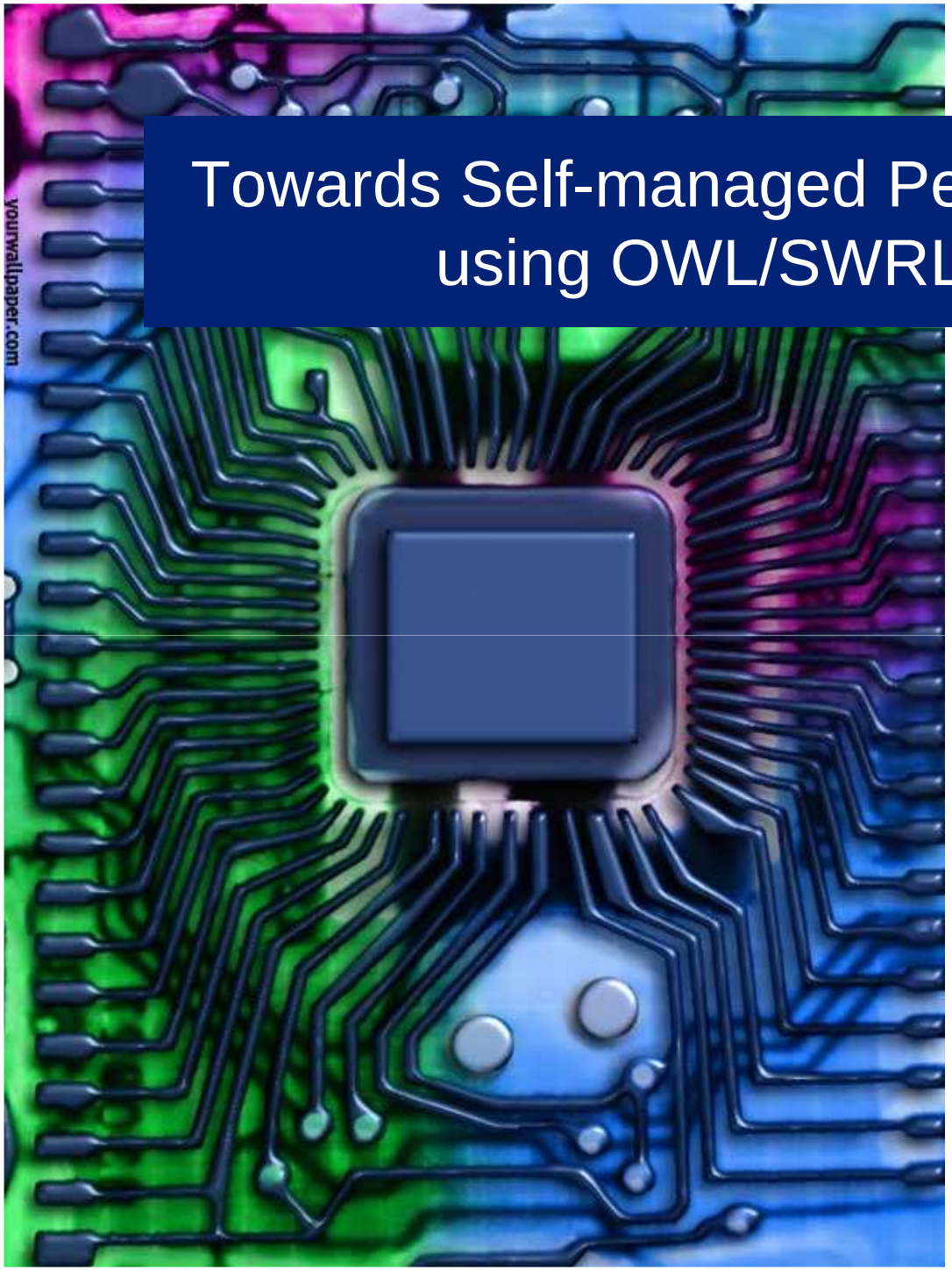


Towards Self-managed Pervasive Middleware using OWL/SWRL ontologies

Weishan Zhang

Klaus Marius Hansen

University of Aarhus

HYDRA

Daimi

IST-2005-034891

Information Society Technologies

Plan

- Motivation
- Dynamic context modeling facilitating self-management
- Self-management Onotologies structure
  - Self-management rules with SWRL
- Architecture of semantic web based self-management
- Evaluations

