TECHNISCHE UNIVERSITÄT
CAROLO-WILHELMINA
ZU BRAUNSCHWEIG

Entwurf und Implementierung einer Policy-Engine für Web-Service-basiertes Netzwerk-Management

Diplomarbeit, Abschlussvortrag

Jan Falkenstern

Betreuer: Torsten Klie

Institut für Betriebssysteme und Rechnerverbund
Technische Universität Braunschweig

Motivation

- **Netzwerke werden immer größer und komplexer**
  - Neue Anforderungen an die Administration

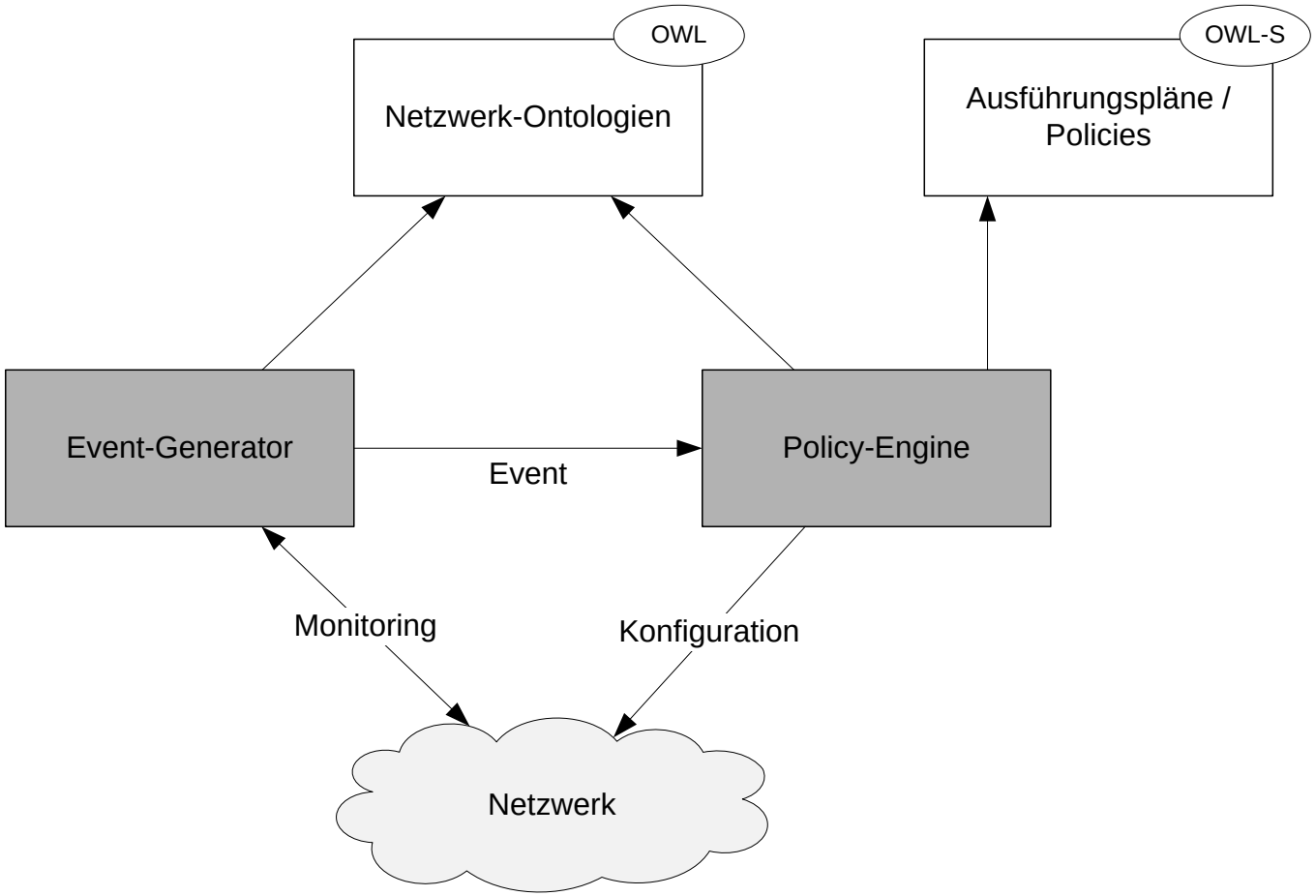
- **Eine Lösung: Policy-basiertes Management**
  - Netzwerk wird in geeigneter Form beschrieben
  - Policies definieren das gewünschte Verhalten des Netzwerkes
  - Automatisierung des Management-Prozesses
    - Administrator definiert Verhaltensregeln
    - Managementsystem entscheidet anhand der Regeln selbstständig, wie es sich beim Eintreffen von *Ereignissen* verhält.

