Active and Available Inventory (AAI)

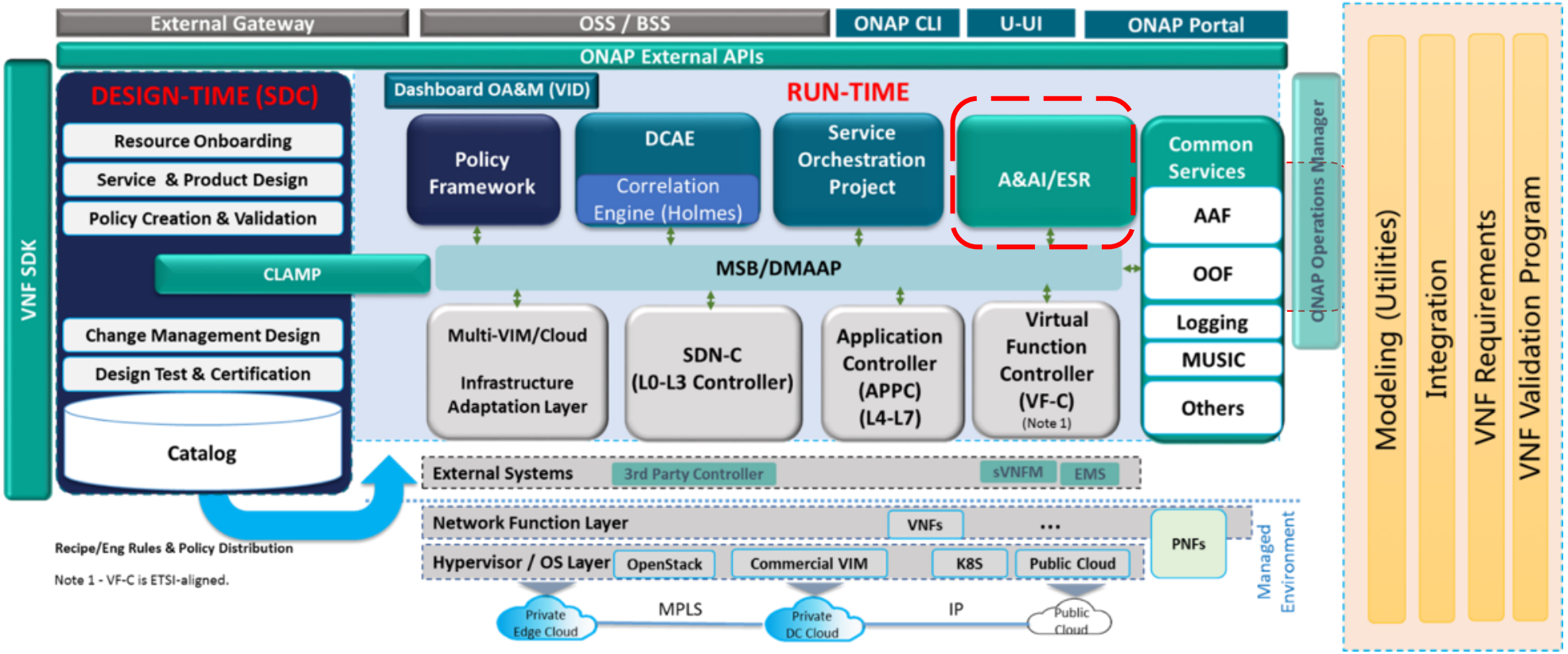
Agenda

- Introduction
- Architecture
- External Interaction & APIs
- Model Overview
- Core Component Insights
  - Schema mS: Create/Manage a new Schema
  - Resource mS: CRUD operations to operate on a given schema
  - Others: Babel, Model Loader, Data Storage, Traversal
- Installation & Demo
- CCVPN

AAI Overview

https://docs.onap.org/projects/onap-aai-aai-common/en/istanbul/platform/architecture.html