Motivation

- Context awareness based self-management
- Dynamic contexts are critical for self-management

This leads to the design:

- SeMaPS: a set of Self-Management for Pervasive Service context ontologies, that considering dynamic contexts
- Three layered Self-management

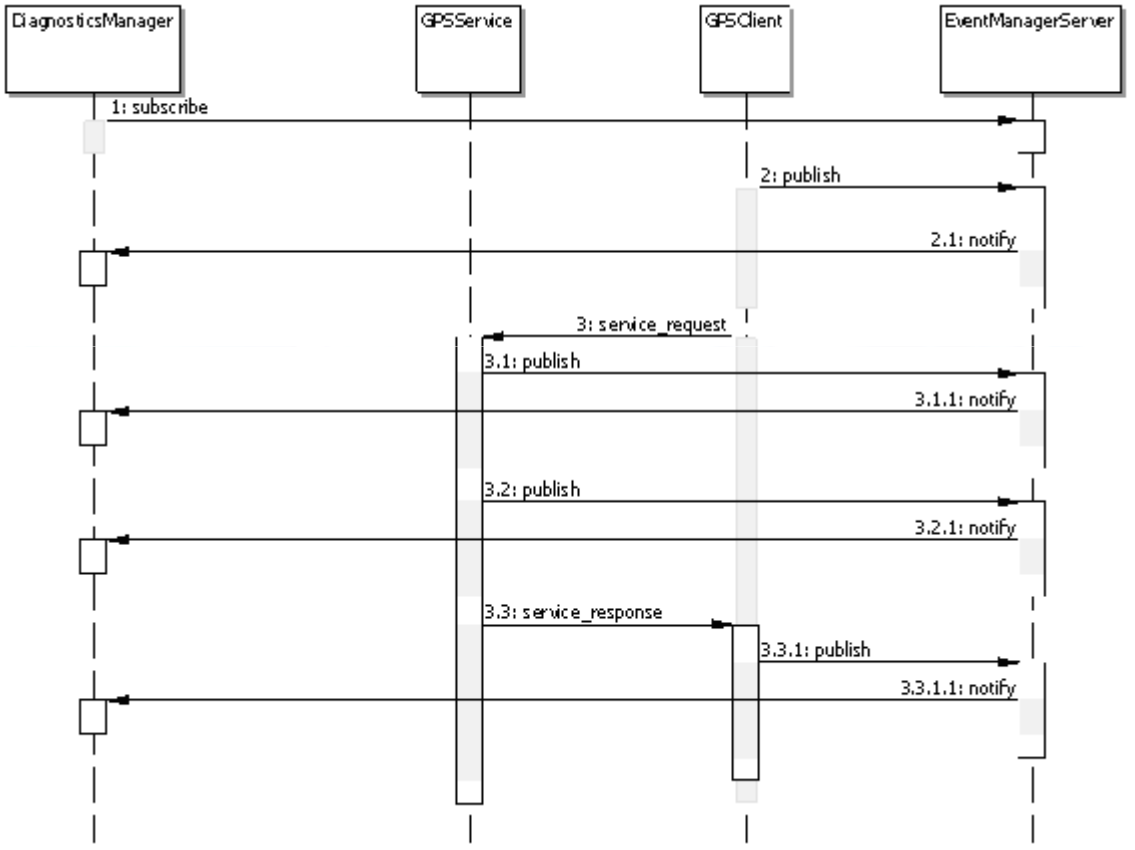
Dynamicity reporting for self-management

- State change reporting. State machines are used to report device state changes as events through the Hydra Event Manager.
- Web service request/reply reporting. The requests and replies (and their associated data) can be used to analyse the runtime structure of the Hydra systems. Here the Flamenco/Probe (IPSniffer) is used.

