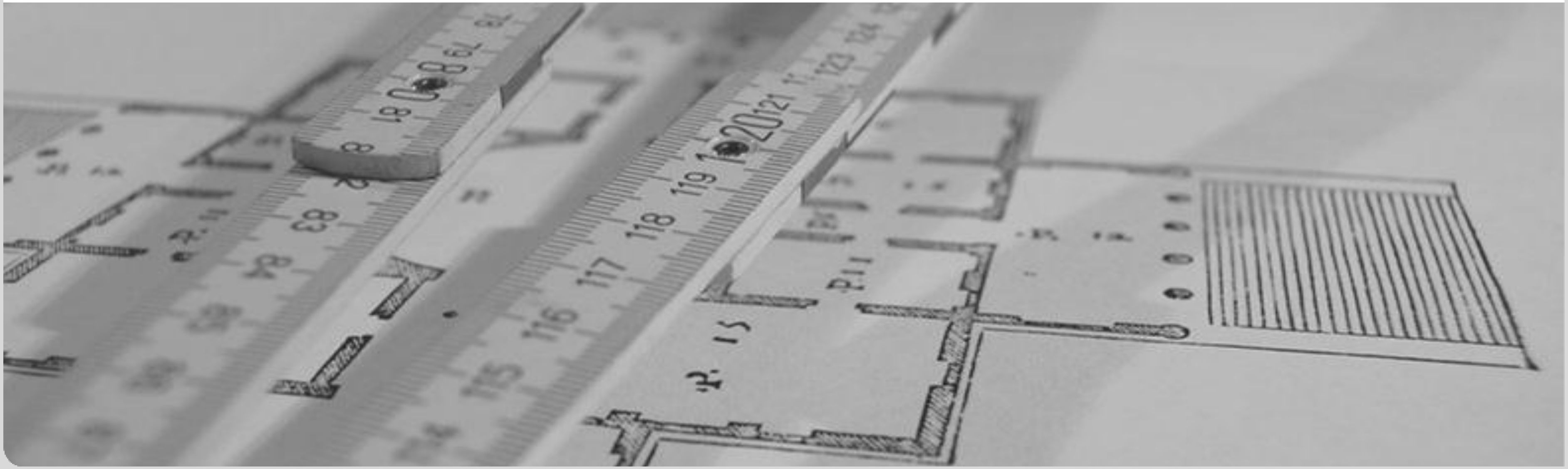
Design-Time vs. Run-Time Models for Quality-of-Service Prediction

Samuel Kounev

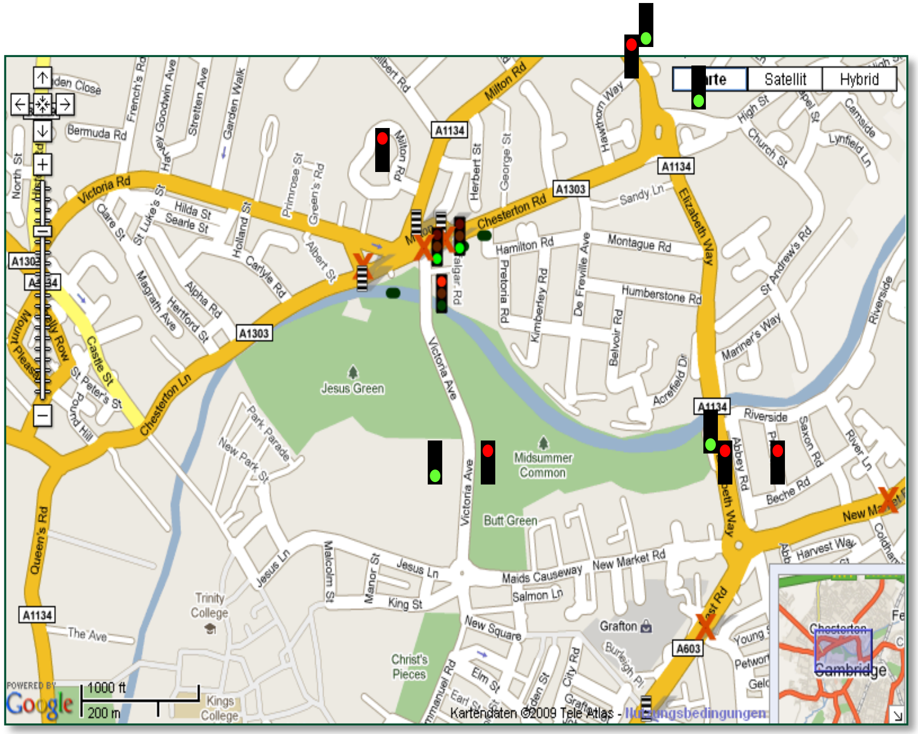
RELATE Open Excellence Workshop, Karlsruhe, December 4th, 2012

