TECHNISCHE UNIVERSITÄT
CAROLO-WILHELMINA
ZU BRAUNSCHWEIG

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

Diplomarbeit, Zwischenvortrag

Jan Falkenstern

Betreuer: Torsten Klie

Institut für Betriebssysteme und Rechnerverbund
Technische Universität Braunschweig

Gliederung

- **Motivation**
  - Policy-basiertes Management
- **Aufgabenstellung**
- **Grundlagen**
  - Semantic Web
- **Anforderungen an das System**
- **Architektur / Design**
- **Implementierung**
- **Fazit**

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

Aufgabenstellung

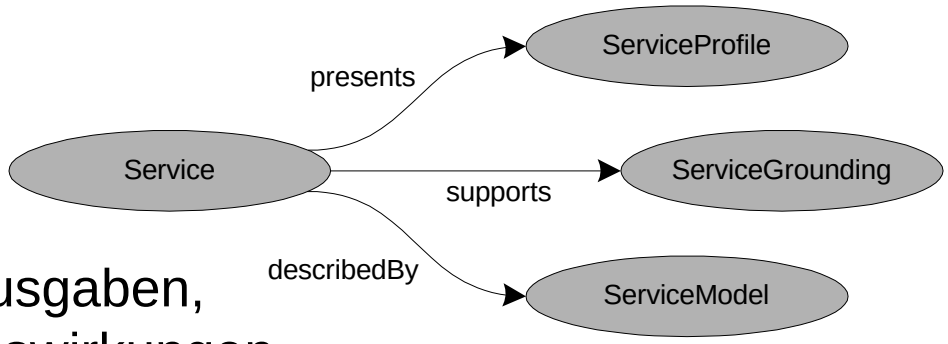
- **Entwurf und Entwicklung eines Policy-basierten Management-Systems**
  - Schnittstellen: Web-Services
  - Eingabe: verfeinerte Policies in Form von Web-Service Ausführungsplänen → OWL-S
- **Schwerpunkte**
  - Anbindung der Monitoring- und Konfigurations-Web-Services
  - Verwalten von Ereignissen
  - Reagieren auf Ereignisse

Grundlagen: Semantic Web

- **Idee**
  - Informationen bereits bei der Speicherung mit Bedeutung versehen
- **Basistechnologien des Semantic Web**
  - URI, XML
  - RDF (*Ressource Description Framework*)
    - Formale Sprache zur Beschreibung von Metadaten
    - Datenrepräsentation durch einen gerichteten Graphen mit Knoten und Kanten
    - Syntax: Beschreibung anhand der Kanten → RDF-Tripel
  - OWL (*Ontology Web Language*)
    - Eine Ontologie besteht aus Klassen, Eigenschaften (Rollen) und Individuen die zueinander in Beziehung stehen
  - OWL-S