Dynamic contexts should reflect this dynamicity of pervasive service systems at run time

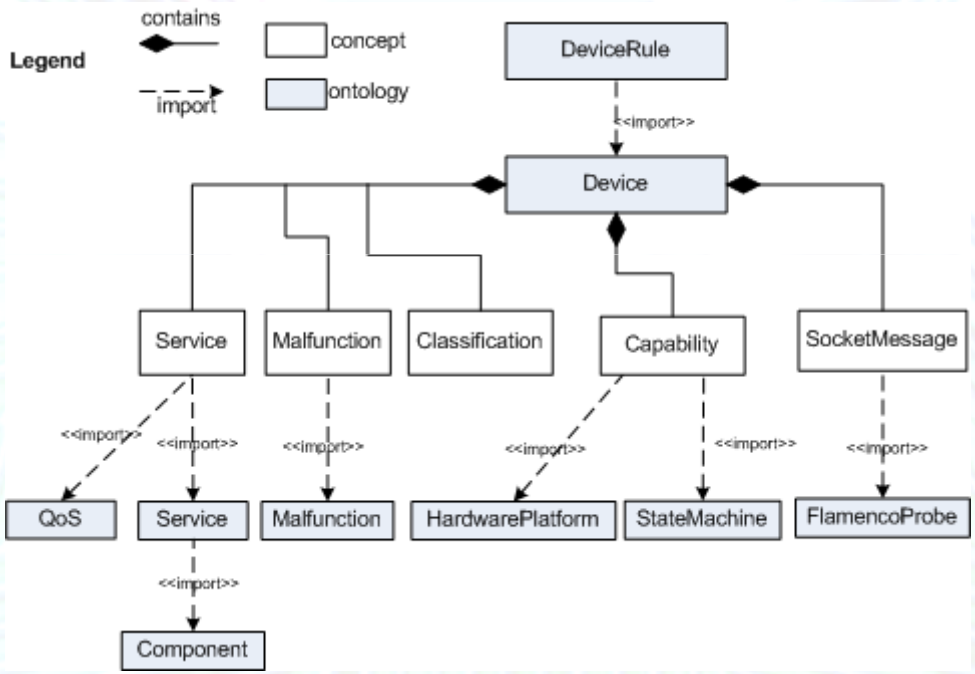
- Limbo (a pervasive service compiler) is used to generate needed code to report invocations and state machines.