DESCARTES RESEARCH GROUP, CHAIR FOR SOFTWARE DESIGN AND QUALITY
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION



Motivation

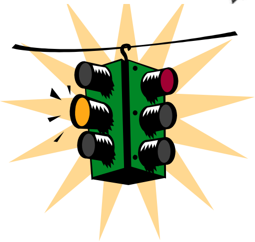
Traffic Management System



Induction Loops

GPS Sensors

Traffic Cameras

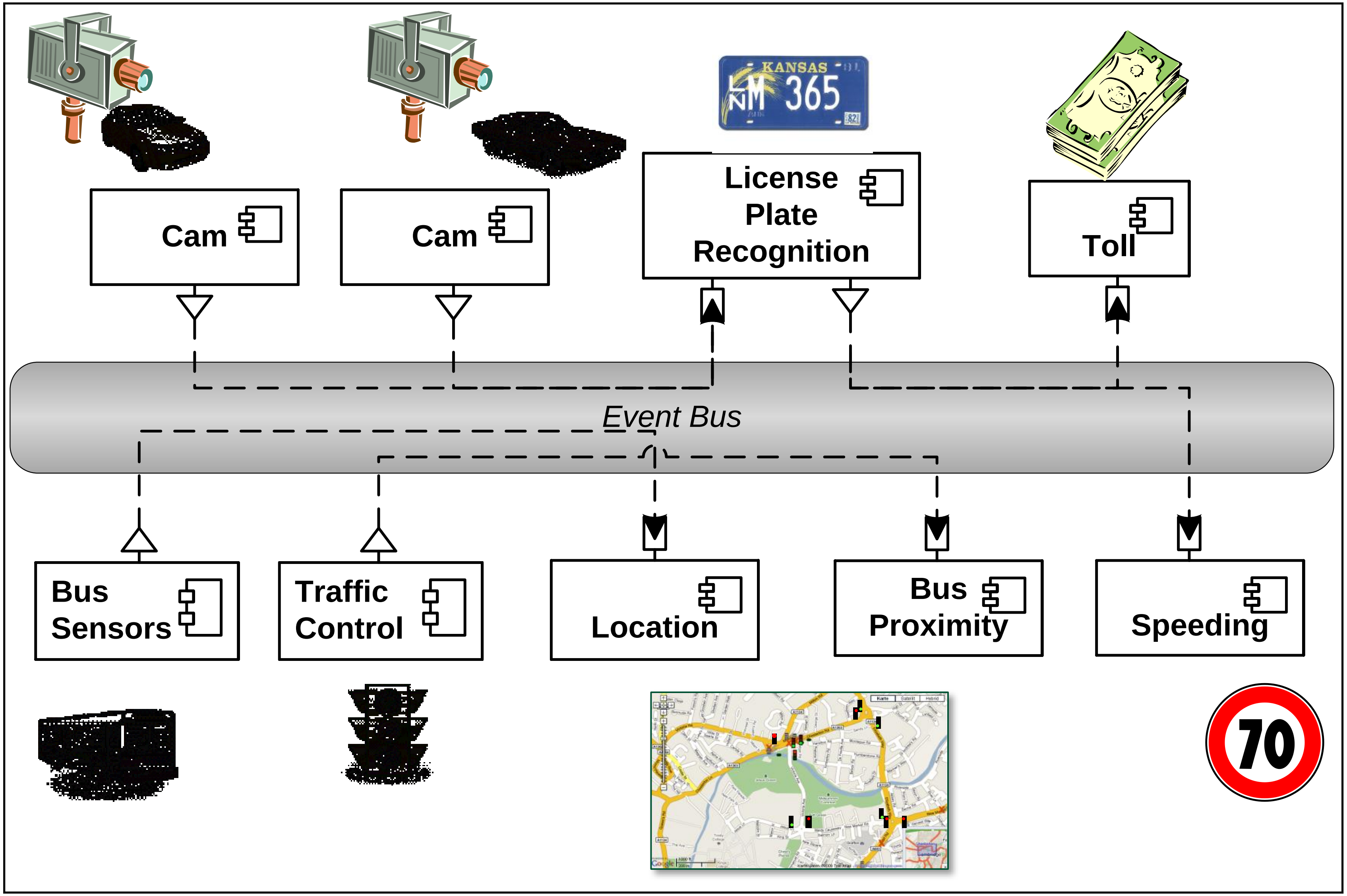


Traffic Light Sensors

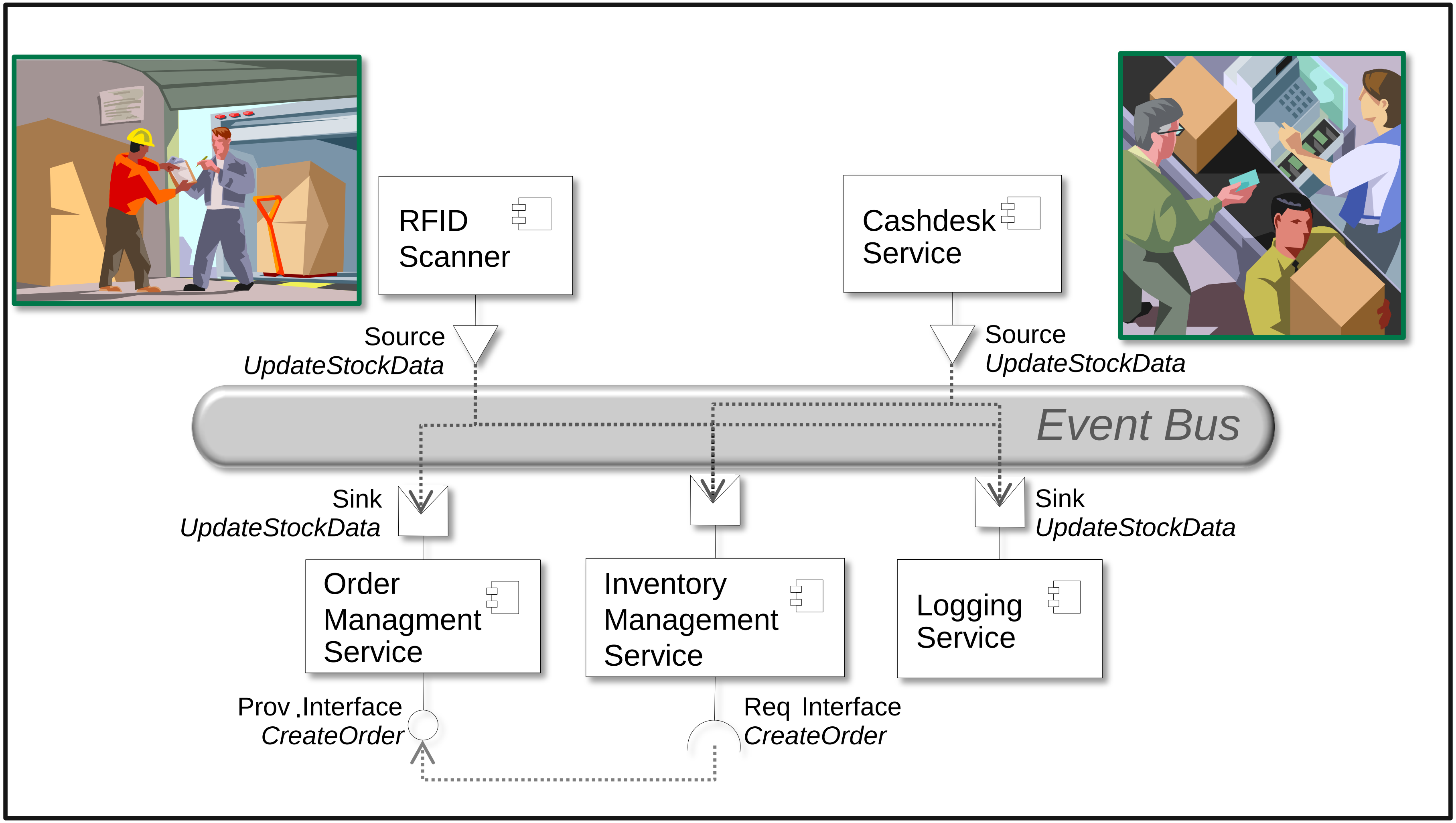
UNIVERSITY OF CAMBRIDGE

http://www.cl.cam.ac.uk/research/time/

Motivation: Traffic Management System



Motivation: Inventory Management System



RFID Scanner

Source
UpdateStockData

Cashdesk Service

Source
UpdateStockData

Event Bus

Sink
UpdateStockData

Sink
UpdateStockData

Order Managment Service

Inventory Management Service

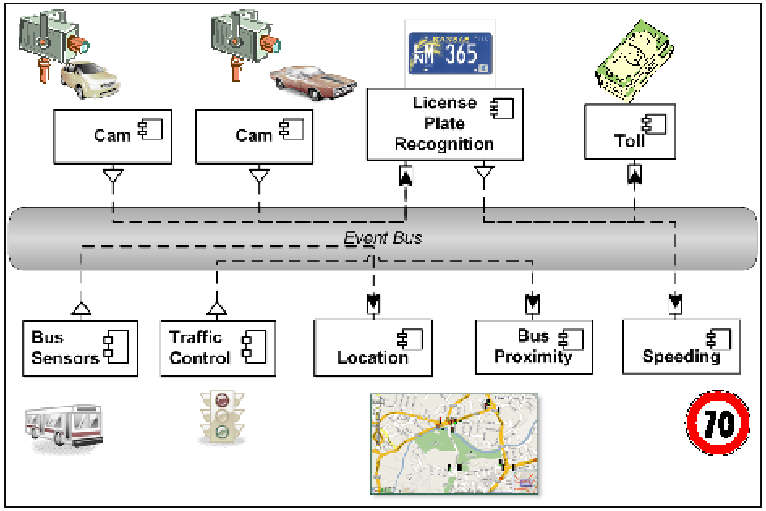
Logging Service

Prov .Interface
CreateOrder

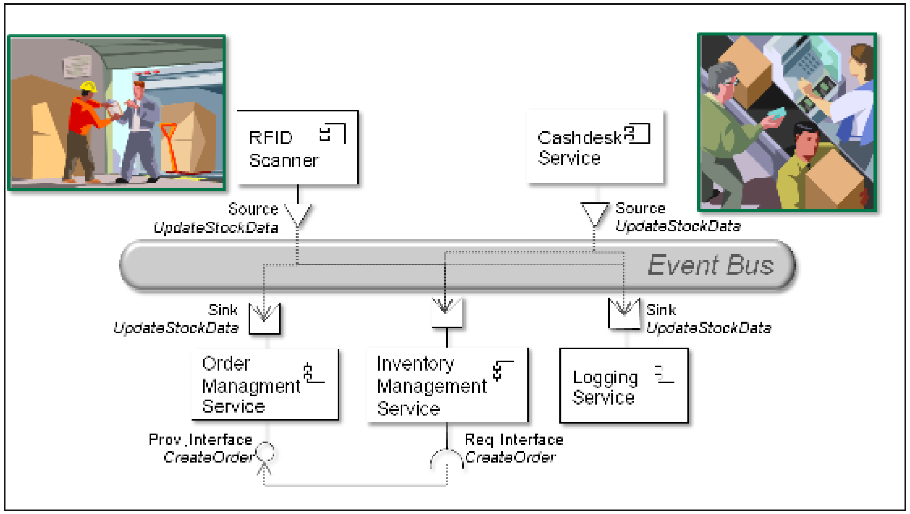
Req Interface
CreateOrder

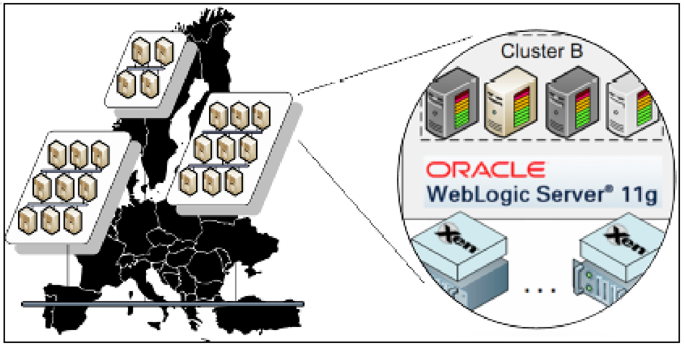
Motivation

Traffic Management System



Inventory Management System

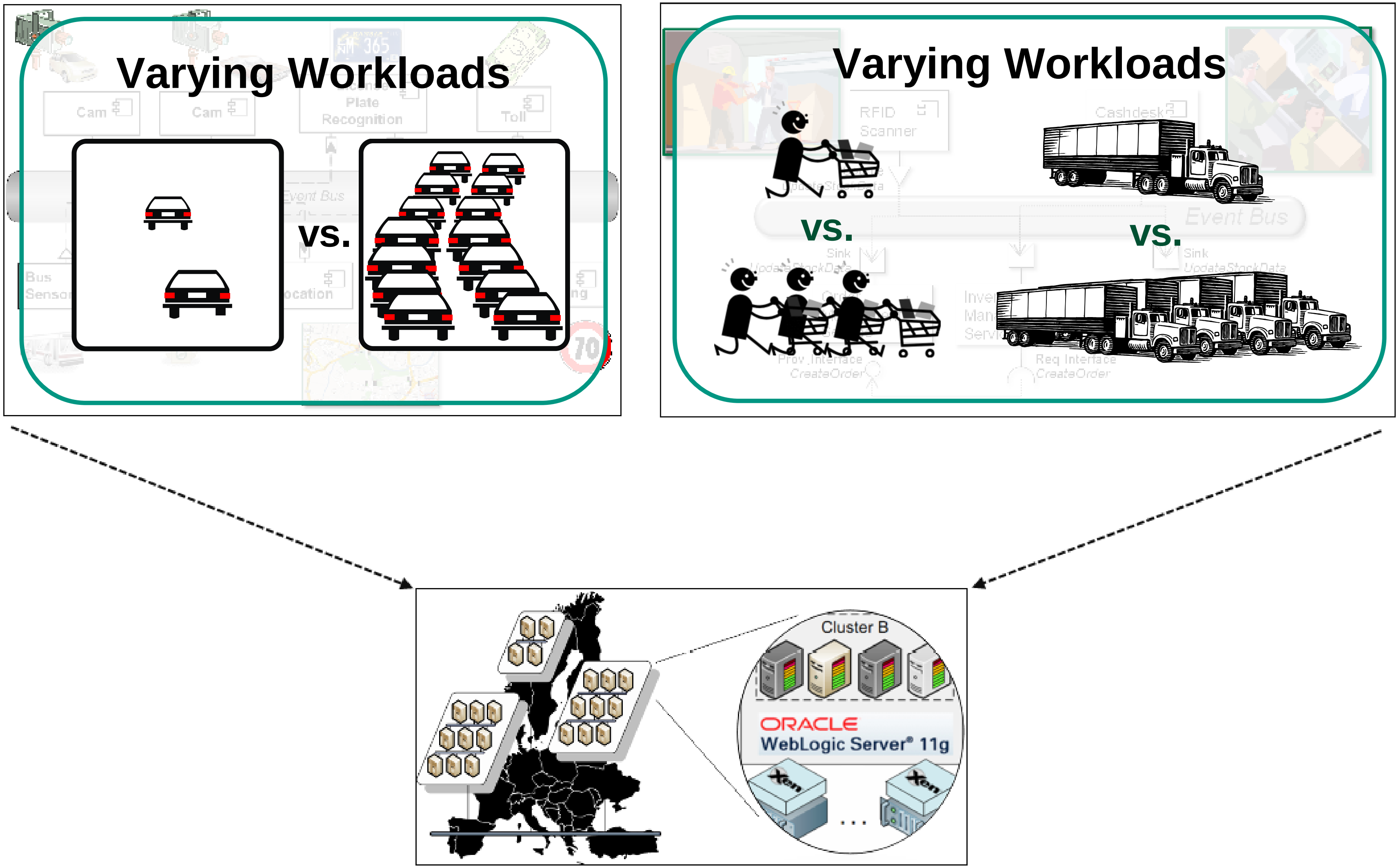




Motivation

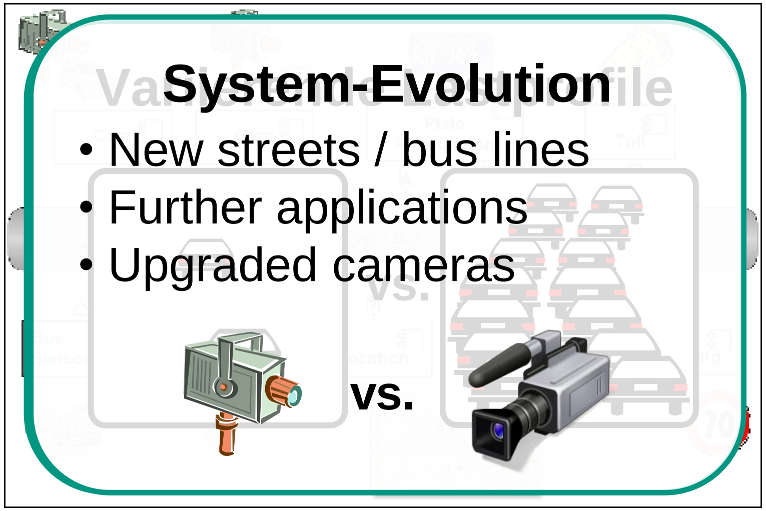
Traffic Management System

Inventory Management System



Motivation

Traffic Management System



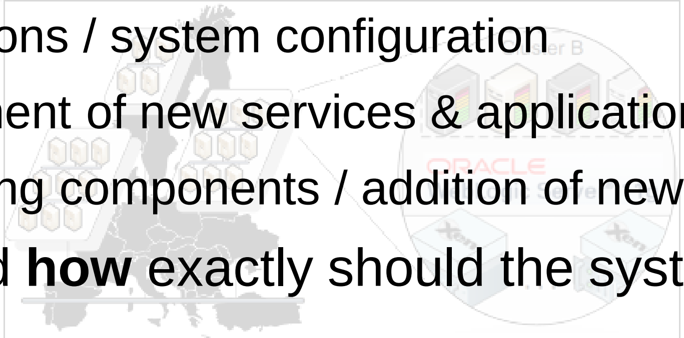
System-Evolution

- New streets / bus lines
- Further applications
- Upgraded cameras

Inventory Management System

System Evolution

- New supermarket stores
- Further applications
- Upgraded RFID readers



- Software systems increasingly **complex** and **dynamic**
- Must be **reconfigured at run-time** more and more frequently
  - Resource allocations / system configuration
  - Dynamic deployment of new services & applications
  - Changes of existing components / addition of new components
- Problem: **When** and **how** exactly should the system be reconfigured?

State-of-the-Art

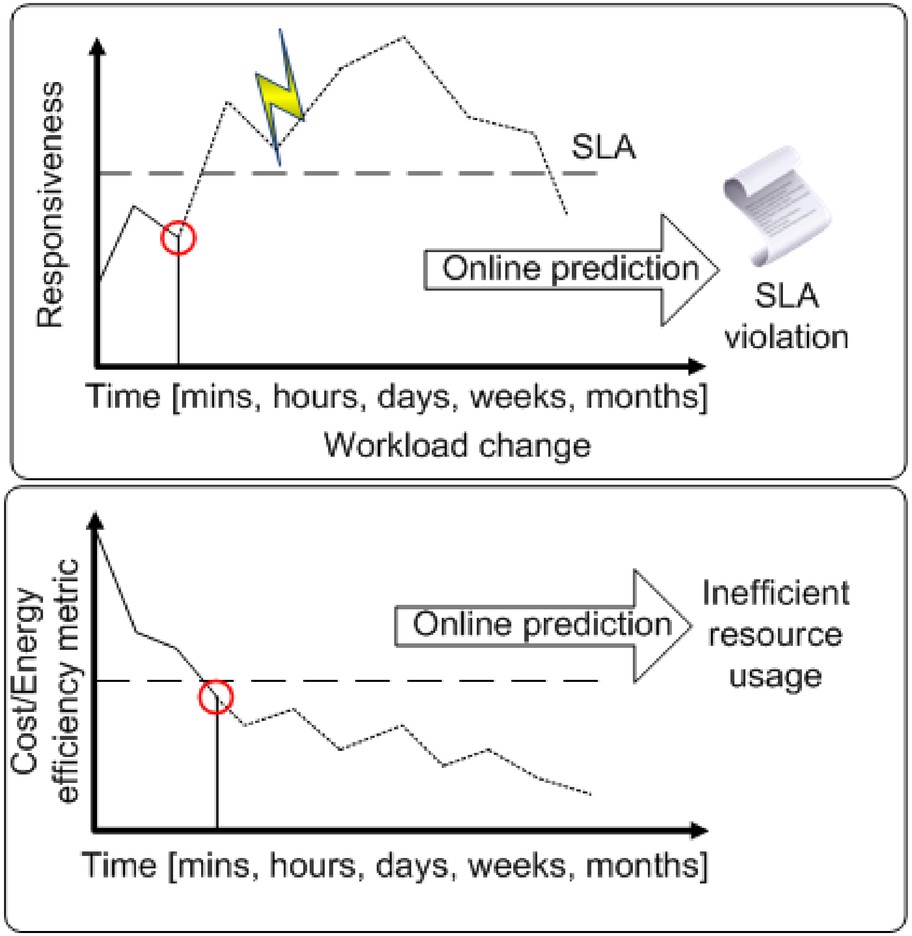
- Hard to predict the effect of dynamic changes on the system performance and resource demands
  - Minimize risks by avoiding the need for reconfiguration
    - Over-provisioning of IT resources
  - Simple rule-based adaptation techniques (“best effort”)
    - Manual adaptation in more complex scenarios

- Consequences: Poor resource efficiency
  - Rising energy costs for IT systems
    - 1600% increase by 2025 [Gartner]
  - Rising global CO2 emissions of ICT sector
    - Today: ca 3%, Increase to 10% expected in 10 years [EU]

Descartes Research Group

- Modeling methods for **predicting at run-time** the effect of dynamic changes on the system Quality-of-Service (QoS)
  - Current focus: availability and performance (response time, throughput and resource/energy efficiency)
- Model-based algorithms and techniques for **autonomic system adaptation** during operation
- Goal:
  - End-to-end QoS guarantees
  - High resource/energy efficiency
  - Low operating costs

