Conflict Resolution in Policy Management

Jan Chomicki
University at Buffalo
http://www.cse.buffalo.edu/~chomicki

Joint work with Jorge Lobo, Bell Labs, and Shamim Naqvi, Convexant.

Policies

Policies are collections of general principles specifying the desired behavior of a system. Potential application areas:

- communications, network management and monitoring (IETF)
- electronic commerce (IBM CommonRules)
- security and access management
- other decision-making applications.

Examples:

- *If a fax from the Chicago office arrives at the main office fax machine, redirect the fax to Joe's office fax machine.*
- *A deploy command for a missile should lead to starting its launch and opening the latch. A launch can be cancelled.*

Policy management

Policy execution:

- evaluation
- **conflict detection and resolution**

Policy maintenance:

- specification
- modification
- analysis
- ...

PDL

PDL policies defined as sets of **Event-Condition-Action** rules.

$$\begin{array}{ll} \mathit{arrivedFaxOff} & \textbf{causes } \mathit{sendFaxJoeOff}(\mathit{arrivedFaxOff.content}) \\ & \textbf{if } \ \mathit{arrivedFaxOff.from} = \text{“Chicago”}. \end{array}$$

Rich **event language**:

- conjunction, negation, sequence, relax-sequence
- event attributes

The formal semantics of PDL can be defined using logic programs.

Action conflicts

A policy manager may specify that several actions **cannot be simultaneously executed**.

Example:

$$requestRes\ \textbf{causes}\ processRes(requestRes.user)$$

Two simultaneous reservation requests cannot both be satisfied:

$$\textbf{never}\ \ processRes(User_1) \wedge processRes(User_2)\ \textbf{if}\ \ User_1 \neq User_2.$$

Reserved resources: bandwidth, airport runway,...

Action constraints

Syntax:

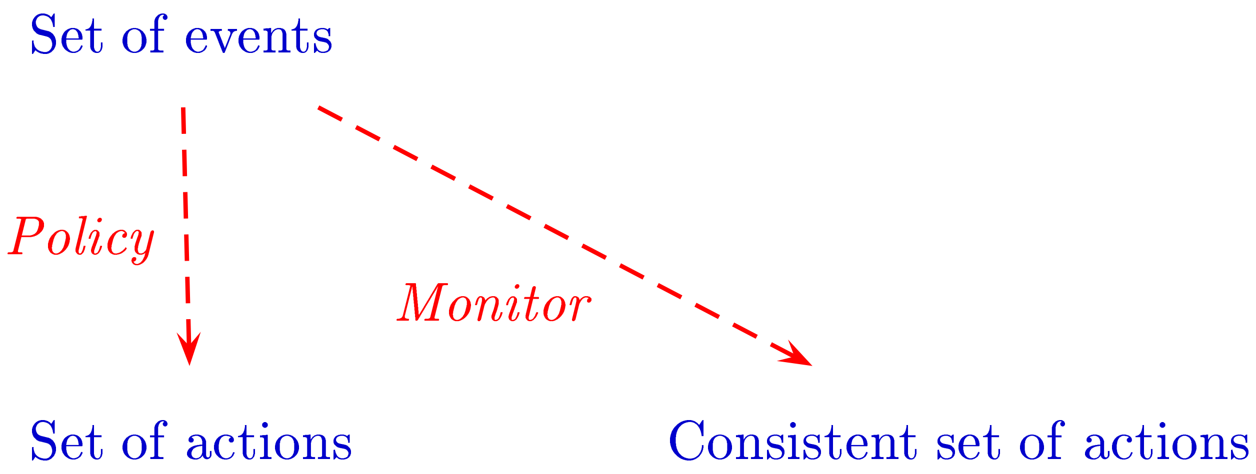
$$\textbf{never} \;\; A_1 \wedge A_2 \wedge \cdots \wedge A_n \;\textbf{if}\;\; C$$

Logical reading:

$$\forall \neg (A_1 \wedge A_2 \wedge \cdots \wedge A_n \wedge C)$$

Conflict resolution

A **monitor** detects and resolves conflicts among the actions generated by a policy.



Classes of monitors

Different resolution strategies.

