UNIVERSAL POLICY THEORY (UPT) AND
ADAPTIVE POLICY EXECUTION (APEX)
Dungeons & Dragons
What do we know about Policy? Examples?
Now, what’s the problem with Policy?
Rethink Policy

A Universal Policy Theory
- From formal models to immutable infrastructure
- Repeating structures everywhere – taming complexity
- Hooks for automation everywhere – reducing costs and bugs
Software update cycles speed up
New insights for all “unknown unknowns”
Everything that can be virtualized will be virtualized
Everything that can be connected will be connected
Users demand seamless / extreme modification and customization
This calls for multi-domain, adaptive automation
This requires a different view on policy
WHAT DO WE KNOW ABOUT POLICY?
Policy governs the choices in behavior of a system

*What does the definition tell us?*

*A difference between the mechanism (of a system) and policies (for a system)!*

- **Mechanisms are invariant**
  - I.e. invariant part of system
  - Relatively static, i.e. hardly change after specification / implementation
  - Need to provide some means of government

- **Policies are variant**
  - I.e. the variant part of a system
  - Relatively dynamic, i.e. change frequently (faster than mechanisms)
  - Need to use mechanism means of government

*What is policy?*

- **What policy is depends on**
  - A (concrete) system with mechanisms
  - How the mechanisms can chose behavior
  - How the mechanisms can be governed

- **Mechanism and policy are relative**
  - Multi-layer systems or systems of systems
  - Higher level policies can be lower level mechanisms

*How to discover and define policies?*

- **Discover policies: analyze the system**
  - Identify invariants, maximize invariants
  - Identify variants, minimize variants
  - The remaining set of variants are very likely policies

- **Define policies**
  - Define the means mechanisms can chose behavior
  - Define the means mechanisms can be governed
  - Build a policy system exploiting the findings in an optimal way
CAN WE GET SOME EXAMPLES?
Some Policy Examples

1. **Linear Feedback Control Loop**
2. **Operating System (Solaris)**
3. **Channel Control (ODP)**
4. **Strategy (Policy) Design Pattern**
5. **Workflow (BPMN)**
6. **Data Processing (Prediction)**
7. **Component Architecture (CORBA)**
8. **Manager/Agent Paradigm**
9. **Autonomic Control Loop (COMPA)**
SOME POLICY EXAMPLES

(a) Linear feedback control loop: reference value $r$ is the policy
(b) OS (e.g. Unix): user-level processes can set arguments for kernel-level process, those arguments are policies
(c) ODP control channel: config params of stub, binder, and protocol adapter are policies
(d) GoF Strategy Pattern: the concrete strategies are policies
(e) BPMN: thresholds and other decision parameters are policies
(f) Data preparation/analysis process: parameters that change field grouping and categorization are policies
(g) CORBA: parts of the APIs are policies
(h) M/A/MO: policies are explicitly defined to control/manage a system
(i) COMPA, the rules inside each component ($A$, $P$, $COM$) are policies, overall inside $P$ are policies
AN ENB EXAMPLE OF POLICIES

a) eNB Configuration Parameters

b) eNB, MO & OSS

c) OSS with ECA Policies

d) SON Function

e) eNB with SON Function

f) eNB with SON Functions

g) eNB SON Coordination

h) Centralized SON Coordination

i) Management of SON Coordination
AN ENB EXAMPLE OF POLICIES

› When we take a Radio Base Station (RBS) or an eNodeB (eNB), the scope we use defines what policy is

(a) eNB: config params are policy
(b) exposed behaviors of an eNB MO are policies
(c) OSS with ECA policies: they are the policies
(d) SON function: configuration parameters are policy
(e) eNB with SON: then the SON function is a policy
(f) eNB with SON functions: all SON functions are policy
(g) Inside SON function: any means that direct choices in its behavior are policies
(h) SON coordination system: then its internal choices are policies
(i) Management of SON coordination system: then the policies are in the management system
WHAT’S THE PROBLEM WITH POLICY?
POLICY ITSELF IS INARIANT(!!?)