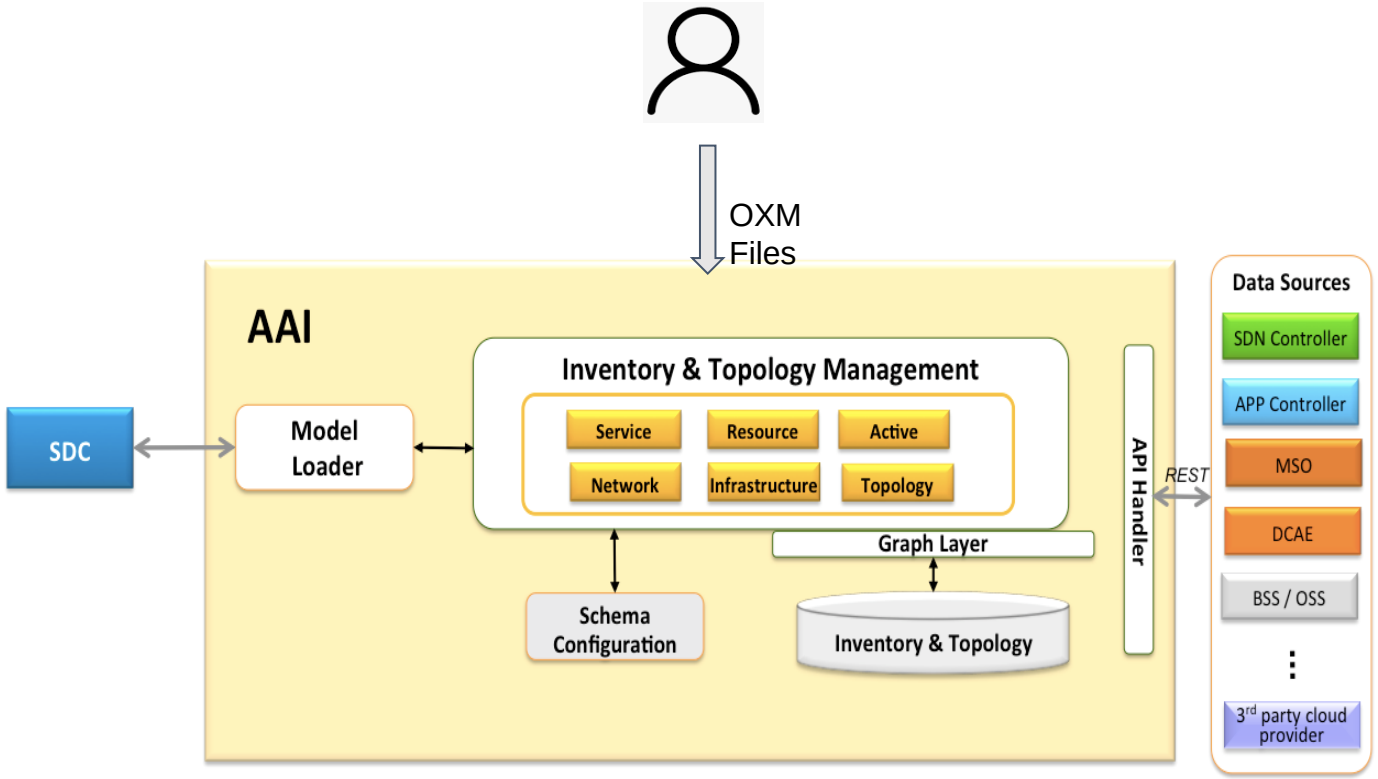
Active and Available Inventory (AAI) is centralized Registry of the active, available, and assigned assets in ONAP. It provide real-time management & views of the **resources, services, and their relationships via dynamic schema-driven APIs**. Data in AAI is continually updated in real-time.



AAI is based on **model/schema driven architecture approach** for inventory subsystem, which means AAI **API** & graph **schema** are driven by its **Data Model** without need of any new coding. AAI uses a Graph Database based on JanusGraph.

AAI Overview: Design to Runtime

https://docs.onap.org/projects/onap-aai-aai-common/en/istanbul/platform/architecture.html