Policy = Bedingung + Aktion

Gliederung

- **Implementierung**
  - Komponenten des Systems
  - Kommunikation der Komponenten
- **Evaluation**
  - Architektur der Testumgebung
  - Verschiedene Testszenarien
  - Komplexes Anwendungsszenario
- **Fazit**
  - Bewertung
  - Ausblick

Systemarchitektur



Implementierung des Event-Generators

- **Implementiert als Kommandozeilen-Anwendung**
  - Einfacher Start als Hintergrundprozess / Dienst
    - Java Service Wrapper
- **Konfiguration über Datei im Java Properties Stil**
  - `java –jar EventGenerator.jar config.properties`
- **Aufbau**
  - Zentrale Controller-Klasse
    - Implementiert als Singelton
    - Initiiert den Start der aktiven Tasks
  - „ServiceChecker“ implementiert als `java.util.TimerTask`

Implementierung der Policy-Engine

- **Zentrale Management-Klasse als Singelton**
  - Ermöglicht einfache Rekonfiguration zur Laufzeit
- **Funktionen**
  - Aufbauen der Knowledgebase
  - Reagieren auf Ereignisse (`performEvent`)
  - Webservice-Schnittstelle(n)
- **Management zur Laufzeit**
  - Lesen/Setzen: URI der Netzwerkontologie, Servicebeschreibung
  - Lesen/Setzen: Log-Level
  - Lesen: Anzahl der registrierten Ontologien, Individuals, Services

Kommunikation der Komponenten

- **Kommunikation auf Basis von Webservices**
  - Austausch von SOAP-Nachrichten

Architektur der Testumgebung

- **Testumgebung mit vier Hosts**
  - Host-System (policy2)
  - Monitoring-Server (nagios-soap)
  - EventGenerator (evgen)
  - EventSimulator (evsim)
- **Verschiedene überwachte Services (Nagios)**
- **Beschreibung der Testumgebung auf Basis der NINO-Ontologie**

Evaluation des implementierten Systems

- **Es folgt:**
  - Einfache Anwendungsfälle / Flexibilität
  - Robustheit
  - Skalierbarkeit

Einfache Anwendungsfälle (1/2)

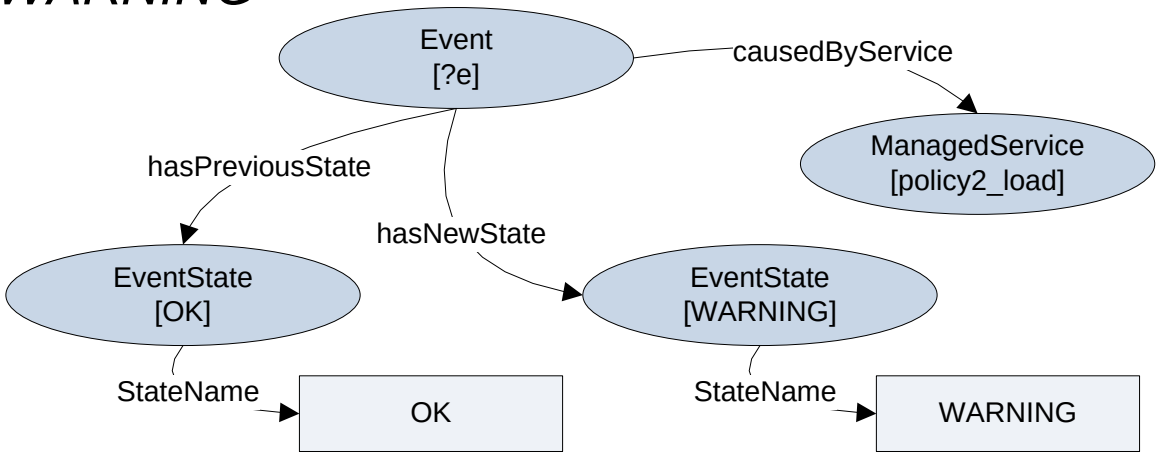
- **Motivation dieses Tests**
  - Grundlegend korrekten Funktion des Systems
  - Flexibilität des Systems

