ONAP
OPEN NETWORK AUTOMATION PLATFORM

ONAP Policy Framework

Frankfurt – Overview
Pamela Dragosh – PTL

June 2020

Overview

- Policy Framework Project
  - Active project since ONAP inception and the Amsterdam Release
  - https://wiki.onap.org/display/DW/Policy+Framework+Project
  - Team meets every Wednesday in a shared meeting with the CLAMP project.

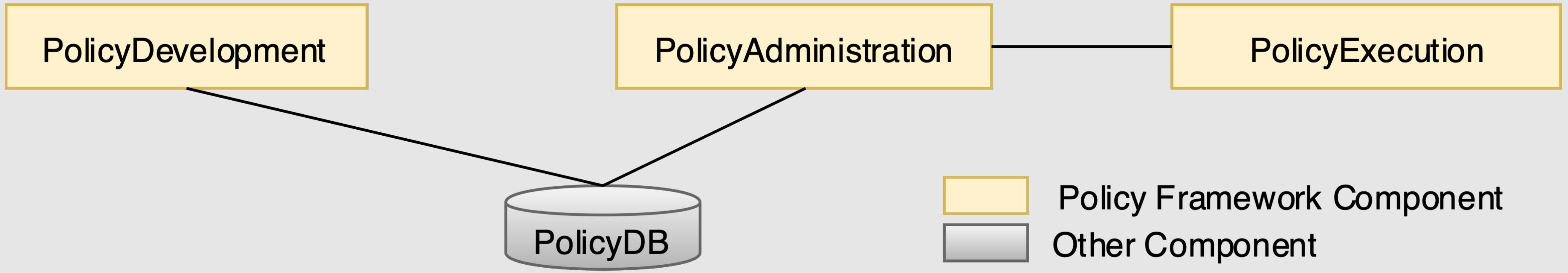
History

- Since Dublin, the project has re-designed and re-built the Policy Framework components.
  - Clear separation of Policy Design and implementation
    - Use of TOSCA Policy Syntax
  - 3 sets of API's
    - Policy Lifecycle API for performing the CRUD
    - Policy Administration API for managing PDP groups and deploying policies to PDPs
    - Decision API for ONAP components to use to render decisions on which policy(s) to enforce
  - 6 Lightweight, scalable microservices that make up the Platform
    - API and PAP
      - Policy data is managed in a MariaDb
    - 3 PDPs: XACML, Drools, Apex
      - Drools requires a nexus database to be available
    - Distribution component that receives SDC Service Distributions