Phase I: AAI Schema Design & Update

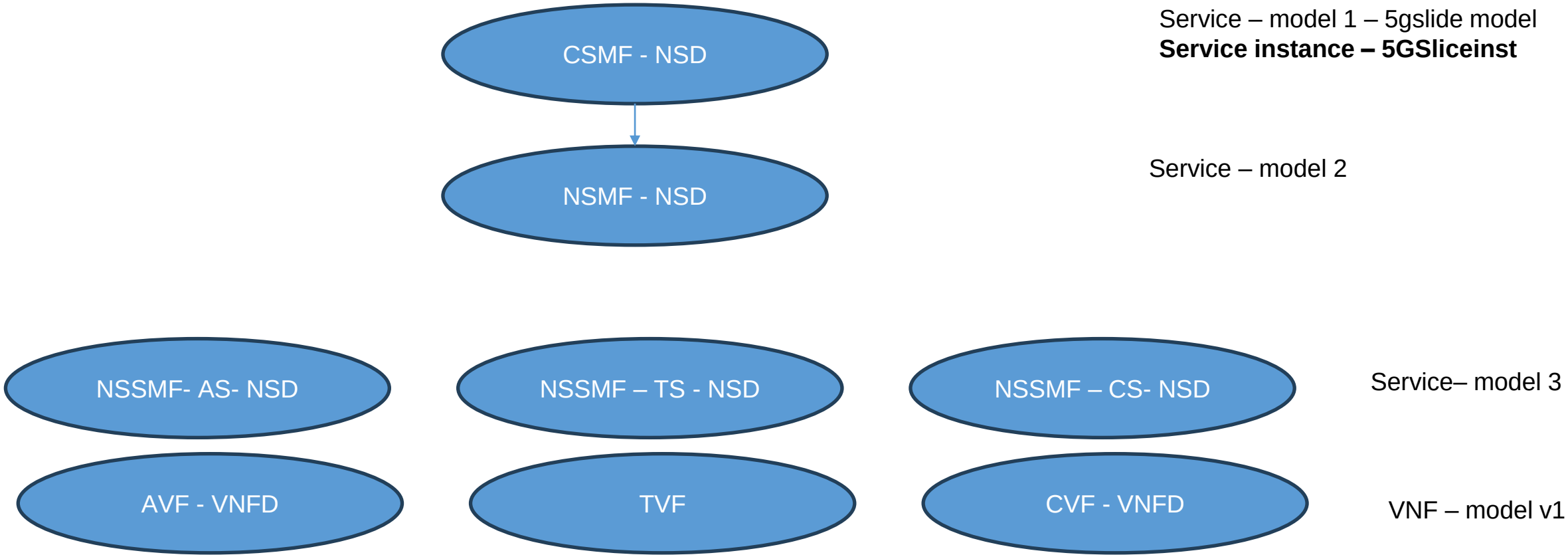
- ❏ AAI schemas are represented in OXM (Object XML mapping) file along with EdgeRule JSON file
- ❏ AAI is recompiled with OXM file to update schema and generate required APIs.
- ❏ Additionally XSD based schema is generated from OXM, this XSD file is used by AAI Client library (for usage by other ONAP components).

Phase II: ONAP Service Design

- ❏ The Model Loader polls the topic for new TOSCA CSAR distributions. It GETs the new CSAR.
- ❏ Model Loader facilitates the distribution and ingestion of new service and resource information from SDC to AAI

Phase III: ONAP Runtime

- ❏ Using AAI client/ direct REST APIs other ONAP components interact with AAI.
- ❏ AAI operates on JANUS Graph DB using Gremlin Query language



Service – model 1 – 5gslide model
Service instance – 5GSliceinst

Service – model 2

Service– model 3

VNF – model v1

AAI Architecture: Components

https://docs.onap.org/projects/onap-aai-aai-common/en/istanbul/platform/architecture.html

ESR: Applications for management of external systems.

aai/esr-gui	External system management ui.
aai/esr-server	ESR backend, mainly include the function of external system reachable check and data pretreatment.

Input abstraction: Applications that serve as entry points to A&AI.