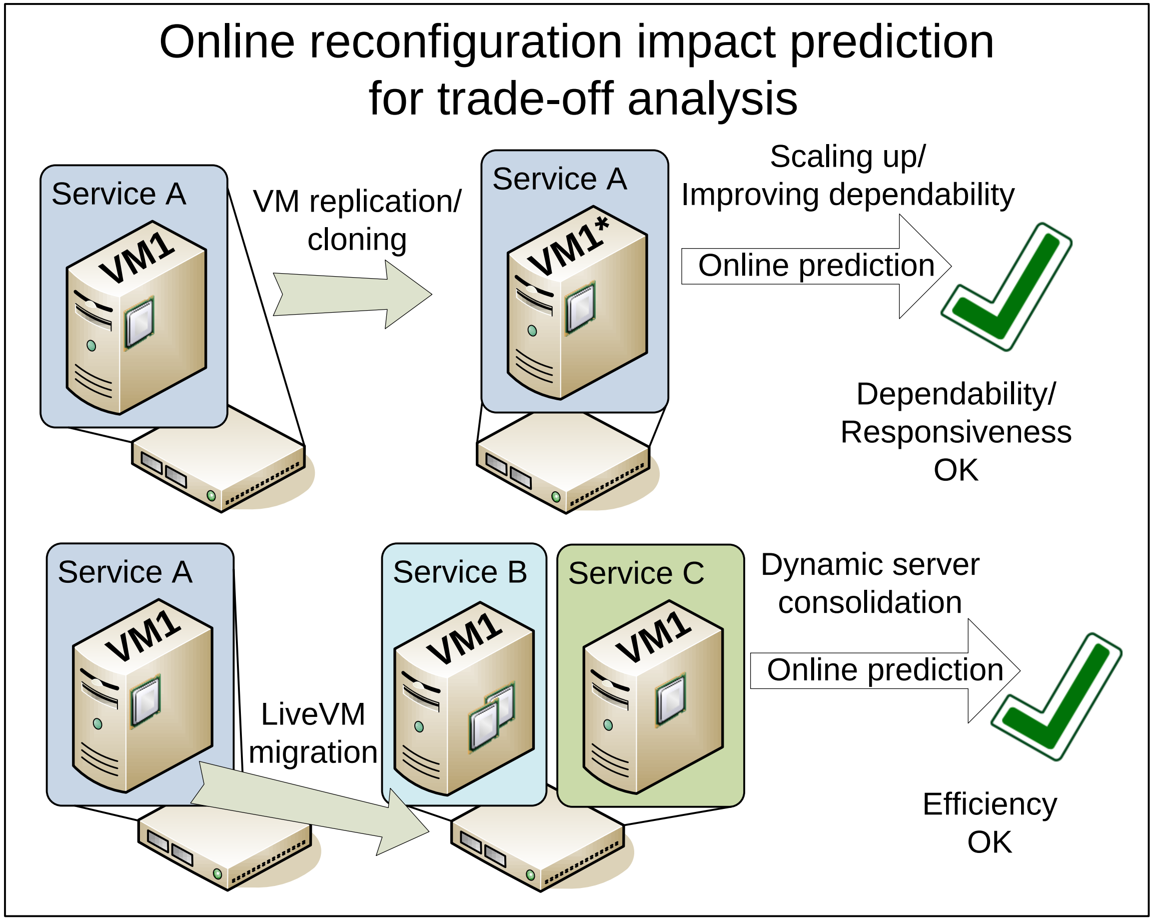
Proactive Self-Adaptive Systems Management



PHASE 1

Online QoS Prediction for Problem Anticipation

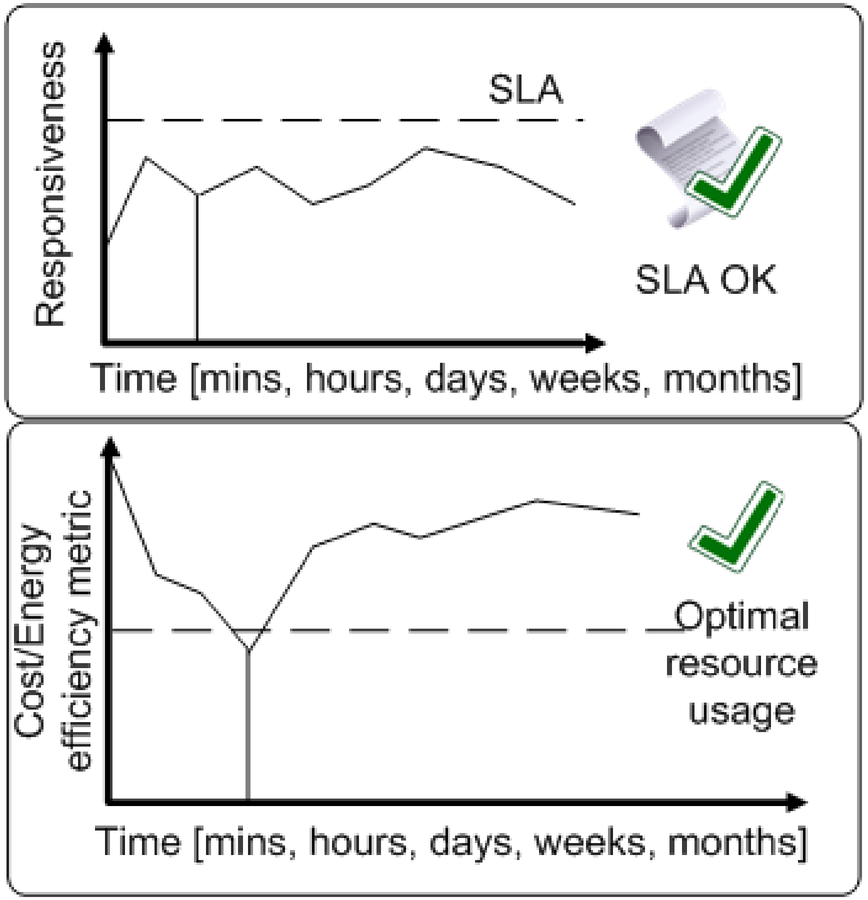
Proactive Self-Adaptive Systems Management



PHASE 2

Online QoS Prediction for Reconfiguration Impact Analysis

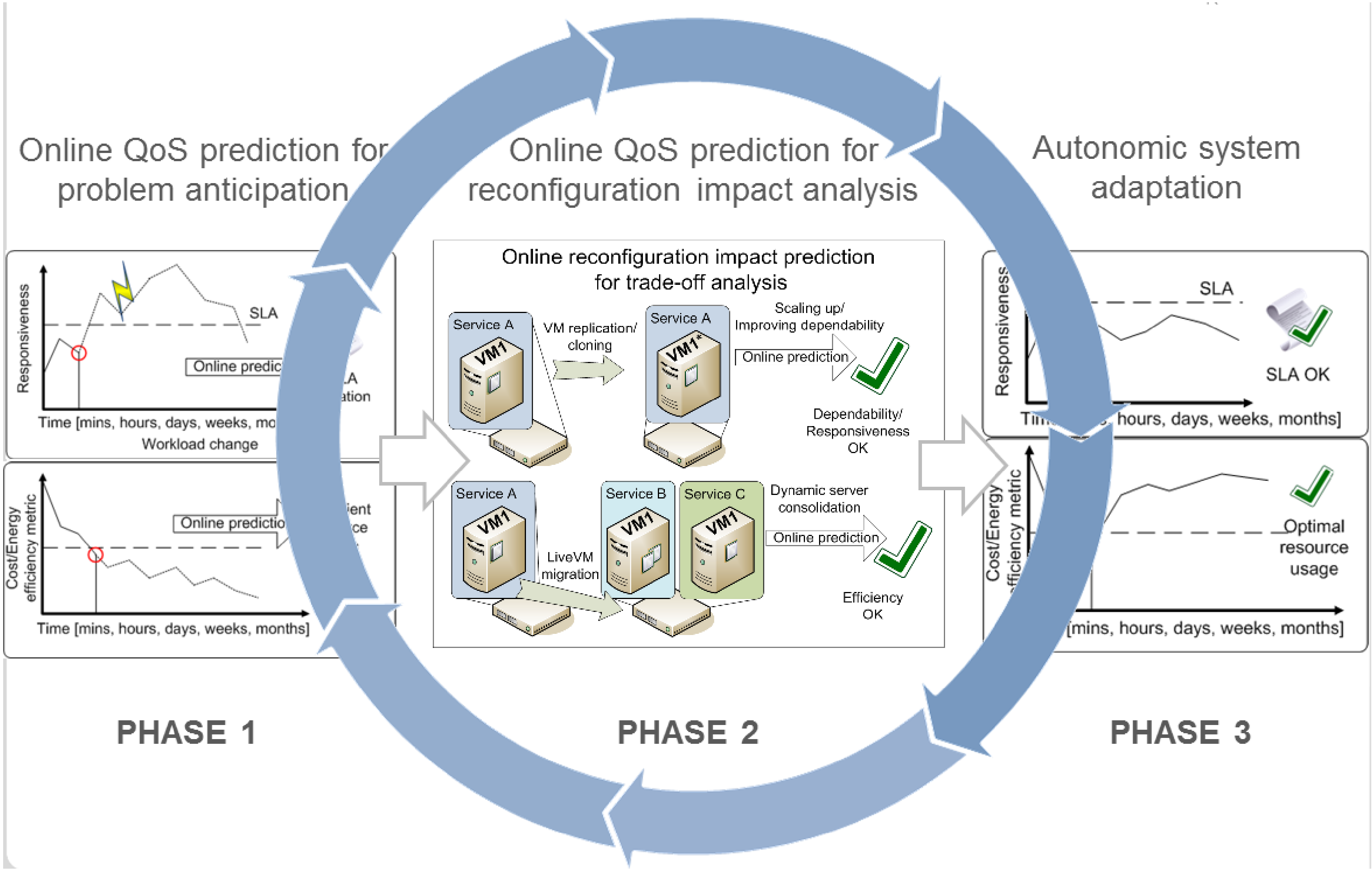
Proactive Self-Adaptive Systems Management



PHASE 3

Autonomic System Adaptation

Proactive Self-Adaptive Systems Management



Motivation ＞ **RUN-TIME MODELS** ＞ Descartes Meta-Model ＞ Case Study ＞ Summary & Outlook

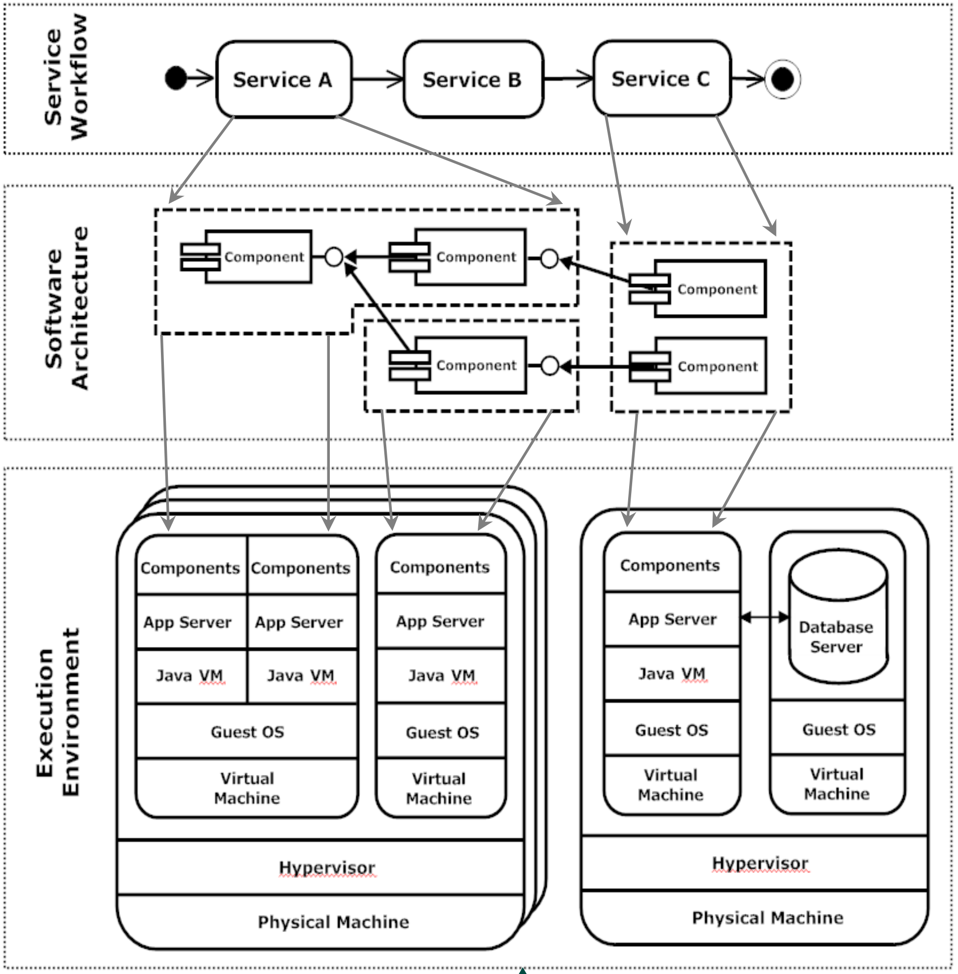
Examples of Performance-Influencing Factors

System workload and usage profile
- Number and type of clients
- Input parameters and input data
- Data formats used
- Service workflow

Software architecture
- Connections between components
- Flow of control and data
- Component resource demands
- Component usage profiles

Execution environment
- Number of component instances
- Server execution threads
- Amount of Java heap memory
- Size of database connection pools

Virtualization layer
- Physical resources allocated to VMs
  - number of physical CPUs
  - amount of physical memory
  - secondary storage devices



Network bandwidth between system nodes

High-Level Research Questions

- What models of the system architecture are appropriate to enable the **prediction of the impact of dynamic changes at run-time**?
    - Resource allocations and configuration parameters in each system layer should be explicitly taken into account
    - How do changes in service workloads and resource allocations impact the system QoS?
- How to deal with the large state space of possible reconfigurations?
- Which model analysis methods and optimization techniques are appropriate for a given adaptation scenario at run-time?
- …

State-of-the-Art: Summary

1. Modeling Approaches for Design-time Analysis

- UML SPT, UML MARTE, CBML, SPE-MM, KLAPER, CSM, PCM, SAMM, …
- Models assume static system architecture
- Dynamic aspects not considered
- Maintaining models at run-time prohibitively expensive

[M. Woodside et al], [D. Petriu et al], [R. Reussner et al], [C. Smith et al], [R. Mirandola et al], [K. Trivedi et al], [V. Cortellessa et al], [I. Gorton et al], [D. Menasce et al], [E. Eskenazi et al], …

