ONAP

OPEN NETWORK AUTOMATION PLATFORM

ERICSSON

NM LAB

State, Context, Adaptability, and Scale for Self-Learning Closed Loop Policies

Liam Fallon
Sven van der Meer
John Keeney

12th December 2017

Quality Attributes for a Policy Framework

The Attributes

- State: Keep track of where we are and what we’re doing
- Context: Keep track of what’s going on around us
- Adaptability: Able to be changed and to change in response to where we are, what we’re doing, and what’s going around us

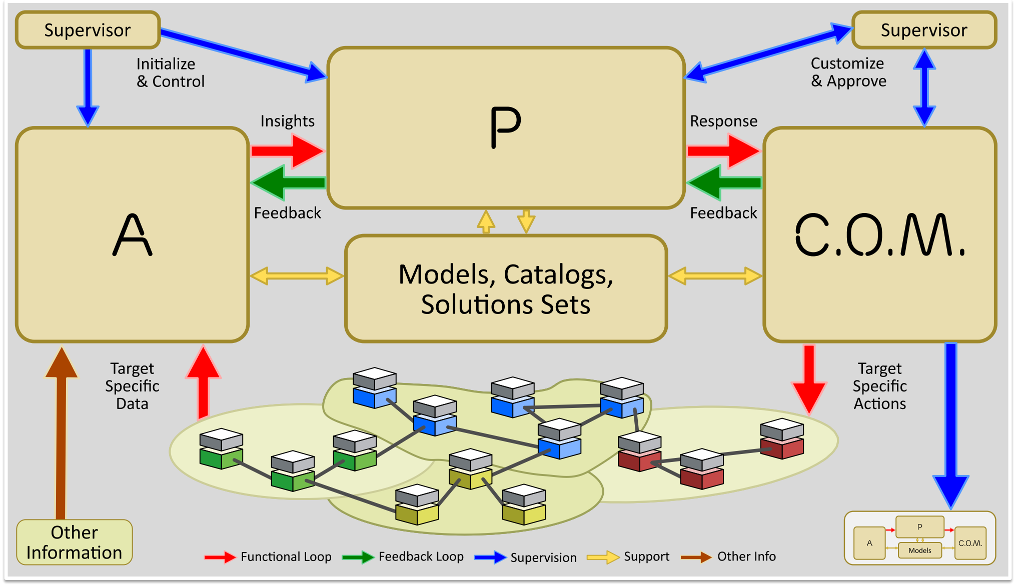
Whilst allowing

- Scale: The ability to do the three bullets above while increasing and decreasing our capacity depending on the load

With the ultimate goal of

- Self-learning: Harnessing and governing machine learning to provide learnt policies

Policy for Closed Loops, Work since 2012



COMPA, a reference architecture for closed loops

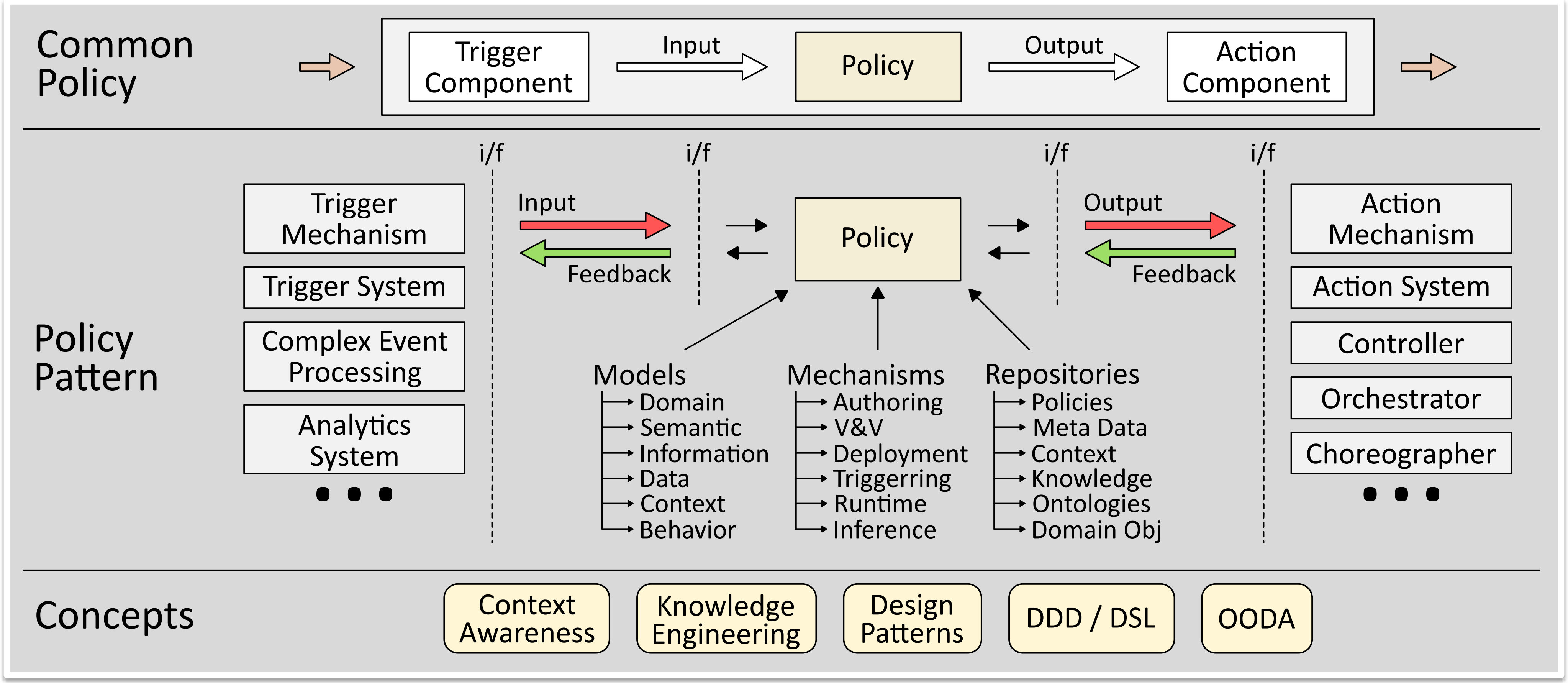
Control, Orchestration, Management, Policy, Analytics

- Analytics, Control and Orchestration becoming complex
- Policies in Control Loops were static, unstructured, and in silos
- Policy Program
  - Established well founded theory for policy in Closed Loops
  - Developed a pre-production policy system to apply that theory in practice

UNIFIED POLICY THEORY (UPT) PRIMER

For full details on Unified Policy Theory, see our research papers in the APEX project in ResearchGate

Policy Patterns and Roles



Common Policy

Trigger Component → Input → Policy → Output → Action Component

Policy Pattern

Trigger Mechanism

Trigger System

Complex Event Processing

Analytics System

• • •

i/f

Input

Feedback

i/f

Policy

i/f

Output

Feedback

i/f

Action Mechanism

Action System

Controller

Orchestrator

Choreographer

• • •

Models
- Domain
- Semantic
- Information
- Data
- Context
- Behavior

Mechanisms
- Authoring
- V&V
- Deployment
- Triggerring
- Runtime
- Inference

Repositories
- Policies
- Meta Data
- Context
- Knowledge
- Ontologies
- Domain Obj