› Policy in/for
  – Security systems, operating systems, communication systems, database systems, IP networks, telecom management, autonomic networks, business rules, production rule, …

› Policy as
  – Rules for access control, artifact of control, rules for control, active triggers, transaction control, unit of governance, …

› SDOs
  – IETF (PCIM, COPS, SUPA), DMTF (CIM), TMF (SID, DEN-ng, Zoom), TMA, 3GPP (PCC), ETSI (MANO), …

› Policy models (some)
  – Control / config. parameter, decision table, E/CA, goal, utility function, promise, RBAC, RETE, semantic, business rule, workflow, declarative, …

› Policy languages (too many)

› Architecture
  – PAP, PDP, PEP, PIP, PR, PXP, …
ADDRESSING THE PROBLEM

1: Identify invariants
▶ Policy stimulated by input, produces output (request, action, answer)
▶ Choices in behavior: set of capabilities and controlled resources

1a: more invariants
▶ Policies have different states, 1-many
▶ Stimuli can be implicit or explicit, naïve or simple or complex

2: Maximize invariants
▶ Express policy decision making as a state machine
▶ Application domain defines resources & capabilities

2a: more maximized invariants
▶ Policies don’t enforce/commit actions
▶ Most in/output information is contextual
▶ We can build a generic executor
GOOD START, BUT NOT ENOUGH

› How to support multiple application domains? For real?
› How to support different policy use cases?
› How to support different input/output systems?
› How to build immutable policy infrastructure?
› Will that work?

› How to support >1 policy model?
› How do support > 1 policy language?
› How to support different architectures?
› How to integrate different SDOs?
› What about semantics?
› What about context?
› What about policy conflicts?
› What about the lifecycle of policies?
LET’S LOOK AT A (NEW) POLICY PATTERN!
OBJECTIVES


Source policies, original format, specific authoring tools
Transformation engine, using templates & transformation rules
Universal policy specification, harmonizing processing (e.g. conflicts) & execution
Universal policy execution & connection to stimulus (e.g. trigger) & response (e.g. actioning) systems

Stimulus (e.g. Trigger) → Response (e.g. Actions)
A NEW POLICY PATTERN

Common Policy

- Trigger Component
- Policy
- Action Component

Policy Pattern

- Trigger Mechanism
- Trigger System
- Complex Event Processing
- Analytics System
- Models
  - Domain
  - Semantic
  - Information
  - Data
  - Context
  - Behavior
- Mechanisms
  - Authoring
  - V&V
  - Deployment
  - Triggering
  - Runtime
  - Inference
- Repositories
  - Policies
  - Meta Data
  - Context
  - Knowledge
  - Ontologies
  - Domain Obj

Concepts

- Context Awareness
- Knowledge Engineering
- Design Patterns
- DDD / DSL
- OODA
A NEW POLICY PATTERN

- Generic input & output
  - Allows to connect to variety of systems
- CEP & analytics
  - Free policy from complex events
- Actioning & orchestrator & …
  - Free policy from complex actions
- Feedback
  - Allows policy optimization & target shifts
- Models
  - Allow for advanced policy behavior
  - E.g. context-awareness, adaptive policies
- Mechanisms
  - For all parts of policy lifecycle
  - For features, such as conflict detection

- OODA loop
  - Clear separation of concern for states
  - Policy with history, tradition & experience
- Context & knowledge
  - Dynamic & semantic link to domain model
  - Flexible coupling to trigger / actioning
- Design patterns
  - Interchangeable policy logic, dynamic behavior
  - Repeating structures
  - Immutable infrastructure (policy execution)
- DDD / DSL
  - DDD: Evolving domain model
  - DSL: Taming the language zoo
A UNIVERSAL POLICY THEORY (UPT)
3 main parts
- UPM, UPEE, Transformer

UPM defines
- Reference Model
- Policy Models

UPEE defines
- Immutable execution infrastructure for policies

Transformer links
- PMs to UEPS to UPEE

Supporting Models / Languages
PM – EXAMPLE ACTION POLICY

\[ M^{eca} = \left( \delta(s, e_{1..k}) = s_c, \delta(s_c, e_1) = s_a, \delta(s_a, e_{1..k}) = \rho \right) \]

Event \( s_e \) (\( s^{ke} \))
- Output \( v \rightarrow_{s_o} \)
- State \( q_0 \)
- Final States \( \varnothing \)

UEPS Template for Action Policy (ECA)
- Policy Type: eca
- State Machine \( M^{eca} \)
- State Set \( Q \)
- State Tuple \( s^{eca} \)
- Event \( s_e \)
- Condition \( s_c \)
- Action \( s_a \)

Alphabet \( \Sigma \)
- Transitions \( \delta \)
- Initial State \( q_0 \)
- Final States \( \varnothing \)

