Specifying and enforcing high-level semantic obligation policies

Zhen Liu  Anand Ranganathan  Anton Riabov

February 3, 2009

Presented by
Alvaro Graves
agraves@cs.rpi.edu
1 Introduction

2 EAGLE language
   - Exemplar individuals
   - Streams

3 Policy enforcement
   - Architecture
   - Models of event sources

4 Conclusions
The problem

High-volume, low-level events (e.g. temperature, humidity, ...)

The problem

High-volume, low-level events (e.g. temperature, humidity, ...) What we want is to be able to process them to create low-volume high-level events and apply policies based on that.
Proposed solution

- Ontology-based policies that defines events, conditions and actions
- Stream monitors that collects the streams, catch the events and apply the policies
An EAGLE policy is defined by

- A policy name
An EAGLE policy is defined by

- A policy name
- Event Pattern (EP)
  - Variable Set (VS) Ex. ?EmpName, ?MgrEmail
An EAGLE policy is defined by

- A policy name
- Event Pattern (EP)
  - Variable Set (VS) Ex. ?EmpName, ?MgrEmail
  - Graph pattern (GP) Set of triple patterns where the subject or object is a variable
An EAGLE policy is defined by

- A policy name
- Event Pattern (EP)
  - Variable Set (VS) Ex. ?EmpName, ?MgrEmail
  - Graph pattern (GP) Set of triple patterns where the subject or object is a variable
- A set of condition-action pairs.
Policy EmployeeCourtesyPolicy

OnEvent

Containing ?EmpName, ?MgrEmail, ?CurtLevel

Where

?Employee
  :Employee;
  :tookCall ?EmpVoIP_Call;
  :hasName ?EmpName;
  :hasMgr ?Manager;
  :employedBy EGS.

?EmpVoIP_Call
  :ServiceVoIP_Call;
  :withCurtLevel ?CurtLevel.

?Manager
  :Manager;
  :hasEmail ?MgrEmail.

PerformAction

IMServ.sendAlert(?MgrEmail, ?EmpName)
  If (?CurtLevel < 0.4)
    AND (IMServ.isOnline(?MgrEmail))

PerformAction

CallRouteServ.suspendEmp(?EmpName)
  If (?CurtLevel < 0.2)
Exemplar individuals and literals

They represent existentially quantified variables. Their function is to allow to describe the semantics of a stream of events. They are prefixed with __

```
Class EmployeeCourtesy{
    com.egs.Dept __Department_1;
    String __MgrName_1;
    String __MgrEmail_1;
    float __CourtesyLevel_1;
    String __EmployeeName_1;
}
```
Definition of a stream

A stream can be represented as a tuple \((SN, SD, SG)\) where

- \(SN\) is the name of the stream
- \(SD\) is the set of data items represented using exemplar literals and variables.
- \(SG\) an RDF graph describing semantics of the event.
EmpCurt_VoIP_CallStream contains
- __CourtesyLevel_1
- Department
- __EmployeeName_1
- __MgrName_1
- Service_VoIPCall
  - withCurtLevel
  - inDept
- __EmployeeVoIP_Call_1
  - placedVia
- IP_Over_VPN_Network
  - ofEmp
  - ofCompany
- EGS_Network
  - ofCompany
- CustServRep
  - hasFormat
  - hasName
- _mgrEmail_1
  - hasEmail
- _mgrName_1
  - hasName
- _Manager_1
  - employedBy
- _Employee_1
  - hasMgr
- _Department_1
  - com.egs.dept
  - inDept
How to match a stream and an event pattern?

We replace all the variables to different terms (individuals and literals). Eventually some of those terms may be exemplar.
How to match a stream and an event pattern?

We replace all the variables to different terms (individuals and literals). Eventually some of those terms may be exemplar. More formally, given a stream $S(SN, SD, SG)$ and an event pattern $EP(VS, GP)$, $EP$ matches $S$ if we find a function $\Theta$ that

- There is a match of variables to RDF terms where we can match $VS \subseteq \Theta(SD)$.
- $SG \cup O \models \Theta(GP)$ i.e., the graph describing the semantics matches the graph pattern of the event pattern.
- Each event source produces one event stream
- You can see pattern events as graphs transformations
When the planner tries to match a stream to a PE as input it considers the expanded stream-graph $SG'$.

Once the plan is deployed in the system, the stream monitor parses the events and
Conclusions

- Very obscure paper in some aspects
Conclusions

- Very obscure paper in some aspects
- Good idea in general and may be useful in several areas
Conclusions

- Very obscure paper in some aspects
- Good idea in general and may be useful in several areas
  but...
- What happens with missing sources?
Conclusions

- Very obscure paper in some aspects
- Good idea in general and may be useful in several areas
  but...
  - What happens with missing sources?
  - Disjointness?
It seems the performance is not very scalable however they claim is an acceptable time, since you can share the pre-reasoning among different policy compilations.

<table>
<thead>
<tr>
<th>#PEs &amp; sources</th>
<th>Streams</th>
<th>Prereason</th>
<th>Plan</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>15</td>
<td>4.66</td>
<td>0.15</td>
</tr>
<tr>
<td>20</td>
<td>40</td>
<td>10.29</td>
<td>0.22</td>
</tr>
<tr>
<td>30</td>
<td>119</td>
<td>27.49</td>
<td>0.76</td>
</tr>
<tr>
<td>40</td>
<td>190</td>
<td>38.48</td>
<td>1.21</td>
</tr>
<tr>
<td>50</td>
<td>288</td>
<td>59.17</td>
<td>2.26</td>
</tr>
<tr>
<td>100</td>
<td>1204</td>
<td>233.03</td>
<td>28.73</td>
</tr>
</tbody>
</table>