Modeling and detection of
Performance Antipatterns in UML

PhD student
Catia Trubiani
catia.trubiani@univaq.it

Advisor
Vittorio Cortellessa
vittorio.cortellessa@univaq.it

Progetto PRIN PaCo (Performability-aware Computing)
L’Aquila, 02-03 marzo 2010

Roadmap

» Motivation
» Problem statement
» UML context
  - Our approach
  - A case study
» Ongoing and future works
What to change to improve the software design?

Architectural Model

Requirements:
- responseTime = 8sec

- Model Solution
- Performance Indices

Model2Model Transformation

Performance Antipatterns

Results Interpretation and Feedback Generation

Performance Model

Analysis

Refactoring
(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!

Our approach in the UML context

- UML and Marte Profile
- PRIMA-UML
- Queueing Network
- Mean Value Analysis
- Utilization, RT...
- Detection and solution of antipatterns
- OCL rules

V. Cortellessa, A. Di Marco, R. Eramo, A. Pierantonio, C. Trubiani
Performance Antipatterns


<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>

Reasoning on the "Blob" problem

» 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.”

1. centralized “Blob”
Reasoning on the "Blob" problem

» 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."

OCL query to detect the "Blob" antipattern

> Each component in the defined context of the model is checked by means of the following rules in order to identify candidate Blob instances.

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

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

body: allComponents.usageRule()
  .interactionRule()
  .utilizationRule() -> asSet()
```
“Blob” - OCL usage rule

» "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"

» Usage Rule: in a Component/Class Diagram a complex controller component/class is surrounded by other components/classes through many usage dependencies.

```plaintext
context Component ::
  usageRule() : Set(Component)

body : cc -> select(oclAsType(Usage) -> size())
      >= getComponentsUsageSize() / getComponentsSize()
```

“Blob” - OCL interaction rule

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

» Interaction Rule: in a Sequence Diagram there are lifelines that generate excessive message traffic (i.e. higher than the average number of messages sent by all lifelines).

```plaintext
context Component ::
  interactionRule() : Set(Component)

def: allLifelines : Set(Lifeline) =
  self.allOwnedElements()
    ->select(oclAsTypeOf(Lifeline))
    .oclAllType(Lifeline) ->asSet()

body : self ->select (getAllComponentLifelines
                  ->select (getSentMessages->size())
                  > allLifelines.getSentMessages->size())
      / allLifelines->size())
```
"Blob" - OCL utilization rule

- "Either manifestation results in excessive message traffic that can degrade performance"
- **Utilization Rule:** device(s) utilization could be critical.

```plaintext
context Component ::
    utilizationRule(): Set(Component)
body: self -> select{
    if singleDeployNode(self) then
        getRunningNode().utilization >= thr;
    else
        getUsingComponent(self); 
        -> iterate(c: Component; result: boolean)
        if getComChannelNode(c).attribute.type 
        .include(self) then
            getComChannelNode(c).attribute 
            .utilization >=thr;
    }
```

Reasoning on the "Blob" solution

- **SOLUTION:** "Refactor the design to distribute intelligence uniformly over the applications top-level classes, and to keep related data and behavior together."

![Diagram of a distributed system with libraryController, bookLibrary, and movieLibrary classes. The diagram shows connectivity and management relationships, including manageCatalog. 1. centralized "Blob"]
Reasoning on the “Blob” solution

» SOLUTION: “Refactor the design to distribute intelligence uniformly over the applications top-level classes, and to keep related data and behavior together.”

A case study in the UML context

» Modeling the Electronic Commerce System (ECS)
**ECS - Performance Analysis**

**Performance Model: Queueing Networks**

- **input parameters**
  - 

**Performance Indices: e.g. Response Time (RT).**

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Required Value</th>
<th>Observed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT (browseCatalog)</td>
<td>1.5 sec</td>
<td>1.61 sec</td>
</tr>
</tbody>
</table>
ECS - Detecting antipatterns

<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.</td>
<td>Refactor the design to keep related data and behavior together. Delegate some work from libraryController to bookLibrary.</td>
</tr>
<tr>
<td>Empty Semi Trucks</td>
<td>An excessive number of requests is required to perform the task of registering new users.</td>
<td>Refactor the design combining items into messages to make better use of available bandwidth.</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>
</tbody>
</table>

ECS - Refactoring (1/2)

» Solving the “Blob” antipattern

UML Component Diagram
ECS - Refactoring (2/2)

» Solving the “Blob” antipattern

UML Sequence Diagram

ECS\{Blob\} system: performance results

» Performance Analysis of the ECS\{Blob\} system

<table>
<thead>
<tr>
<th>Requirement</th>
<th>Required Value</th>
<th>ECS Observed Value</th>
<th>ECS{Blob} Observed Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RT(browseCatalog)</td>
<td>1.5 sec</td>
<td>1.61 sec</td>
<td>1.44 sec</td>
</tr>
</tbody>
</table>
Ongoing works: a Framework

Main activities

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

Specifying Antipatterns

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

Goal: Antipatterns across different languages

Validate the scope of the whole approach to assess the independence of any concrete notation.
Our metamodel in concrete notations

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

![Diagram](image)

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).

![Diagram](image)
Detecting antipatterns

» Translating the occurrence of antipatterns with OCL expressions.

Future works: solving Antipatterns

» Apply concepts of difference models.
## Open issues

- **Requirements issues**
  - Functional requirement
    - Legacy components cannot be split or re-deployed
  - Non-Functional requirement
    - Budget limitations

- **Coherency issues**
  - Incoherences among antipattern solutions

- **Maintenance issues**
  - What happens if the design and the architectural changes are performed at run-time (e.g. pervasive systems)? How do the performance antipatterns change across the run-time reconfigurations of the system?

- **Further issues**
  - Can an antipattern solution introduce another antipattern? How do the workload and the operational profile affect the antipatterns identified?