Specifying and Enforcing High-Level Semantic Obligation Policies

Zhen Liu,
Anand Ranganathan,
Anton Riabov
The first part of ECA is $E$

- Obligation Policies often take the form of ECA (Event-Condition-Action) rules
- In this work, we focus on the Event part of ECA
- Large scale pervasive and distributed computing systems often produce event streams containing large volumes of low-level events
  - Logging and audit messages
  - Events from sensors
  - Unstructured data (containing text and multimedia) from video cameras, microphones
- Policies are often at a higher-level
- Hence often difficult to specify policies
How can I specify the \( E \) in an ECA rule more easily?

- Specify obligation policies based on high-level events
  - derived after performing appropriate processing on raw, low-level events

- **Semantic obligation policy specification language called Eagle**
  - based on high-level events
  - represented as RDF graph patterns
  - based on terms defined in domain OWL ontologies

- **Policy Enforcement through a Compiler**
  - Uses an AI Planning approach to build a workflow for producing a stream of the required high-level events as specified in a policy.
  - Workflow consists of a number of event sources and event processing components, which are described semantically
Key advantages of Eagle Policies

- **Ease of Policy Specification and Maintenance**
  - Policies are independent of low-level events or of specific processing components

- **Highly expressive**
  - Use of Description Logic Reasoning for matching an event (or a stream of events) to a policy

- **Adaptive**
  - Use of planning allows adaptation to current conditions of a system
Example from a call center domain

- Customer Service Department of a company receives and makes VoIP calls
- Managers create policies that specify actions for certain kinds of events
  - If there are many dropped calls
  - If the average call waiting time is high
  - If an employee is discourteous
- Raw events in system
  - VoIP based call traffic
  - Session management events (call transfer, call termination, etc.)
Example Policy

- If the courtesy level of an employee in a VoIP service call is
  - less than 0.4, then an instant message alert should be sent to the manager;
  - if the courtesy level is less than 0.2, then the employee should not be allowed to answer any further customer service calls.
Example Policy

Name of Policy

Data Elements contained in Event

Semantic Description of Event (expressed as an RDF graph pattern)

Actions to be Performed under given conditions

```
Policy EmployeeCourtesyPolicy

OnEvent

Containing ?EmpName, ?MgrEmail, ?CurtLevel

Where

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

?EmpVoIP_Call a :ServiceVoIP_Call;
    :withCurtLevel ?CurtLevel.

?Manager a :Manager;
    :hasEmail ?MgrEmail.

PerformAction

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

PerformAction

CallRouteServ.suspendEmp(?EmpName)
  If (?CurtLevel < 0.2)
```
Event Specification in Eagle

- **Streaming model of events**

- **An Eagle Policy describes the data elements that must be contained in the event**
  - Described as a set of variables that are bound to values by a given event on the stream

- **Also specifies the semantics (or the meaning) of the elements in an event**
  - Described as a set of constraints on the variables
  - Expressed as an RDF graph pattern
Employee Courtesy Event Stream

contains

?CurtLevel

withCurtLevel

Service VolIPCall

?EmpVolP_Call

?EmpName

Employee

hasName

?Manager

Manager

hasEmail

?MgrEmail

hasEmail

tookCall

EGS

employedBy

?Employee

hasMgr
Terms in Event Specification defined in OWL Ontologies

- **OWL Ontologies Describe Entities in Domain**
  - Classes, Properties, Individuals

```
CALL subclassOf VolICall
VolICall subclassOf Service VolICall

Location atLocation City

Employee ofEmp Company
employedBy CustServRep

Manager hasMgr Customer
employedBy EGS
employedBy Bob

hasName “Bob Roberts”
```

**Diagram:**
- **Location** atLocation **City**
- **Employee** ofEmp **Company**
- **Manager** hasMgr **Customer**
- **EGS** employedBy **Bob**
  - **Bob** hasName “Bob Roberts”
The Eagle Policy Language

**Policy** <PolicyName>

**OnEvent**

**Containing** <VariableSet>

**Where** <Graph Pattern>

**PerformAction** <ActionDefinition_1>

**If** <ConditionDefinition_1>

...