2. Modeling Approaches for Run-time Analysis

- Queueing networks, „Reinforcement Learning“-Models, LPV-Models, …
- Models at a high level of abstraction: Components as „Black-Box“
- Architecture layers and configuration parameters not modeled explicitly

[G. Pacifici et al], [A. D‘Ambrogio et al], [G. Tesauro et al], [D. Menasce et al], [C. Adam et al], [Rashid A. Ali et al], [I. Foster er al], [S. Bleul et al], [A. Othman et al], [P. Shivam et al], …

Design-time vs. Run-time Models

- Two orthogonal dimensions
  - Modeling of design-time vs. run-time aspects
  - Use of models at design-time vs. run-time

- Fine granular differentiating factors
  1. Model purpose
  2. Model target users / consumers
  3. Degrees of freedom in model use case scenarios
  4. Model structure & parameterization
  5. Possibilities for model calibration
  6. Required model flexibility

1. Model Purpose

- Design-time
  - Evaluate and compare different design alternatives
  - Optimize system architecture
  - Sizing and capacity planning
- Run-time
  - Anticipate QoS issues resulting from
    - E.g., changing workloads, deployment of new services
  - Predict impact of possible dynamic reconfiguration
  - Adapt system configuration in a predictable manner
    - Elastic resource provisioning
    - Intrusion prevention
    - Failover after a server crash

2. Model Target Users / Consumers

- Design-time
  - System architect / performance engineer
  - Use by humans in an offline setting
  - Could also serve as architecture documentation

- Run-time
  - System administrator and/or the system itself
  - Use by humans and/or the system itself in an online setting

3. Degrees-of-Freedom

- Design-time
  - Theoretically every single aspect of the system can be varied
  - Degrees of freedom focused on
    - Software and system architecture
    - Deployment platforms
    - System configuration
- Run-time
  - Software architecture is relatively stable
  - Degrees of freedom focused on
    - Workloads / usage profiles
    - System deployment and configuration (incl. resource allocations)
    - Deployment of new services and/or change of service providers

4. Model Structure & Parameterization

- Design-time
  - Aligned with software development processes
    - Development phases and developer roles
    - Component: Unit of composition at design-time
  - Assumption: clear separation of concerns
  - Sub-models parameterized to capture their context dependencies

- Run-time
  - Aligned with system layers
    - Component: Unit of composition at run-time
  - Sub-models parameterized according to their dynamic reconfiguration aspects
  - Explicit distinction between static and dynamic aspects

5. Possibilities for Model Calibration

- Design-time
  - Flexibility to run experiments in a controlled environment
  - Possible lack of complete implementations of system components
  - Possible lack of a realistic production-like testing environment

- Run-time
  - All system components implemented and deployed
  - Monitoring in the production environment possible
  - Less control over the system to run experiments
  - Monitoring in a non-intrusive manner

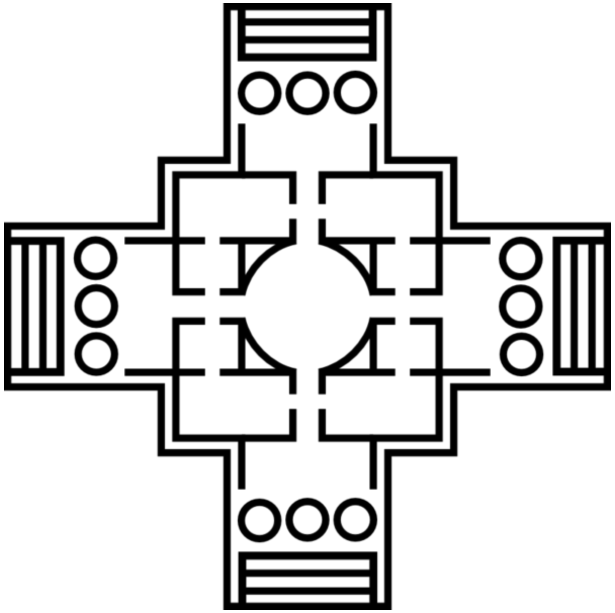
6. Required Model Flexibility

- Design-time
  - Plenty of time to analyze the model
  - Can run detailed time-intensive simulations
  - Generally accuracy more important than analysis overhead

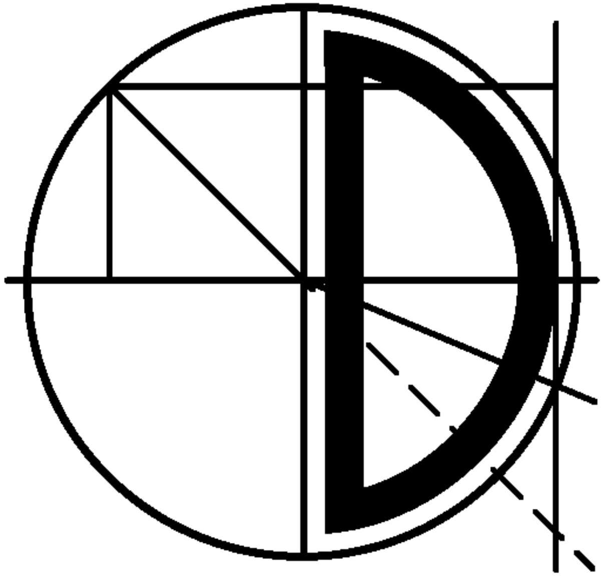
- Run-time
  - Model may have to be solved in seconds, minutes, hours, or days
  - Trading-off btw. accuracy and overhead critically important
  - Generally more flexibility required
    - Support for multiple abstraction levels, parameter granularities
    - Support for different analysis techniques

PCM and DMM

Palladio Component Model (PCM)



Descartes Meta-Model (DMM)



Design-time aspects

Run-time aspects

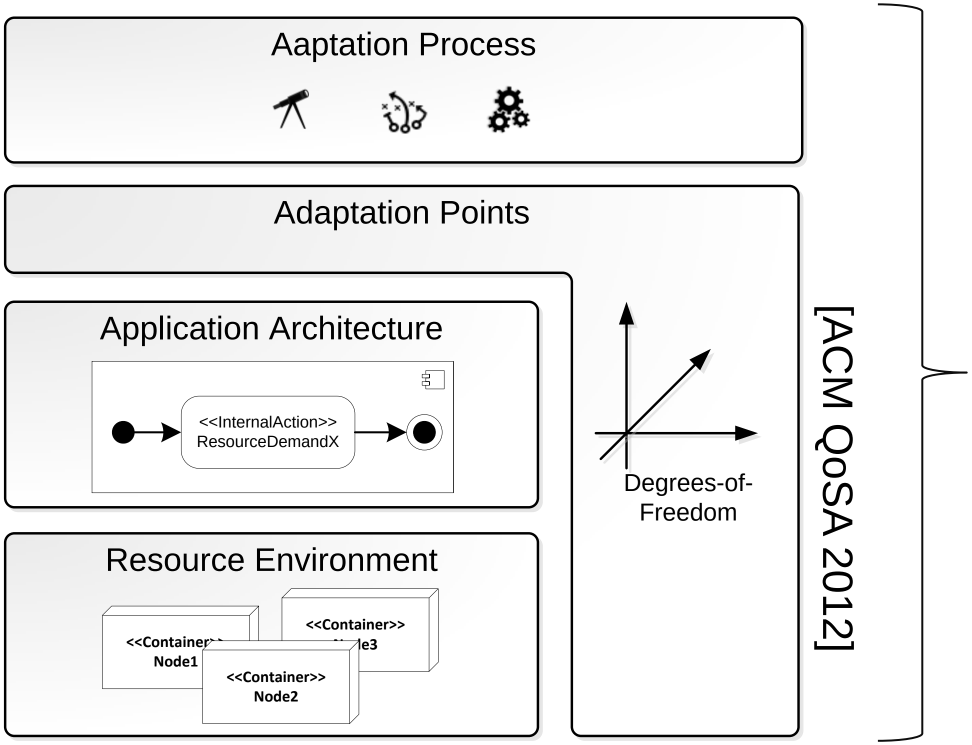
Descartes Meta-Model (DMM)

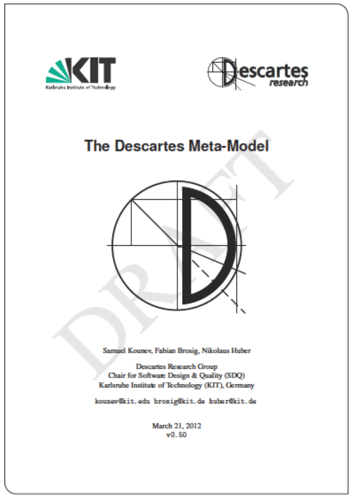
- Architecture-level modeling language for modeling QoS and resource management related aspects of IT systems, infrastructures and services
  - Prediction of the impact of dynamic changes at run-time
  - Autonomic performance and resource management
  - Current version focused on performance, capacity and efficiency aspects

[IEEE ICEBE 2012]

[ACM CBSE 2012]
[IEEE/ACM ASE 2011]

[ACM QoSA 2012]
[CLOSER 2011]
[DOA 2010]



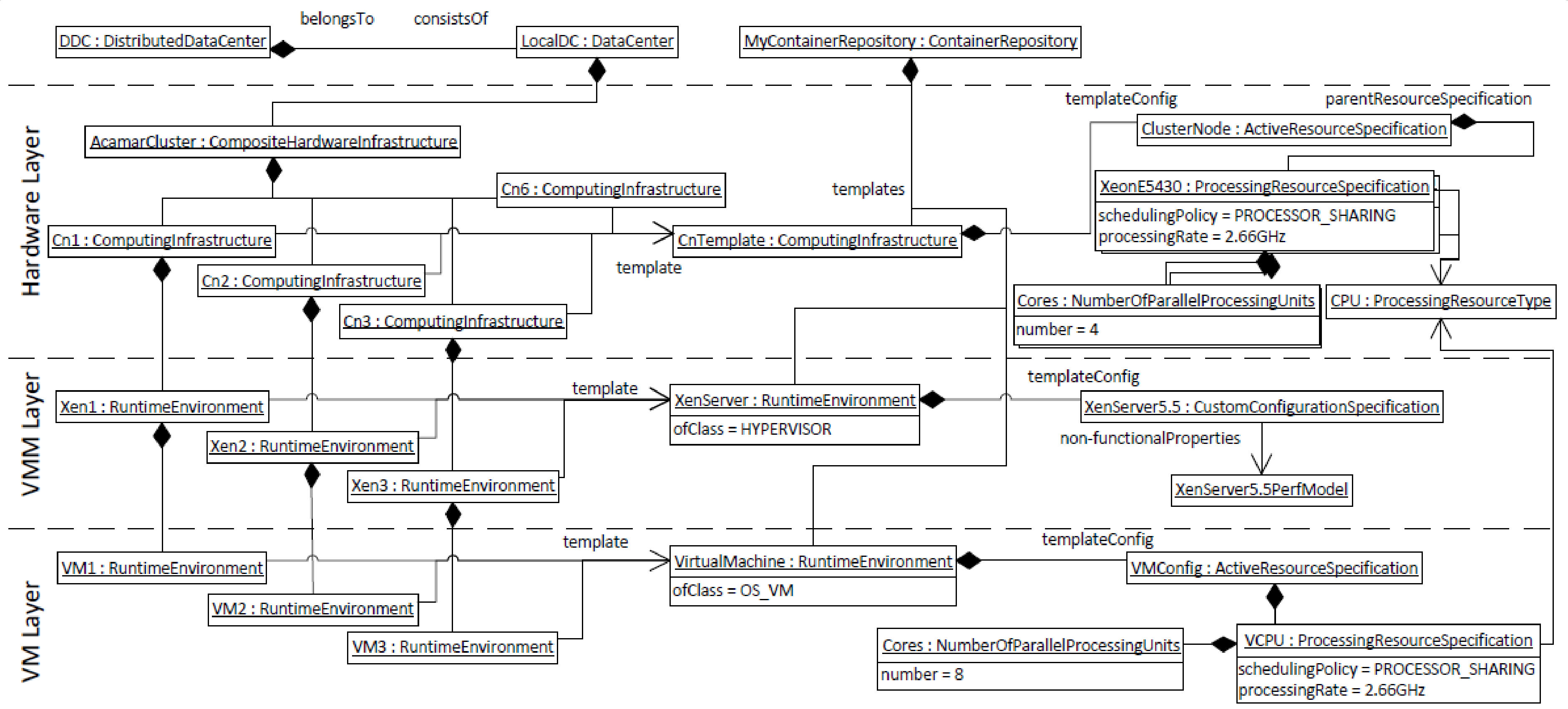

The Descartes Meta-Model

Samuel Kounev, Fabian Brosig, Nikolaus Huber
Descartes Research Group
Chair for Software Design & Quality (SDQ)
Karlsruhe Institute of Technology (KIT), Germany
kounev@kit.edu brosig@kit.de huber@kit.de