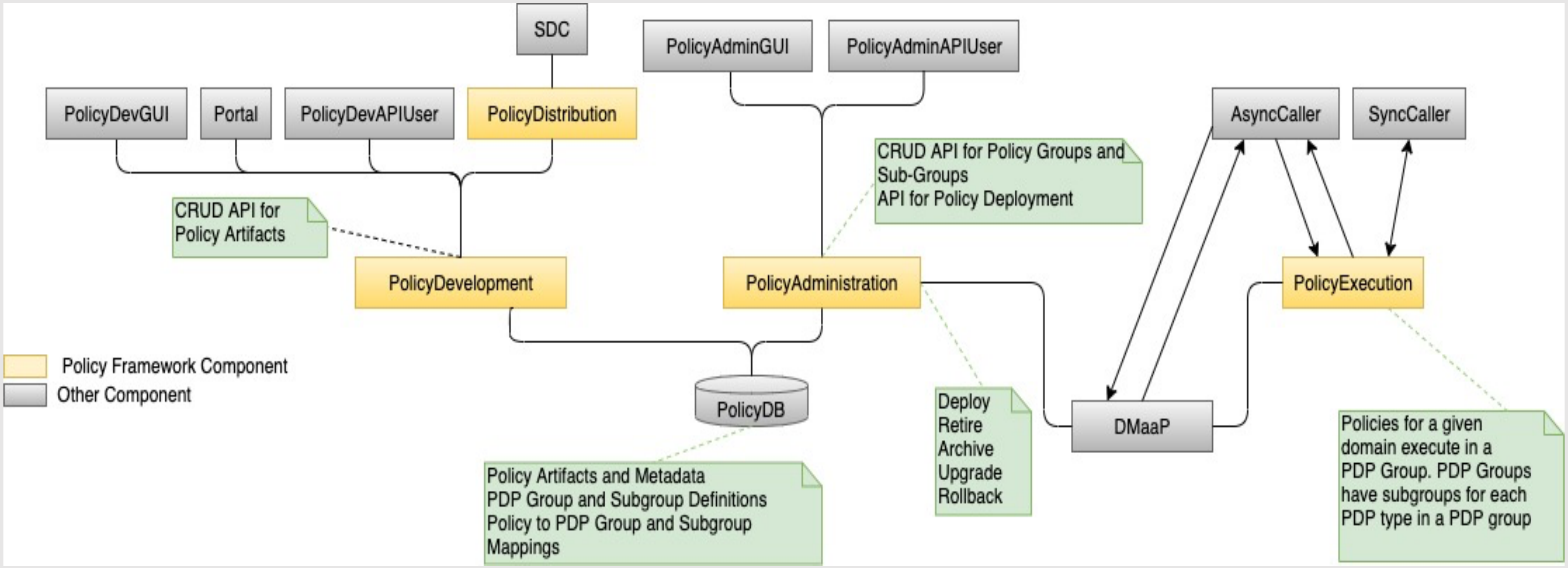
History continued

- Flexibility and extensibility
  - Pick and choose the PDP (Policy Decision Point) to use for your use case
    - Ability to build your own PDP if desired
  - Design your own Policy Types and create your own applications that translate/implement those Policy Types
  - Configure SDC distribution to automate lifecycle API and PAP API
- Frankfurt full integration with the following ONAP components was completed:
  - CLAMP: control loop policies
  - DCAE: control loop monitoring policies
  - OOF: optimization policies
  - SDNC: naming policies

Policy Execution at Highest Level



Abstract View of the Architecture


SDC
PolicyAdminGUI
PolicyAdminAPIUser
PolicyDevGUI
Portal
PolicyDevAPIUser
PolicyDistribution
AsyncCaller
SyncCaller
CRUD API for Policy Groups and Sub-Groups
API for Policy Deployment
CRUD API for Policy Artifacts
PolicyDevelopment
PolicyAdministration
PolicyExecution
Policy Framework Component
Other Component
PolicyDB
Deploy
Retire
Archive
Upgrade
Rollback
DMaaP
Policies for a given domain execute in a PDP Group. PDP Groups have subgroups for each PDP type in a PDP group
Policy Artifacts and Metadata
PDP Group and Subgroup Definitions
Policy to PDP Group and Subgroup Mappings

Policy API's – 3 sets of RESTful API's

- Lifecycle API
  - RESTful CRUD API – used by Policy Designers
  - Policy Designers perform lifecycle for Policy Types and their Policies
  - Implemented in the Policy API component: policy/api
  - This API should **NOT** be used by runtime components to retrieve their policies. Only the Decision API should be used by runtime components to return policies are to be enforced based on conditions.
- Administration API
  - RESTful CRUD for Grouping PDPs and deploying policies to PDPs – used by DevOps team
  - Implemented in the Policy PAP component: policy/pap
  - The Policy PAP component is also responsible for Dmaap notifications of Policy deployment changes, to enable ONAP components option to dynamically update the policy they are enforcing
- Decision API
  - Simple REST POST to query for decisions
  - Runtime Decisions for both ONAP components and Policy PDP's
  - Implemented in the Policy XACML PDP component: policy/xacml-pdp

