

# An Expandable Extraction Framework for Architectural Performance Models

Jürgen Walter<sup>\*</sup>, Christian Stier<sup>\*\*</sup>, Heiko Koziolk<sup>\*\*\*</sup>, and  
Samuel Kounev<sup>\*</sup>

\* University of Würzburg

\*\* FZI Karlsruhe

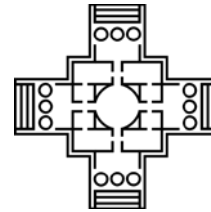
\*\*\* ABB Corporate Research

April 27, 2017

QUDOS 2017 L'Aquila, Italy



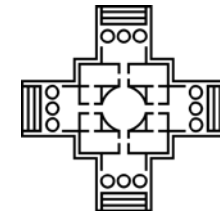
- Architectural models can be applied for design time analysis and reconfigurations at runtime ...





Performance Engineer

Manual creation



...

Architectural  
Performance  
Model

Huge effort per  
application

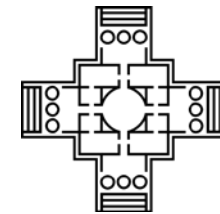


## Automated model learning

Components, controlflow, resource  
demands, workload, ...



**Performance Engineer**



...

Architectural  
Performance  
Model

**Huge effort  
for each  
formalism**



- **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**

Decouple learning of  
generic aspects from  
object creation routines



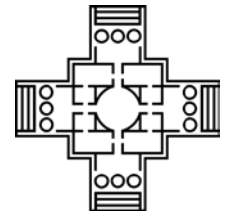
Performance Engineer



PMX



Automated learning  
of generic aspects



...

Architectural  
Performance  
Model

## 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

- 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.



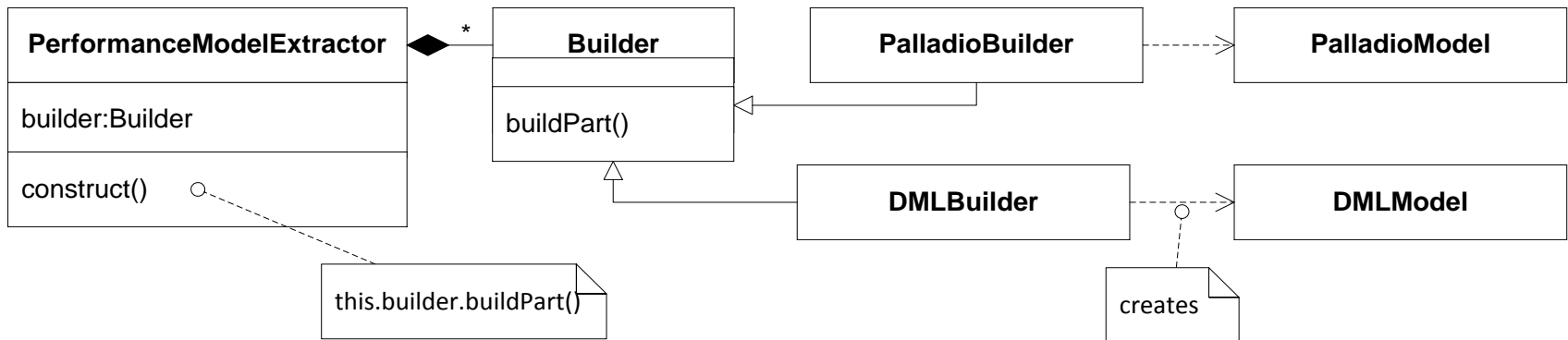
# Learning of generic aspects

- 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



- Decouple learning and model creation using builder pattern

- 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);
```

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