Grundlagen: OWL-S

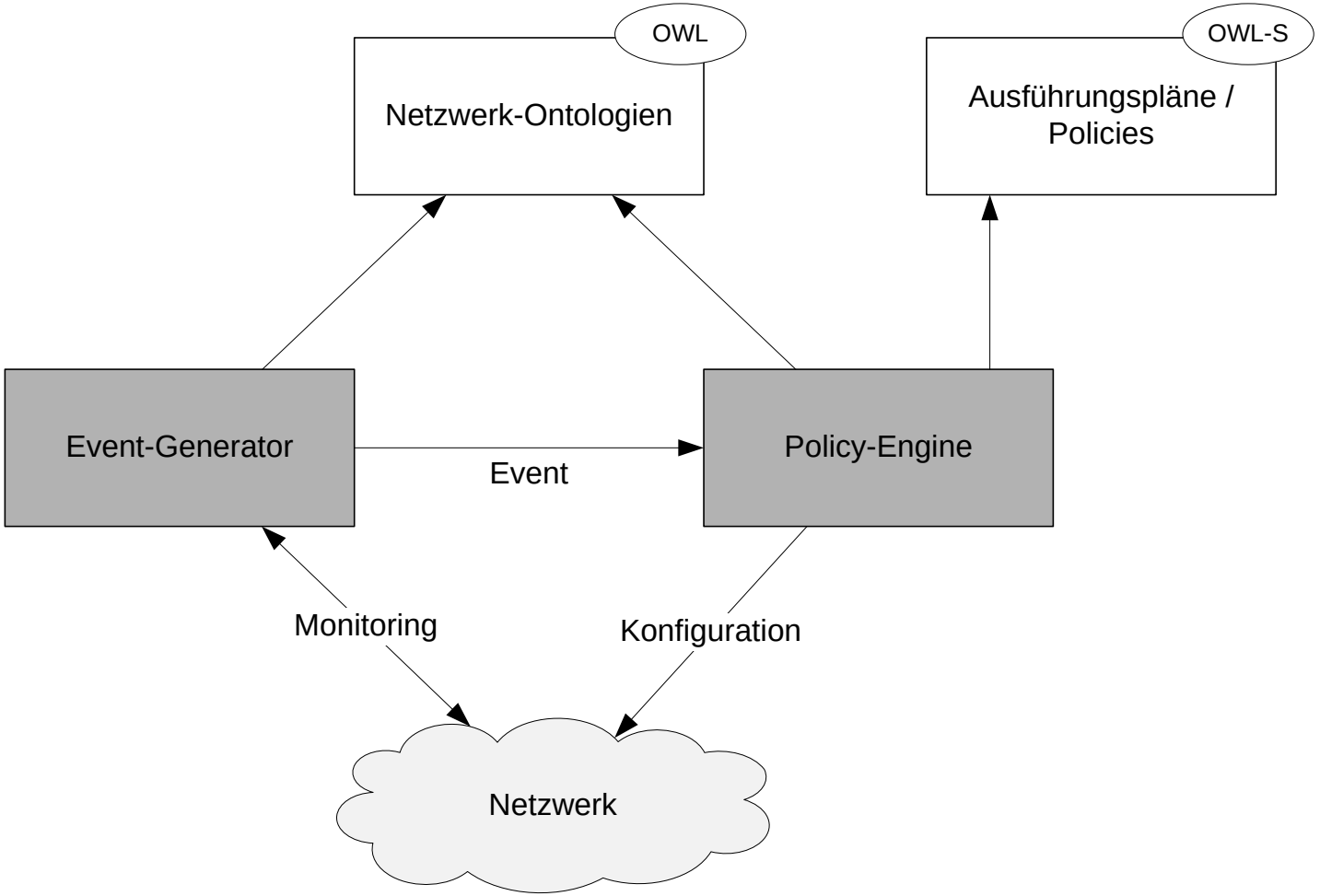
- **OWL-S ermöglicht automatisches**
  - Auffinden, Ausführen, Zusammensetzen und Überwachen von Web-Services.
- **Beschreibung anhand von**
  - ServiceProfile
    - „Was tut der Service“
    - Beschreibt Eingaben, Ausgaben, Vorbedingungen und Auswirkungen
  - ServiceModel
    - „Wie arbeitet der Service“
  - ServiceGrounding
    - „Wie wird der Service aufgerufen“