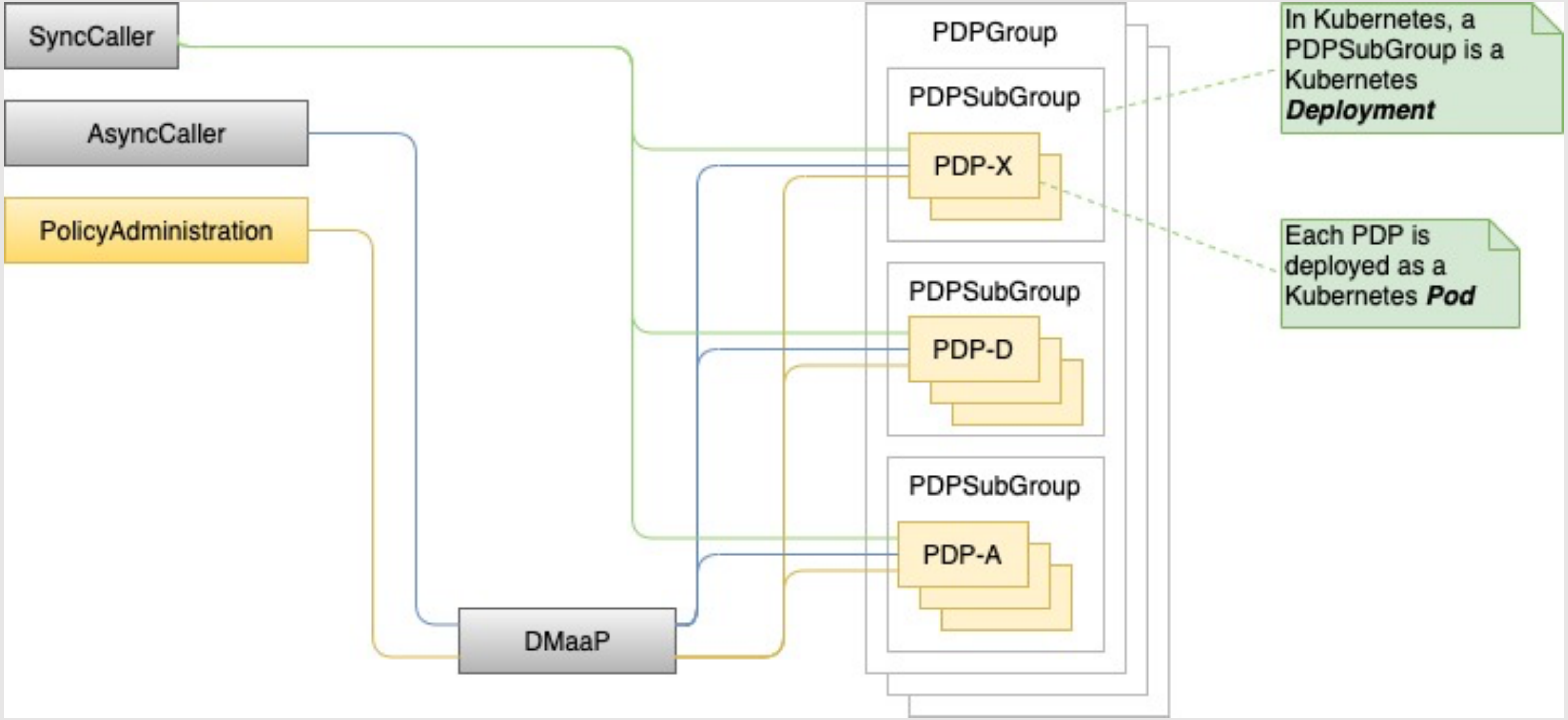
Policy Distribution

- Policy has a distribution component capable of receiving to SDC Service distributions
  - The distribution component is fully configurable and integrated with the latest lifecycle and pap API's
  - Users can configure the distribution component to automate the creation and deployment of policy types and their policies when a service is distributed
  - Implemented in the policy/distribution component
  - https://onap-doc.readthedocs.io/projects/onap-policy-parent/en/frankfurt/distribution/distribution.html

Policy PDP’s – 3 PDPs available

- Apex PDP
  - Adaptive chained state-machine driven solution
- Drools PDP
  - BPMN-based Drools Rules
  - Can utilize a nexus repo for storage/retrieval of Drools Rules and Java artifacts supporting those Rules
- XACML PDP
  - Implements the Decision API
  - Fine-grained attribute-based question/answer decision making

PDP Execution – from a logical view


SyncCaller
AsyncCaller
PolicyAdministration
DMaaP
PDPGroup
PDPSubGroup
PDP-X
PDPSubGroup
PDP-D
PDPSubGroup
PDP-A
In Kubernetes, a PDPSubGroup is a Kubernetes ***Deployment***
Each PDP is deployed as a Kubernetes ***Pod***

What Policy Types are available to use out-of-the box?

- These Policy Types are supported by the XACML PDP
  - Monitoring Policy Types supported by DCAE collectors/analytics
    - tca (threshold crossing), datafile-app-server
    - Used in Control Loops
    - Integrated with CLAMP/DCAE
  - Guard Policy Types for protection regarding Control Loop operations
    - Blacklist, frequency limiters, min/max, coordination
    - Used in Control Loops (optional)
    - Integrated with CLAMP
  - Optimization Policy Types for OOF Services
    - All inherit from service and resource specific policy types
    - Affinity, distance, hardware placement, optimization algorithm, VIM fit, vnf, pci, query, subscriber
  - Naming Policy Types for SDNC Naming Services

What Policy Types are available to use out-of-the box?

- Operational Policy Types for enforcement of Control Loop operations
  - Used in Control Loops
  - Both Drools and Apex support these policy types
  - Integrated into CLAMP

- NOTE: Users can configure which Policy Types are pre-loaded in the policy/api component's configuration file.
  - See example config:
  - https://github.com/onap/policy-api/blob/frankfurt/packages/policy-api-tarball/src/main/resources/etc/defaultConfig.json

Where can I see the Policy Types?

- Located in the policy/models repository
- https://github.com/onap/policy-models/tree/master/models-examples/src/main/resources/policytypes

Control Loop vs Non-Control Loop Policy Types

- These Policy Types support Control Loop implementation
  - onap.policies.Monitoring
  - onap.policies.controlloop.operational.Common
    - Apex and drools extensions
  - onap.policies.controlloop.guard.Common
    - Frequency limiter, blacklist, min/max, and coordination

Can I Design my own Policy Type?

- Yes!
- Some existing Policy Types can be extended and implemented by out-of-box ONAP framework
  - onap.policies.Monitoring: For new DCAE collectors/analytics
  - onap.policies.Optimization: For new extensions to OOF use cases
- For new Policy Types, you will need to build an application for the PDP you choose to use to support your Policy Type
  - Each PDP has its own design for building applications to support a policy type
  - Please see the in-depth video tutorials for each PDP

Which PDP should I use?