Concepts

Context Awareness

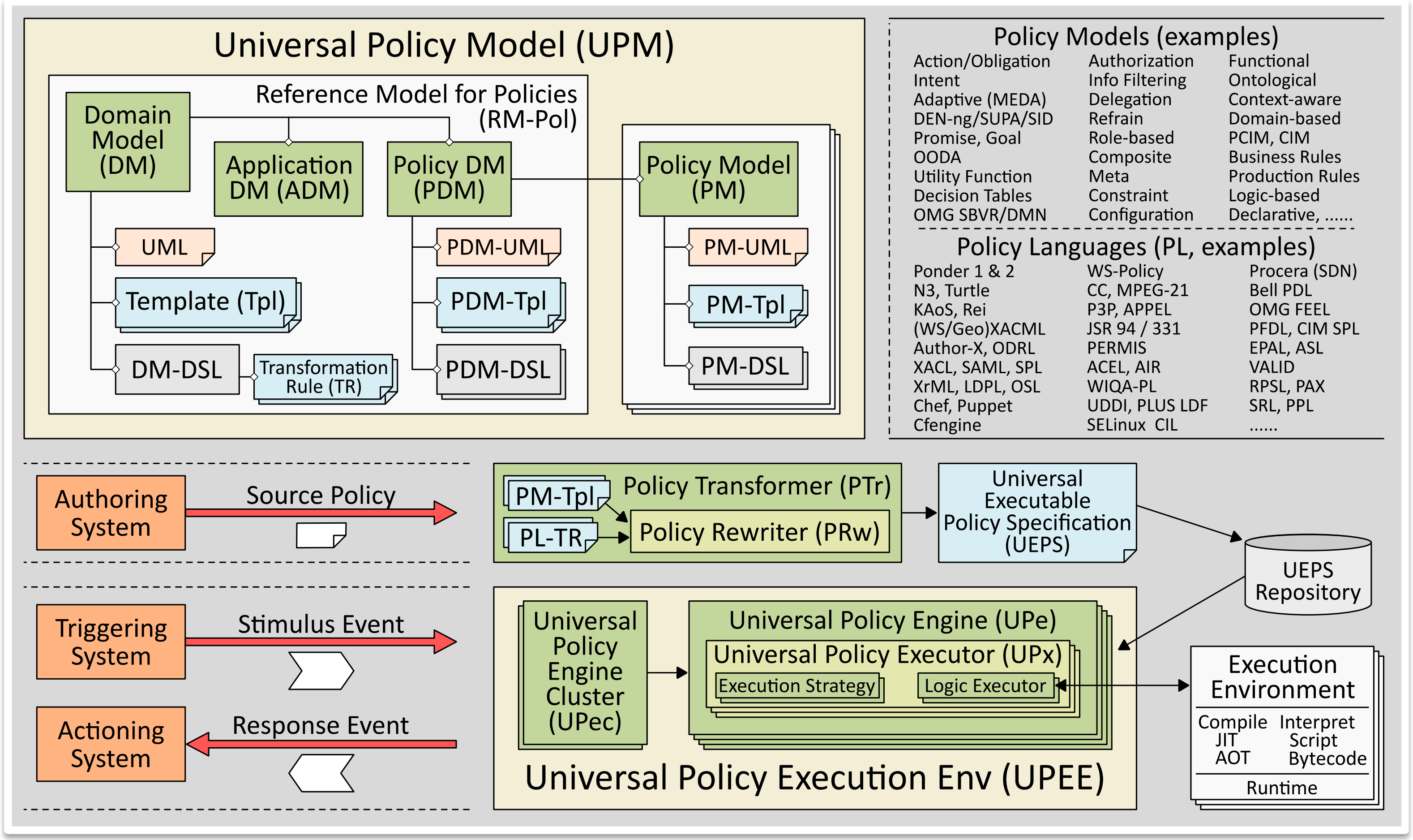
Knowledge Engineering

Design Patterns

DDD / DSL

OODA

Unified Policy Model



Policy Models (examples)

Action/Obligation
Intent
Adaptive (MEDA)
DEN-ng/SUPA/SID
Promise, Goal
OODA
Utility Function
Decision Tables
OMG SBVR/DMN

Authorization
Info Filtering
Delegation
Refrain
Role-based
Composite
Meta
Constraint
Configuration

Functional
Ontological
Context-aware
Domain-based
PCIM, CIM
Business Rules
Production Rules
Logic-based
Declarative, ...

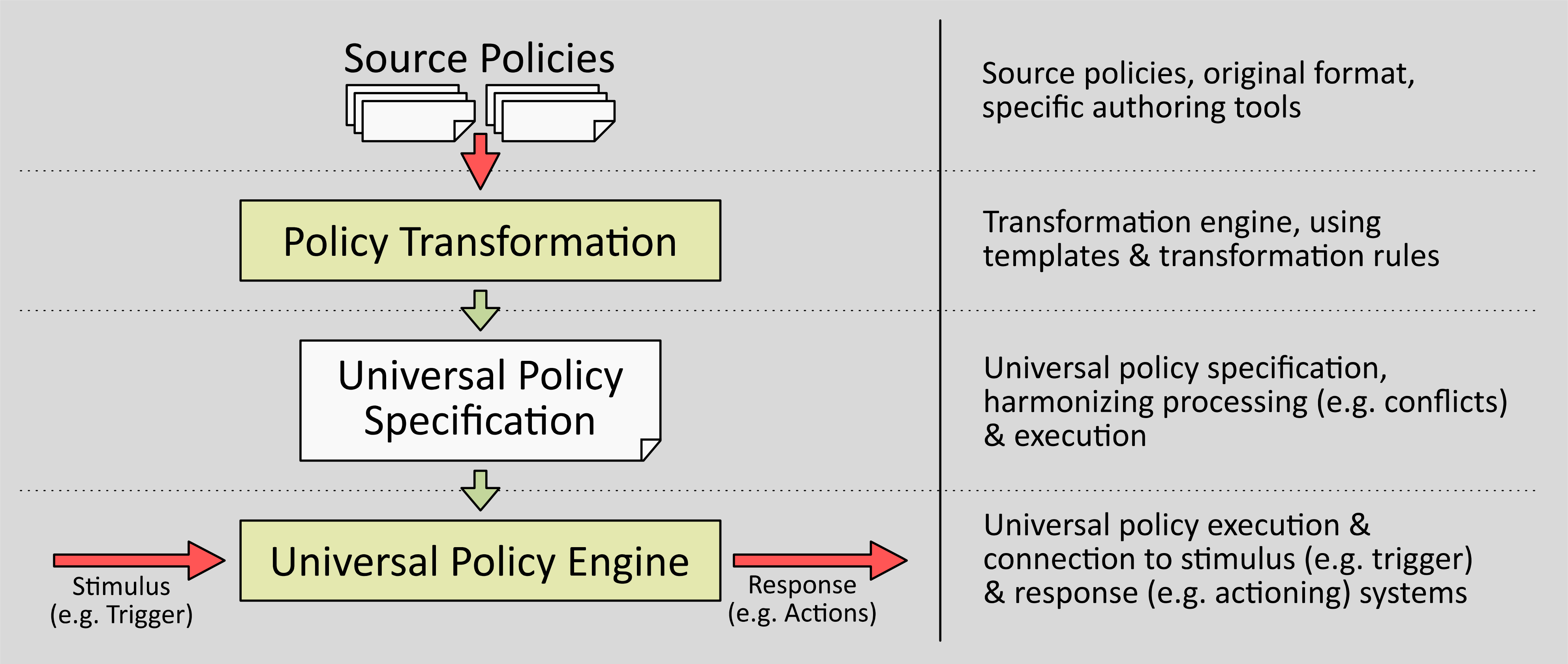
Policy Languages (PL, examples)

