dipartimentoinformatica
Università degli Studi dell'Aquila

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

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

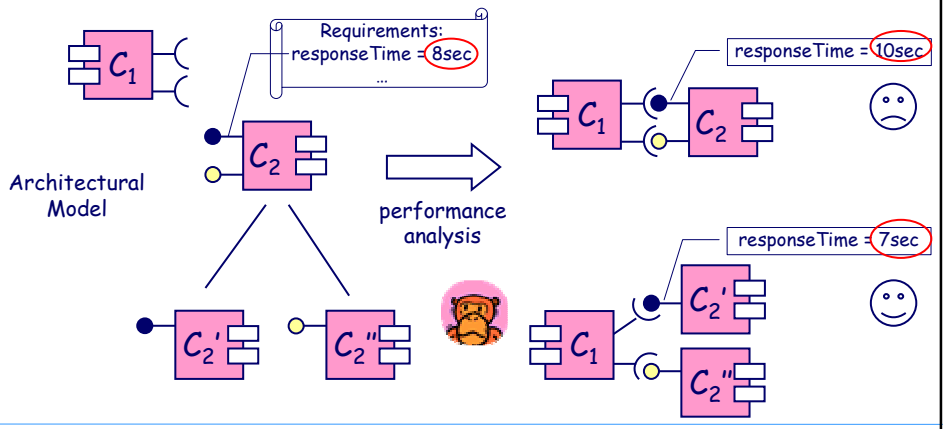
dipartimentoinformatica
Università degli Studi dell'Aquila

Roadmap

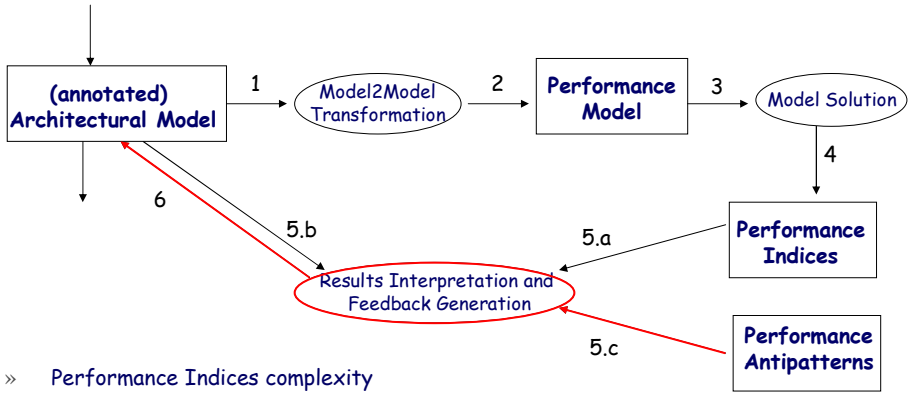
- Motivation
- Software Performance Analysis Process
- Our approaches
- Related work
- Ongoing and future work

Motivation

» What to change to improve the software design?

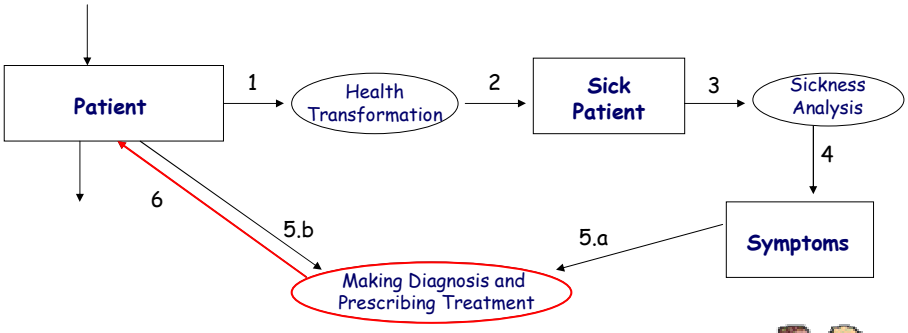


Software Performance Analysis Process



» Performance Indices complexity
- Numbers to be interpreted
- Different levels of granularity
- Cross-checking of software system characteristics

A corresponding "Process"



- Physician correspondances
  - Patient -> Model
  - Sick -> Performance
  - Indices -> Symptoms

(Performance) Antipatterns

- W.J.Brown, R.C. Malveau, H.W. Mc Cornich III, and T.J. Mowbray. "Antipatterns: Refactoring Software, Architectures, and Project in Crisis", 1998.

  -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

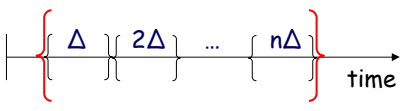
» C. U. Smith and L. G.Williams. "More new software performance antipatterns: Even more ways to shoot yourself in the foot", 2003

	Antipattern		Problem	Solution
Snapshot-based	Unbalanced Processing	Concurrent Processing Systems	Processing cannot make use of available processors.	Restructure software or change scheduling algorithms to enable concurrent execution.
		"Pipe and Filter" Architectures	The slowest filter in a "pipe and filter" architecture causes the system to have unacceptable throughput.	Break large filters into more stages and combine very small ones to reduce overhead.
		Extensive Processing	Extensive processing in general impedes overall response time.	Move extensive processing so that it doesn't impede high traffic or more important work.
	...		...	...
Monitoring-based	One-Lane Bridge		At a point in execution where only one, or a few, processes may continue to execute.	To alleviate the congestion, use the Shared Resources Principle to minimize conflicts.