Web service call reporting

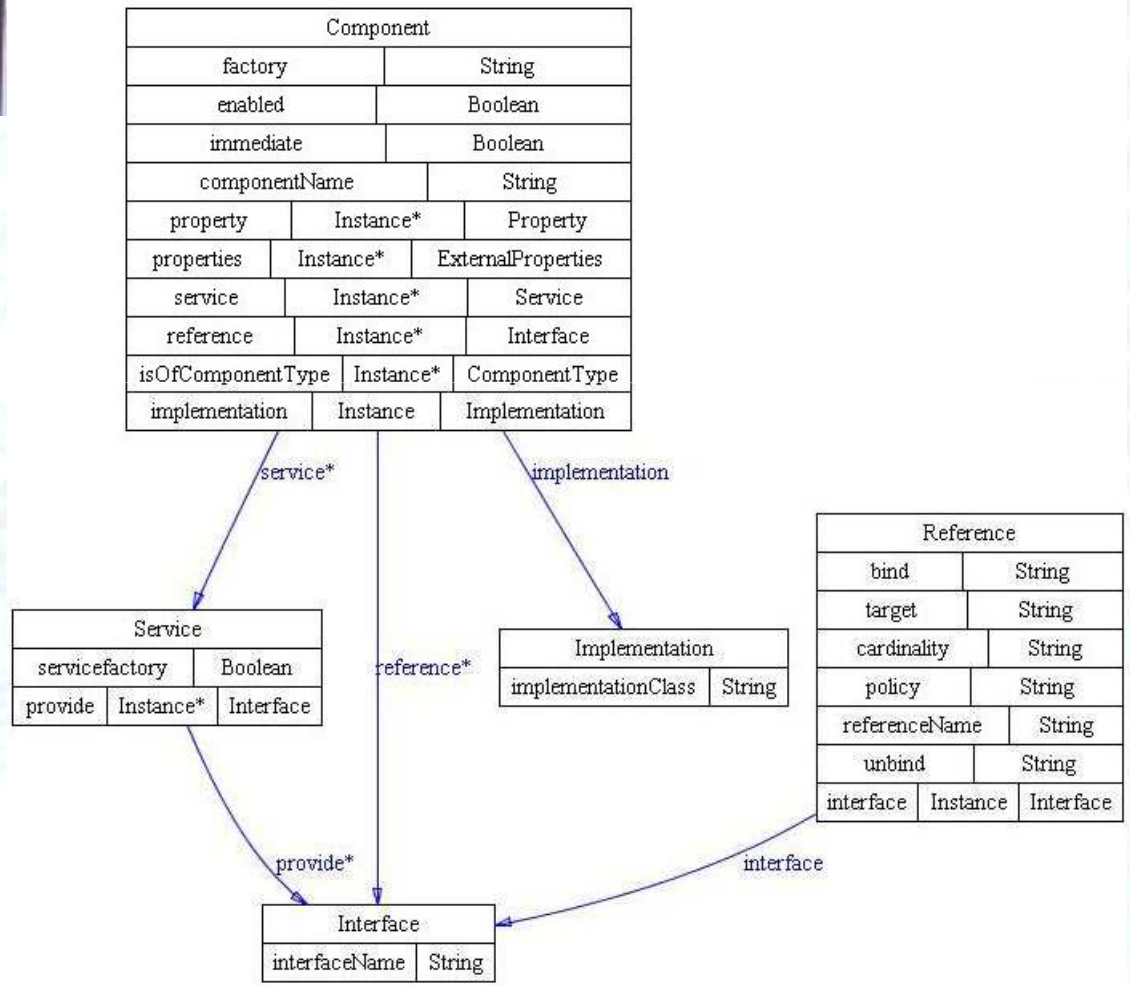


HYDRA

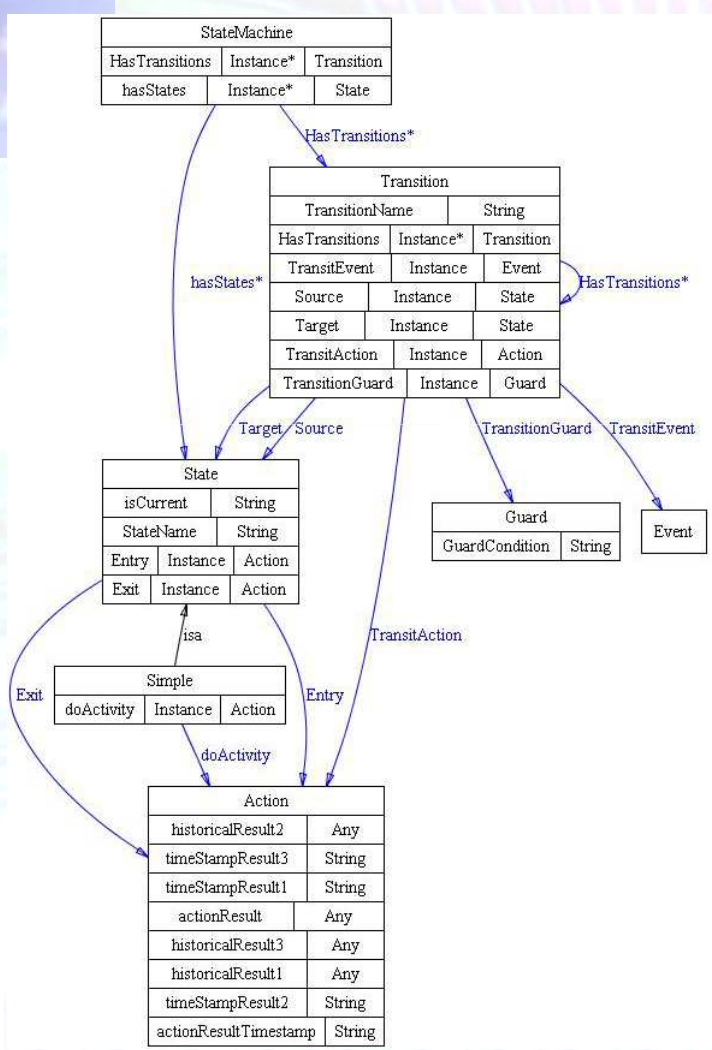
SeMaPS ontologies structure

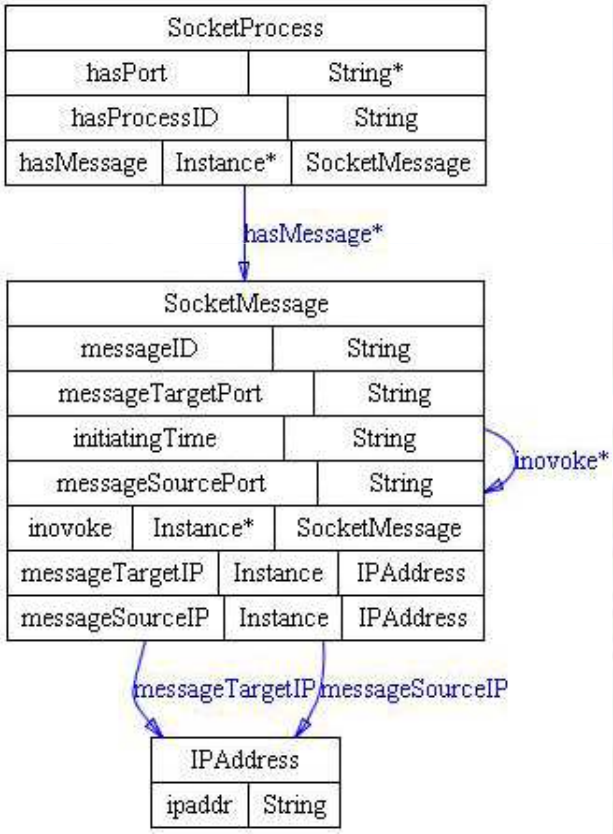


Component ontology



State machine and FlamencoProbe ontology





Self-management rule specification with SWRL

- A simple diagnosis example

SWRL Rule

Name | Comment

Name

http://localhost:9999/ontology/DeviceRule.owl#flowmeterToohighPig

SWRL Rule

```
device:FlowMeter(?device) ∧
device:hasStateMachine(?device, ?statemachine) ∧
statemachine:hasStates(?statemachine, ?state) ∧
statemachine:doActivity(?state, ?action) ∧
statemachine:actionResult(?action, ?result) ∧
abox:isNumeric(?result) ∧
swrlb:greaterThan(?result, 16.0)
  → device:currentMalfunction(device:Flowmeter, error:PumpBroken)
```

B ≠ = ∧ → ( ) [ ]

Daimi

HYDRA

Self-management rule specification with SWRL

- A not simple example for checking current configuration

```
ComponentBased(?con) ∧
hasComponent(?con, ?comp1) ∧
osgicomponent:reference(?comp1, ?ref1) ∧
osgicomponent:cardinality(?ref1, ?car1) ∧
swrlb:containsIgnoreCase(?car1, "1.") ∧
osgicomponent:interface(?ref1, ?inter1) ∧
osgicomponent:interfaceName(?inter1, ?name1) ∧
hasComponent(?con, ?comp2) ∧
osgicomponent:service(?comp2, ?ser2) ∧
osgicomponent:provide(?ser2, ?inter2) ∧
osgicomponent:interfaceName(?inter2, ?name2) ∧
swrlb:equal(?name1, ?name2)
  → sqwrl:selectDistinct(?con, ?comp1, ?comp2, ?name1, ?name2) ∧
sqwrl:select("validConfiguration")
```