Ponder 1 & 2
N3, Turtle
KAoS, Rei
(WS/Geo)XACML
Author-X, ODRL
XACL, SAML, SPL
XrML, LDPL, OSL
Chef, Puppet
Cfengine

WS-Policy
CC, MPEG-21
P3P, APPEL
JSR 94 / 331
PERMIS
ACEL, AIR
WIQA-PL
UDDI, PLUS LDF
SELinux CIL

Procera (SDN)
Bell PDL
OMG FEEL
PFDL, CIM SPL
EPAL, ASL
VALID
RPSL, PAX
SRL, PPL
......

Policy Transformations



Source Policies

Policy Transformation

Universal Policy Specification

Universal Policy Engine

Stimulus (e.g. Trigger)

Response (e.g. Actions)

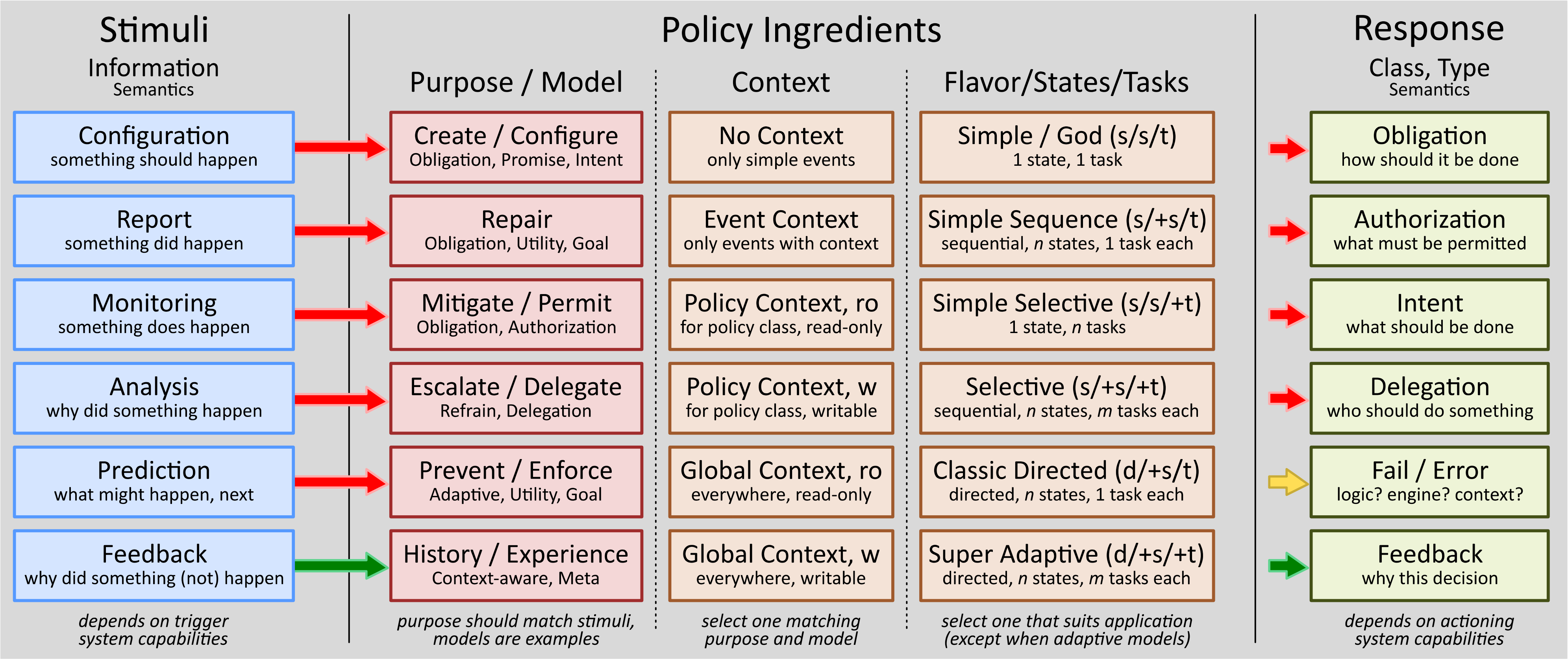
Source policies, original format, specific authoring tools

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

Policy Matrix



Stimuli Information Semantics	Policy Ingredients Purpose / Model	 Context	 Flavor/States/Tasks	Response Class, Type Semantics
Configuration something should happen	Create / Configure Obligation, Promise, Intent	No Context only simple events	Simple / God (s/s/t) 1 state, 1 task	Obligation how should it be done
Report something did happen	Repair Obligation, Utility, Goal	Event Context only events with context	Simple Sequence (s/+s/t) sequential, *n* states, 1 task each	Authorization what must be permitted
Monitoring something does happen	Mitigate / Permit Obligation, Authorization	Policy Context, ro for policy class, read-only	Simple Selective (s/s/+t) 1 state, *n* tasks	Intent what should be done
Analysis why did something happen	Escalate / Delegate Refrain, Delegation	Policy Context, w for policy class, writable	Selective (s/+s/+t) sequential, *n* states, *m* tasks each	Delegation who should do something
Prediction what might happen, next	Prevent / Enforce Adaptive, Utility, Goal	Global Context, ro everywhere, read-only	Classic Directed (d/+s/t) directed, *n* states, 1 task each	Fail / Error logic? engine? context?
Feedback why did something (not) happen	History / Experience Context-aware, Meta	Global Context, w everywhere, writable	Super Adaptive (d/+s/+t) directed, *n* states, *m* tasks each	Feedback why this decision
depends on trigger system capabilities	*purpose 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*