Action-based monitors:

- blocking a conflicting action (**action cancellation** monitor)
- delaying a conflicting action (**action delay** monitor)
- action priorities may be included

Event-based monitors:

- ignoring the events causing a conflicting action(**event cancellation** monitor)
- postponing the events that cause a conflicting action (**event delay** monitor)

Example

Policy:

$$\mathit{cancel}\ \ \textbf{causes}\ \ \mathit{stop\_launch}$$

$$\mathit{deploy}\ \ \textbf{causes}\ \ \mathit{start\_launch}$$

$$\mathit{deploy}\ \ \textbf{causes}\ \ \mathit{open\_latch}$$

Constraint:

$$\textbf{never}\quad \mathit{stop\_launch} \wedge \mathit{start\_launch}$$

The latch shouldn't be opened if the launch was stopped.

Unobtrusiveness

An unobtrusive monitor mimicks the policy on some subset E' of input events:

- input event $e \in E'$: every action caused by e succeeds
- input event $e \notin E'$: every action caused by e fails unless it is also caused by an event in E'

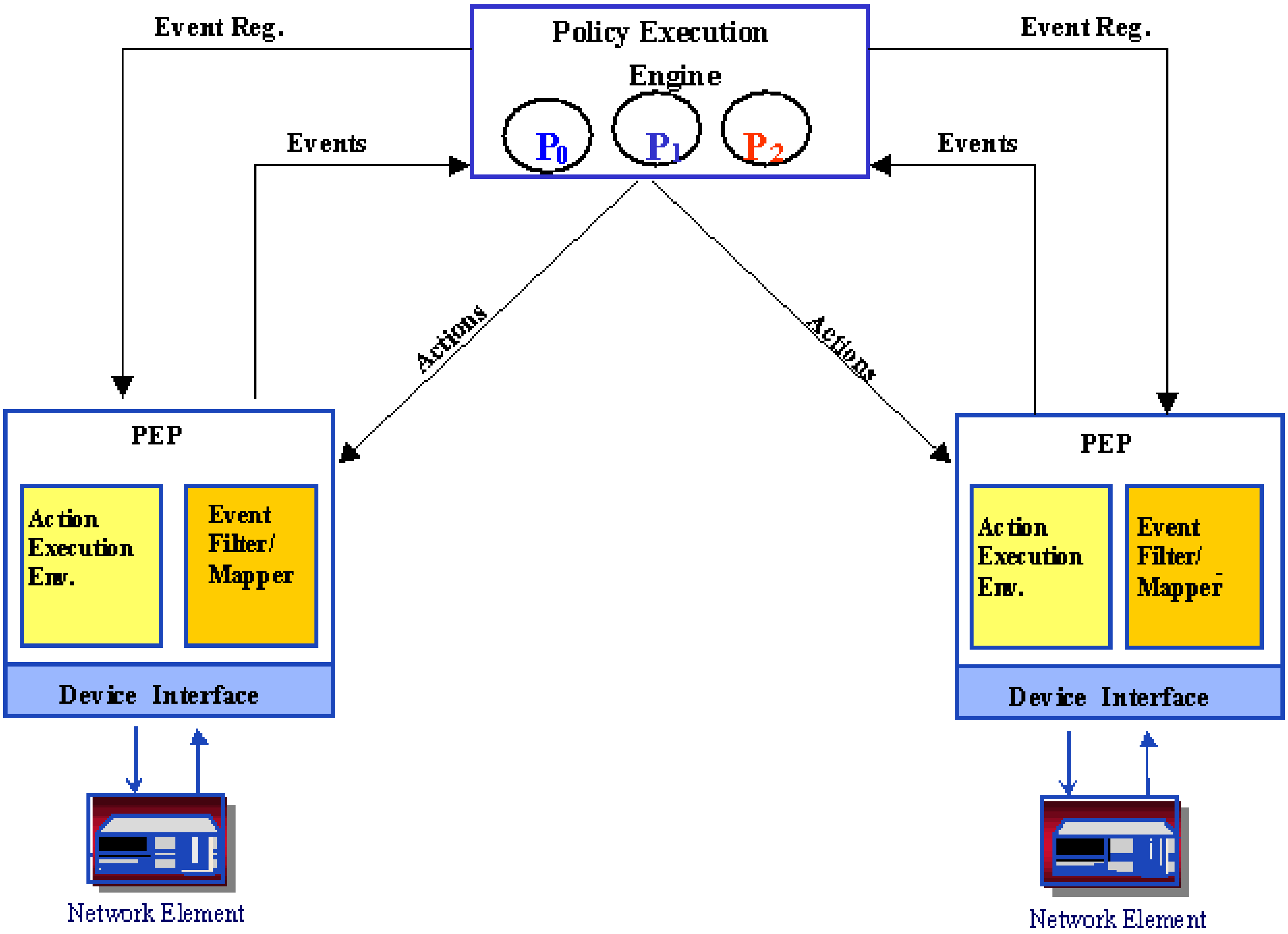
$$\begin{tikzcd}
\text{Set of events} \arrow[r, dashed] \arrow[d, dashed, "\textit{Policy}"'] \arrow[dr, dashed, "\textit{Monitor}"'] & \text{Reduced set of events} \arrow[d, dashed, "\textit{Policy}"] \\
\text{Set of actions} & \text{Consistent set of actions}
\end{tikzcd}$$