**PerformAction** <ActionDefinition_n>

**If** <ConditionDefinition_n>
Policy Enforcement consists of 2 steps

- **Policy Compilation**
  - Create a workflow that can generate a stream of events that match the pattern defined in the policy
  - Matching is based on Description Logic (DL) reasoning

- **Stream Monitoring**
  - Monitor events on the matched stream and perform actions if certain conditions are true.
Policy Compiler

- Constructs Workflow (DAG) of sources and processing elements to produce high level event
Goal of Policy Compiler: Automatic Generation of Workflows

- **Ease of specification**
  - Policy-writer doesn’t have to be aware of actual event sources and software components

- **Shorter Time to Enforcement**

- **Reusable Software Components**

---

**Policy Compiler**

- SPA → Streaming Protocol Annotator
- VBF → Value-Based Filter
- SCF → Stream Courtesy Finder
- JoinCC → Join Cust & Emp to Courtesy Level
- IPN → Identified Person Notification
- SGD → Speech Gait Detection
- PCP → Progressive Conversation Pairing
- JAE → Join Async Events
- DSN → Denoise via Social Network
- DUPP → Dig Up Person Pair Info
- N(CE)S^2 → Employee Selector
- SPM → Stream Progression Monitor
- DFE → Distributed Front-End
- PSD → Progressive Speaker Detection
- IPN → Identified Person Notification

---

Policy-writer
What is required for policy specification?

- **Expressive semantic descriptions of**
  - Outputs of different streaming event sources
  - Input requirements and output capabilities of stream processing components

- **Method for determining if a stream produced from an event source can match the input requirements of a stream processing component**

- **Method for determining if any stream matches the event specification in the policy**
Semantic Description of a Data Source

GSM_VoIP_DataSource produces contains

hasDataRate ?DataRate

hasMedium ?Medium

Information about the Data Source

Application Concepts

Semantic description of information emitted from data source

Domain Concepts

PayloadFormat

UDP_Header

RTP_Header

RTP_Packet

UDP_Packet

GSM

hasPayloadFormat

type

NewObjectCreated

Protocol

RTP

type

hasHeader

subClass

PayloadFormat

type

RTP_Header

subClass

RTP_Packet

type

Information about the Data Source

Policy 2007

June 14, 2007

© 2006 IBM Corporation
Domain Concepts

Semantic description of information required in input stream

Semantic description of information emitted on output stream

StreamingProtocolAnnotator

requires

produces

contains

contains

RTP_Packet

Protocol

RTP

?RTP_Packet

protocol

type

RTP_Packet

Protocol

RTP

?RTP_Packet

protocol

type

?GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet

GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet

GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet

GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet

GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet

GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet

GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet

GSM_VoIP_Channel

ServiceSet

carriesContent

type

fromServiceSet

?ServiceSet
Using Semantic Descriptions to Match and Assemble Plan Graph Components
Policy Compilation through use of AI Planning

We have developed a semantic planner for composing a workflow that can…

– Make use of rich, expressive domain knowledge in ontologies
– Perform Description Logic (DL) reasoning to allow flexible matching
– Be scalable even in the presence of possibly time-consuming DL reasoning
– Be capable of planning on streams
2-phase semantic planning algorithm

- **First phase, which occurs offline**
  - Takes OWL files describing components and data sources.
  - Uses a DL reasoner to obtain all inferred facts about the components.
  - E.g. Asserted: Data Source produces Traffic Video Segments
    Inferred: Data Source produces Video Segment

- **Second phase, which occurs when the system is given a goal, for which plans have to be found.**
  - Searches for plan using more complete description of components.
  - Uses a forward chaining, branch and bound search to produce optimal plans.
  - No DL reasoning required in this phase.
  - Checks data security and privacy constraints.
Stream Monitoring

- Stream monitoring components are instantiated to monitor high-level events produced by workflow
- Performs actions if given conditions in Eagle Policy are satisfied
Conclusion

- **Method for specifying high-level obligation policies based on high-level events**
  - Using RDF Graph Patterns to describe the event
  - Using terms defined in ontologies to define the semantics of the event more precisely

- **Method for matching a stream of events to an event pattern**
  - Uses DL reasoning

- **Method of compiling a policy by constructing a workflow of processing elements and event sources**
  - Workflow produces event stream that satisfies high-level event pattern
Grazie e Buon Giorno

Delle Domande??
Backup