APPLYING UNIFIED POLICY THEORY IN A SYSTEM

Concept & Implementation

Theory

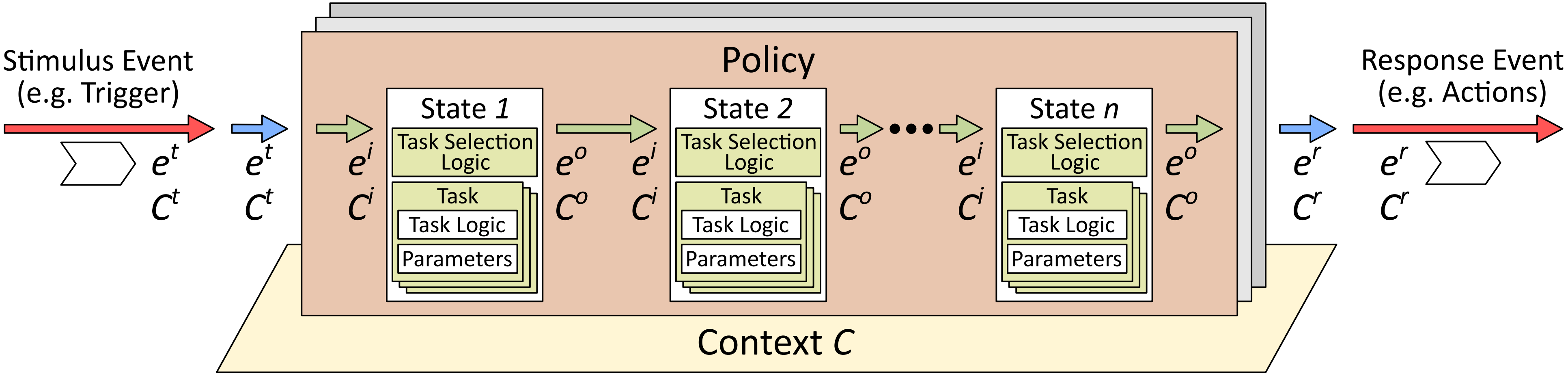
- Harmonize policy models
- Provide single execution environment
- Facilitate conflict processing
- Features
  - Context aware
  - Adaptive logic selection
  - Flexible clustering options
  - Flexible deployment options
  - Flexible policy deployment

Practice

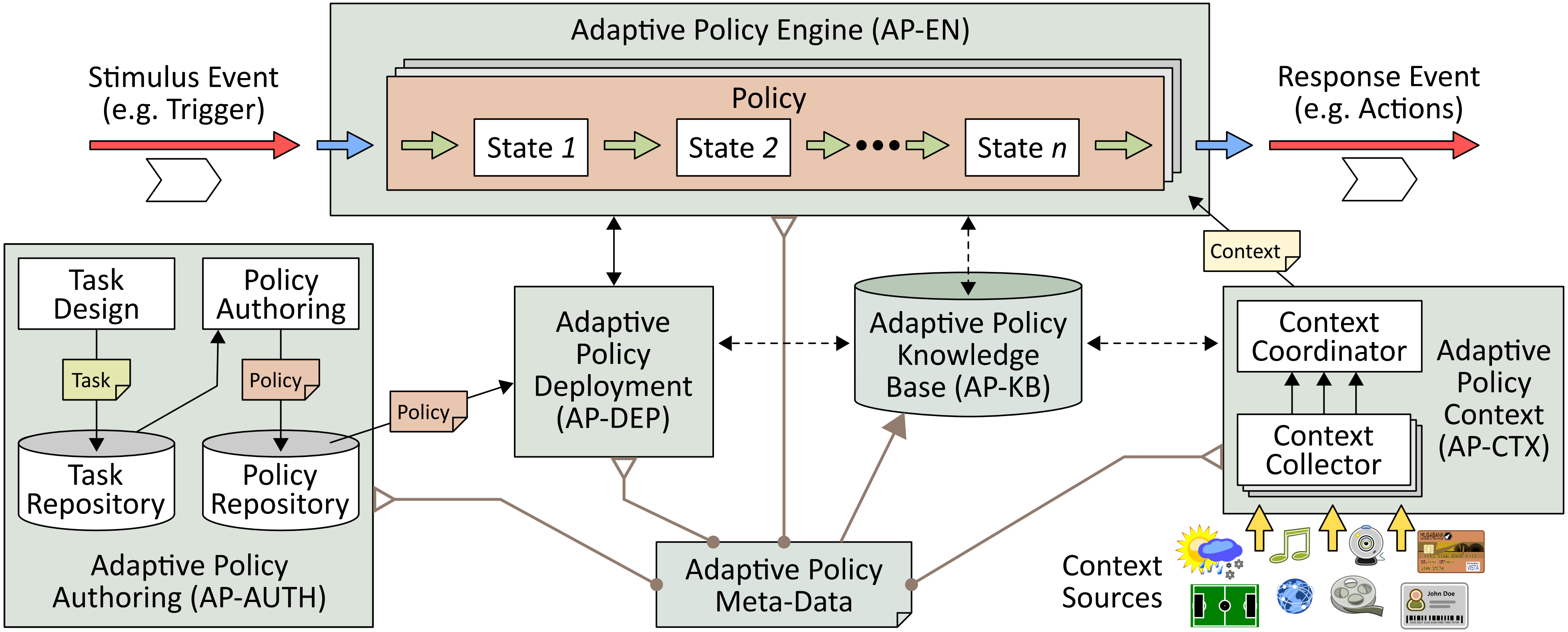
- An editor to create policy models
- An engine to run policy models
- Control of state and context
- Features
  - Context defined at run time using metadata
  - Logic loaded at run time
  - Policy Deployed as metadata
  - Policies/context distributed for scale

Adaptive Policy Engine: Event Flow & Context

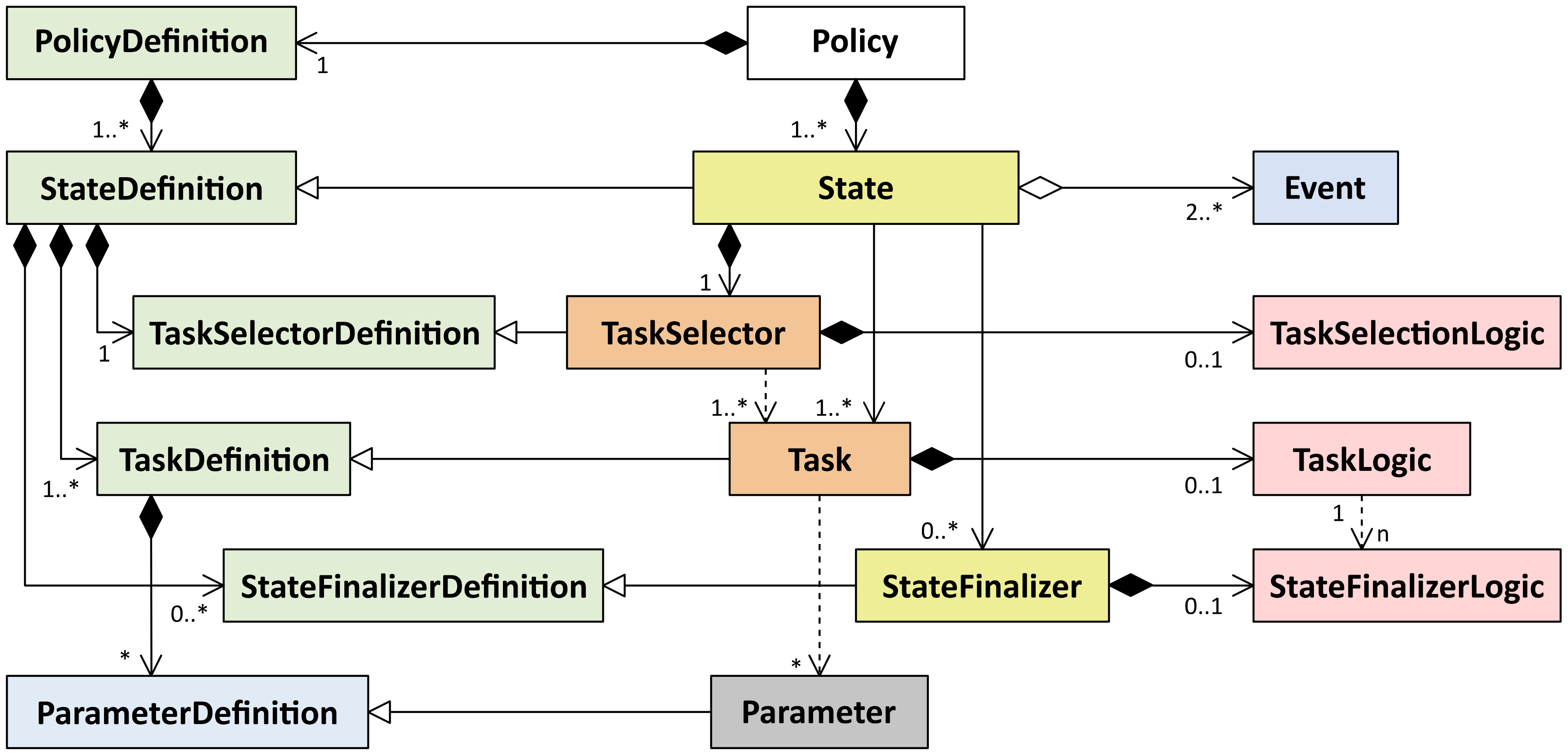
- Flow: Trigger → Engine → Policy → Engine → Actioning System
- Context: in all events, per policy type, global (r/w), external (r)



