» The negative logical predicate is built with the negation of the views predicates

\[
\neg LP_{\text{antipatName}} = \neg SVP_{\text{antipatName}} \land \neg DyVP_{\text{antipatName}} \land \neg DeVP_{\text{antipatName}}
\]

» Each (static, dynamic, deployment) View predicate has to be negated

\[
\neg ^*VP_{\text{antipatName}} = \neg (*BP_1, 0, 0) \land \ldots \land (0, 0, *BP_n)
\]

A first proposal: XML-based

» **An example:** "Blob" problem formalization

\[
LP_{\text{Blob}} \equiv SVP_{\text{Blob}} \land DyVP_{\text{Blob}} \land DeVP_{\text{Blob}}
\]

\[
SVP_{\text{Blob}} \equiv SBP_1
\]

\[
DyVP_{\text{Blob}} \equiv DyBP_1
\]

\[
DeVP_{\text{Blob}} \equiv DeBP_1 \land DeBP_2
\]

\[
\forall E \mid F_{\text{numConnects}}(E) \geq \text{Th}_{\text{connect}} \land F_{\text{numMsgs}}(E) \geq \text{Th}_{\text{msgs}} \land
(F_{\text{maxHWUtil}}(P_E) \geq \text{Th}_{\text{HWUtil}} \land F_{\text{maxNetUtil}}(P_E) \geq \text{Th}_{\text{NetUtil}})
\]
A first proposal: XML-based

An example: “Blob” solution formalization

\[ \neg \text{LP}_{\text{blob}} = \neg (\text{SVP}_{\text{blob}} \land \text{DyVP}_{\text{blob}} \land \text{DeVP}_{\text{blob}}) \]

\[ \neg \text{SVP}_{\text{blob}} \equiv \neg \text{SBP}_1 \]

\[ \neg \text{DyVP}_{\text{blob}} \equiv \neg \text{DBP}_1 \]

\[ \neg \text{DeVP}_{\text{blob}} \equiv \neg \text{DBP}_1 \land \neg \text{DBP}_2 \]

\[ \neg \text{E} \mid \neg \text{F}_{\text{numConnects}}(E) \land \text{F}_{\text{connect}}(E) \land \neg \text{F}_{\text{numMags}}(E) \land \text{F}_{\text{mags}}(E) \land \neg \text{F}_{\text{mmaxHwUtil}}(P_E) \land \neg \text{F}_{\text{mmaxNetUtil}}(P_E) \land \neg \text{F}_{\text{mnetUtil}}(E) \]

A second proposal: Model-driven

Solving antipatterns with difference models.

Architectural Model

Solution₁ \quad Solution₂ \quad \ldots \quad Solutionₙ
Related Work

  - Informal interpretation matrices from the analysis of Layered Queueing Networks (LQNs)

  - Performance antipattern detection (PAD) tool for Enterprise Java Bean (EJB) applications

  - Analysis of LQNs performance model for bottlenecks and long paths

Ongoing work

» Validate the scope of the whole approach across languages, to assess the independence of any concrete notation.

[Diagram showing relationships between APML, UML-Marte, Stochastic Process Algebras, and Architecture Description Languages]
Future work

- Real case studies to study the **usability** and the **scalability** of the approach.