Snapshot vs Monitoring

» Snapshot: average values in a small interval of time

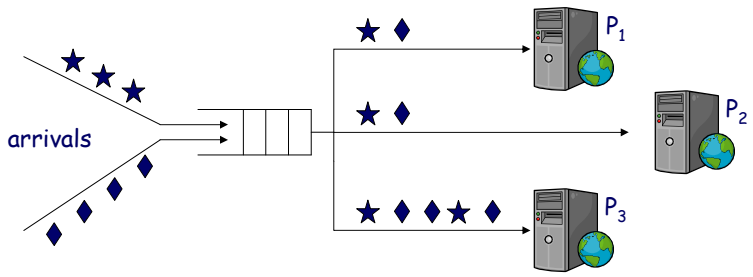
» Monitoring: multiple intervals of time (i.e., a set of several snapshots)



Snapshot-based example

» Cuncurrent Processing Systems

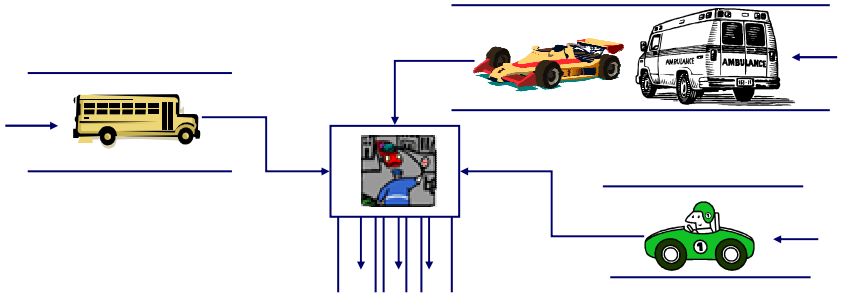
Problem	Solution
Processing cannot make use of available processors.	Restructure software or change scheduling algorithms to enable concurrent execution.



Monitoring-based example

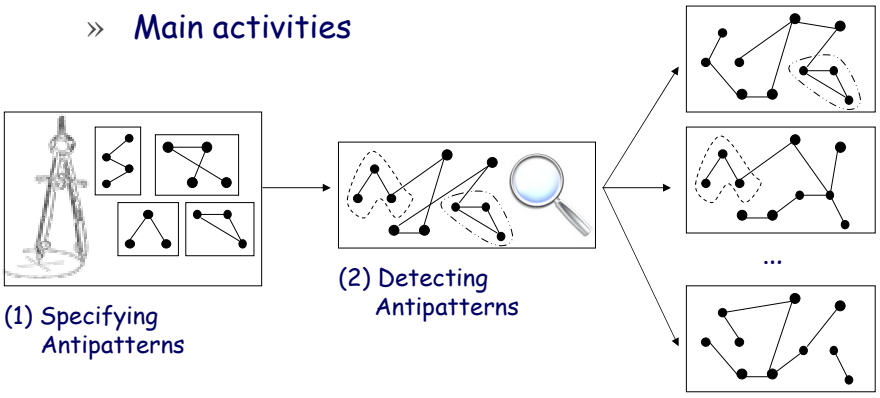
» One-Lane Bridge

Problem	Solution
At a point in execution where only one, or a few, processes may continue to execute.	To alleviate the congestion, use the Shared Resources Principle to minimize conflicts.



Our Approach: Conceptual Framework

» Main activities



(1) Specifying Antipatterns

(2) Detecting Antipatterns

(3) Solving Antipatterns

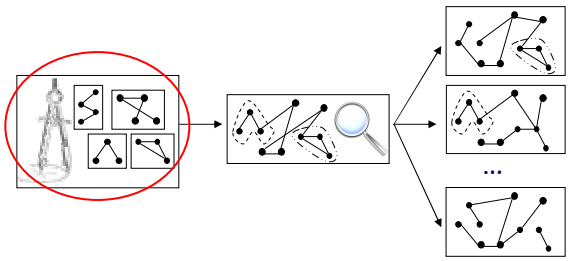
V.Cortellessa, A.Di Marco, R.Eramo, A. Pierantonio, C.Trubiani "Approaching the model-driven generation of feedback to remove software performance flaws", to appear in the proceedings of Euromicro SEAA 2009.

Contributions

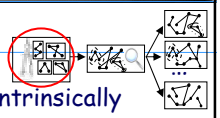
» (1) Specifying Antipatterns

- Structuring system elements
- Formalizing antipatterns as logical predicates on these elements

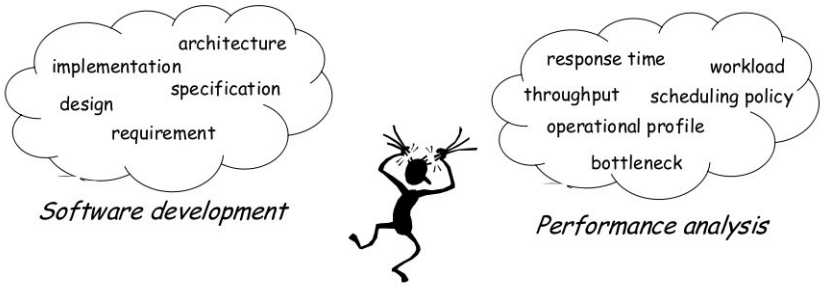
- Model-driven techniques: defining a metamodel for antipatterns



Structuring system elements

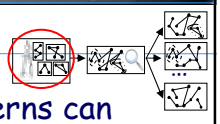


» Problem: software developers' world (vocabulary) is intrinsically distant from performance analysts' one.