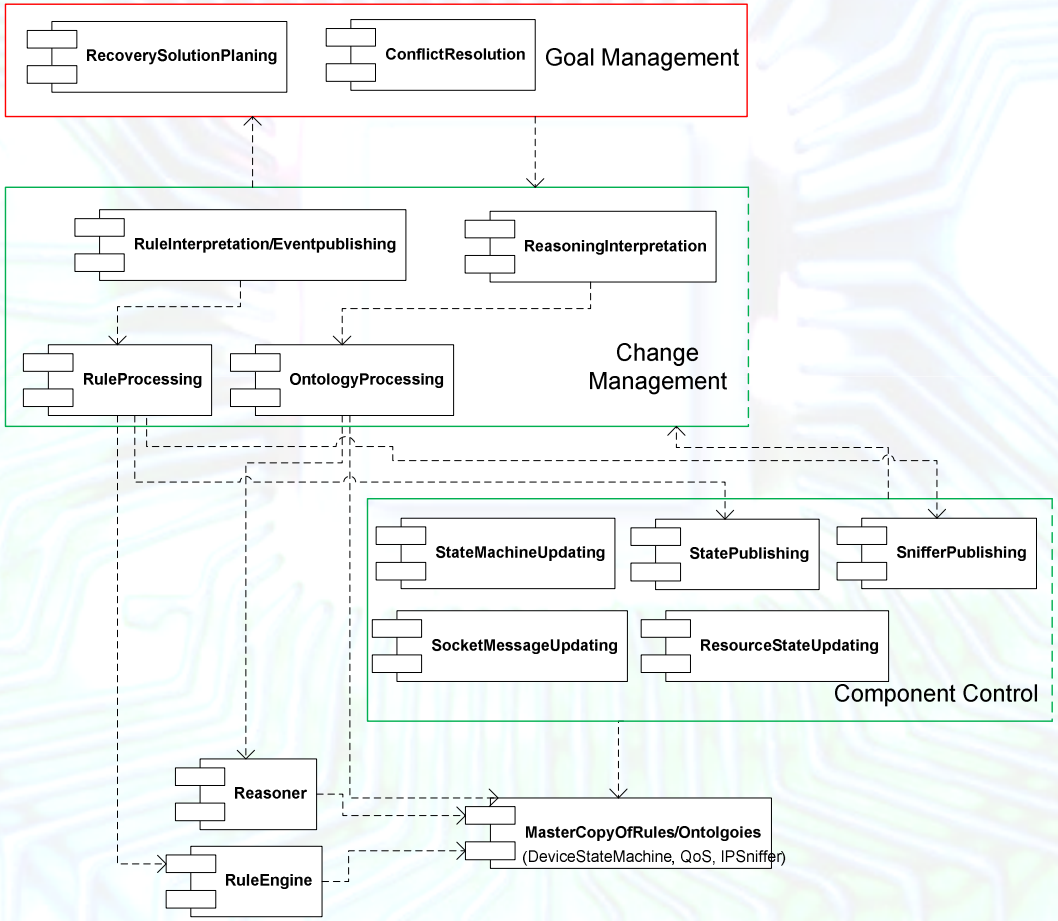
Self-management rule specification with SWRL