- **Test anhand von drei konkreten Testfällen**
  1. Auf Host *policy2* wechselt der Service *Current Load* vom Status *OK* in den Status *WARNING*
  2. Auf einem beliebigen Host wechselt ein Service der Kategorie *NetworkService* in den Status *CRITICAL*
  3. Auf einem beliebigen Host der Domain *ibr.cs.tu-bs.de* findet eine Statusänderung statt

Einfache Anwendungsfälle (2/2)

Formulierung als SWRL-Regel

- „Auf Host *policy2* wechselt der Service *Current Load* vom Status *OK* in den Status *WARNING*“



- Human Readable Syntax:

Event(?e) ^ causedByService(?e;HostPerformanceService_policy2_load) ^ hasPreviousState(?e;EventState_OK) ^ hasNewState(?e;EventState_WARNING) →

Robustheit

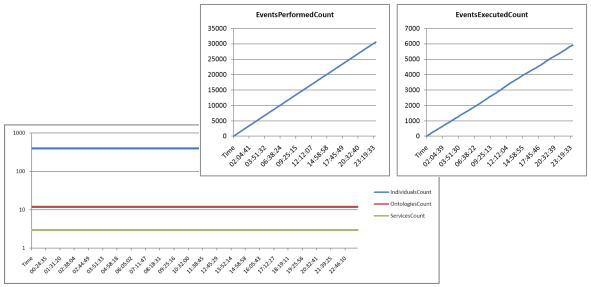
- **Betrachtet werden Probleme der**
  - Kommunikation
    - Host vorübergehend nicht erreichbar
    - Ontologiebeschreibung nicht erreichbar
    - …
  - Infrastruktur
    - PolicyEngine wird vom AppServer nicht bereitgestellt
  - Konfiguration
    - Webservice-Schnittstelle nicht verfügbar
    - Ungültige URL für Netzwerk- und Ontologie-Beschreibung
    - …

Skalierbarkeit (1/2)

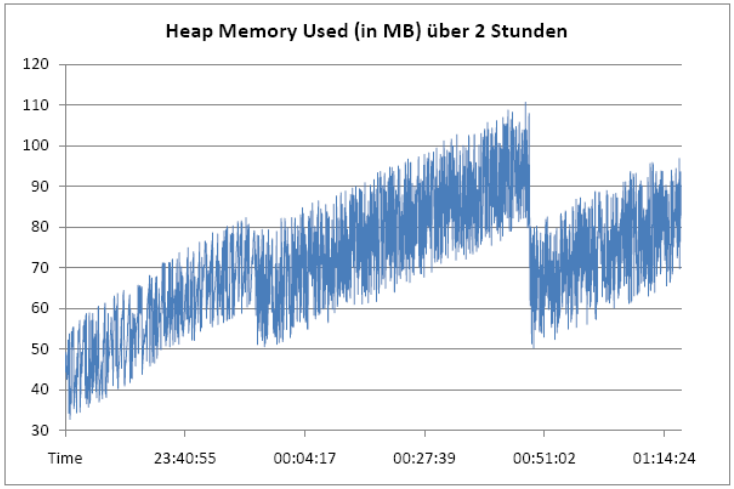
- **Testaufbau**
  - Zufällige Events werden in kurzer Folge generiert und an die Policy-Engine übergeben
- **Auswertung**
  - Dauer des Tests: 25h 29m 13s
  - 30.648 Events wurden generiert
  - Ausgeführte Services (Vorbedingung erfüllt)
    - 102 x Testcase1
    - 933 x Testcase2
    - 4891 x Testcase3

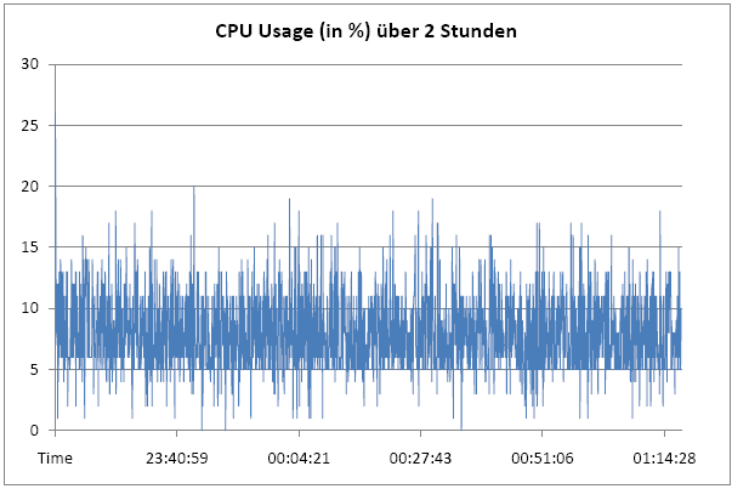
Skalierbarkeit (2/2)