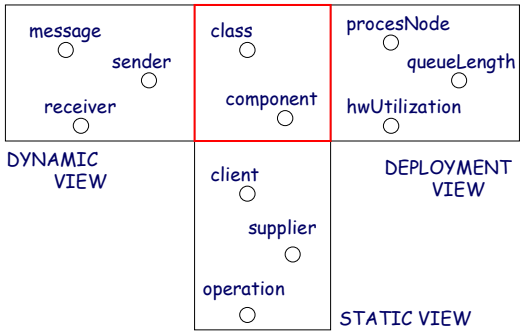


» What are the system elements needed for specifying performance antipatterns?

Structuring system elements

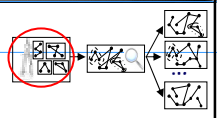


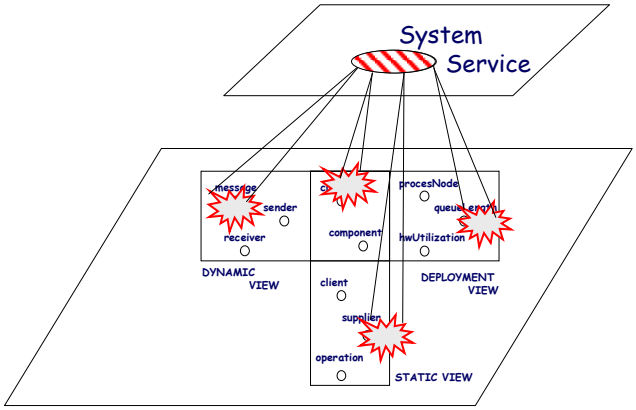
» Different system elements of antipatterns can be organized into "Views"



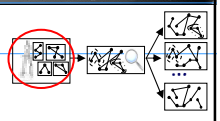
C.M.Woodside. "A Three-View Model for Performance Engineering of concurrent software", 1995.

Projection on system elements





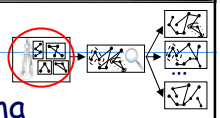
Some proposals



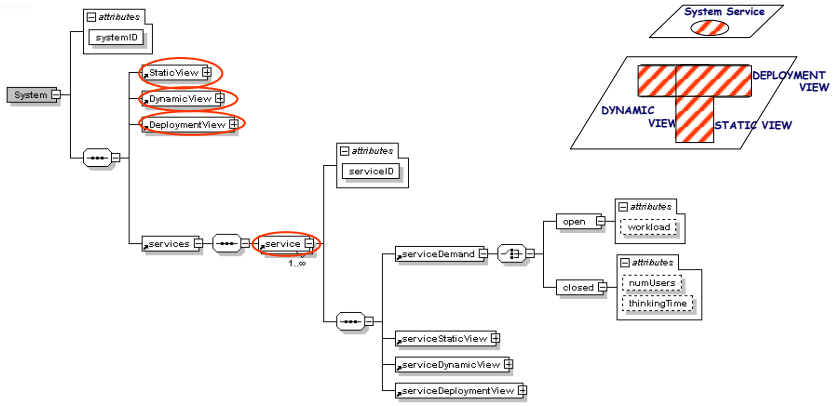
- Key Idea: **Neutral** structured data
- **Parametric** notations that can be instantiated with any concrete modeling notation (e.g. UML+Marte, SPA, Aemilia, ADLs).
- **Notation-independent** approach
  - A first proposal: XML based approach
  - A second proposal: Model driven approach