- That is up to the user to evaluate which PDP is preferable to use
- Each PDP has its own strengths and weaknesses
- Neither is better than the other, just a different way of accomplishing your solution
- Each PDP is used by various Use Cases in ONAP
  - We encourage you to contact Use Case owners to determine why and how they use the PDP for their solution

Can I build my own PDP?

- Yes!
- The interface between PDP’s that register with the PAP is available to implement for any developer
- Code is shared in policy/models repo
- There is a PDP simulator in the policy/models repo one can reference
- See any of the existing PDP’s as well as a PDP simulator code to get an understanding on how to create your own PDP

How do I build an application for my own Policy Type?

- XACML applications can be built to support Policy Types that require an ONAP component to query the Decision API
  - Simple question/answer Decisions to support an ONAP component to be policy-driven
  - "What policy(s) should my app enforce given these conditions?"
  - Allows fine-grained attribute-based Policy Decisions
  - There are standard policy translators available to use.
    - Matchable: gives the user the ability to designate which properties are matched in a decision
    - Combined: simple combination of all the ids or types of policies.
  - Creating your own translator is possible as the design of the XACML PDP is extendible
  - Best examples: monitoring, optimization, naming, guard Policy Types
  - https://onap-doc.readthedocs.io/projects/onap-policy-parent/en/frankfurt/xacml/xacml.html

How do I build an application for my own Policy Type?

- Drools applications can be built to support Policy Types that can work with the drools BPMN rules
  - Maintains state
  - Allows flexibility in writing rules and java artifacts to support those rules
  - Applications are called “controllers” which can be configured to serialize/deserialize objects to/from Dmaap
  - Best example: operational
  - https://onap-doc.readthedocs.io/projects/onap-policy-parent/en/frankfurt/drools/drools.html

How do I build an application for my own Policy Type?

- Apex applications can be built to drive a chain of adaptive states
  - Apex is adaptive in that you each state can be fed by the previous state
  - MEDA state model: Match Establish Decide Act
  - ECA state model: Event Condition Action
  - https://onap-doc.readthedocs.io/projects/onap-policy-parent/en/frankfurt/apex/apex.html

What does Policy Execution Mean?

- Policy Execution can mean different things
  - Enforcement: an application must enforce policy(s)
  - Decisions: a policy decision is made that returns either permit/deny or one or more policies that an application must enforce

What about the GUI?

- The legacy GUI will be deprecated in Guilin and should not be used any longer.
- A POC that developed a PDP Monitoring GUI was done in Frankfurt and is available for evaluation
  - Located in policy/gui repository
  - https://onap-doc.readthedocs.io/projects/onap-policy-parent/en/frankfurt/gui/Monitoring-GUI-Demo-Guide.html
- Long term roadmap is to re-build a new GUI to support Policy Lifecycle API and the Administration API

Where to get information

- Documentation – is the starting place to get an understanding on the platform
  - https://onap-doc.readthedocs.io/projects/onap-policy-parent/en/frankfurt/index.html#master-index
- Codebase – is the starting place to understand how the code is built
  - Be sure to checkout the **frankfurt** branch for code that was delivered in Frankfurt.

Codebase locations - common

- Codebase – is the starting place to understand how the code is built
  - policy/docker build the common base Policy docker images consumed by the Policy components
    - https://github.com/onap/policy-docker
  - policy/parent, policy/common and policy/models hold the base code shared by the other repos
    - They produce java artifacts only
    - https://github.com/onap/policy-parent
    - https://github.com/onap/policy-common
    - https://github.com/onap/policy-models

Codebase locations - api and pap

- policy/api holds the Policy Lifecycle API
  - Produces both java artifacts and docker images
  - https://github.com/onap/policy-api
- policy/pap holds the Policy Administration API
  - Produces both java artifacts and docker images
  - https://github.com/onap/policy-pap

Codebase locations – pdp and their applications

- policy/apex-pdp, policy/drools-pdp and policy/xacml-pdp hold PDP code
  - These repos produce both java artifacts and docker images
  - https://github.com/onap/policy-apex-pdp
  - https://github.com/onap/policy-drools-pdp
  - https://github.com/onap/policy-xacml-pdp
  - Both apex and xacml have application code in those repos
  - policy/drools-applications holds drools application code, it also produces a docker image
  - https://github.com/onap/policy-drools-applications

Codebase locations – distribution and gui

- policy/distribution has the SDC distribution code
  - This repo produces both java artifacts and a docker image
  - https://github.com/onap/policy-distribution
- policy/gui has the Monitoring GUI code (evaluation only)
  - This repo only produces java artifacts
  - https://github.com/onap/policy-gui