dipartimentoinformatica
Università degli Studi dell'Aquila
Via Vetoio 1, 67010 Coppito, L'Aquila, Italy
www.di.univaq.it

KIT
Karlsruhe Institute of Technology

Institute for Programme Structure
and Data Organisation
Software Design and Quality Group
Am Fasanengarten 5
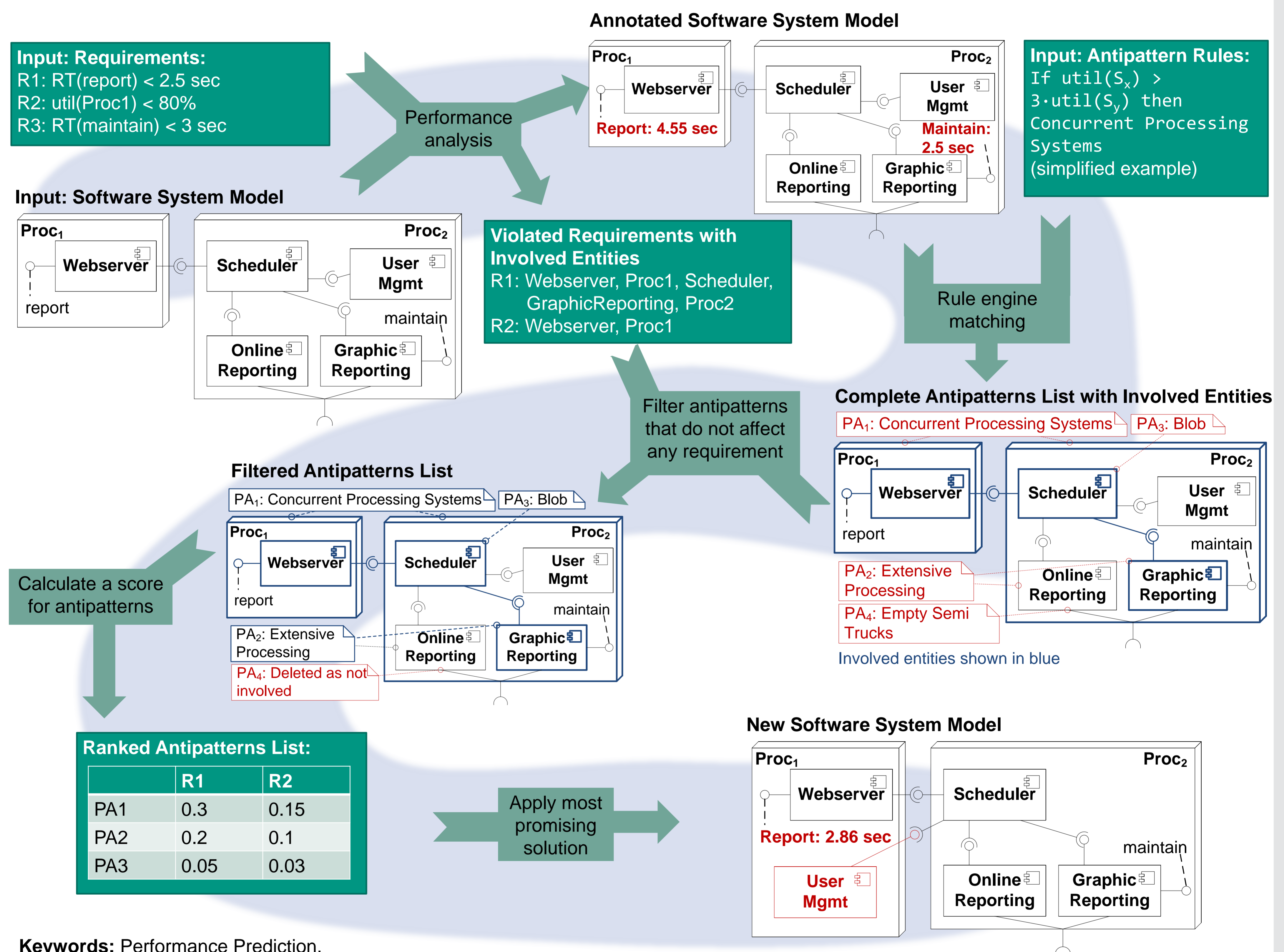Karlsruhe, Germany
www.sdq.ipd.kit..edu

Towards the Identification of "Guilty" Performance Antipatterns

Vittorio Cortellessa, Anne Martens, Ralf Reussner, Catia Trubiani

The problem of interpreting the results of software performance analysis is very critical. Software developers expect feedback in terms of architectural design alternatives (e.g., re-deploy a component), whereas the results of performance analysis are pure numbers. Support to the interpretation of such results that helps to fill the gap between numbers and software alternatives is still lacking. Performance antipatterns can play a key role in the search of performance problems and in the formulation of their solutions.