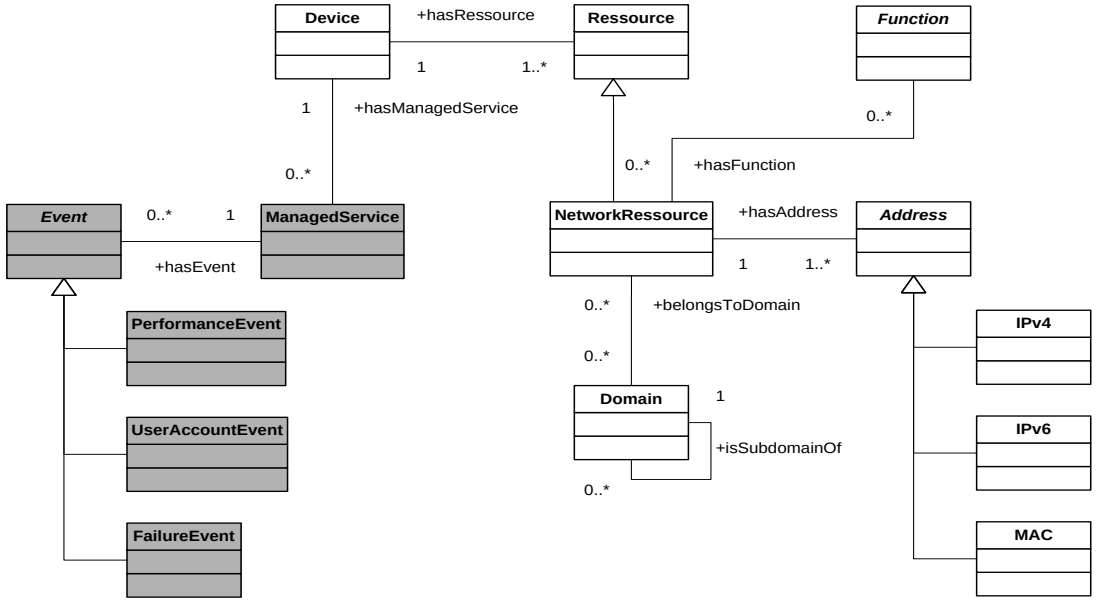
Anforderungsanalyse

- **Beschreibung des Netzwerkes, der Ereignisse und der Reaktionen**
  - Möglichst außerhalb des Quellcodes
- **Schnittstelle zum Melden von Ereignissen**
  - Implementiert als Webservice
  - Möglichst frei zugänglich → bestehende Konzepte
- **Skalierbarkeit**
- **Flexibilität**