March 21, 2012
v0.50

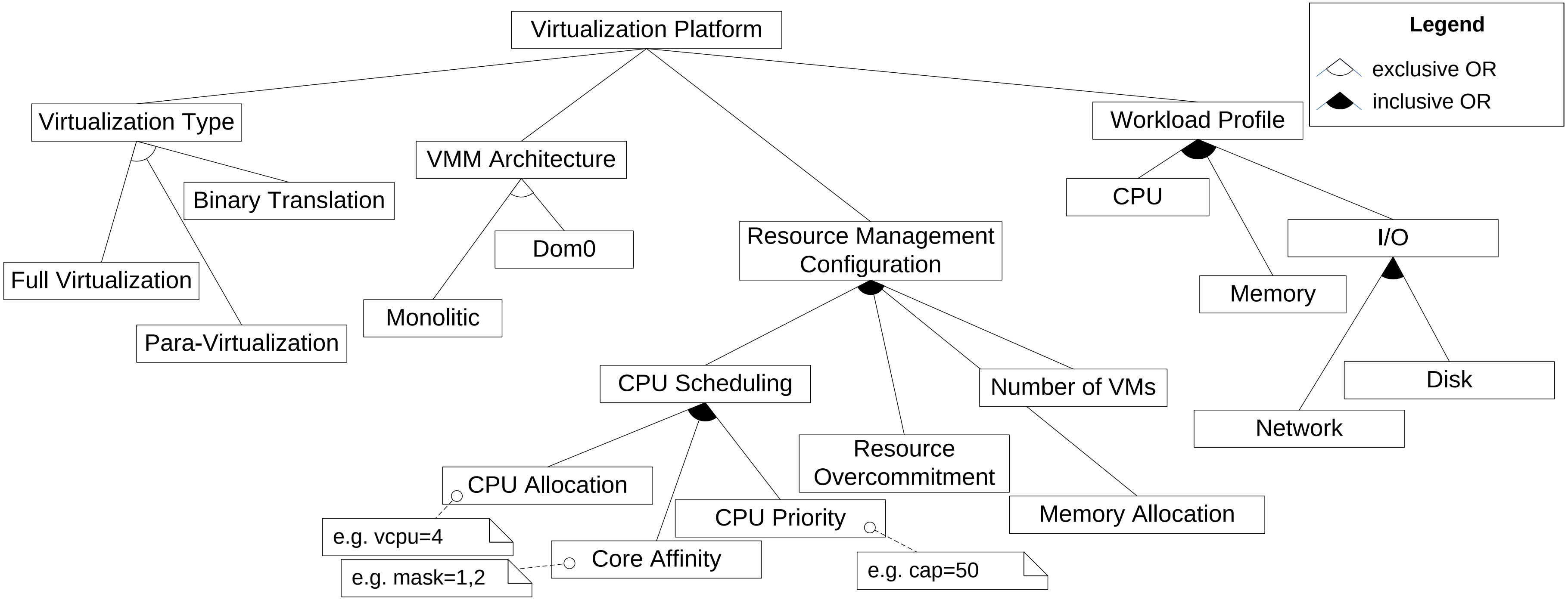
Technical
Report

Example: Resource Environment



N. Huber, F. Brosig and S. Kounev. **Modeling Dynamic Virtualized Resource Landscapes**. *In 8th ACM SIGSOFT International Conference on the Quality of Software Architectures (QoSA 2012),* Bertinoro, Italy, June 25-28, 2012.

Example: Resource Environment
Influence Factors of the Virtualization Layer



Virtualization Platform

Virtualization Type

Binary Translation

Full Virtualization

Para-Virtualization

VMM Architecture

Dom0

Monolitic

Resource Management Configuration

CPU Scheduling

CPU Allocation

e.g. vcpu=4

Core Affinity

e.g. mask=1,2

CPU Priority

e.g. cap=50

Resource Overcommitment

Number of VMs

Memory Allocation

Workload Profile

CPU

Memory

I/O

Disk

Network

Legend

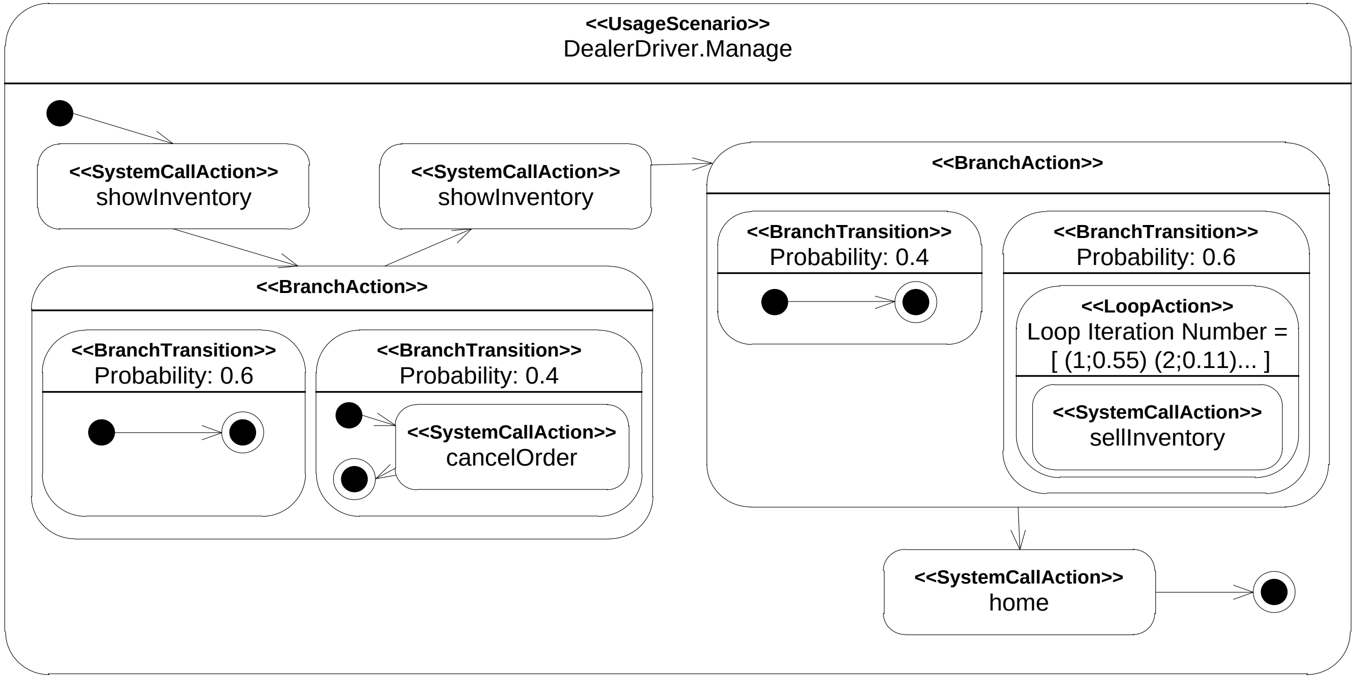
exclusive OR

inclusive OR

N. Huber, M. Quast, M. Hauck, and S. Kounev. **Evaluating and Modeling Virtualization Performance Overhead for Cloud Environments**. *International Conference on Cloud Computing and Services Science (CLOSER 2011), Noordwijkerhout, The Netherlands*, May 7-9, 2011. Best Paper Award.

Example: Application Architecture

- Control flow and data flow
- Service resource demands
- Parameter and context dependencies



<<UsageScenario>>
DealerDriver.Manage

<<SystemCallAction>>
showInventory

<<SystemCallAction>>
showInventory

<<BranchAction>>

<<BranchTransition>>
Probability: 0.6

<<BranchTransition>>
Probability: 0.4

<<SystemCallAction>>
cancelOrder

<<BranchAction>>

<<BranchTransition>>
Probability: 0.4

<<BranchTransition>>
Probability: 0.6

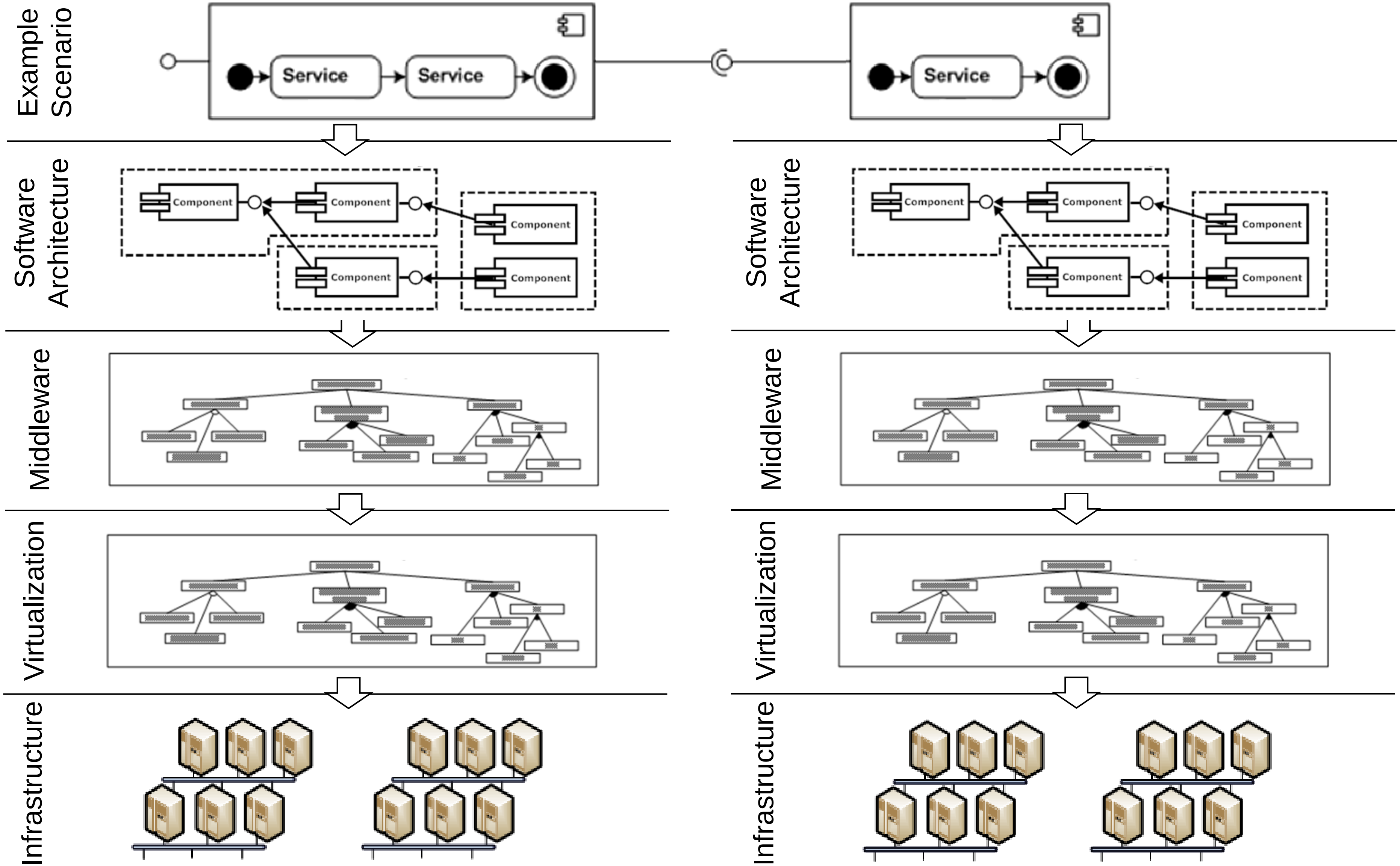
<<LoopAction>>
Loop Iteration Number =
[(1;0.55) (2;0.11)...]

<<SystemCallAction>>
sellInventory

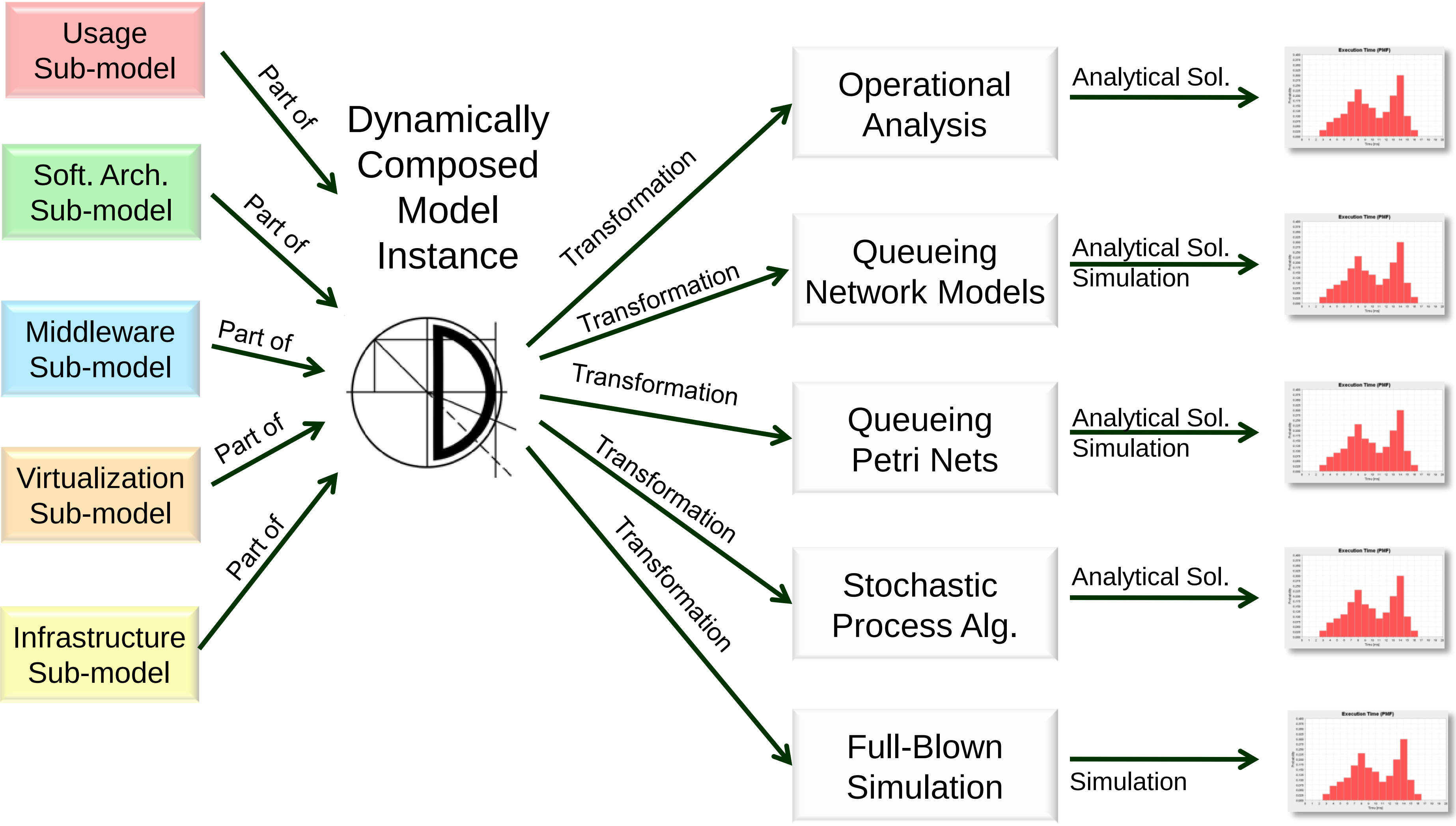
<<SystemCallAction>>
home

F. Brosig, N. Huber, and S. Kounev. **Modeling Parameter and Context Dependencies in Online Architecture-Level Performance Models**. *15th ACM SIGSOFT Intl. Symposium on Component Based Software Engineering (CBSE 2012),* June 26-28, 2012.

