Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

An Expandable Extraction Framework for Architectural Performance Models

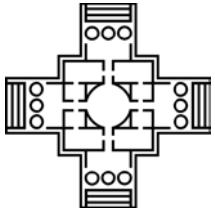
Jürgen Walter*, Christian Stier**, Heiko Koziolek***, and Samuel Kounev*

* University of Würzburg
** FZI Karlsruhe
*** ABB Corporate Research

April 27, 2017
QUDOS 2017 L'Aquila, Italy

Motivation

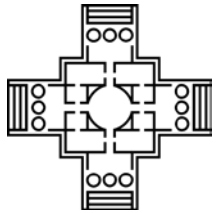
- Architectural models can be applied for design time analysis and reconfigurations at runtime …



Motivation

Performance Engineer

Manual creation



…

Huge effort per application

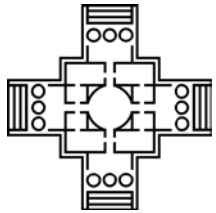
Architectural Performance Model

Motivation

Automated
model learning

Components, controlflow, resource demands, workload, …

Performance Engineer



…

Huge effort
for each
formalism

Architectural
Performance
Model

Problem Statement

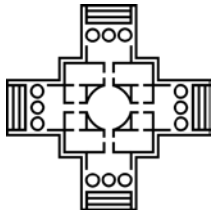
- **Analysis Toolchain Parallelism**
  - As a user, I would like to use different analysis approaches in parallel.
- **Analysis Toolchain Flexibility**
  - As a user, I prefer not to be forced to decide about the toolchain in advance.
- **Extraction Toolchain Reuse**
  - As a user, I would like to include performance model extraction for newly emerging formalisms without bothering about extraction complexity.

Performance Engineer

Idea

Decouple learning of generic aspects from object creation routines

Performance Engineer



…

Automated learning of generic aspects

Architectural Performance Model

Two step model learning

1. learning of generic aspects

Algorithm 1 Model Extraction Using Generic Builder

```
1: function CONSTRUCT(Path path, Builder builder)
2:     logs ← readLogFiles(path)
3:     analyzer ← compose analysis filters
4:     analyzer.analyze(logs)
5:     operationGraph ← analyzer.getOperationGraph()
6:     rds ← analyzer.getResourceDemands()
7:     workload ← analyzer.getWorkload()
8:     buildModel(operationGraph, rds, workload, builder);
9:     builder.save()
10: end function
```

Algorithm 2 Application of builder for Performance Model Generation

```
1: function BUILDMODEL(systemModel, operationGraph, resourceDemands, workload, builder)
2:     createHosts(systemModel, builder);
3:     createComponents(systemModel, builder);
4:     createInterfaces(systemModel, builder);
5:     createAllocations(systemModel, builder);
6:     for all source : operationGraph.vertices do
7:         component ← source.component.name
8:         host ← source.host.name
9:         assembly ← component + host
10:        builder.addAssembly(assembly);
11:        builder.assign(assembly, component);
12:        for all edge : source.outgoingEdges do
13:            target ← edge.getTargetVertice;
14:            tComponent ← target.component.name
15:            tHost ← target.host.name
16:            tAssembly ← tComponent + tHost
17:            builder.assign(tAssembly, tComponent);
18:            builder.connect(assembly, tAssembly);
19:            calls ← outgoing.getExternalCalls();
20:        end for
21:        rd ← resourceDemands.get(signature)
22:        builder.addBehavior(component, signature, calls, host, rd);
23:    end for
24: end function
```

2. model element creation

Approach

- Develop a framework that provides developers with a solution that integrates established tooling for monitoring, log processing, and resource demand estimation.
- To leverage the framework for model construction developers only have to implement a model builder interface that maps language independent concepts to language specific representations.