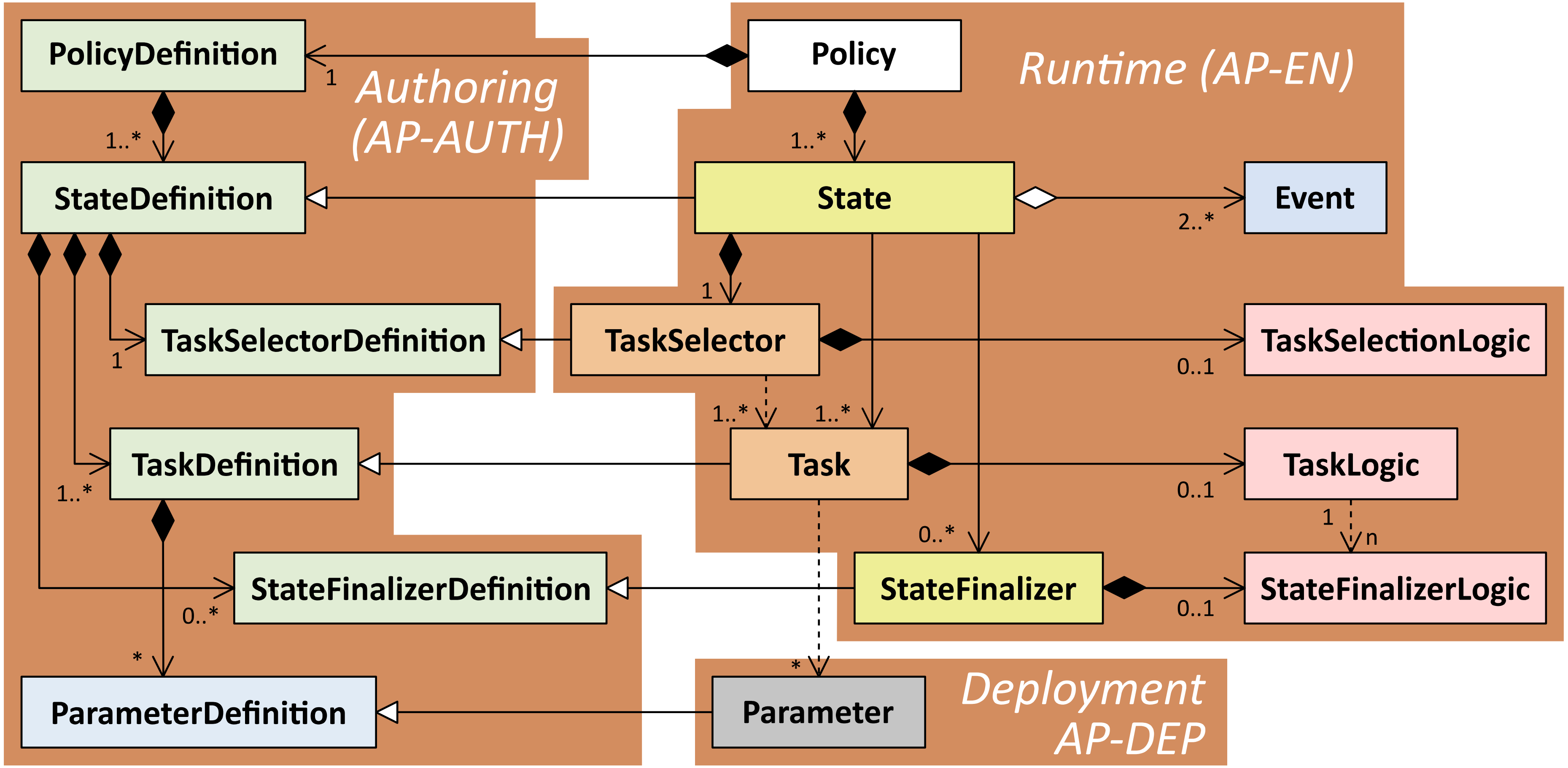
Policy Environment Components & Flow



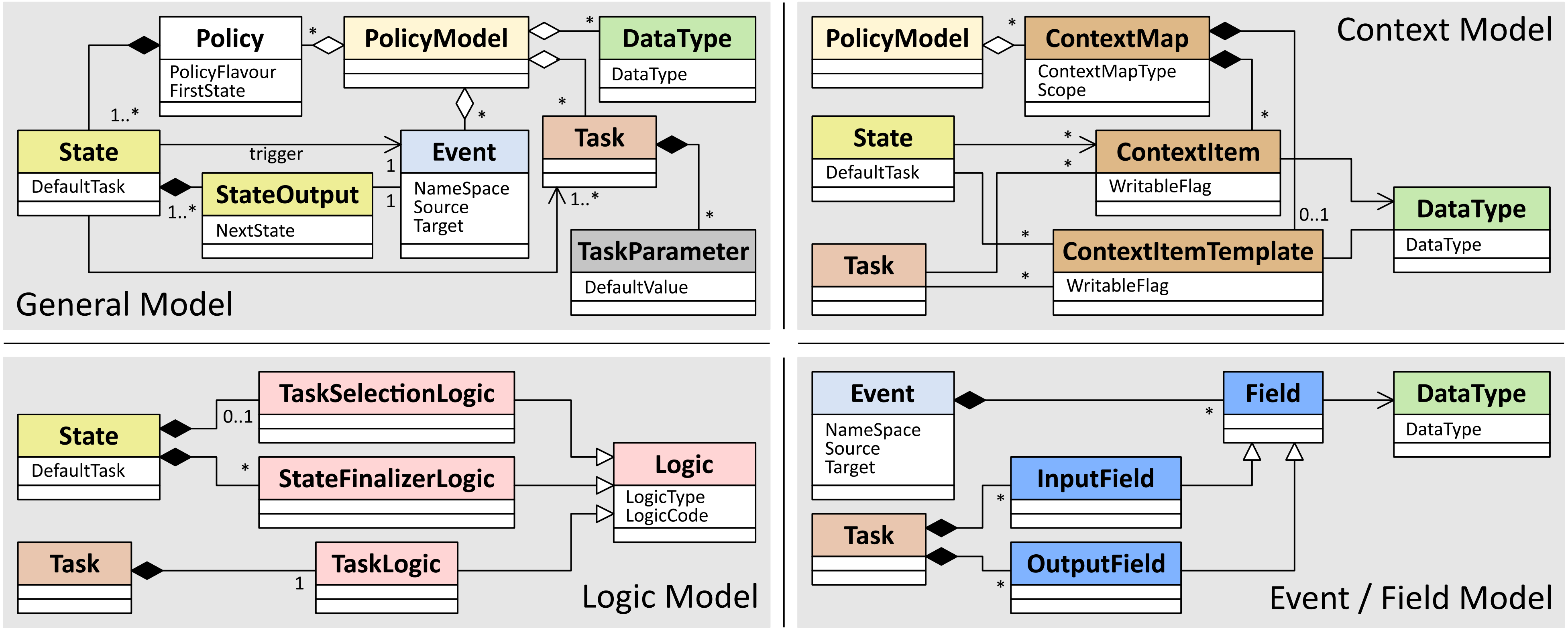
The Universal Policy Theory (UPT) Policy Model as UML



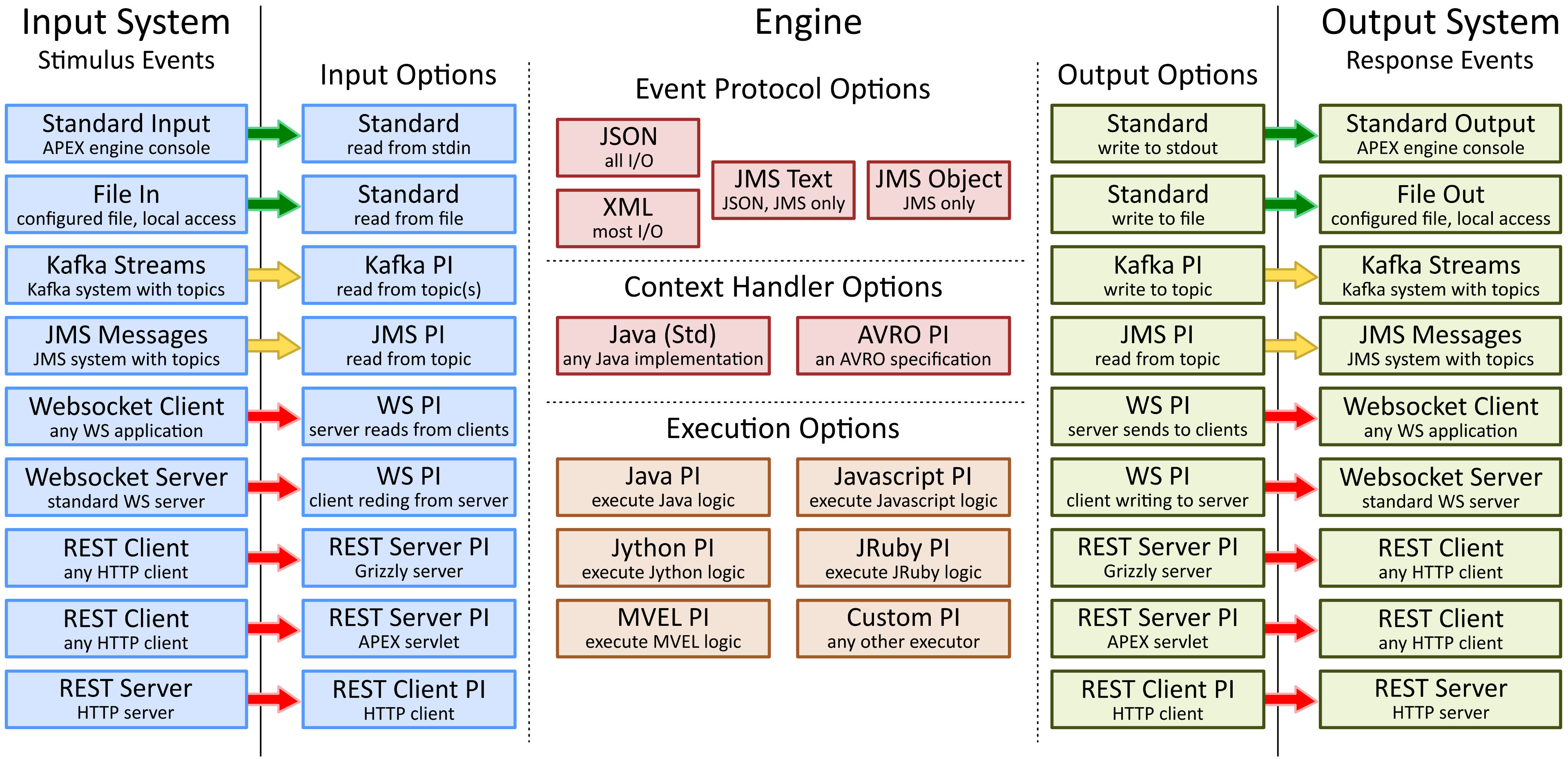
The UPT Policy Model mapped to a Policy System



Engine Execution Model



Engine Configuration and Plugins



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 Client — any HTTP client
- REST Server — HTTP server

Input Options

- Standard — read from stdin
- Standard — read from file
- Kafka PI — read from topic(s)
- JMS PI — read from topic
- WS PI — server reads from clients
- WS PI — client reding from server
- REST Server PI — Grizzly server
- REST Server PI — APEX servlet
- REST Client PI — HTTP client

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
- Javascript PI — execute Javascript logic
- Jython PI — execute Jython logic
- JRuby PI — execute JRuby logic
- MVEL PI — execute MVEL logic
- Custom PI — any other executor

Output Options

- Standard — write to stdout
- Standard — write to file
- Kafka PI — write to topic
- JMS PI — read from topic
- WS PI — server sends to clients
- WS PI — client writing to server
- REST Server PI — Grizzly server
- REST Server PI — APEX servlet
- REST Client PI — HTTP client

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 Client — any HTTP client
- REST Server — HTTP server

DISTRIBUTED CONTEXT FOR POLICY

Distributed Context for Policy

- Policy work required context sharing across policy engines
- We wanted structured context just like what’s available in management models
  - Think MIBs, Yang objects, UML classes, Java Beans, XML entities, JSON objects, …
- We went looking for a model distribution system that
  - Provides distributed context (somehow classified information)
  - Supports locking
  - Supports monitoring
  - Supports persisting
- No such distributed context framework existed