dipartimentoinformatica
Università degli Studi dell'Aquila

A first proposal : XML-based

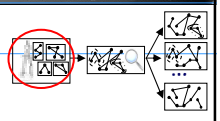


» **Neutral** structured data: an XML schema

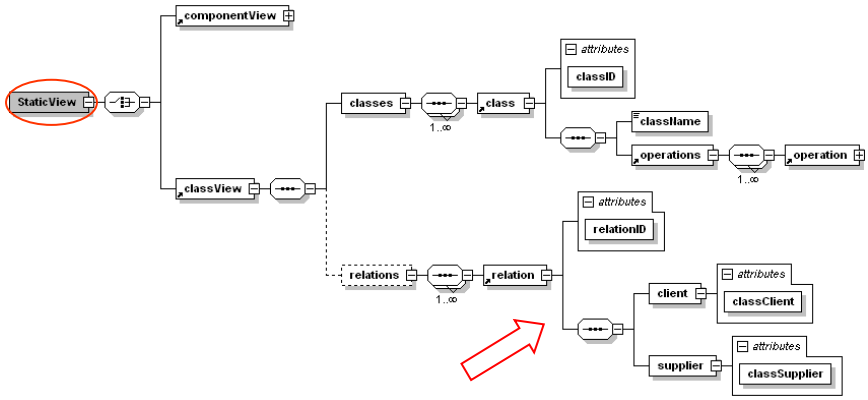


dipartimentoinformatica
Università degli Studi dell'Aquila

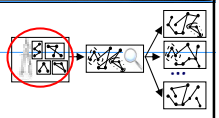
A first proposal : XML-based



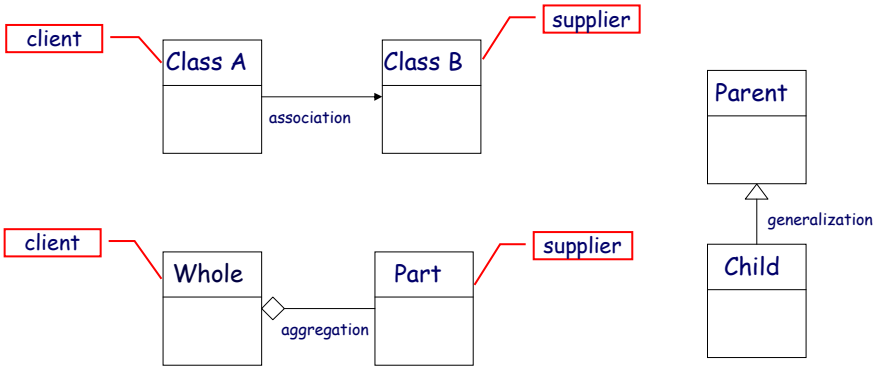
» Static View details



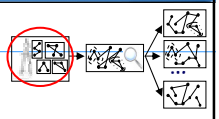
Client/Supplier Paradigm



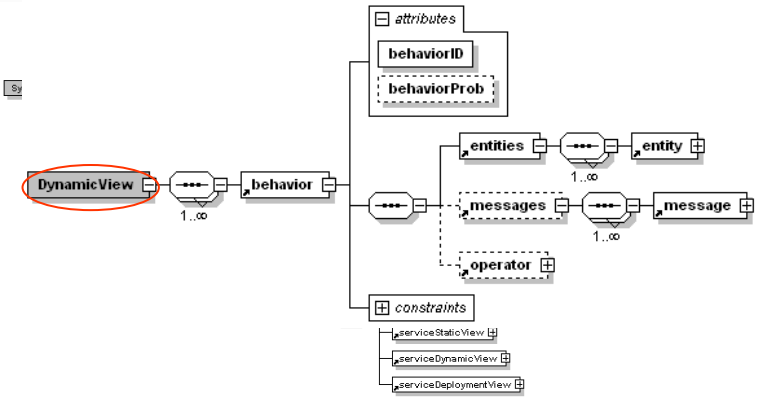
» Flatten relations types



A first proposal : XML-based

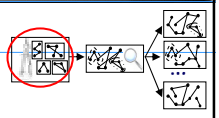


» Dynamic View details

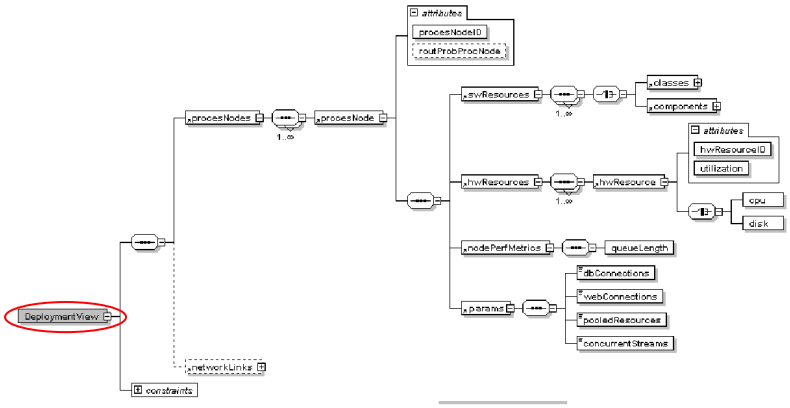


dipartimentoinformatica
Università degli Studi dell'Aquila

A first proposal : XML-based

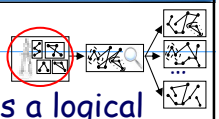


» Deployment View details



dipartimentoinformatica
Università degli Studi dell'Aquila

Formalizing antipatterns



» Basic Idea: a performance antipattern is a logical predicate $LP_{antipatName}$

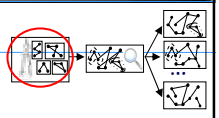
» A logical predicate is built with views predicates: StaticView predicate $SP_{antipatName}$, DynamicView predicate $DyP_{antipatName}$, DeploymentView $DeP_{antipatName}$

$LP_{antipatName} = SVP_{antipatName} \square DyVP_{antipatName} \square DeVP_{antipatName}$

» Each (static, dynamic, deployment)View predicate is built with a set of basic predicates BP_i

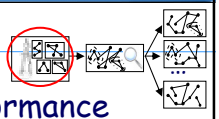
$*VP_{antipatName} = *BP_1 (\square, \square) \ldots (\square, \square) *BP_n$