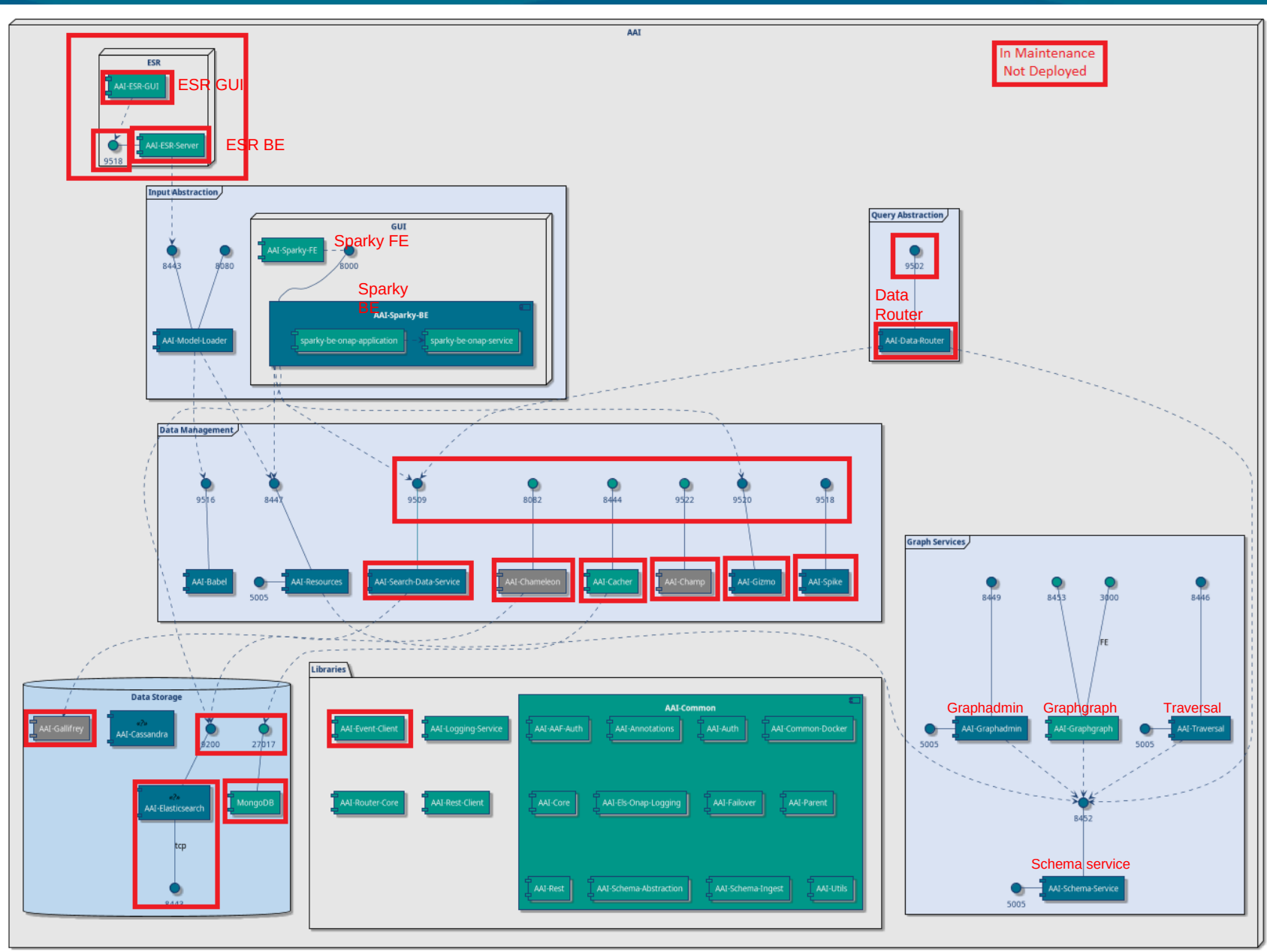
aai/model-loader	Obtains SDC artifacts and loads them into the A&AI Resources service for storage.
aai/sparky-be	AAI user interface back end.
aai/sparky-fe	AAI user interface front end.

Query abstraction: Query abstraction point for clients that routes AAI queries and event data.

aai/data-router	Makes decisions about workloads to be dispatched to search and tabular microservces. Includes logic to recognize and direct requests based on request archetypes

Graph service: Set of components, which store, provide or display schemas.

aai/graphadmin	Microservice with various functions for graph management.
aai/graphgraph	Microservice used to provide view of AAI model, schema and edge rules.
aai/schema-service	Centralized hub for all A&AI schema needs
aai/traversal	AAI Traversal Micro Service providing REST APIs for traversal/search of inventory resources.