Specification and implementation of monitors

Both action and event-based monitors can be specified using disjunctive logic programs.

Implementation (event conjunction only):

- **action-based monitors**: computing a maximal consistent subset of the set of actions generated by the input epoch
- **event-based monitors**: computing a maximal subset of the input epoch that generates a consistent set of actions
- polynomial time


Policy Execution
Engine
P0
P1
P2
Event Reg.
Event Reg.
Events
Events
Actions
Actions
PEP
Action
Execution
Env.
Event
Filter/
Mapper
Device Interface
PEP
Action
Execution
Env.
Event
Filter/
Mapper
Device Interface
Network Element
Network Element

Negation

Problems:

- a policy may fail to have a monitor at all
- ignoring events may trigger new actions

Solution:

- ignoring an event makes it undefined

History-based policies

Temporal dimensions:

- **sequence** events
- **delay** monitors
- **temporal** action constraints

$$
\begin{tikzcd}
\text{Event history} \arrow[r, dashed] \arrow[d, dashed, "\textit{Policy}"'] \arrow[dr, dashed, "\textit{Monitor}"'] & \text{Reduced event history} \arrow[d, dashed, "\textit{Policy}"] \\
\text{Action history} & \text{Consistent action history}
\end{tikzcd}
$$

Related work

Event notification systems and languages:

- rich syntax (events), informal semantics
- explicit conflict resolution only recently addressed
- typically static, not dynamic, conflicts

Production rules:

- resolution of **rule conflicts**
- interpreted actions
- simple event model
- meta-language for controlling rule executions [Jagadish, Mendelzon, Mumick, PODS'96].

Agent-based systems

Policies are simple **reactive agents**.

Commands can be viewed as **events**.

Action constraints determined by the physical or virtual environment.

Unobtrusiveness: no partially executed commands.

[Eiter, Subrahmanian, Pick, AI Journal, 1999]:

- deontic specifications of agent systems
- action cancellation monitors (w/o logical framework)
- only atomic events

Unobtrusive agents

Waiter agent:

$pour\_request$ **causes** $hold\_cup$

$pour\_request$ **causes** $tilt\_bottle$

$serve\_request$ **causes** $hold\_plate$

never $hold\_cup \wedge hold\_plate$

What happens if $pour\_request$ and $serve\_request$ arrive simultaneously?

Conclusions

A general formal framework for defining policy monitors:

- broad classes of policies and monitors
- results optimal: maximal monitors
- extensible:
  - negation
  - history-based policies

Further work

Policy **analysis**:

- conflict-freeness

More general classes of policies:

- internal events (already in PDL)
- complex, long-duration actions
- preferences, policy hierarchies, inheritance
- databases, XML documents
- concurrency control monitors
- more general agents: coordination, communication

Papers:

1. J. Lobo, R. Bhatia, S. Naqvi, *"A Policy Description Language."* AAAI'99.

2. J. Chomicki, J. Lobo, S. Naqvi, *"A Logic Programming Approach to Conflict Resolution in Policy Management."* KR'2000.

3. J. Chomicki, J. Lobo, *"Monitors for History-Based Policies."* POLICY'2001, January 2001, Bristol, UK.

POLICY'2001

Workshop on Policies for Distributed Systems and Networks

January 29-31, Bristol, U.K.

`http://www-dse.doc.ic.ac.uk/events/policy-2001`