UEPS Template for Action Policy (ECA)
- Policy Type: eca
- State Machine \( M^{eca} \)
- State Set \( Q \)
- State Tuple \( s^{eca} \)
- Event \( s_e \)
- Condition \( s_c \)
- Action \( s_a \)

Alphabet \( \Sigma \)
- Transitions \( \delta \)
- Initial State \( q_0 \)
- Final States \( \varnothing \)

Condition \( s_c \) (\( s^{ch} \))
- Output \( v \rightarrow_{s_o} \)
- State \( q_0 \)
- Final States \( \varnothing \)

Action \( s_a \) (\( s^{ac} \))
- Output \( v \rightarrow_{s_o} \)
- State \( q_0 \)
- Final States \( \varnothing \)

ECAPolicy \( \rightarrow \) State \( \rightarrow \) EventState \( \rightarrow \) ConditionState \( \rightarrow \) ActionState

\[ \pi = eca, S^{eca} = (s_e, s_c, s_a), \gamma^{eca}_0 = \gamma_a = (\text{condition}, \text{true}|\text{false}) \]

\[ M^{eca} = \left( \{\bullet, \rho\} \cup S^{eca}, E^{eca}_c \cup E^{eca}_c \cup E^{eca}_a \cup \varnothing \right) \]

\[ \delta(s, e_{1..k}) = s_c, \delta(s_c, e_1) = s_a, \delta(s_a, e_{1..k}) = \rho, \delta(s_a, \varnothing) = \rho \]
PM - EXAMPLE MEDA POLICY

\[ \pi = \text{meda}, S_{\text{meda}} = (s_m, s_e, s_d, s_a) \]

\[ M_{\text{meda}} = \left\{ \{\cdot, \circ\} \cup S_{\text{meda}}, e^t_m \cup e^e_m \cup e^o_m \cup e^o_d \cup E^a \cup \{\emptyset\}, \delta(s_e, e^e) = s_d, \delta(s_d, e^o_d) = s_a, \delta(s_a, e^1_{1..k}) = \circ, \delta(s_m, \emptyset) = \circ, \delta(s_d, \emptyset) = \circ, \delta(s_a, \emptyset) = \circ, \rangle, \cdot, \{\circ\} \right\} \]
### POLICY MATRIX

<table>
<thead>
<tr>
<th>Stimuli</th>
<th>Purpose / Model</th>
<th>Context</th>
<th>Flavor/States/Tasks</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>Configuration</td>
<td>Create / Configure</td>
<td>No Context</td>
<td>Simple / God (s/s/t)</td>
<td>Obligation</td>
</tr>
<tr>
<td>Report</td>
<td>Repair</td>
<td>Event Context</td>
<td>Simple Sequence (s/+s/t)</td>
<td>Authorization</td>
</tr>
<tr>
<td>Monitoring</td>
<td>Mitigate / Permit</td>
<td>Policy Context, ro</td>
<td>Simple Selective (s/+s+t)</td>
<td>Intent</td>
</tr>
<tr>
<td>Analysis</td>
<td>Escalate / Delegate</td>
<td>Policy Context, w</td>
<td>Selective (s/+s/+t)</td>
<td>Delegation</td>
</tr>
<tr>
<td>Prediction</td>
<td>Prevent / Enforce</td>
<td>Global Context, ro</td>
<td>Classic Directed (d/+s/t)</td>
<td>Fail / Error</td>
</tr>
<tr>
<td>Feedback</td>
<td>History / Experience</td>
<td>Global Context, w</td>
<td>Super Adaptive (d/+s/+t)</td>
<td>Feedback</td>
</tr>
</tbody>
</table>

- **Stimuli**: Information, Semantics
- **Policy Matrix**: Purpose / Model, Context, Flavor / States / Tasks
- **Response**: Class, Type, Semantics

- **Create / Configure**: Obligation, Promise, Intent
- **Repair**: Obligation, Utility, Goal
- **Mitigate / Permit**: Obligation, Authorization
- **Escalate / Delegate**: Refrain, Delegation
- **Prevent / Enforce**: Adaptive, Utility, Goal
- **History / Experience**: Context-aware, Meta

- **Policy Matrix**:
  - **Purpose / Model**: Create / Configure
  - **Context**: No Context
  - **Flavor / States / Tasks**: Simple / God (s/s/t)

- **Response**:
  - **Obligation**: how should it be done
  - **Authorization**: what must be permitted
  - **Intent**: what should be done
  - **Delegation**: who should do something
  - **Fail / Error**: logic? engine? context?
  - **Feedback**: why this decision