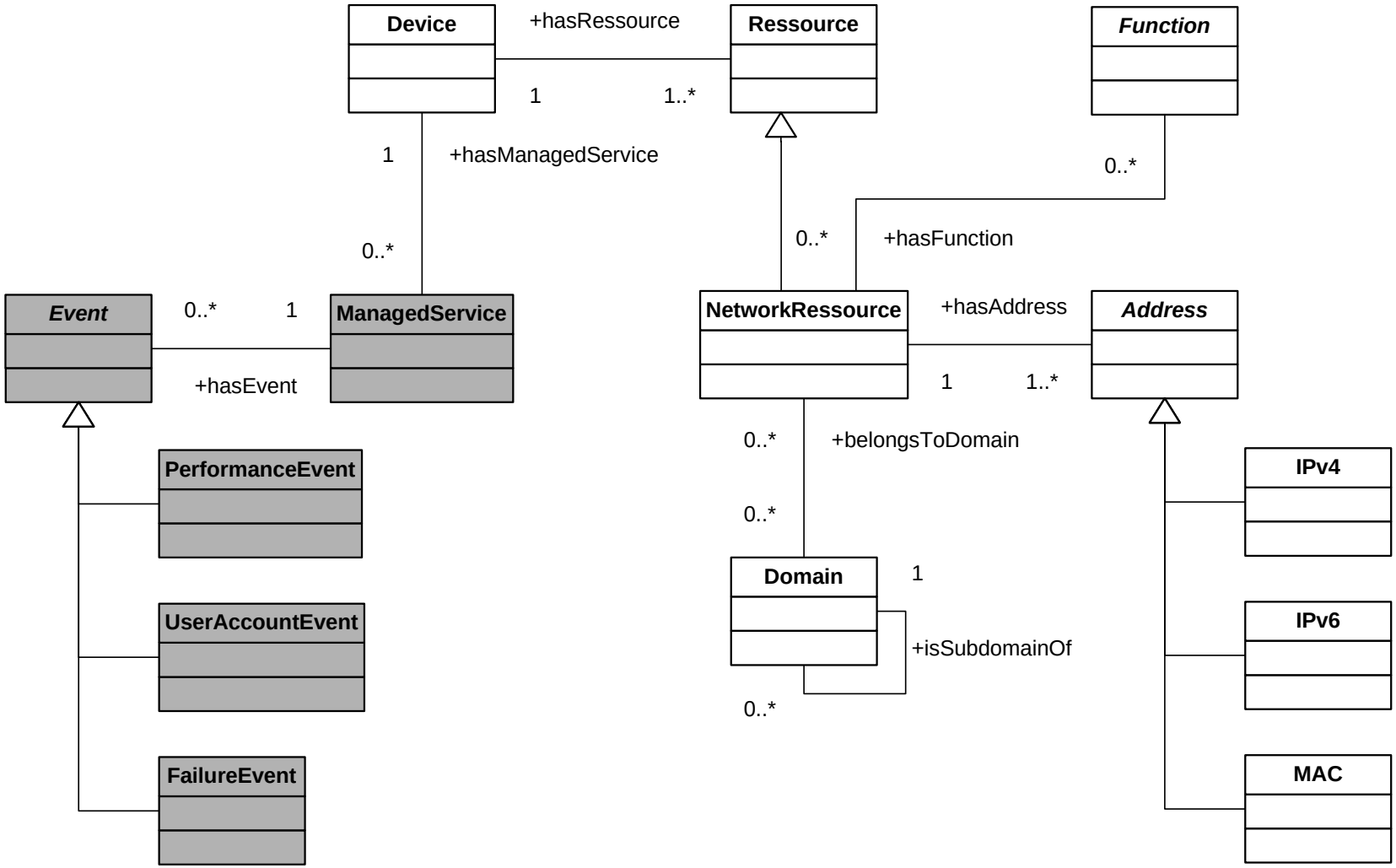
Systemarchitektur



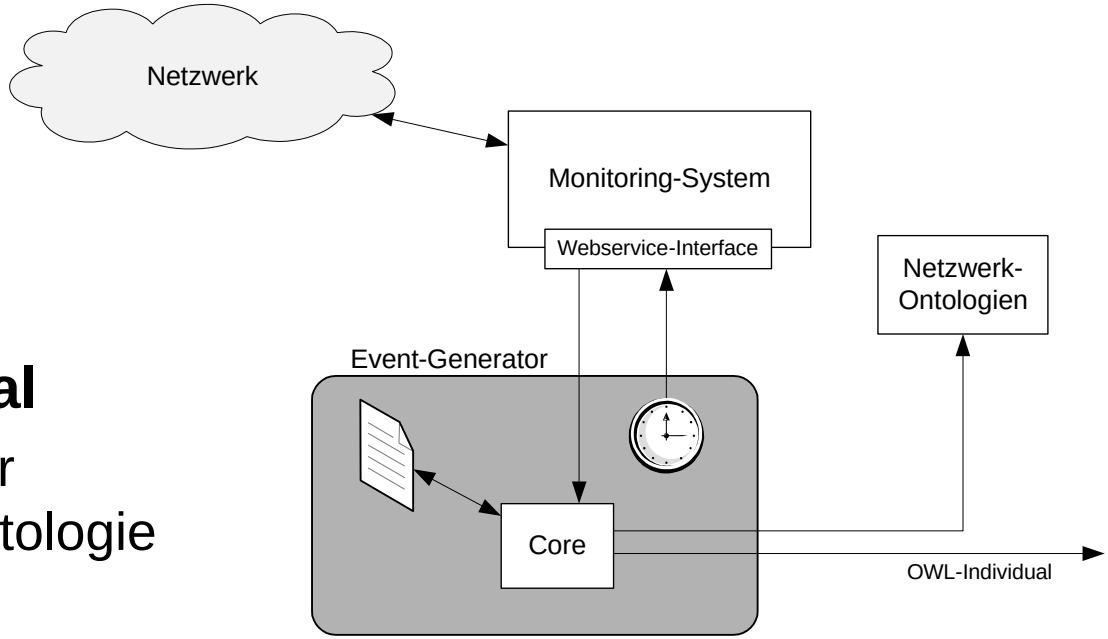
- **NINO wurde in einer anderen Arbeit entwickelt**
  - Network Infrastuctur Ontology
  - Einigen Anpassungen (grau) notwendig, dann erfüllt sie alle Anforderungen



Netzwerk-Ontologie NINO



- **Status der überwachten Services wird in regelmäßigen Abständen aktiv erfragt und zwischengespeichert**
  - Hier keine direkte Kommunikation mit den Geräten
- **Vergleich der Status**
  - Statusänderung führt zum Auslösen eines Events
- **Event = OWL Individual**
  - Generiert auf Basis der jeweiligen Netzwerkontologie



Zuordnung: Ereignis → Aktion (1/3)

- **Die Ereignisse müssen mit den Ausführungsplänen assoziiert werden**
  - Forderungen:
    - Eindeutige Zuordnung
    - Möglichst Flexibel (außerhalb des Quellcodes)
    - Änderungen möglichst auch zur Laufzeit möglich
  - Drei Denkansätze werden im Folgenden vorgestellt