Prediction Method:
Step 1: Dynamic Model Composition



Prediction Method:
Step 2: Tailored Model-to-Model Transformation



Usage Sub-model

Soft. Arch. Sub-model

Middleware Sub-model

Virtualization Sub-model

Infrastructure Sub-model

Part of

Dynamically Composed Model Instance

Transformation

Operational Analysis — Analytical Sol.

Queueing Network Models — Analytical Sol. Simulation

Queueing Petri Nets — Analytical Sol. Simulation

Stochastic Process Alg. — Analytical Sol.

Full-Blown Simulation — Simulation

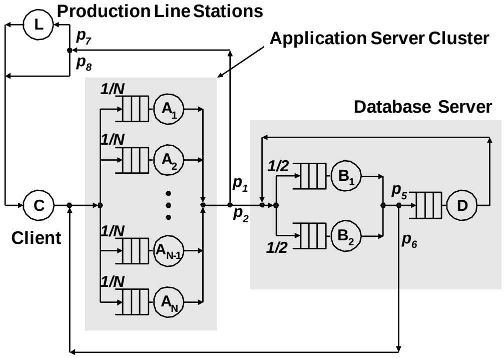
Example Transformations

Simple Bounds Analysis

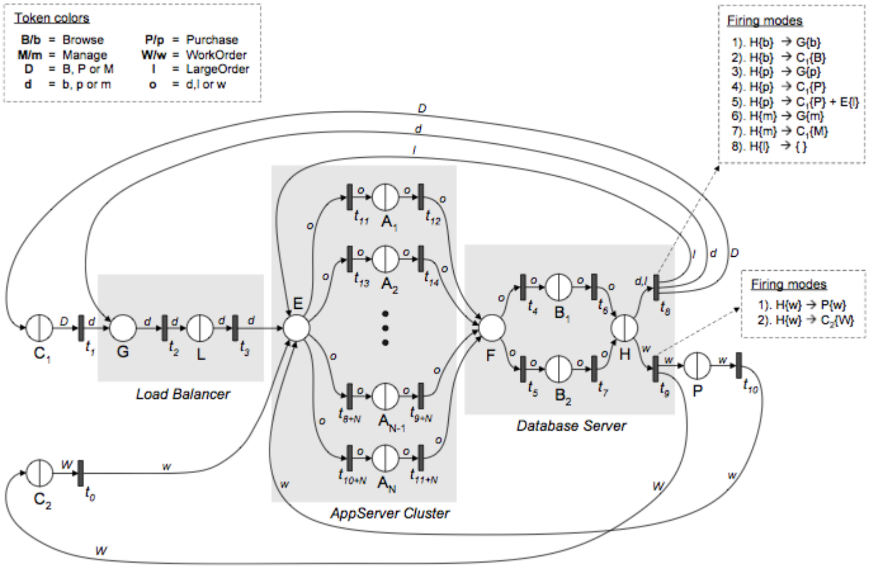
$$R \geq \max\left[N \times \max\{D_i\}, \sum_{i=1}^{K} D_i \right] \quad X_0 \leq \min\left[\frac{1}{\max\{D_i\}}, \frac{N}{\sum_{i=1}^{K} D_i} \right]$$

$$\frac{N}{\max\{D_i\}[K+N-1]} \leq X_0 \leq \frac{N}{avg\{D_i\}[K+N-1]}$$

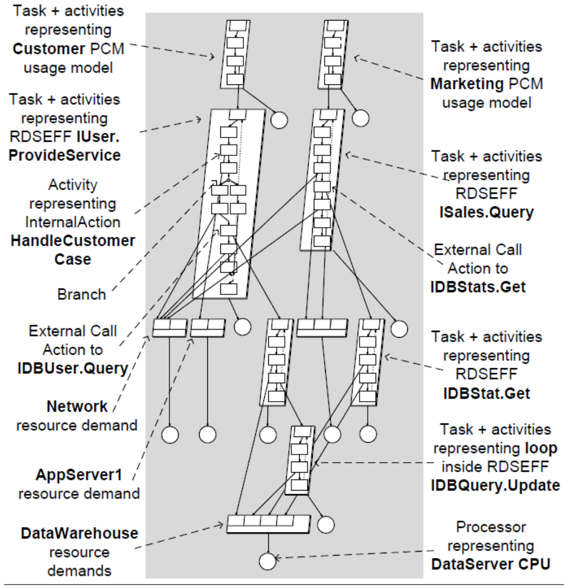
Queueing Network Model (Product Form)



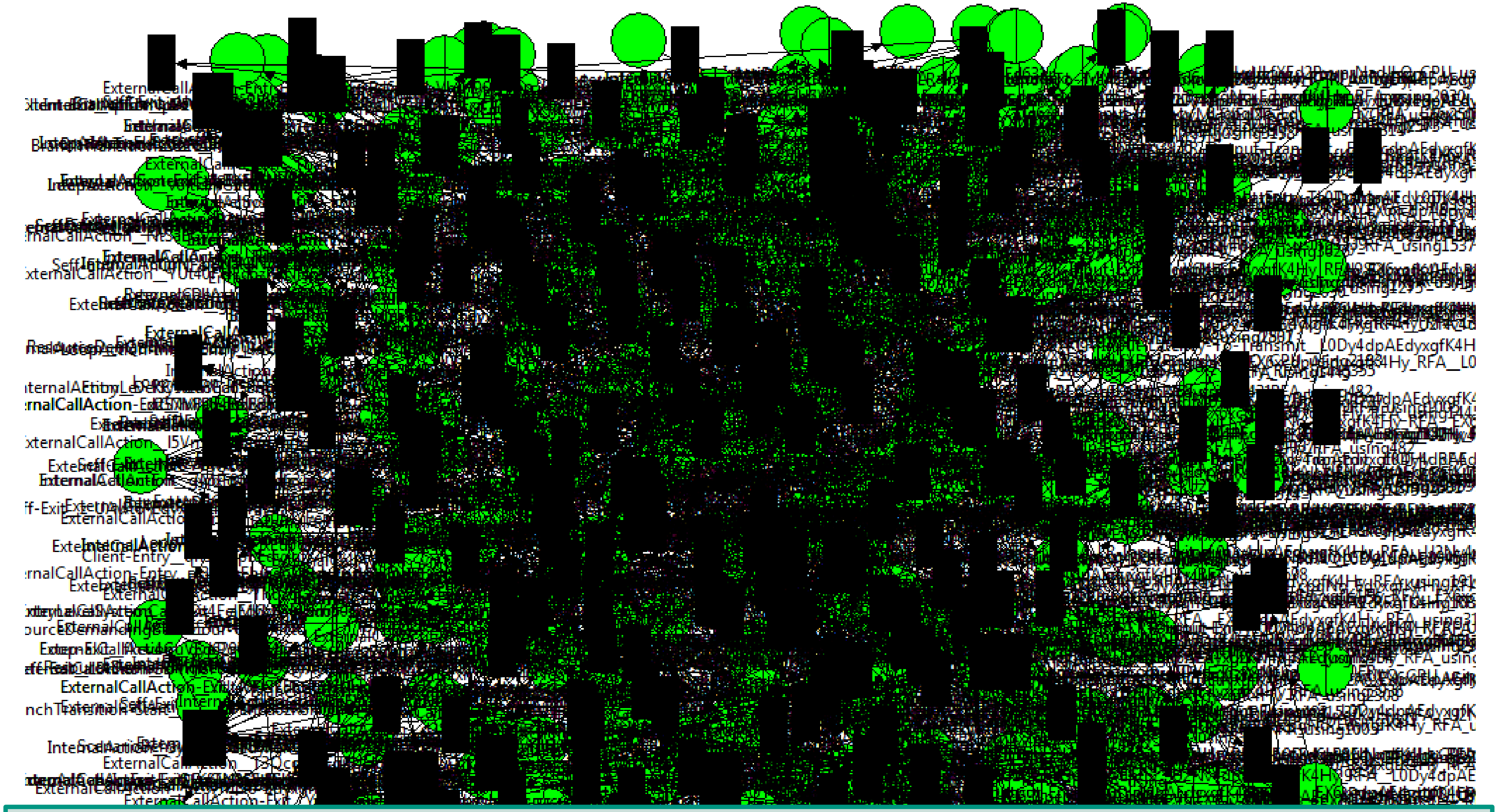
Queueing Petri Net (QPN) Model



Layered Queueing Network (LQN) Model



Case Study: Process Control System (ABB)



P. Meier, S. Kounev and H. Koziolek. **Automated Transformation of Palladio Component Models to Queueing Petri Nets**. *19th IEEE/ACM International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS 2011)*, Singapore, July 25-27, 2011.

Modeling with Queueing Petri Nets

- Modeling methodology [TSE 2006]
- Efficient discrete event simulation [PerfEval 2006]
- Modeling tool
  - “Queueing Petri net Modeling Environment” (QPME)
  - “Eclipse Public License (EPL) v1.0”
  - Distributed under 130 organizations worldwide
  - Website: http://qpme.sourceforge.net/
  - Further details:
    - [Petri Nets 2012] [LNCS 6462] [PER 2009] [QEST 2006]

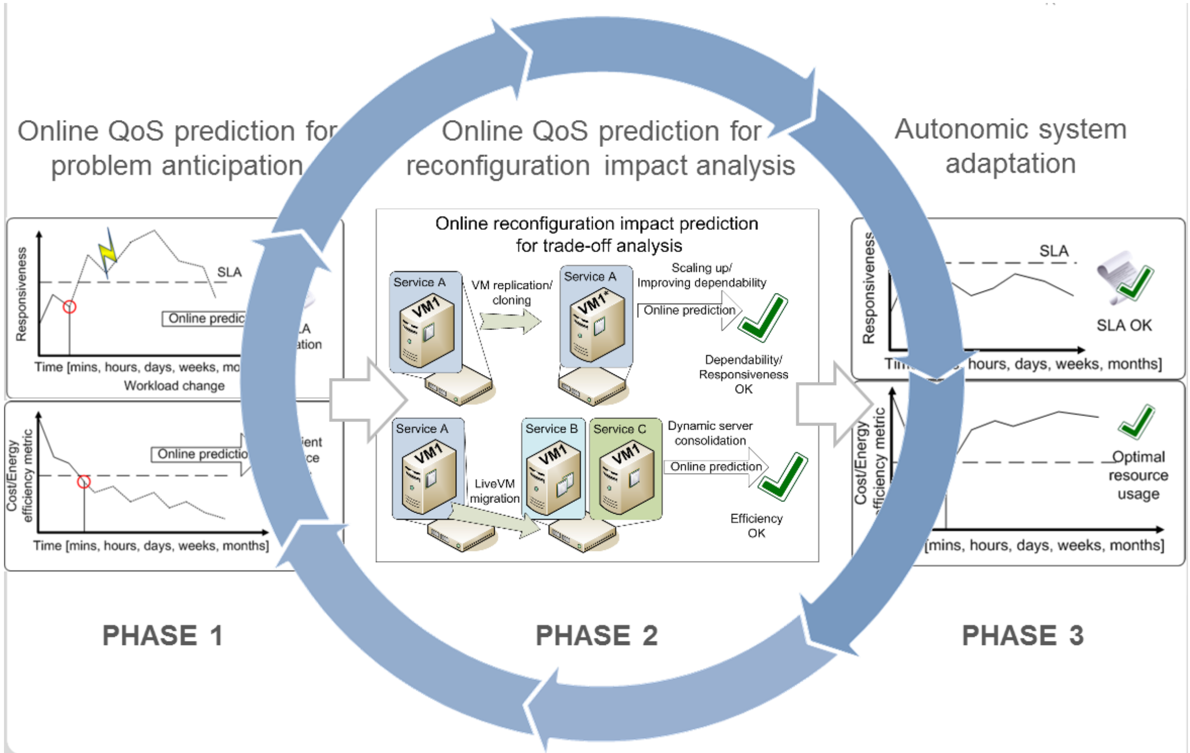
S. Kounev. **Performance Modeling and Evaluation of Distributed Component-Based Systems using Queueing Petri Nets**. *IEEE Transactions on Software Engineering (TSE)*, 32(7):486-502, July 2006.

S. Kounev and A. Buchmann. **SimQPN - a tool and methodology for analyzing queueing Petri net models by means of simulation**. *Performance Evaluation*, 63(4-5):364-394, May 2006.