- **depends on trigger system capabilities**
- **purposes should match stimuli, models are examples**
- **select one matching purpose and model**
- **select one that suits application (except when adaptive models)**
- **depends on actioning system capabilities**
UPEE – CLUSTERING EXAMPLES

(a) stimulus → UPx → UPe → response
(b) stimulus → UPx → UPe → response
(c) stimulus → UPx → UPx → UPx → UPe → response
(d) stimulus → UPx → UPx → UPx → UPe → response
(e) stimulus → UPx → UPx → UPe → response
(f) stimulus → UPx → UPx → UPe → response
(g) stimulus → UPx → UPx → UPe → response
(h) stimulus → UPx → UPx → UPe → response
(i) stimulus → UPx → UPx → UPe → response
(k) stimulus → UPx → UPx → UPe → response
UPEE – CLUSTERING EXAMPLES

(a) Single source/target, single UPx
   - Simple forward
(b) Multiple sources/targets, single UPx
   - Simple forward
(c) Single source/target, multiple UPx
   - Multithreading (MT) in UPe
(d) Multiple sources/targets, multiple UPx instances
   - Simple forward & MT in UPe
(e) Multiple non-MT UPe in UPec
   - Simple event routing
(f) Multiple MT UPe in UPec
   - Simple event routing
(g) Mixed UPe in UPec
   - Simple event routing
(h) Multiple non-MT UPec in UPec
   - Intelligent event routing
(i) Multiple mixed UPec in UPec
   - Intelligent event routing
(k) Mix of UPec in multiple UPec
   - External intelligent event routing
   - Optimized with UPec internal routing
UPEE – DEPLOYMENT OPTIONS

(a) for an i/f or a class
(b) for an application
(c) for a component
(d) as a service
(e) in a control loop
(f) on cloud compute nodes
(g) cloud example
UPEE – DEPLOYMENT OPTIONS

(a) For an interface or class
- Either UPx or UPe as association

(b) For an application
- UPx as object for single policies
- UPe as object for multiple policies

(c) For a component (as service)
- UPe as service for requests
- UPe as service for requests

(d) As a service (PolaS)
- One or more UPe with service i/f
- One or more Upec/UPec with service i/f
- One or more Upec/UPec with service i/f

(e) In a control loop
- UPe as decision making part
- UPe as decision making part

(f) On cloud compute nodes
- Nodes with only UPe or Upec
- Nodes with any combination of UPe, UPe

(g) A cloud example
- Left: 2 UPe managing several UPe on different cloud nodes
- Right: 2 large UPe with different UPe/UPec deployments
QUICK APEX INTRO

APEX: ADAPTIVE POLICY EXECUTION
UPEE is the Concept
› Harmonize policy models
› Provide single execution environment
› Facilitate conflict processing
› Features
  – Context aware
  – Adaptive logic selection
  – Flexible clustering options
  – Flexible deployment options
  – Flexible policy deployment

APEX is the Implementation
› Fully featured policy engine
› Many logic executors
  – e.g. Java, JavaScript, Python, MVEL, Ruby
› Integrated access to context
  – shared information
› Flexible deployment options
› Flexible link to trigger/actioning systems
APEX FEATURES

› Flexibility similar to programming languages, rule engines, workflows etc.
› Fully context-aware, adaptable, and adaptive decision making
› Facilitates intelligent conflict detection and mitigation
   – at authoring time, deployment time, and runtime
› Event-sourced system
   – connectable to virtually any trigger/actioning system
› A simple, flexible, event-sourced state machine as execution model
› Light, fast, scalable policy engine
   – Engine: 131 kB, Core+services: ~800 kB (optimizable for deployment)
   – Everything (APEX + all external dependencies): ~97 MB
   – Performance: > 150k Policy Transactions per second (non-optimized system)
COMPONENT, ENGINE & APP

Stimulus

Policy Application

Policy Engine

Response
Policy Application

Policy Engine

Policy Statements
any statement translatable to policy

Policy Model, including Context
JSON (also XML)

Policy MEDA
Policy ECA

Context Model

API
APEX Model

Statement(s)
any abstraction

REST
HTML5

CLI
Shell

XTxt
DSLs

Your Policy Statements
APEX Boundary

Your Use Cases, Applications, or Products

APEX Authoring (AP-AUTH) / Domain-specific tooling

APEX Policy Model
(Executable Policy Statement)
& Deployment (AP-DEP)

APEX Engine (AP-EN)