```
public EObject getRole(String role);
```

```
public EObject getAssembly(String assemblyName);
```

```
public EObject getMethod(String methodName);
```

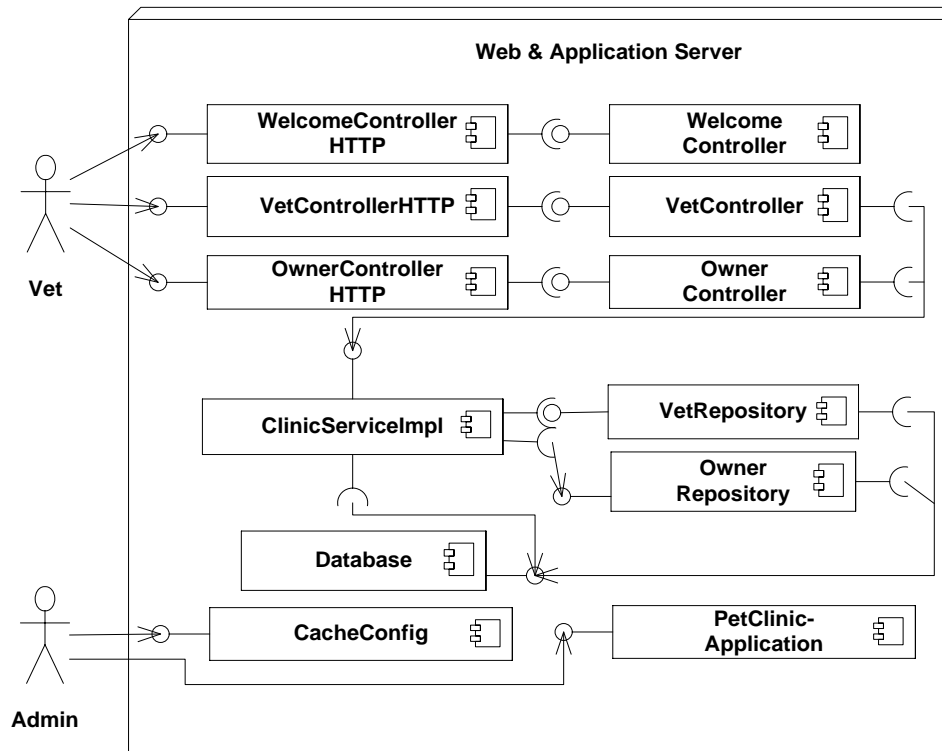
```
public EObject getInterface(String interfaceName);
```

```
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

- Pet Clinic application
- Deployed on a 42 core VM



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.

- Automated model extraction approaches
  - Closed source: PMW, Epassa
  - 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 (DUALY)

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

The screenshot shows the website for the Performance Model eXtractor (PMX) tool. The page is part of the Descartes Research portal, which is hosted by the Chair of Computer Science II Software Engineering at the University of Würzburg. The website features a navigation menu on the left, a central content area with a diagram of the PMX tool, and a right sidebar with various links and events.

**UNIVERSITÄT WÜRZBURG** Chair of Computer Science II Software Engineering

Imprint + Privacy Policy | Sitemap

Fakultät für Mathematik und Informatik  
Institut für Informatik  
Lehrstuhl für Informatik II

News  
People  
Research  
Publications  
Projects  
Tools  
DML  
DNI  
LIMBO  
WCF  
LibReDE  
DQL  
PMX  
Download  
License  
SPA  
BUNGEE  
hInjector  
QPME

**PMX**

**Performance Model eXtractor**

The manual creation of architectural performance models is very complex, time intense and error prone. The Performance Model eXtractor (PMX) tool automates the extraction of architectural performance models from measurement data. Currently, PMX supports logs of the [Kieker Monitoring Framework](#) as input data format. PMX separates the learning of generic aspects from model creation and is able to extract models of different formalisms. Currently there are builder implementations for [Palladio Component Model](#) and [Descartes Modeling Language](#). More information can be found on the following pages:

- Download (eclipse update site and standalone archive)
- source code
- jenkins (currently only accessible within network of the university of wuerzburg)
- License

If you have any questions, please contact [Jürgen Walter](#).

**Mailing List**

To stay updated on our tools, please subscribe to our descartes-tools mailing list (low traffic, only announcements related to our tools)

Your E-mail address:

Your Name (optional):

**Descartes research**

Important Links

[Self-Aware Computing](#)

[SPEC Research Group](#)

[ICPE 2017, L'Aquila, Italy](#)

[ICAC 2017, Columbus, USA](#)

[SeAC @ ICAC 2017, Columbus, USA](#)



- 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

- 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.



# Thank You!

Jürgen Walter<sup>\*</sup>, Christian Stier<sup>\*\*</sup>, Heiko Koziolk<sup>\*\*\*</sup>, and Samuel Kounev<sup>\*</sup>

<sup>\*</sup> University of Würzburg

<sup>\*\*</sup> FZI Karlsruhe

<sup>\*\*\*</sup> ABB Corporate Research

April 27, 2017

QUDOS 2017 L'Aquila, Italy