Formalizing antipatterns



- How to define a basic predicate BP_i?
- Supporting elements
  - Auxiliary Functions F_{func}
  - Thresholds $Th_{threshold}$
    - > System monitoring
    - > Heuristic evaluations

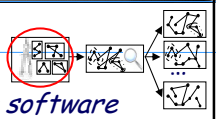
Formalizing "Blob"antipattern



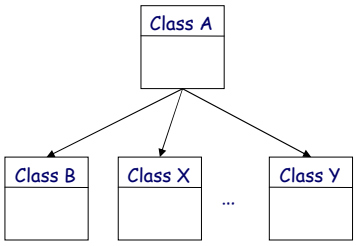
- An example: how to formalize the performance antipattern "Blob"

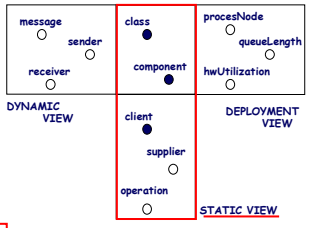
Antipattern	Problem
Blob	Occurs when a single class or component either 1) performs all of the work of an application or 2) holds all of the applications data. Either manifestation results in excessive message traffic that can degrade performance.

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"*





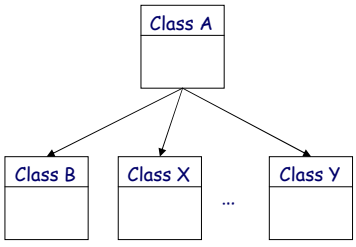
- SBP_1: ∃E | $F_{numConnects}(E) \geq Th_{connect}$

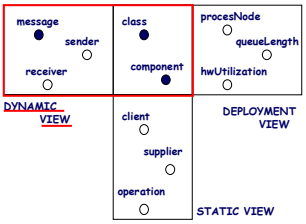
auxiliary function — threshold

$SVP_{blob} \equiv SBP_1$

Formalizing "Blob"antipattern

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





- $DyBP_1$: $F_{numMsgs}(E) \geq Th_{msgs}$

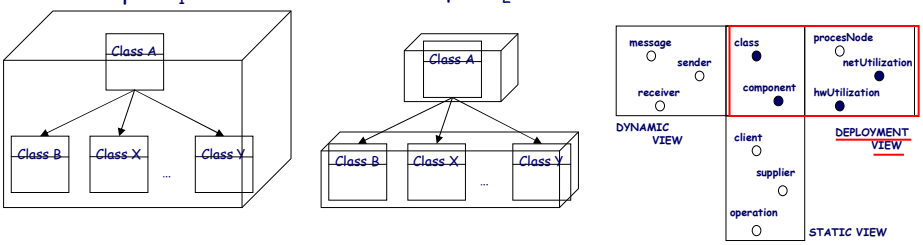
auxiliary function — threshold

$DyVP_{blob} \equiv DyBP_1$

Formalizing "Blob"antipattern

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

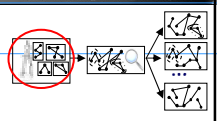
option$_1$ option$_2$



- $DeBP_1$: $F_{maxHwUtil}(P_E) \geq Th_{hwUtil}$
- $DeBP_2$: $F_{maxNetUtil}(P_E) \geq Th_{netUtil}$

$DeVP_{blob} \equiv DeBP_1 \wedge DeBP_2$

Formalizing "Blob" antipattern



» Blob problem formalization

$LP_{blob} = SVP_{blob} \wedge DyVP_{blob} \wedge DeVP_{blob}$

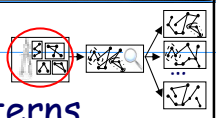
$SVP_{blob} \equiv SBP_1$

$DyVP_{blob} \equiv DyBP_1$

$DeVP_{blob} \equiv DeBP_1 \wedge DeBP_2$

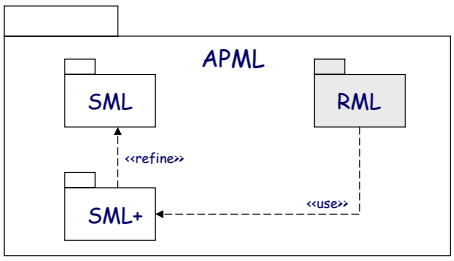
$\Rightarrow \exists E \mid F_{numConnects}(E) \geq Th_{connect} \wedge F_{numMsgs}(E) \geq Th_{msgs} \wedge$
$(F_{maxHwUtil}(P_E) \geq Th_{hwUtil} \wedge F_{maxNetUtil}(P_E) \geq Th_{netUtil})$

A second proposal : Model-driven

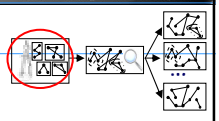


» Defining a metamodel for antipatterns

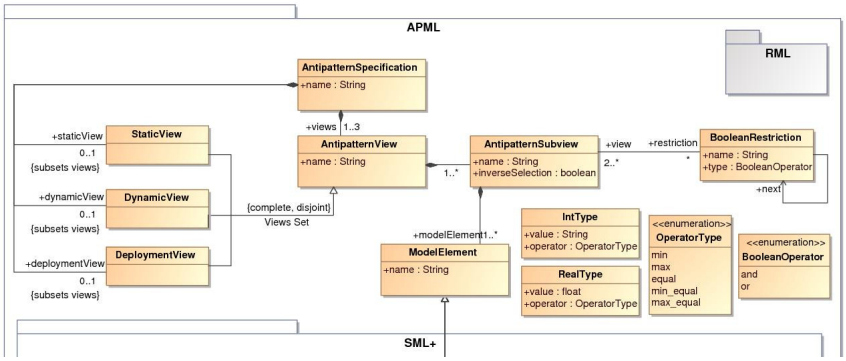
- APML -> AntiPattern Modeling Language
  - > SML -> Software Modeling Language
  - > SML+ -> An enrichment of SML
  - > RML -> Refactoring Modeling Language