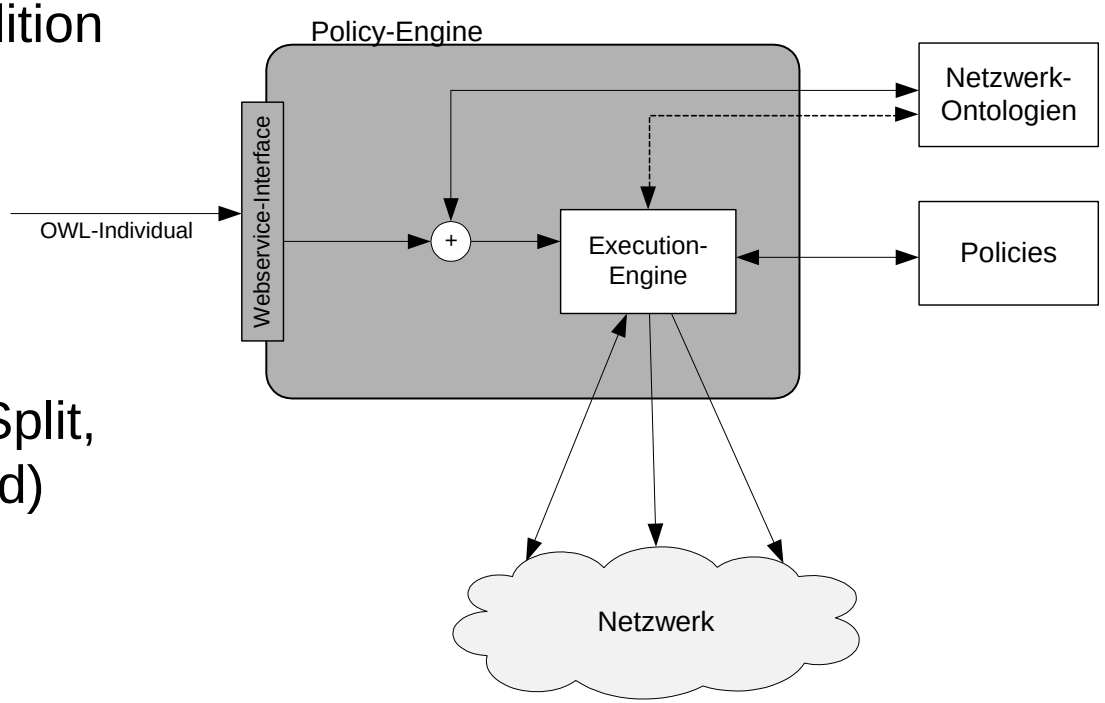
Zuordnung: Ereignis → Aktion (2/3)

- **Idee1: Zuordnung über XML-Beschreibung**
  - Trivial aber praktikabel
  - Nachteil: eigenes Beschreibungsformat, zusätzliche Datei
- **Idee2: Zuordnung über OWL-Beschreibung**
  - Vorteil: OWL-Editoren können genutzt werden
  - Nachteil: zusätzliche Datei

Zuordnung: Ereignis → Aktion (3/3)

- **Idee 3: Zuordnung über OWL-S Preconditions**
  - Vorteile
    - Keine separate Datei zur Beschreibung notwendig
    - Format durch W3C Submission (OWL-S) festgelegt
      - Auslösende Ereignisse werden in Form einer *Condition* direkt als Teil des *ServiceProfile* definiert (*hasPrecondition*)
      - *Condition* ist die Existenz eines bestimmten Individuals
    - Anpassungen zur Laufzeit möglich
      - *Preconditions* werden beim Eintreffen eines Ereignisses ausgewertet
  - Nachteil
    - Derzeit noch relativ schlechte Tool-Unterstützung

- **Policy-Engine wartet auf eintreffende Events**
  - Das Event wird mit Netzwerk-Ontologie verknüpft
  - Für alle Ausführungspläne findet eine Überprüfung statt, ob ihre jeweiligen Vorbedingungen erfüllt sind.
  - Pläne deren Precondition erfüllt ist, werden ausgeführt
  - Es können sowohl einfache als auch kombinierte Services ausgeführt werden (Split, Sequence, Unordered)



Implementierung (Ausblick)

- **Implementierung erfolgt in Java**
  - Gute Unterstützung für die Implementierung von Webservices
  - Mit Jena ist ein gutes Framework für RDF-/OWL-Implementierung vorhanden
  - OWL-S API für Java (mindswap.org) erlaubt das Lesen, Schreiben und Ausführen von OWL-S Service-Beschreibungen

- **Möglichst aufbauend auf bestehende Konzepte**
  - Beschreibung des Netzwerkes durch beliebige Ontologie
  - Assoziation zwischen Ereignis und aufgeführtem Service direkt innerhalb der Service-Beschreibung

Zusammenfassung

- **Netzwerke werden immer komplexer**
  - Policy-basiertes Management löst Probleme
- **Entwickelt wird ein Netzwerk-Management-System**
  - Webservice-basiert
  - Policy-basiert
- **Architektur des Systems**
  - Netzwerk-Ontologie NINO
  - Event-Generator und Policy-Engine
  - Ereignis-Aktion-Mapping über OWL-S Preconditions

Ende

Vielen Dank für Ihre Aufmerksamkeit!