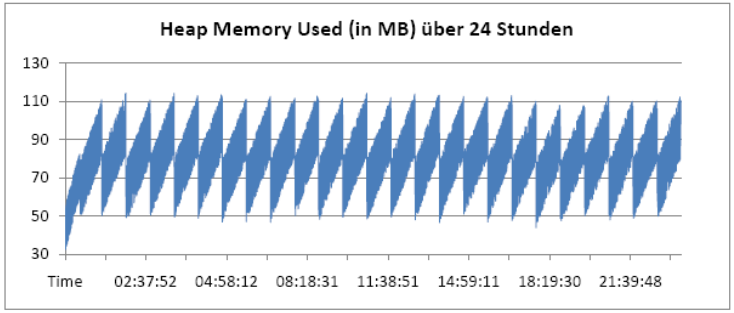
- **Überwachte Performanceparameter**
  - Innerhalb der JVM
    - HeapMemoryUsed
    - LiveThreads
    - Classes Loaded
    - CPU Usage
  - Leistungsparameter der PolicyEngine
    - EventsPerformedCount
    - EventsExecutedCount
    - LastExecutionTime
    - IndividualsCount, OntologiesCount, ServicesCount

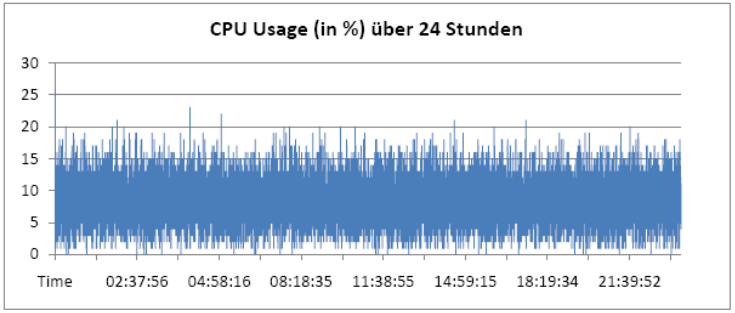


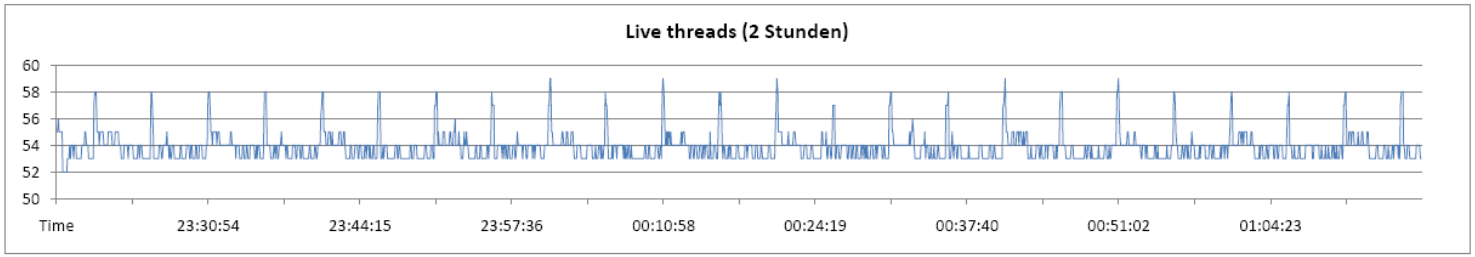
Skalierbarkeit (2/2)











Komplexes Anwendungsszenario

- **Motivation**
  - Darstellung eines komplexeren und möglichst realistischen möglichen Anwendungsszenarios

Traffic-Balancing auf verschiedenen Netzwerk-Segmenten innerhalb des Weitverkehrsnetzes (WAN) eines Unternehmens
Multiple Spanning Tree Protocol (MSTP)