Case Studies (Selection)

- **Java EE-based systems**
  - [IEEE Trans. on SE 2006] [Elsevier PerfEval 2006]
  - [IEEE ISPASS]
- **Enterprise data fabrics**
  - [ICST SIMUTools 2011]
- **Enterprise Grid Environments**
  - [Elsevier JSS 2009] [VALUETOOLS 2007]
- **Message-oriented systems**
  - [Springer SoSyM 2012]
- **Distributed event-based systems**
  - [IEEE ISORC 2008] [Springer SoSyM 2012]
- **Component-based software architectures**
  - [IEEE MASCOTS 2012] [Elsevier SciCo 2012]

Empirical Validation ("Proof-of-Concept")

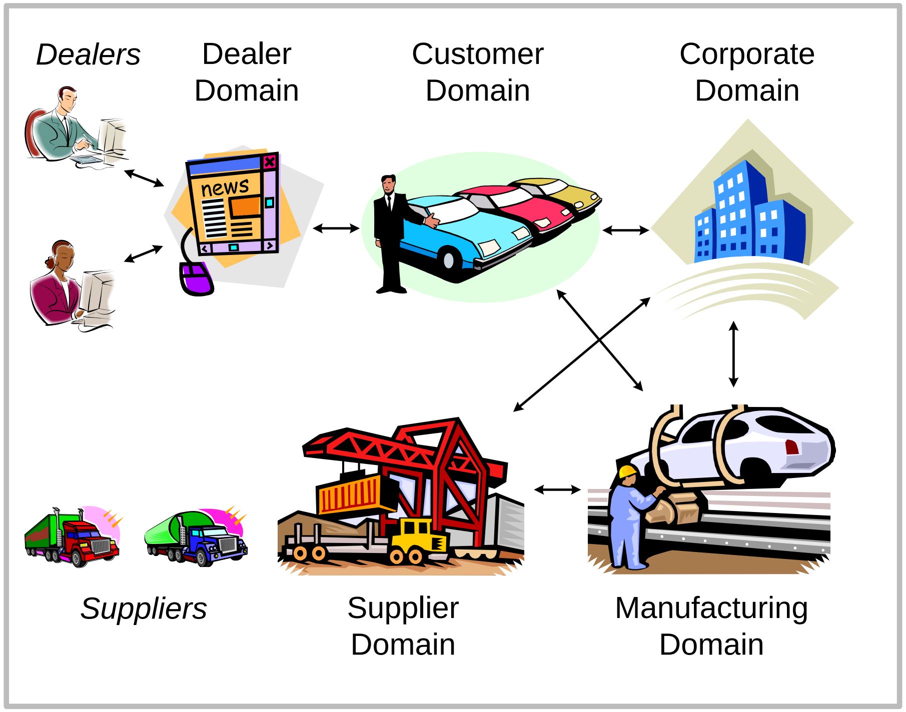


F. Brosig, N. Huber and S. Kounev. **Automated Extraction of Architecture-Level Performance Models of Distributed Component-Based Systems**. 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011), Oread, Lawrence, Kansas, November 2011.

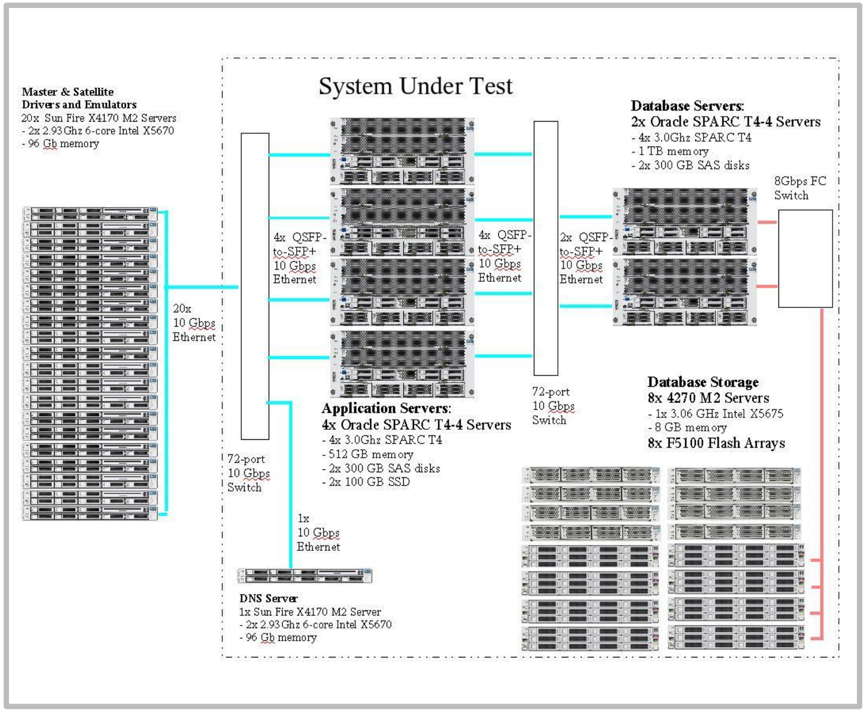
N. Huber, F. Brosig, and S. Kounev. **Model-based Self-Adaptive Resource Allocation in Virtualized Environments**. In 6th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS 2011), Honolulu, HI, USA, May 23-24, 2011.

Case Study: SPECjEnterprise2010

Business Logic



Example Deployment (Oracle)

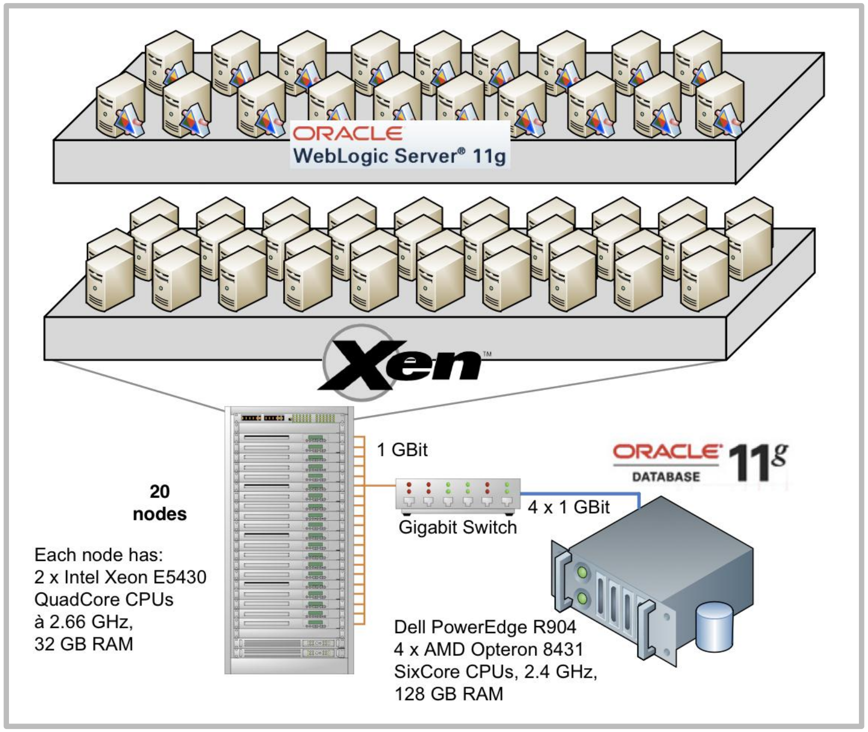


- Customer Relationship Management (CRM)
- Manufacturing
- Supply Chain Management (SCM)

- SPARC T4-4 Server + Sun Fire X4270 M2
- 444 CPU-Cores @ 3 GHz
- Oracle WebLogic + Database Server 11g

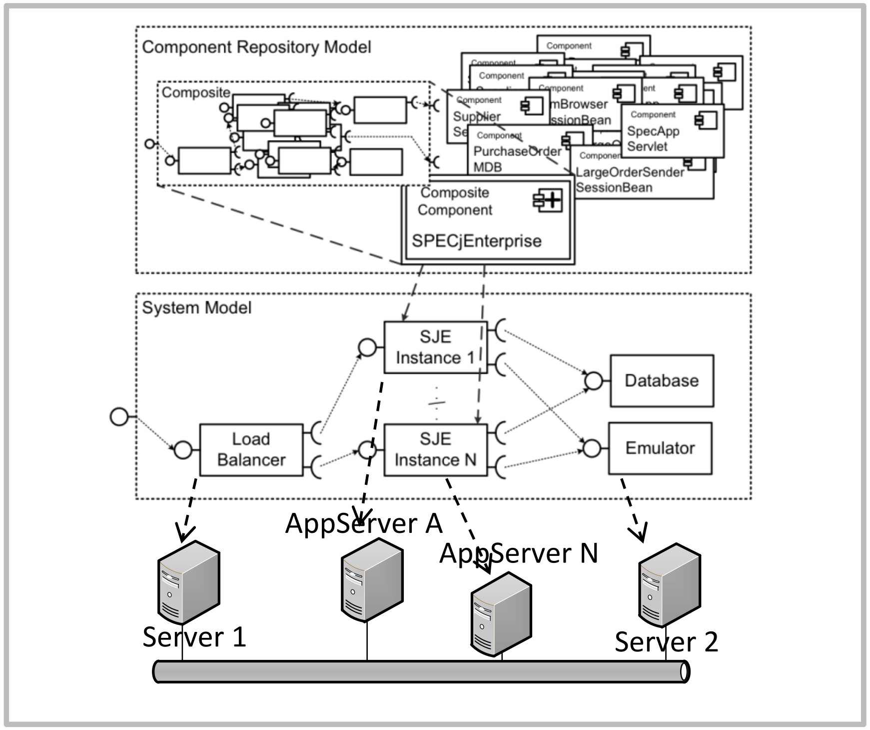
Scenario

Experimental environment at KIT



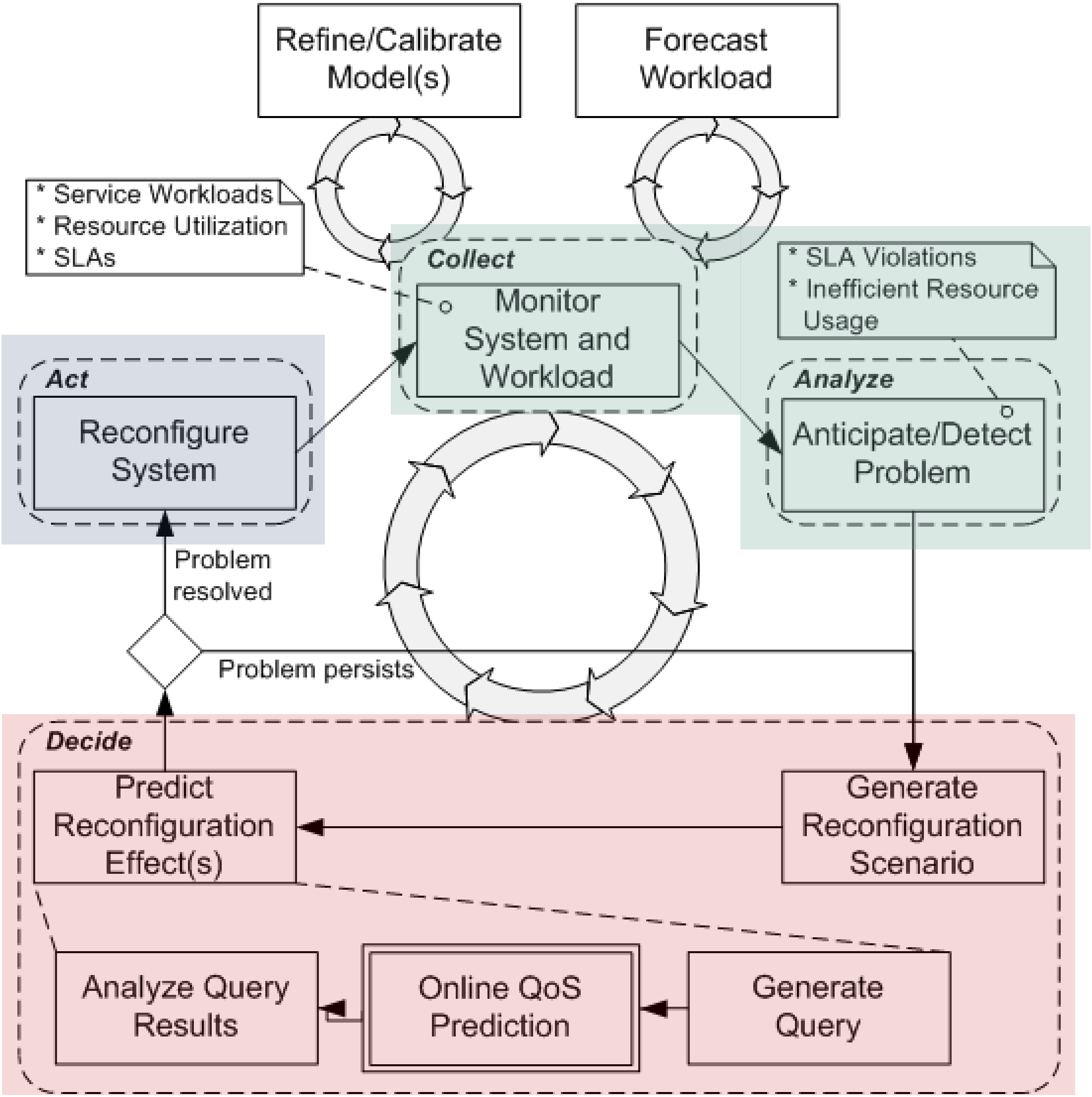
- AppServer up to 20 nodes
  - 8 CPU cores per server
- Database server
  - 24 CPU cores

High-level architecture model overview



- 28 software components
- 63 behavior specifications
  - Control flow and data flow
  - Service resource demands
  - Parameteric dependencies

System Control Loop

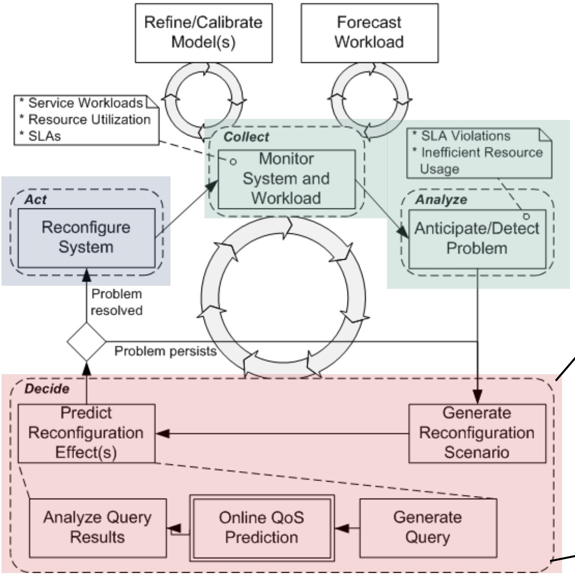


Phase 1

Phase 3

Phase 2

System Control Loop



Decision phase

PUSH

- Add resources till SLAs are fulfilled
- vCPUs & AppServer cluster nodes

PULL

- Release resources as long as no SLAs are violated

PUSH

```
while $\exists c \in \widetilde{C} : \neg P_R(c)$ do
    for all $t \in V(c[s]) : \neg P_U(t)$ do
        while $cap(c,t) \leq \overline{cap}(c,t)$ do
            if $\exists i \in F(c[s],t) : i[\kappa] < i[\overline{\kappa}]$ then
                $i[\kappa] \leftarrow i[\kappa] + 1$
            else
                $F(c[s],t) \leftarrow F(c[s],t) \cup \{\hat{i}\}$
            end if
        end while
    end for
end while
```