Gibt es Fragen?

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

Code-Beispiel: Event als OWL-Individual

```
<rdf:RDF
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:j.0="http://www.ibr.cs.tu-bs.de/theses/tklie/policy-engine/ninoNetwork.owl#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#">
  <j.0:PerformanceEvent rdf:nodeID="A0">
    <j.0:hasPreviousState>ok</j.0:hasPreviousState>
    <j.0:hasPreviousState>warning</j.0:hasPreviousState>
    <j.0:causedByService>
      <j.0:ManagedService>
        <j.0:hasEvent rdf:nodeID="A0"/>
        <j.0:hostedByDevice rdf:resource="http://www.ibr.cs.tu-
    bs.de/theses/tklie/policy-engine/individuals.owl#Device_localhost"/>
        <j.0:hasServiceName>Current Load</j.0:hasServiceName>
      </j.0:ManagedService>
    </j.0:causedByService>
  </j.0:PerformanceEvent>
</rdf:RDF>
```

Code-Beispiel: Mapping mittels Precondition

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:j.0="http://www.falkenstern.de/ontologies/individuals.owl#"
    xmlns:p1="http://www.falkenstern.de/ontologies/individuals.owl#"
    xmlns:p2="http://www.ibr.cs.tu-bs.de/users/tklie/nino/ninoBase.owl#"
    xmlns:j.2="http://www.falkenstern.de/ontologies/ninoNetwork.owl#"
    ...
  xml:base="http://www.falkenstern.de/ontologies/individuals.owl">

  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://www.falkenstern.de/ontologies/individuals.owl"/>
  </owl:Ontology>

  <service:Service rdf:ID="testService1">
    <service:describedBy>
      <process:AtomicProcess rdf:ID="testProcess1">
        <process:hasOutput>...</process:hasOutput>
        <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">testProcess1</rdfs:label>
        <service:describes rdf:resource="#testService1"/>
        <process:hasPrecondition
      rdf:resource="http://www.falkenstern.de/ontologies/individuals.owl#Condition_localhost_load_ok_to_warning"/>
      </process:AtomicProcess>
    </service:describedBy>
    <service:supports>
      <grounding:WsdlGrounding rdf:ID="helloGrounding1">...</grounding:WsdlGrounding>
    </service:supports>
    <service:presents>
      <profile:Profile rdf:ID="testProfile1">
        <profile:hasPrecondition
      rdf:resource="http://www.falkenstern.de/ontologies/individuals.owl#Condition_localhost_load_ok_to_warning"/>
        <profile:serviceName rdf:datatype="http://www.w3.org/2001/XMLSchema#string">**TestService** localhost / OK ->
      WARNING</profile:serviceName>
        <profile:hasOutput rdf:resource="#return1"/>
        <service:presentedBy rdf:resource="#testService1"/>
      </profile:Profile>
    </service:presents>
  </service:Service>
  ...
</rdf:RDF>
```

Code-Beispiel: Mapping mittels Precondition

```
...
<service:Service rdf:ID="testService1">
  <service:describedBy>
    <process:AtomicProcess rdf:ID="testProcess1">
      <rdfs:label rdf:datatype="http://www.w3.org/2001/XMLSchema#string">testProcess1</rdfs:label>
      <service:describes rdf:resource="#testService1"/>
      <process:hasOutput>...</process:hasOutput>
      <process:hasPrecondition
  rdf:resource="http://www.falkenstern.de/ontologies/individuals.owl#Condition_localhost_load_ok_to
   _warning"/>
    </process:AtomicProcess>
  </service:describedBy>
  ...
```

Code-Beispiel: SWRL-Condition (in OWL-S)