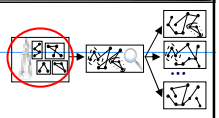
Metamodel for antipatterns



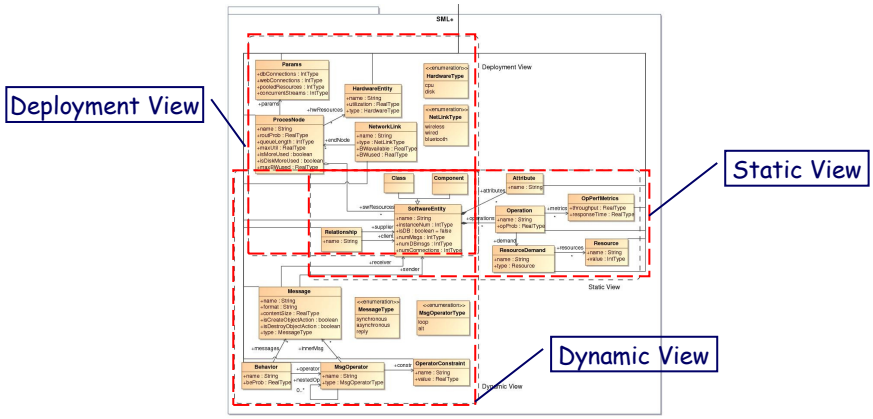
» An excerpt of APML



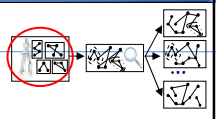
Metamodel for antipatterns



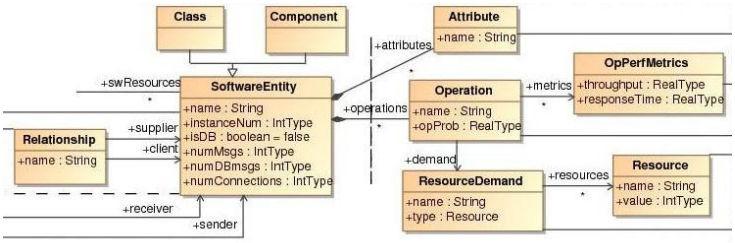
» Software Modeling Language SML+



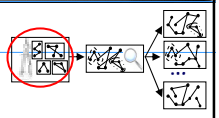
Metamodel for antipatterns



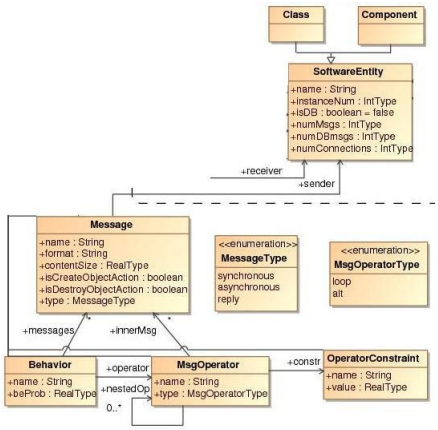
» Static View of SML+



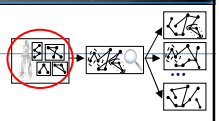
Metamodel for antipatterns



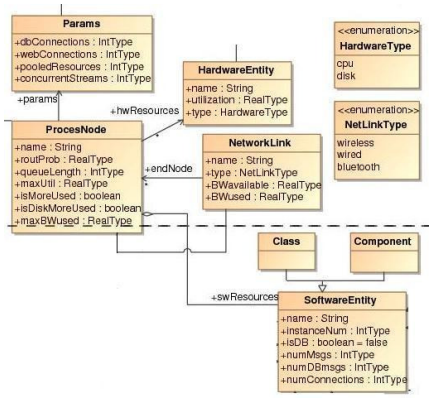
» Dynamic View of SML+



Metamodel for antipatterns

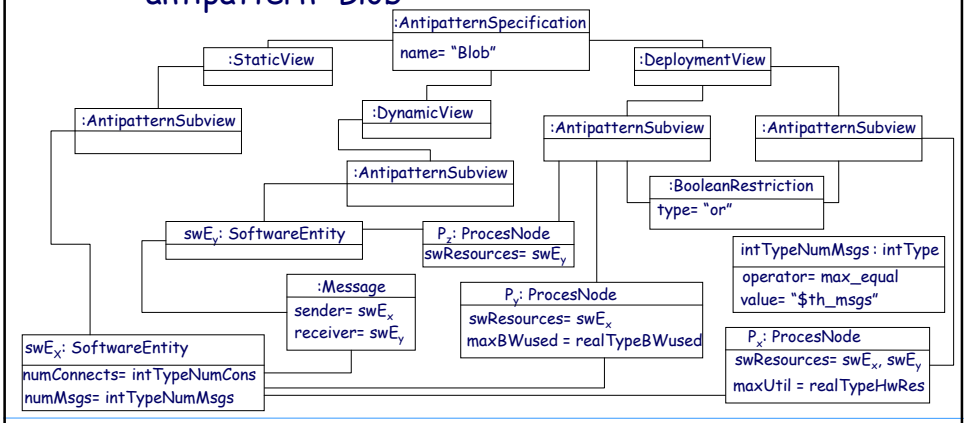


» Deployment View of SML+



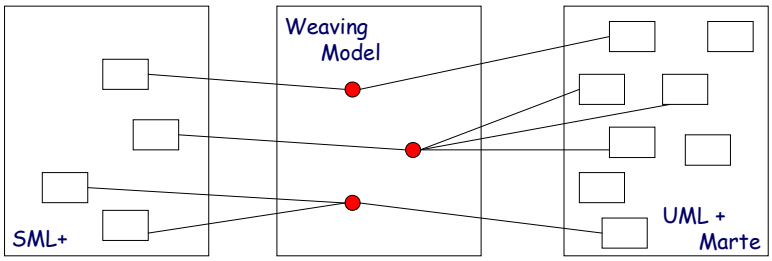
Modeling with APML

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



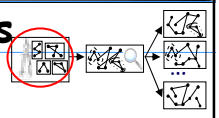
Our metamodel in concrete notations

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

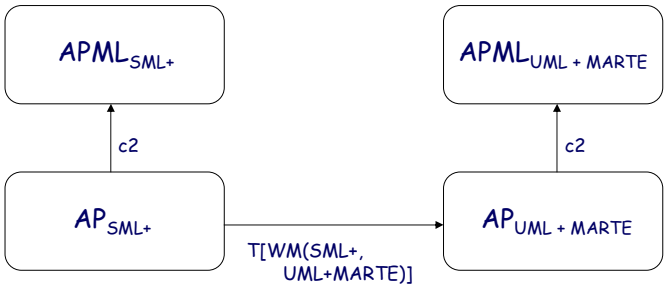


dipartimentoinformatica
Università degli Studi dell'Aquila

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).

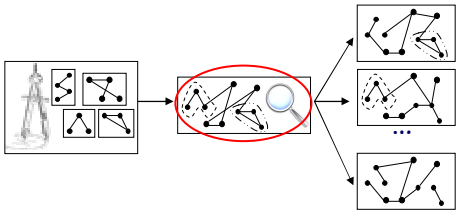


dipartimentoinformatica
Università degli Studi dell'Aquila

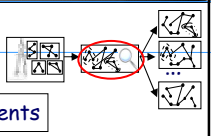