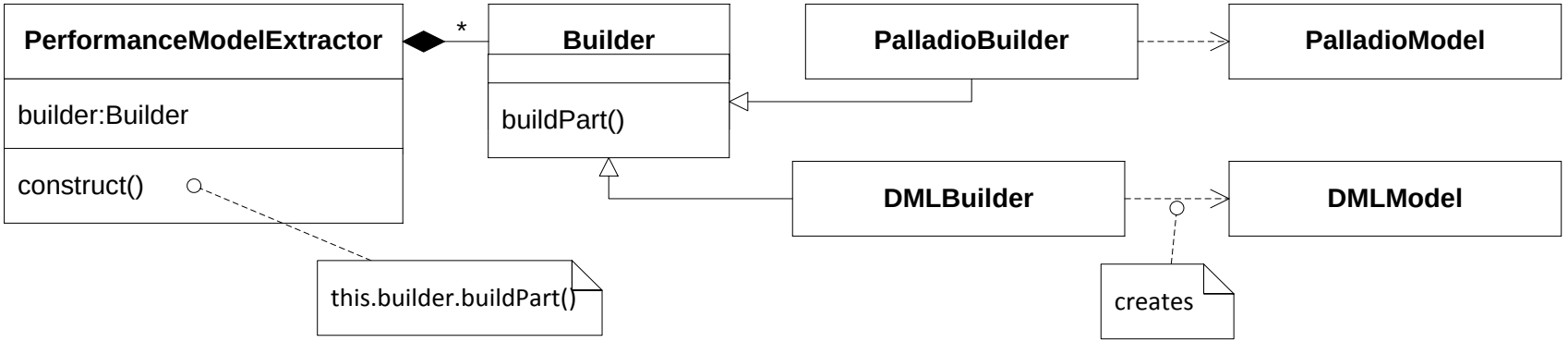
Learning of generic aspects

- PMX internaly uses a pipes and filter architecture
- PMX reuses existing libraries were possible

- Operation call graph
- Resource landscape
- Deployment
- Job arrival rates

- Resource demands

Approach



- Decouple learning and model creation using builder pattern

Builder Interface

- The interface includes object creation routines …

```
public EObject createHost(String hostName, int numberOfCores);
public EObject createComponent(String componentName);
public EObject createInterface(String InterfaceName);
public EObject createMethod(String interfaceName, Signature signature);
public EObject createAssembly(String assemblyName, String componentName);
public EObject createAllocation(String assemblyName, String hostName);
public EObject createProvidedRole(String componentName, String interfaceName);
public EObject createRequiredRole(String componentName, String interfaceName);
public EObject createServiceBehavior(String componentName, String methodName,
List<ExternalCall> externalCalls, String processingResource, double meanResourceDemand);
public void createResourceDemand(String service);
public void createWorkload(HashMap<String, List<Double>> workload);
```

Builder Interface II

- …and getter and connector functions
  - Meta-model elements have cross references e.g., deployment referes to infrastructure and component definitions

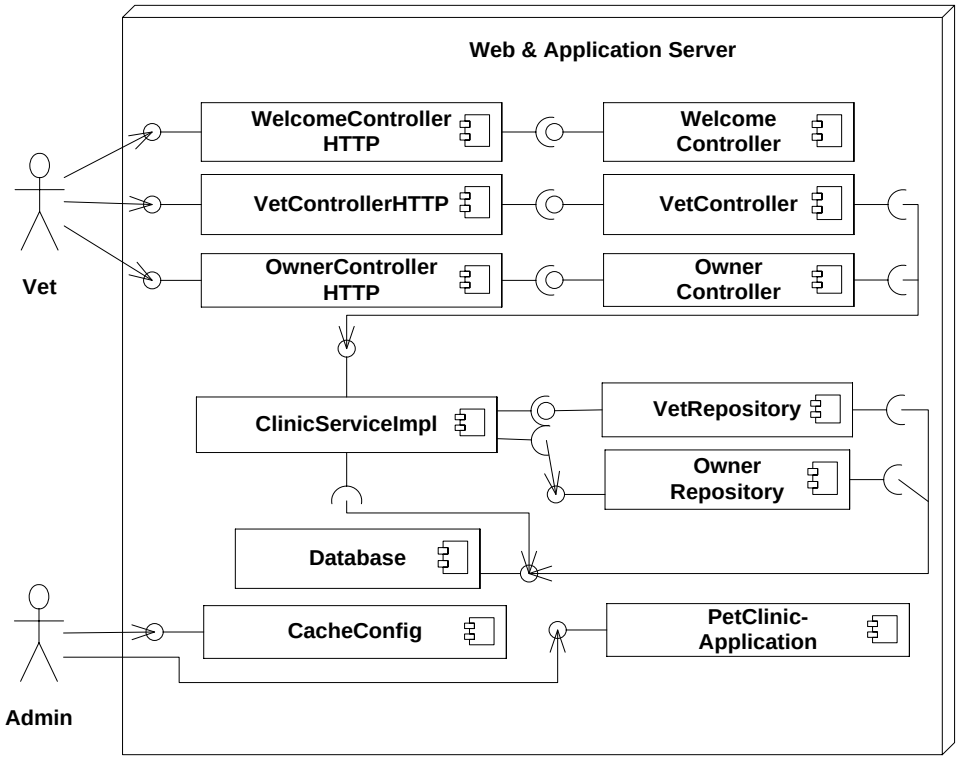
```
public EObject getRole(String role);
public EObject getAssembly(String assemblyName);
public EObject getMethod(String methodName);
public EObject getInterface(String interfaceName);
public EObject getServiceBehavior(String componentName,String methodName);
```

…

  - Implementation of getters can be alleviated inheriting from a provided *AbstractBuilder* class that stores created elements into HashMaps

Evaluation Setting

- Pet Clinic application
- Deployed on a 42 core VM



Evaluation Results

Workload in requests per second	CPU utilization (average)			session response time in ms (average)		
	Actual	DML	PCM	Kieker	DML	PCM
1(calibration)	0.33%	0.35%	0.34%	14.24	14.13	14.13
732	25.22%	24.64%	24.84%	14.35	14.14	14.54
940	33.12%	31.64%	31.77%	15.65	14.14	14.69

Table 2: Evaluation Results Pet Clinic Case Study.