In this poster, we introduce a process to elaborate the analysis results and to score performance requirements, model entities and guilty performance antipatterns. We illustrate the process with exemplary values.

Input: Requirements:
R1: RT(report) < 2.5 sec
R2: util(Proc1) < 80%
R3: RT(maintain) < 3 sec

Performance analysis

Annotated Software System Model
Proc$_1$ — Webserver — Report: 4.55 sec
Proc$_2$ — Scheduler, User Mgmt — Maintain: 2.5 sec, Online Reporting, Graphic Reporting

Input: Antipattern Rules:
If util(S$_x$) > 3·util(S$_y$) then Concurrent Processing Systems (simplified example)

Input: Software System Model
Proc$_1$ — Webserver — report
Proc$_2$ — Scheduler, User Mgmt, Online Reporting, Graphic Reporting — maintain

Violated Requirements with Involved Entities
R1: Webserver, Proc1, Scheduler, GraphicReporting, Proc2
R2: Webserver, Proc1

Rule engine matching

Filter antipatterns that do not affect any requirement

Complete Antipatterns List with Involved Entities
PA$_1$: Concurrent Processing Systems   PA$_3$: Blob
Proc$_1$ — Webserver — report
Proc$_2$ — Scheduler, User Mgmt, Online Reporting, Graphic Reporting — maintain
PA$_2$: Extensive Processing
PA$_4$: Empty Semi Trucks
Involved entities shown in blue

Filtered Antipatterns List
PA$_1$: Concurrent Processing Systems   PA$_3$: Blob
Proc$_1$ — Webserver — report
Proc$_2$ — Scheduler, User Mgmt, Online Reporting, Graphic Reporting — maintain
PA$_2$: Extensive Processing
PA$_4$: Deleted as not involved

Calculate a score for antipatterns

Ranked Antipatterns List:

	R1	R2
PA1	0.3	0.15
PA2	0.2	0.1
PA3	0.05	0.03

Apply most promising solution

New Software System Model
Proc$_1$ — Webserver — Report: 2.86 sec — User Mgmt
Proc$_2$ — Scheduler, Online Reporting, Graphic Reporting — maintain

Keywords: Performance Prediction, Software Architecture, Antipatterns

Related Work

Xu et al.[4] present a semi-automated approach to find configuration and design improvement on the performance model level based on LQN models.

Parsons et al. [3] present a framework for detecting performance anti-patterns in Java EE architectures. The method requires an implementation of a component-based system, which can be monitored for performance properties.

Diaz Pace et al. [2] have developed the ArchE framework. ArchE assists the software architect during the design to create architectures that meet quality requirements. It helps to create architectural models and suggests improvements.

A more detailed presentation of this work can be found in [1].

References

[1] V. Cortellessa, A. Martens, R. Reussner and C. Trubiani. A Process to Effectively Identify "Guilty" Performance Antipatterns. In *Proc. of Fundamental Approaches to Software Engineering (FASE'10),* to appear.

[2] A. Díaz Pace, H. Kim, L. Bass, P. Bianco, and F. Bachmann. Integrating quality-attribute reasoning frameworks in the ArchE design assistant. In S. Becker, F. Plasil, and R. Reussner, editors, *Proc. of 4th International Conference on the Quality of Software-Architectures (QoSA'08)*, volume 5281 of *LNCS*, pages 171--188. Springer, 2008.

[3] T. Parsons and J. Murphy. Detecting performance antipatterns in component based enterprise systems. *Journal of Object Technology*, 7(3):55--90, 2008.

[4] J. Xu. Rule-based automatic software performance diagnosis and improvement. In *Proc. of the 7th International Workshop on Software and Performance (WOSP'08)*, pages 1--12, New York, NY, USA, 2008. ACM.

KIT – University of the State of Baden-Württemberg and
National Large-scale Research Center of the Helmholtz Association

www.kit.edu