Contributions

» (2) Detecting Antipatterns

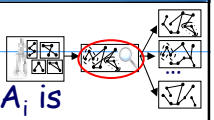
- Most promising antipatterns
- Model driven techniques: translating the occurrence of antipatterns with OCL expressions



Most promising antipatterns



Which ones and in what order?



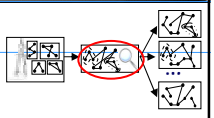
» How much a Performance Antipattern PA_i is important for the Violated Requirement Rj?

Violated Requirements

Performance Antipatterns

	R_1	...	R_j
PA_1			
...			
PA_i			weight (PA_i, R_j)

A first proposal : XML-based



» Java Engine for XML parsing

```
import org.w3c.dom.*;
...
Document doc = newDocumentBuilder().parse(new
   File("swSystem.xml"));
```

Loading XML file

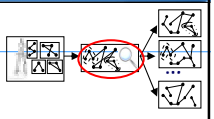
```
NodeList listOfProcesNodes =
   doc.getElementsByTagName("procesNode");
```

Creating a list of processing nodes

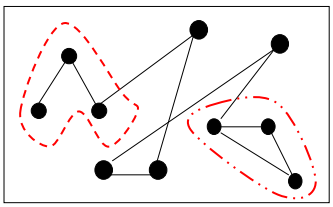
```
System.out.println("Total number of processors: "
   + listOfProcesNodes.getLength());
...
```

Printing the number of processing nodes

A second proposal : Model-driven



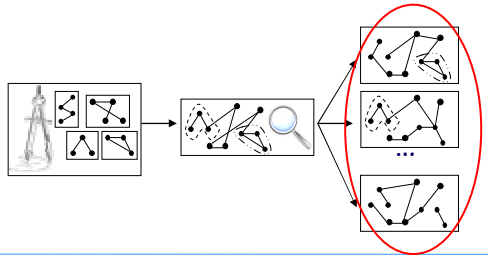
» Translating the occurrence of antipatterns with OCL expressions.