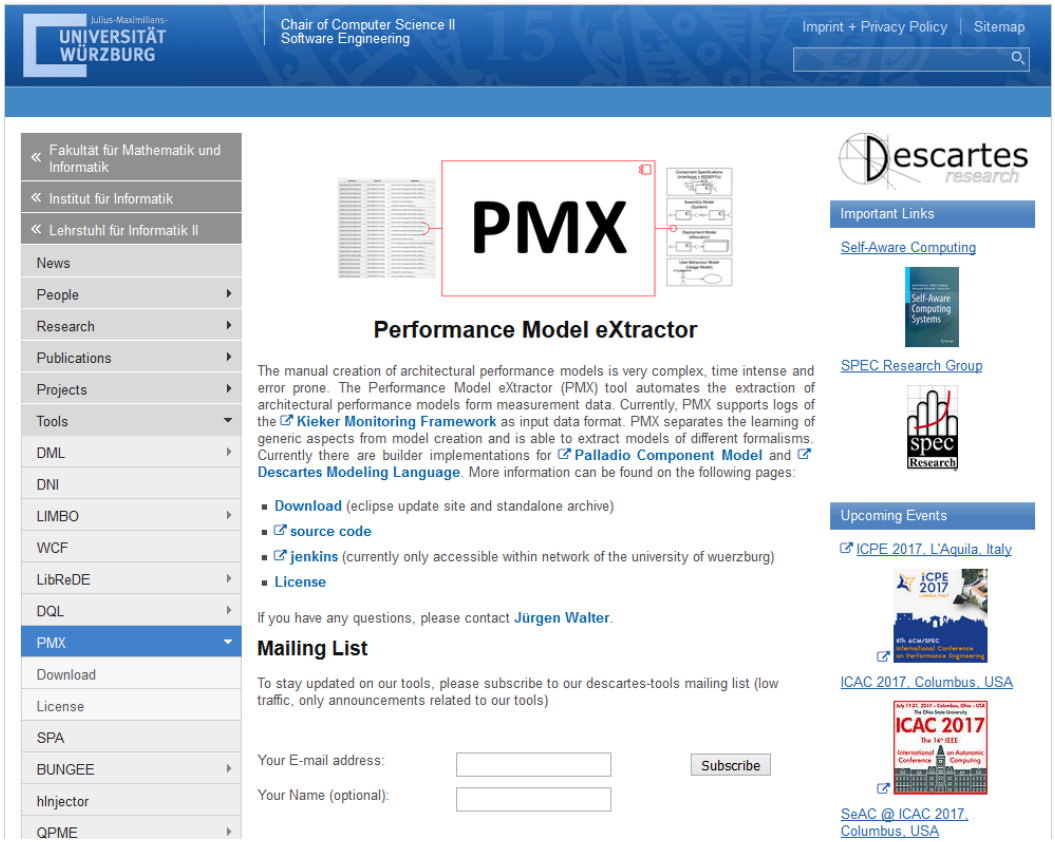
Deviation for utilization is below 2% and below 10% for response times.

Related Work

- Automated model extraction approaches
  - Closed source: PMW, Epasa
  - limited to a single modeling language
- Subparts of model learning
  - Extraction of resource demands, e.g., LibEeDE (Spinner2014, Spinner2015)
- Flexibility
  - Intermediate models (PMIF, Klapper, CSM,…)
  - Generic meta-model (SAMM)
  - Interchange format (DUALLY)

Available online

- PMX core as well as builders are available online http://descarte.tools/pmx/



Future Work

- Provide more builder implementations
- Conduct more case studies
- Allow for different monitoring tools and formats using OPEN.xtrace (formerly Common Trace API (CTA)) as input
- Use extracted models …
  - to integrate in load testing e.g., using a Jenkins plugin
  - for runtime resource management

Conclusion

- We present a framework for the extraction of architectural performance models generalizing over the target modeling language.
- Using the presented approach, the user only has to implement our builder interface to create a performance model generation tool for a specific modeling language.

Julius-Maximilians-
UNIVERSITÄT
WÜRZBURG

Thank You!

Jürgen Walter*, Christian Stier**, Heiko Koziolek***, and Samuel Kounev*

* University of Würzburg
** FZI Karlsruhe
*** ABB Corporate Research

April 27, 2017
QUDOS 2017 L'Aquila, Italy