```
<expr:SWRL-Condition rdf:ID="Condition_localhost_load_ok_to_warning">
  <expr:expressionLanguage rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#SWRL"/>
  <expr:expressionBody rdf:parseType="Literal">
      <swrl:AtomList>
        <rdf:first>
          <swrl:IndividualPropertyAtom>
            <swrl:propertyPredicate rdf:resource="http://www.falkenstern.de/ontologies/ninoNetwork.owl#causedByService"/>
            <swrl:argument1>
              <swrl:Variable rdf:about="http://www.example.org/service.owle"/>
            </swrl:argument1>
            <swrl:argument2 rdf:resource="http://www.falkenstern.de/ontologies/individuals.owl#ManagedService_localhost_load"/>
          </swrl:IndividualPropertyAtom>
        </rdf:first>
        <rdf:rest>
          <swrl:AtomList>
            <rdf:first>
              <swrl:DatavaluedPropertyAtom>
                <swrl:propertyPredicate rdf:resource="http://www.falkenstern.de/ontologies/ninoNetwork.owl#hasPreviousState"/>
                <swrl:argument1 rdf:resource="http://www.example.org/service.owle"/>
                <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">ok</swrl:argument2>
              </swrl:DatavaluedPropertyAtom>
            </rdf:first>
            <rdf:rest>
              <swrl:AtomList>
                <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                <rdf:first>
                  <swrl:DatavaluedPropertyAtom>
                    <swrl:propertyPredicate rdf:resource="http://www.falkenstern.de/ontologies/ninoNetwork.owl#hasNewState"/>
                    <swrl:argument1 rdf:resource="http://www.example.org/service.owle"/>
                    <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#string">warning</swrl:argument2>
                  </swrl:DatavaluedPropertyAtom>
                </rdf:first>
              </swrl:AtomList>
            </rdf:rest>
          </swrl:AtomList>
        </rdf:rest>
      </swrl:AtomList>
  </expr:expressionBody>
</expr:SWRL-Condition>
```

Code-Beispiel: SWRL-Condition (in OWL-S)

```
<expr:SWRL-Condition rdf:ID="Condition_localhost_load_ok_to_warning">
  <expr:expressionLanguage rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/Expression.owl#SWRL"/>
  <expr:expressionBody rdf:parseType="Literal">
      <swrl:AtomList>
        <rdf:first>
          <swrl:IndividualPropertyAtom>
            <swrl:propertyPredicate rdf:resource="http://[...]/ninoNetwork.owl#causedByService"/>
            <swrl:argument1>
              <swrl:Variable rdf:about="http://www.example.org/service.owle"/>
            </swrl:argument1>
            <swrl:argument2 rdf:resource="http://[...]/individuals.owl#ManagedService_localhost_load"/>
          </swrl:IndividualPropertyAtom>
        </rdf:first>
        <rdf:rest>...</rdf:rest>
      </swrl:AtomList>
  </expr:expressionBody>
</expr:SWRL-Condition>
```

Geschichte des Semantic Web

- **Das Semantic Web geht zurück auf die Idee semantischer Netze (Quilian, 1969)**
  - Modell zur Beschreibung von Begriffen und ihrer Beziehung untereinander
  - Netz repräsentiert durch Graphen mit Knoten und Kanten
- **Aufgegriffen durch Tim Berners-Lee im Jahr 2001**
  - Vision eines „bedeutungsvollen Internet“
  - Informationen unter Berücksichtigung ihres Kontextes
  - Neue Informationen können *erschlossen* werden
- **Idee**
  - Informationen werden bereits bei ihrer Speicherung mit einer Bedeutung versehen
  - Syntax und Struktur von Daten

„Ein Webservice ist eine Software-Anwendung auf die über fest definierte Schnittstellen und unter Benutzung von Standard-Internet-Protokollen zugegriffen werden kann“

- **Webservice über URI eindeutig identifiziert / erreichbar**
- **Schnittstellen sind als XML-Artefakte definiert**
  - Interaktion mit anderen Software-Agenten wird unterstützt
- **Was tut ein Webservice?**
  - Vergleich mit einer Webseite für den menschlichen Benutzer
  - Request-Response-Prinzip
- **Architektur**
  - Konsument, Anbieter und Verzeichnis

Netzwerk-Management

- **Größe und Komplexität heutiger Netzwerke erfordert ein effizientes Management**
- **Aufgabenbereiche beschrieben durch FCAPS-Modell der ISO/ITU**
  - Fehlermanagement (*Fault Management*)
  - Konfigurationsmanagement (*Configuration Management*)
  - Abrechnungsmanagement (*Accounting Management*)
  - Leistungsmangement (*Performance Management*)
  - Sicherheitsmanagement (*Security Management*)
- **Policy-basiertes Netzwerk-Management**
  - Regeln beschreiben das gewünschte *Verhalten* des Netzwerkes