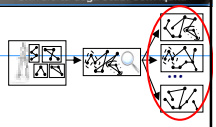
Contributions

» (3) Solving Antipatterns

- Deciding which antipatterns to solve and in what order
- Planning a simultaneous combination of antipatterns solutions
- Model-driven techniques: solving antipatterns with difference models



Analysing some problems



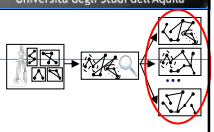
» Requirement problems

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

» Coherency problems

- Incoherences among antipattern solutions

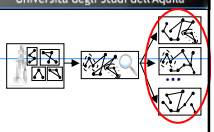
Discussion

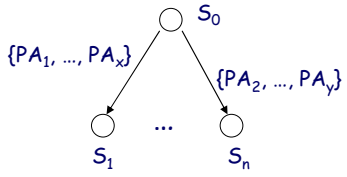


» Further issues

- 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

A first proposal : XML-based

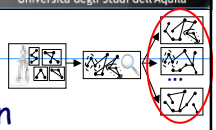
- Basic Idea: solving a performance antipattern by the negation of the logical predicate $\neg LP_{antipatName}$
- The negative logical predicate is built with the negation of the views predicates

$$\neg LP_{antipatName} = \neg SVP_{antipatName} \vee \neg DyVP_{antipatName} \vee \neg DeVP_{antipatName}$$

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

$$\neg *VP_{antipatName} = \neg (*BP_1 (\_, \_) \ldots (\_, \_) *BP_n)$$

A first proposal : XML-based



- An example: "Blob" problem formalization

$$LP_{blob} = SVP_{blob} \wedge DyVP_{blob} \wedge DeVP_{blob}$$

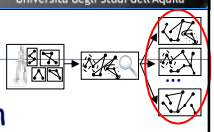
$$SVP_{blob} \equiv SBP_1$$

$$DyVP_{blob} \equiv DyBP_1$$

$$DeVP_{blob} \equiv DeBP_1 \wedge DeBP_2$$

$$\Rightarrow \exists E \mid F_{numConnects}(E) \geq Th_{connect} \wedge F_{numMsgs}(E) \geq Th_{msgs} \wedge (F_{maxHwUtil}(P_E) \geq Th_{hwUtil} \wedge F_{maxNetUtil}(P_E) \geq Th_{netUtil})$$

A first proposal : XML-based



» An example: "Blob" solution formalization

$$\neg LP_{blob} = \neg (SVP_{blob} \wedge DyVP_{blob} \wedge DeVP_{blob})$$

$$\neg SVP_{blob} \equiv \neg SBP_1$$

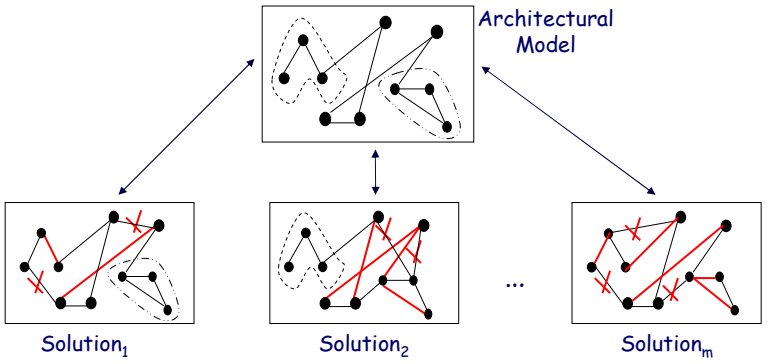
$$\neg DyVP_{blob} \equiv \neg DyBP_1$$

$$\neg DeVP_{blob} \equiv \neg DeBP_1 \vee \neg DeBP_2$$

$$\exists E \mid F_{numConnects}(E) < Th_{connect} \wedge F_{numMsgs}(E) < Th_{msgs} \wedge$$
$$(F_{maxHwUtil}(P_E) < Th_{hwUtil} \wedge F_{maxNetUtil}(P_E) < Th_{netUtil})$$

A second proposal : Model-driven

» Solving antipatterns with difference models.

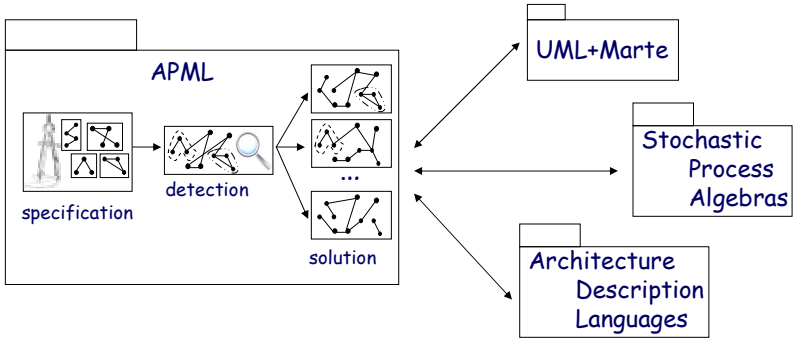


Related Work

- V.Cortellessa, L.Frittella, "A framework for automated generation of feedback from software performance analysis", 2007.
  - Informal interpretation matrices from the analysis of Layered Queueing Networks (LQNs)
- T.Parsons, J.Murphy, "Detecting Performance Antipatterns in component-based enterprise systems", 2008.
  - Performance antipattern detection (PAD) tool for Enterprise Java Bean (EJB) applications
- J.Xu, "Rule-based automatic software performance diagnosis and improvement", 2008.
  - Analysis of LQNs performance model for bottlenecks and long paths

Ongoing work

- Validate the scope of the whole approach across languagues, to assess the indipendence of any concrete notation.



Future work

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