- **Szenario**
  - Unternehmen mit 3 Standorten
  - Daten und VoIP in separaten VLANs im gleichen WAN
  - Negative Beeinflussung der VoIP-Pakete soll verhindert werden
- **Lösung**
  - Routing der einzelnen VLANs wird flexibel variiert

Bewertung und Ausblick

- **Wichtigste Eigenschaft des vorgestellten Systems**
  - Einfache Integration in heterogene Systemlandschaften
  - Hauptargument: Flexibilität
  - Sämtliche Entscheidungen werden auf Basis einer systemweisen Wissensbasis (Ontologie) getroffen
  - Nicht an NINO gebunden
- **Weitere Arbeit**
  - Verzicht auf Event-Generator
  - Zwischenspeichern versendeter Events
    - WS-ReliableMessaging
  - Verschlüsselung / Authentifizierung der Kommunikationspartner
    - WS-Security

Zusammenfassung

- **Konzepte des Semantic Web**
  - Beschreibung des Netzwerkes als Ontologie (NINO)
  - Definition der Ereignisse als OWL-Instanz
  - Definition der Ausführungspläne als OWL-S Service
- **Implementierung**
  - Event-Generator und Policy-Engine
  - Kommunikation über Webservices/SOAP
- **Evaluation**
  - System ist flexibel, korrekt und erweiterbar

Ende

Vielen Dank für Ihre Aufmerksamkeit!

Gibt es Fragen?

Webseite: www.ibr.cs.tu-bs.de/theses/tklie/policy-engine

Verwendete Technologien

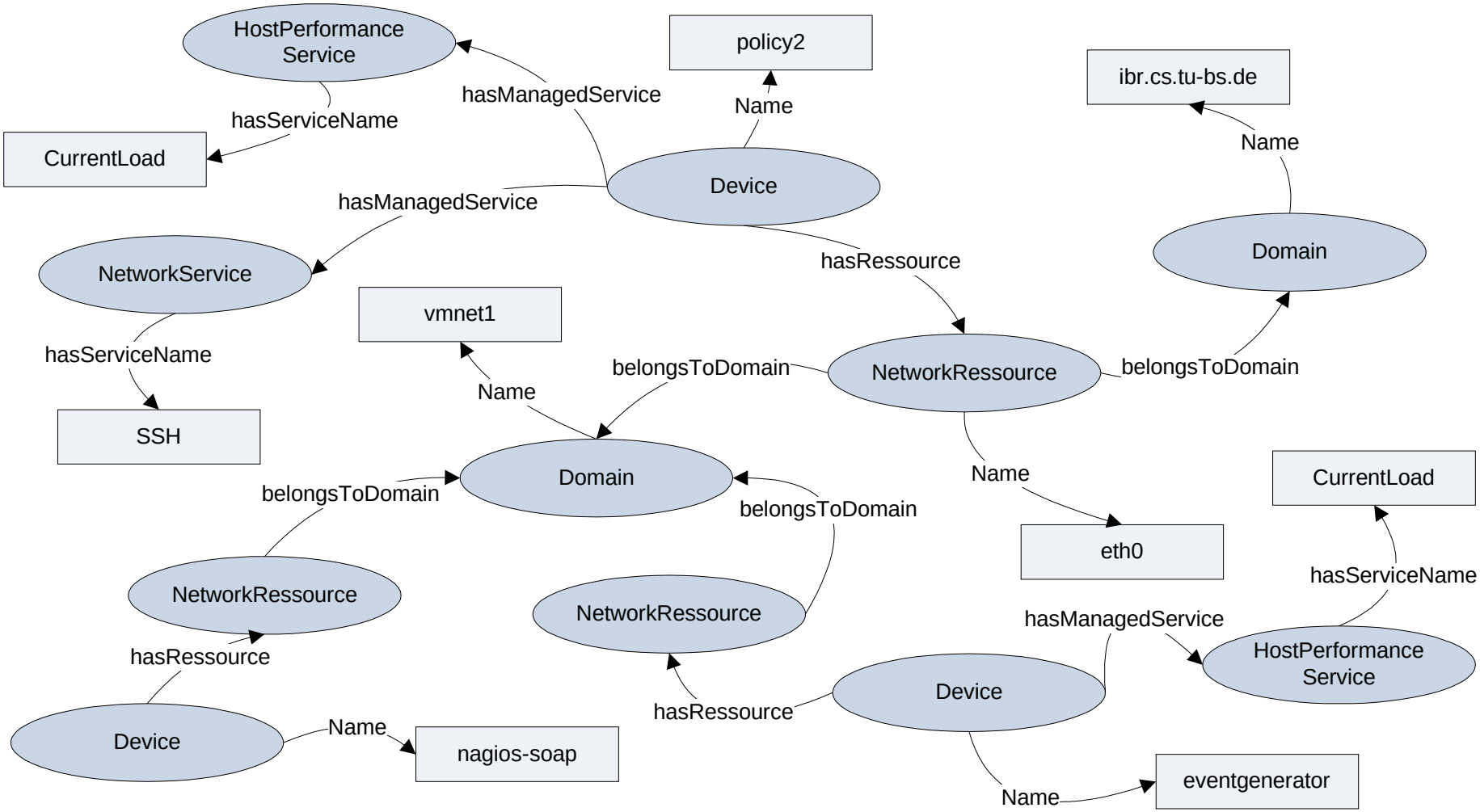
- **Java 6**
  - Annotationen (erweitert gegenüber Java 5)
    - Kennzeichnung einer Methode als Webservice
  - JMX (Java Management Extensions)