- A complex example

SWRL Rule

```
ipsniffer:messageID(?message1, ?message1id) ∧
ipsniffer:messageID(?message2, ?message2id) ∧
ipsniffer:messageID(?message3, ?message3id) ∧
ipsniffer:messageID(?message4, ?message4id) ∧
swrlb:equal(?message1id, ?message2id) ∧
swrlb:equal(?message2id, ?message3id) ∧
swrlb:equal(?message3id, ?message4id) ∧
abox:hasURI(?message1, ?u1) ∧
abox:hasURI(?message2, ?u2) ∧
abox:hasURI(?message3, ?u3) ∧
abox:hasURI(?message4, ?u4) ∧
swrlb:containsIgnoreCase(?u1, "clientbegin") ∧
swrlb:containsIgnoreCase(?u2, "servicebegin") ∧
swrlb:containsIgnoreCase(?u3, "serviceend") ∧
swrlb:containsIgnoreCase(?u4, "clientend") ∧
ipsniffer:messageSourceIP(?message1, ?ip1) ∧
ipsniffer:ipaddr(?ip1, ?ipa1) ∧
ipsniffer:ipaddr(?ip2, ?ipa2) ∧
ipsniffer:hasMessage(?process1, ?message1) ∧
ipsniffer:hasProcessID(?process1, ?pid1) ∧
ipsniffer:messageTargetIP(?message1, ?ip2) ∧
ipsniffer:initiatingTime(?message1, ?time1) ∧
ipsniffer:messageSourceIP(?message2, ?ip3) ∧
ipsniffer:messageTargetIP(?message2, ?ip4) ∧
ipsniffer:ipaddr(?ip3, ?ipa3) ∧
ipsniffer:ipaddr(?ip4, ?ipa4) ∧
ipsniffer:messageTargetPort(?message2, ?port2) ∧
ipsniffer:hasMessage(?process2, ?message2) ∧
ipsniffer:hasProcessID(?process2, ?pid2) ∧
ipsniffer:initiatingTime(?message2, ?time2) ∧
ipsniffer:messageSourceIP(?message3, ?ip5) ∧
ipsniffer:messageTargetIP(?message3, ?ip6) ∧
ipsniffer:ipaddr(?ip5, ?ipa5) ∧
ipsniffer:ipaddr(?ip6, ?ipa6) ∧
ipsniffer:messageTargetPort(?message3, ?port3) ∧
ipsniffer:hasMessage(?process3, ?message3) ∧
ipsniffer:hasProcessID(?process3, ?pid3) ∧
ipsniffer:initiatingTime(?message3, ?time3) ∧
ipsniffer:messageSourceIP(?message4, ?ip7) ∧
ipsniffer:messageTargetIP(?message4, ?ip8) ∧
ipsniffer:ipaddr(?ip7, ?ipa7) ∧
ipsniffer:ipaddr(?ip8, ?ipa8) ∧
ipsniffer:messageTargetPort(?message4, ?port4) ∧
ipsniffer:hasMessage(?process4, ?message4) ∧
ipsniffer:hasProcessID(?process4, ?pid4) ∧
ipsniffer:initiatingTime(?message4, ?time4) ∧
temporal:duration(?d1, ?time1, ?time4, temporal:Milliseconds) ∧
temporal:duration(?d2, ?time2, ?time3, temporal:Milliseconds)
  → ipsniffer:inovoke(?message1, ?message2) ∧
sqwrl:select(?ip1, ?ipa1, ?pid1, ?ipa2, ?port2, ?pid2, ?d1, ?d2)
```

Architecture of the semantic web based self-management



Evaluations

- Using rule group
- Executing all rules

Update	InferringTime	AfterEventTillInferred
328	328	328
297	281	297
297	297	297
297	281	297
344	344	344

Performance after rule grouping

Update	InferringTime	AfterEventTillInferred
843	843	843
906	906	906
922	906	922
719	719	719
953	938	938

Performance before rule grouping

Protege 3.4 Build 130, JVM 1.6.02-b06, Heap memory is 266M, Windows XP SP3. The hardware platform is: Thinkpad T61P T7500 2.2G CPU, 7200rpm hard disk, 2G DDR2 RAM. The time measurement is in millisecond. The size of DeviceRule ontology is 238,824 bytes, and contains 20 rules, including 6 rules for the Pig system, 12 generic rules which can be used in a number of domains, 3 rules (2 are shared with Pig rules) for the Weather Station, and 1 rule for FlamencoProbe related rules which is the biggest rule in the DeviceRule ontology.

Thank you..

HYDRA