Your Use Cases, Applications, or Products

APEX Authoring (AP-AUTH) / Domain-specific tooling

APEX Policy Model
(Executable Policy Statement)
& Deployment (AP-DEP)

APEX Engine (AP-EN)
APEX CONCEPTS
POLICY LIFE CYCLE

Authoring (AP-AUTH)

- PolicyDefinition
  - StateDefinition
    - TaskSelectorDefinition
      - TaskDefinition
        - StateFinalizerDefinition
          - ParameterDefinition
          - StateFinalizer
            - Parameter
            - TaskLogic
              - TaskSelectionLogic
              - StateFinalizerLogic

Runtime (AP-EN)

- Policy
  - State
    - Task
      - 1..* TaskSelectionLogic
      - 1..* TaskLogic

Deployment AP-DEP

- 2..* Event
APEX COMPONENTS & FLOW
GOALS & CONTEXT

› Flexible policy execution
› In an Adaptive Policy Engine (AP-EN)
› Link policy with changeable goals
  – Business goals
  – Domain goals
› Link policy with other context
  – Retrieved in real time

› Policies can adapt to changes at runtime
Flexible policy execution
- In an Adaptive Policy Engine (AP-EN)
- Connect to any I/O system

Register policy trigger
- Trigger context defined by trigger system
- Receive response from policy
  - Response context defined by policy

APEX I/O & CONTEXT

Stimulus Event (e.g. Trigger) → Policy Engine (AP-EN) → Response Event (e.g. Actions) → Actioning System

with Incoming Context from triggering system

Policy (retrieved in real time)

with Outgoing Context from policy
› Flow: Trigger $\rightarrow$ Engine $\rightarrow$ Policy $\rightarrow$ Engine $\rightarrow$ Actioning System

› Context: in all events, per policy type, global (r/w), external (r)
APEX COMPONENTS

- APEX defines and implements a set of sub-systems
  - **AP-AUTH** – policy authoring
    - Programmatic, CLI editor, REST editor, Xtext DSL editor
  - **AP-DEP** – policy deployment to one or more engines
    - Simple deployment
  - **AP-EN** – the actual policy engine
    - Fully featured, multithreaded engine with various configuration options
  - **AP-CTX** – adapters and collectors for external context information
    - Distributed context across engines, VMs, hosts, networks
  - **AP-KB** – a knowledgebase with all information about policies
### Engine Configuration

#### Input System
- **Stimulus Events**
  - Standard Input APEX engine console
  - File In configured file, local access
  - Kafka Streams Kafka system with topics
  - JMS Messages JMS system with topics
  - Websocket Client any WS application
  - Websocket Server standard WS server
  - REST Client any HTTP client
  - REST Server HTTP server

#### Input Options
- **Standard Input**
  - Standard read from stdin
  - Standard read from file
- **Kafka Streams**
  - Kafka PI read from topic(s)
- **JMS Messages**
  - JMS PI read from topic
- **Websocket Client**
  - WS PI server reads from clients
- **Websocket Server**
  - WS PI client reading from server
- **REST Client**
  - REST Server PI Grizzly server
  - REST Server PI APEX servlet
  - REST Client PI HTTP client

#### APEX Engine
- **Event Protocol Options**
  - JSON all I/O
  - XML most I/O
  - JMS Text JSON, JMS only
  - JMS Object JMS only

#### Context Handler Options
- **Java (Std)**
  - any Java implementation
- **AVRO PI**
  - an AVRO specification

#### Execution Options
- **Java PI**
  - execute Java logic
- **Jython PI**
  - execute Jython logic
- **MVEL PI**
  - execute MVEL logic
- **Javascript PI**
  - execute Javascript logic
- **JRuby PI**
  - execute JRuby logic
- **Custom PI**
  - any other executor

#### Output System
- **Response Events**
  - **Standard Output** APEX engine console
  - **File Out** configured file, local access
  - **Kafka Streams** Kafka system with topics
  - **JMS Messages** JMS system with topics
  - **Websocket Client** any WS application
  - **Websocket Server** standard WS server
  - **REST Client** any HTTP client
  - **REST Server** HTTP server
APEX RESOURCES
SOURCE CODE, DOCUMENTATION, TALKS, PAPERS

APEX Public (Github)
› Documentation (GH Pages)
› Code Repository

APEX Public (ONAP)
› Talk at Developer Forum Dec’2017
  – PDF at the bottom of the page

Related Papers
› In APEX Docs
› Research Gate APEX Project