- **Applikationsserver Glassfish**
  - Java EE 5 Referenzimplementierung
    - Java API for Webservices (JAX-WS 2.0)
  - Servletcontainer: Apache Tomcat

Verwendete Bibliotheken

- **Jena**
  - Open Source Framework
  - Entwicklung von Semantic Web Applikationen unter Java
  - Umfangreiche RDF- und OWL-API
- **OWL-S API**
  - Umfangreiche API zum Lesen, Schreiben und Ausführen von OWL-S Services
  - Execution Engine erlaubt die Ausführung einfacher und zusammengesetzter Services
- **Apache log4j**
  - Umfassende Logging-Mechanismen für Java

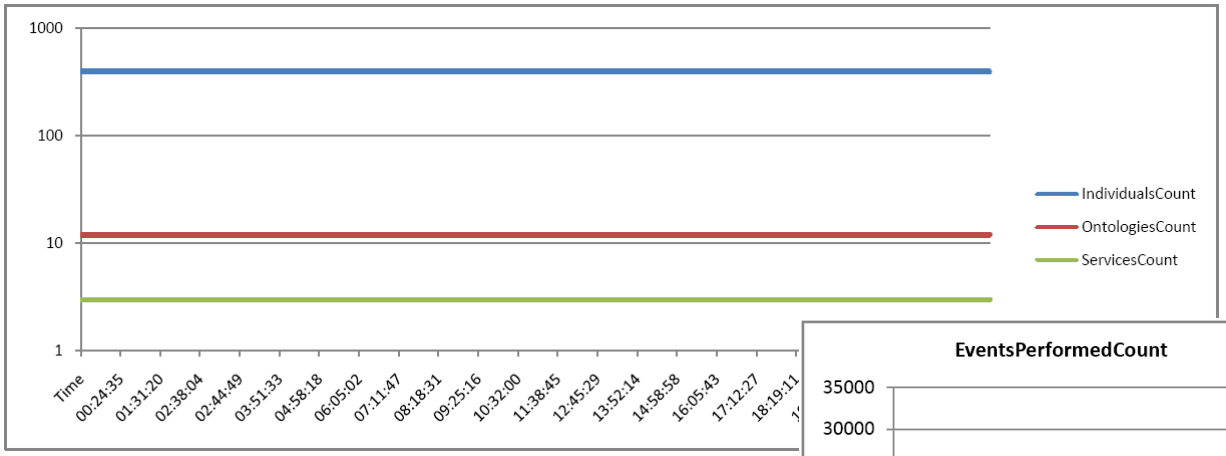
Architektur der Testumgebung (2/2)