Distribution Frameworks

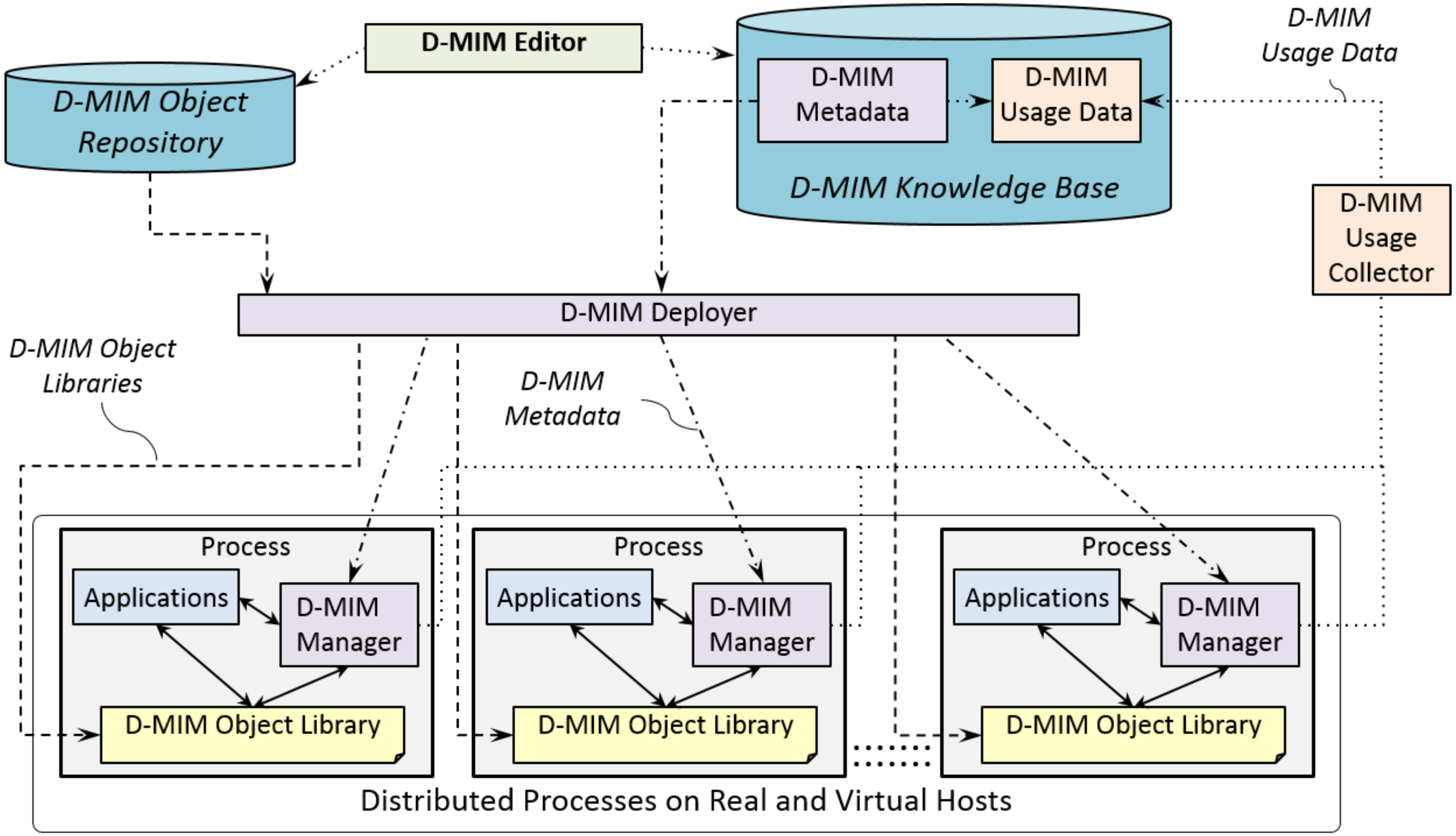
- Numerous frameworks for distributing unstructured hash maps
  - Distribution of maps of objects keyed by objects
  - E.g.: Hazelcast and Infinispan

- Some frameworks for locking
  - Transactional frameworks such as Narayana (Jboss JTA implementation) are slow
    - and very expensive in resource usage
  - No locking support on specific entries on distributed maps
  - No integrated persistence support
  - No integrated monitoring of CRUD operations on map entries