PULL

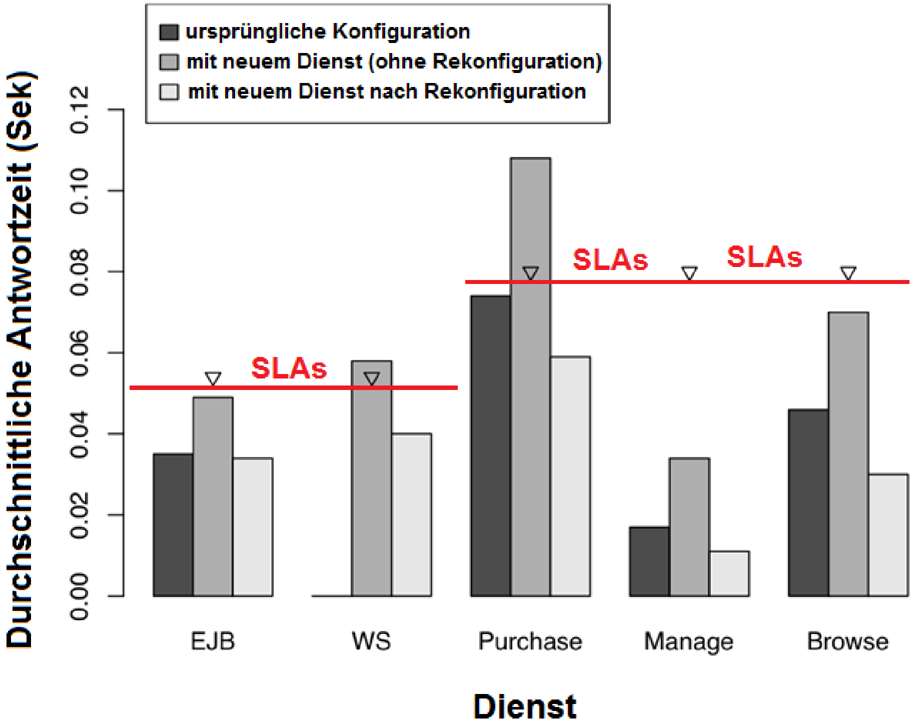
```
for all $c \in C$ do
    while $\exists t \in V(c[s]) : \overline{U}(t) - U(t) \geq \epsilon$ do
        if $\exists i \in F(c[s],t) : i[\kappa] > 0$ then
            $i[\kappa] \leftarrow i[\kappa] - 1$
            if $\neg P_R(c)$ then
                $i[\kappa] \leftarrow i[\kappa] + 1$
            end if
            if $i[\kappa] = 0$ then
                $F(c[s],t) \leftarrow F(c[s],t) \setminus \{i\}$
            end if
        end if
    end while
end for
```

Evaluation

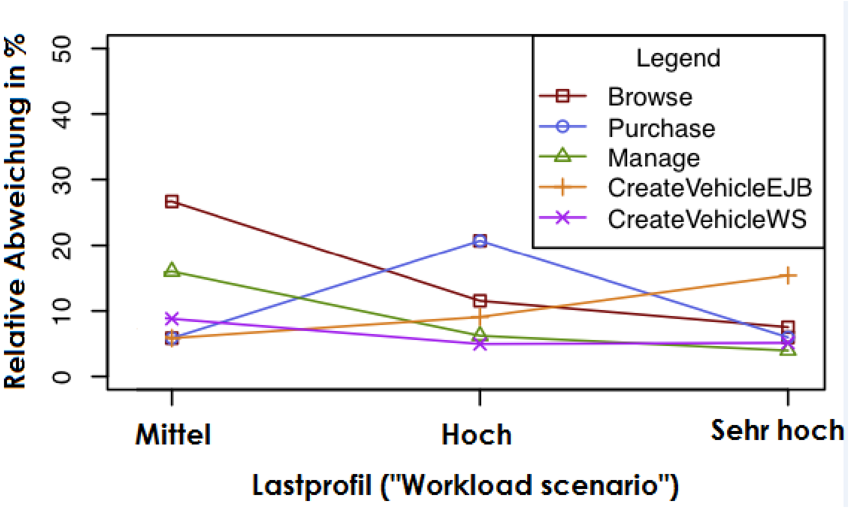
Comparison of the model predictions with measurements on the real system

Prediction error: for utilization/throughput: < 5%, for response time: up to 30%

Example scenario: Deployment of a new service



Prediction error for response time in different workload scenarios



Cooperation with VMware, Inc.

- Market leader in virtualization technology
- Cooperation since 2009
- “VMware Academic Research Award 2012”

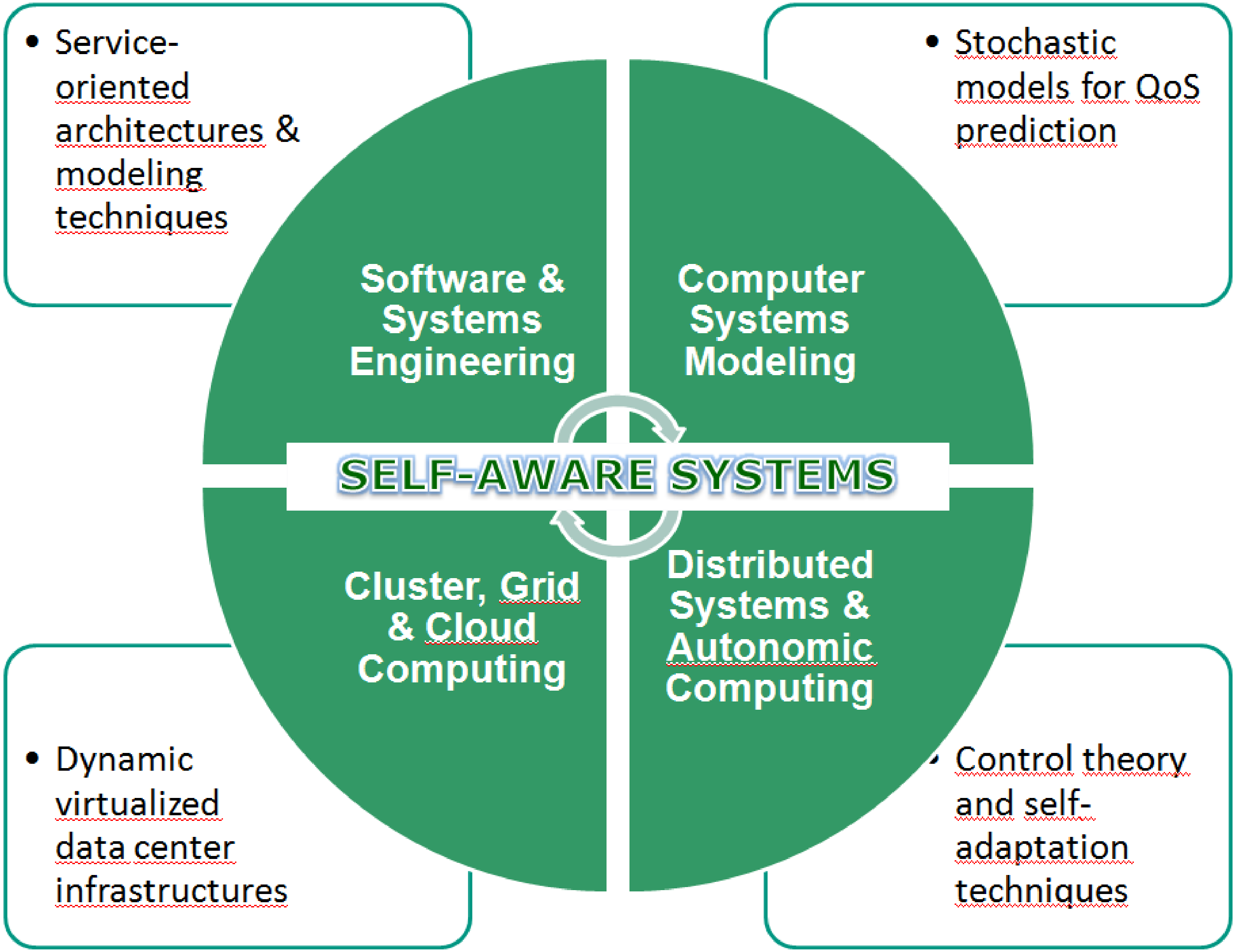
- 3 year project aiming at
  - Model-based performance and resource management
  - Integration into virtualization platforms

Self-Aware Software Systems

- **Self-Reflective**
  - Aware of their software architecture, execution environment and hardware infrastructure, as well as of their operational goals (e.g., QoS and efficiency)
- **Self-Predictive**
  - Able to anticipate and predict the effect of dynamic changes in the environment, as well as the effect of possible adaptation actions
- **Self-Adaptive**
  - Proactively adapting as the environment evolves to ensure that their operational goals are continuously met

"I think,therefore I am…"
-- René Descartes

“Self-Aware Complex Systems Engineering”



- Service-oriented architectures & modeling techniques

- Stochastic models for QoS prediction

Software & Systems Engineering

Computer Systems Modeling

SELF-AWARE SYSTEMS

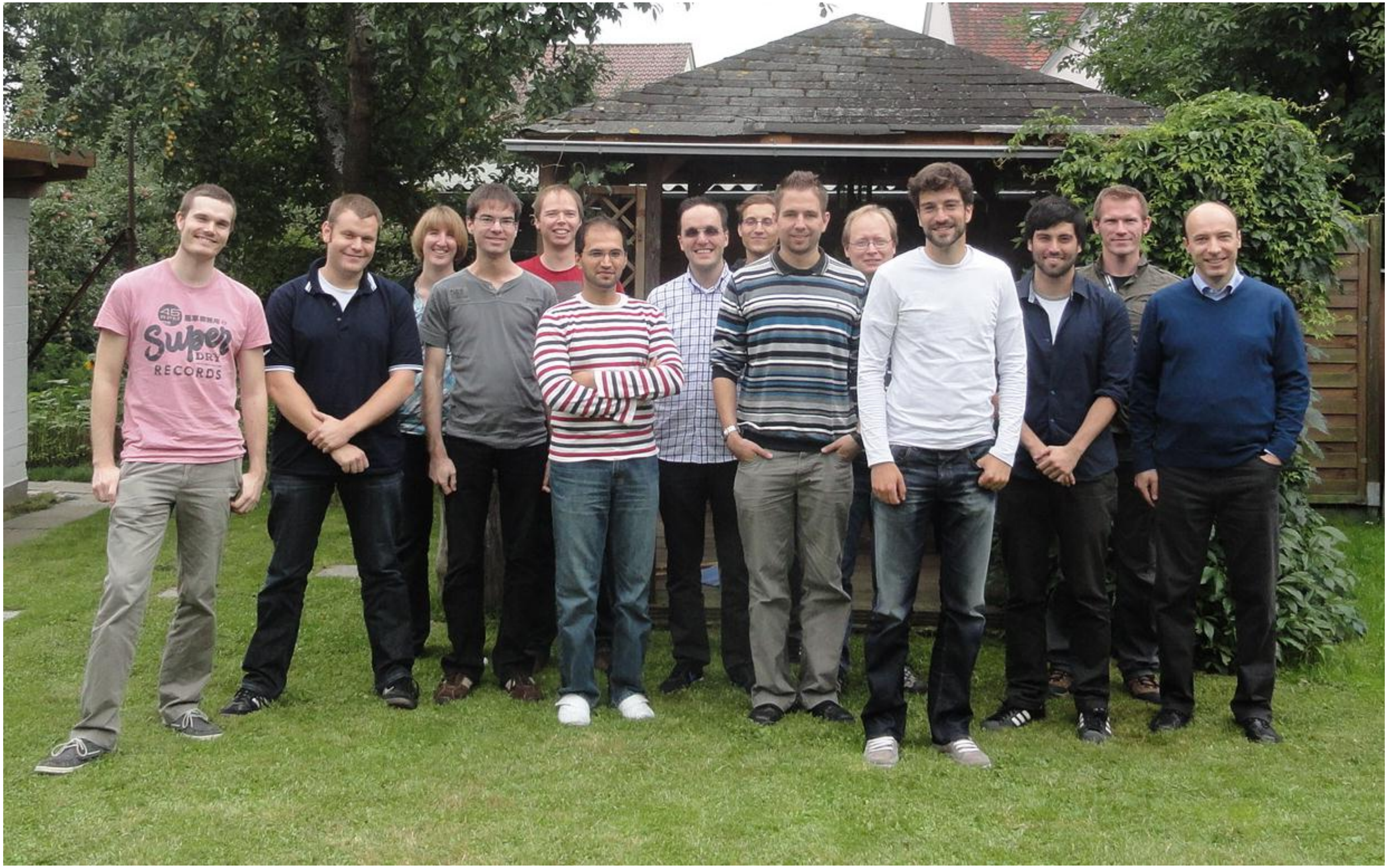
Cluster, Grid & Cloud Computing

Distributed Systems & Autonomic Computing

- Dynamic virtualized data center infrastructures

- Control theory and self-adaptation techniques

DFG-Nachwuchsgruppe “Descartes”



Vielen Dank!

http://www.descartes-research.net