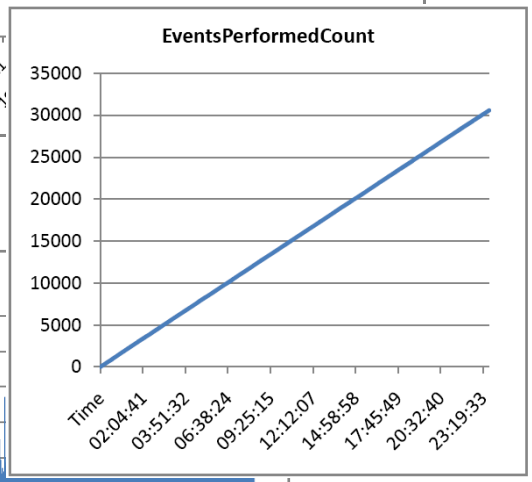


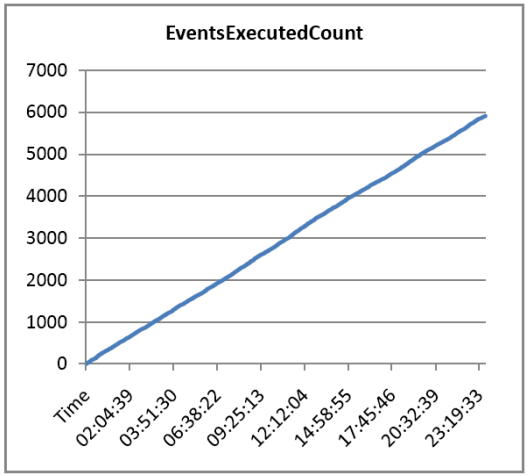
Skalierbarkeit (1/3)

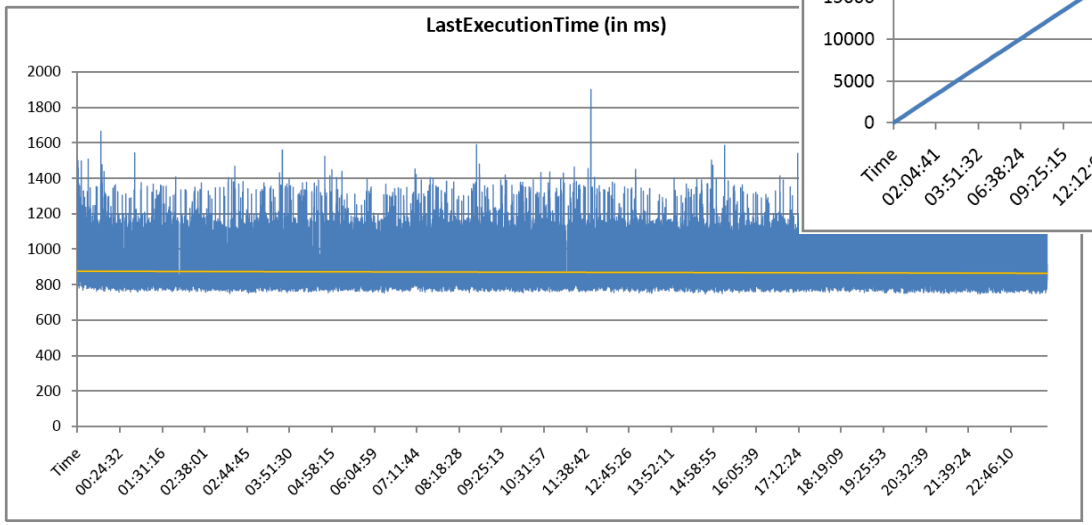
- **Testaufbau**
  - Zufällige Events werden in kurzer Folge generiert und an die Policy-Engine übergeben
- **Algorithmus**
  1) Erfrage eine Liste aller Hosts und Services von der Nagios-Schnittstelle
  2) Wähle zufällig einen Host aus der Liste
  3) Wähle zufällig einen Service für den im letzten Schritt gewählten Host
  4) Wähle zufällig zwei (unterschiedliche) Status
  5) Generiere auf Basis der in Schritt 2-4 bestimmten Werte ein Event und sende dieses an die Policy-Engine
  6) Warte eine zufällige Zeit zwischen 0 und 4000 Millisekunden und beginne wieder bei Schritt 1

Skalierbarkeit (3/3)









Vergleich mit anderen Arbeiten

- **Ponder2 Framework**
  - Universelles und erweiterbares Objakt-Management-Framework
  - Sehr umfangreich, aber eigene Sprache notwendig
- **Web Services Policy Framework (WS-Policy)**
  - Betrachtet das Management *von* Webservices
  - Notation innerhalb der WSDL-Beschreibung
  - Schwerpunkt: Sicherheitspolicies
- **Policy Management für Autonomic Computing (PMAC)**
  - Kontext: „*Autonomic Computing*“ (IBM)
    Selbstkonfiguration, -heilung, -optimierung, -schutz von IT-Systemen
  - 2 Sprachen zur Definition der Policies
  - Wird aktuell nicht mehr weiterentwickelt