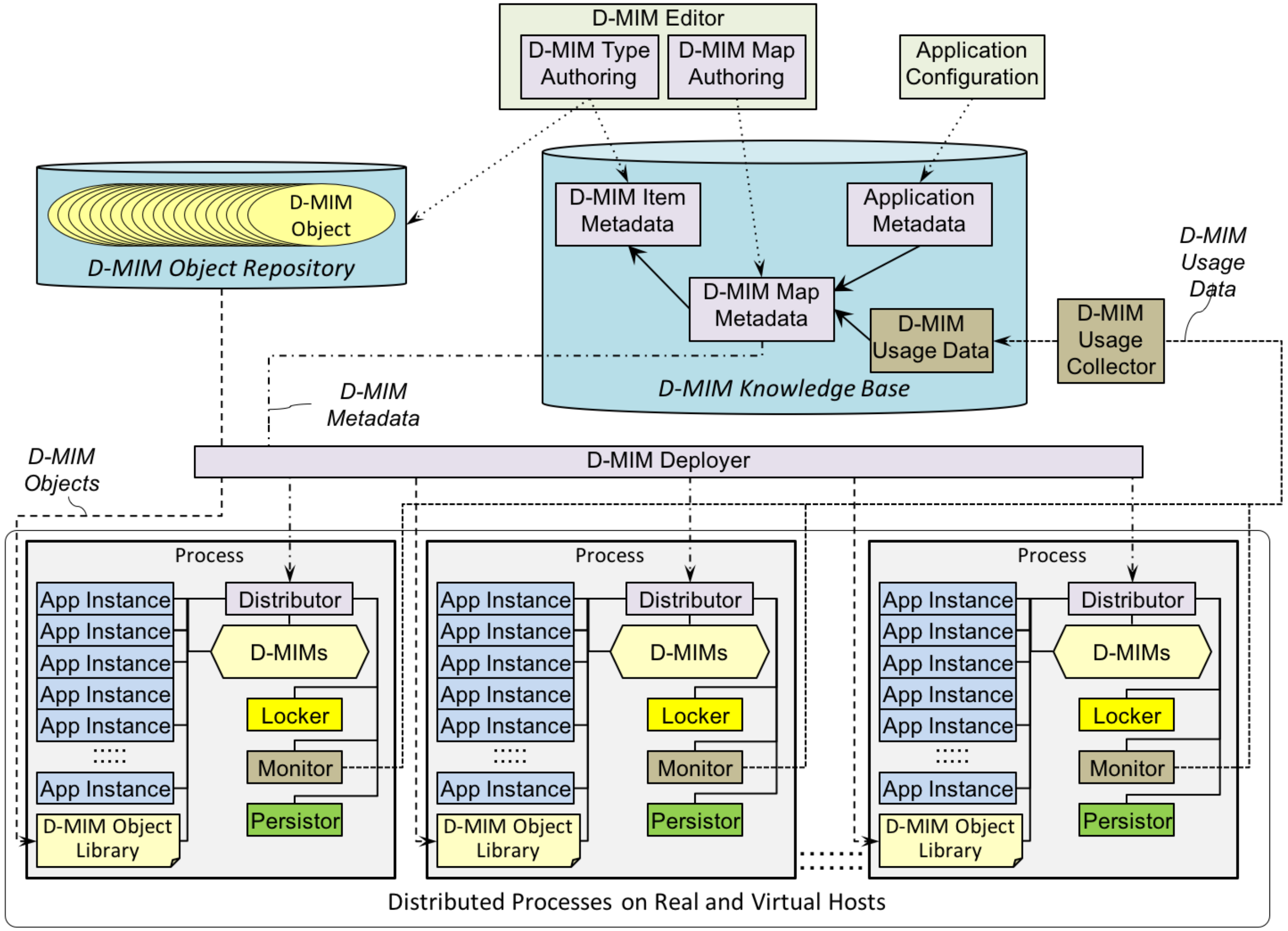
Distributed MIMs

- We decided to build a system for context that had strong structure support (just like MIMs)
  - Provides an interface to users to define highly structured context
  - Provides an interface to users to read, write, create, delete, persist, lock, and monitor context
  - Provides a plug-in architecture that allows existing distribution, locking, persistence and monitoring frameworks to be used
- D-MIM users can use distributed MIM maps
  - transparently on multiple processes, hosts, and geographic locations
  - Changes to one D-MIM copy are propagated to all others
  - Unified monitoring is supported
  - Unified locking is supported

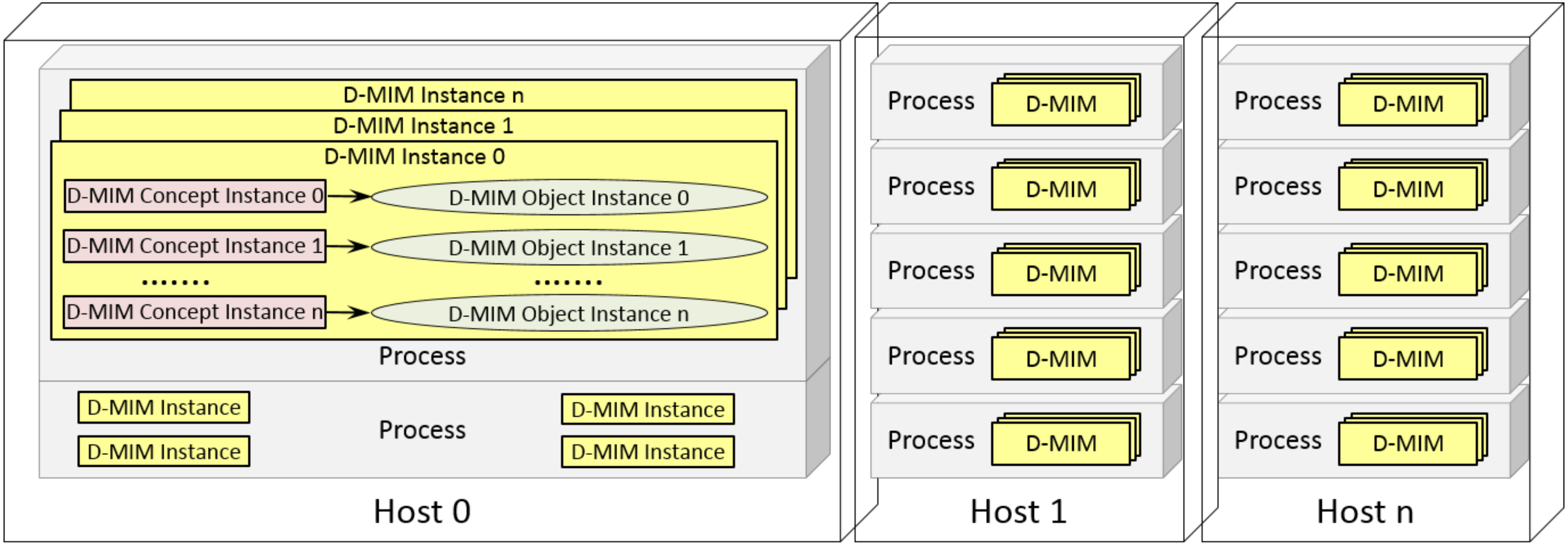
Distributed MIMs (D-MIMs) in Policy Engines



D-MIMs in a Distributed System



Distributed MIM instances on Hosts



APPLYING TO ONAP

UPT and UPEE in ONAP

- A model for policies and policy engines
  - Drools and beyond Drools, state handling
  - XACML engine, stateless
- UPM model distribution using ONAP Policy Framework
- D-MIMs and Context in Drools/plugin for Drools?
- Editor integration for policy authoring
- Context and conflict
  - Design time
  - Deployment time
  - Runtime identification
  - Runtime mitigation

ERICSSON

ADDENDUM:
SOME POLICY TERMINOLOGY

Policy & Engine

- *Policy* is an artefact that governs the choices in behaviour of a system
  - Separation of mechanism from policy
  - Has capabilities, defined in a specification, explicit / implicit trigger
  - Has multiple, partially relative, dimensions
  - Example for choices: in Network Management choices are operations on managed objects

- A *Policy Engine* is responsible for executing policies
  - Receiving triggers
  - Execute relevant policies
  - Receive actions from executed policies
  - Return / forward them
  - With all "-ities: scalability, performance, security, …

System & Application

- A *Policy System* controls and manages life cycle of policies
  - Functional and non-functional capabilities
  - Life cycle
    - Authoring
    - Deployment
    - Execution (using the engine)

- A *Policy Application* realizes a policy system
  - Builds / implements functional and non-functional capabilities

Policy Variants

Context-aware, Adaptive & Adaptable

Context-aware Policy

- makes different decisions based on context information
- static decision-making behavior
- *Fewer policies*: same trigger, different context

Policies are more flexible

Adaptable Policy

- Can change its decision making behavior
- Based on an external activity
  - outside the policy
- *Fewer policies*: same policy, multiple behaviors

Adaptive Policy

- Can change its decision making behavior
- Based on an internal activity
  - inside the policy
- *Fewer policies*: same policy, multiple behaviors
- *Policy can adapt* to target shifts

Policy Variants

Context-aware, Adaptive & Adaptable

Context-aware Policy

- Different trigger context results in different situations
- External context as part of situation or decision making can change decision
- Not understood context might signal shift in automation target

Adaptable Policy

- Non-context-aware
  - Set policy parameter
  - Set policy state logic
- Context-aware (trigger/external)
  - Based on context, set policy parameters and/or use different policy state logic

Adaptive Policy

- Change on automation target resulting in new/altered context
- Set policy parameter/state logic due to policy internal context
- Change state logic

ERICSSON