AAI Architecture: Components

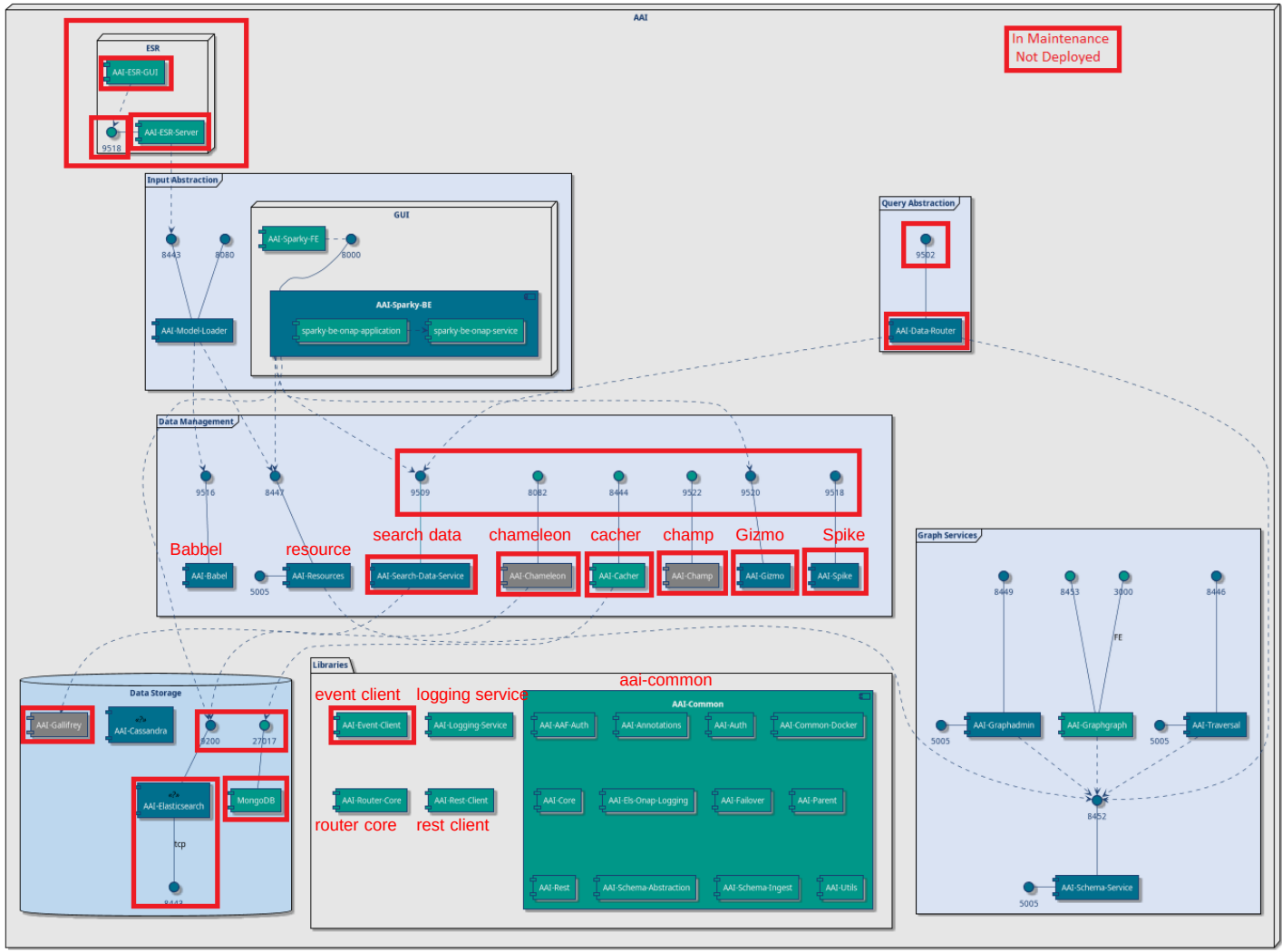
https://docs.onap.org/projects/onap-aai-aai-common/en/istanbul/platform/architecture.html

Data Management: Microservices that facilitate data management of AAI objects

aai/babel	AAI Microservice to generate AAI model XML from SDC TOSCA CSAR artifacts.
aai/cacher	Cacher is a generic service that can be used to snapshot json responses,.
aai/chameleon	(deprecated) Abstraction service for historical database.
aai/champ	(deprecated)Abstraction from underlying graph storage systems that A&AI would interface with.
aai/gizmo	(deprecated) CRUD Rest API endpoint for resources and relationships, delivering atomic interactions with the graph for improved scalability.
aai/resources	AAI Resources Micro Service providing CRUD REST APIs for inventory resources. This microservice provides the main path for updating and searching the graph - java-types defined in the OXM file for each version of the API define the REST endpoints - for example, the java-type “CloudRegion” in aai-common/aai-schema/src/main/resources/oxm/aai_oxm_v11.xml maps to /aai/v11/cloud-infrastructure/cloud-regions/cloud-region.
aai/search-data-service	Used by UI, for complex search by using indexing from Elasticsearch.
aai/spike	(deprecated) Microservice used to generate events describing changes to the graph data.

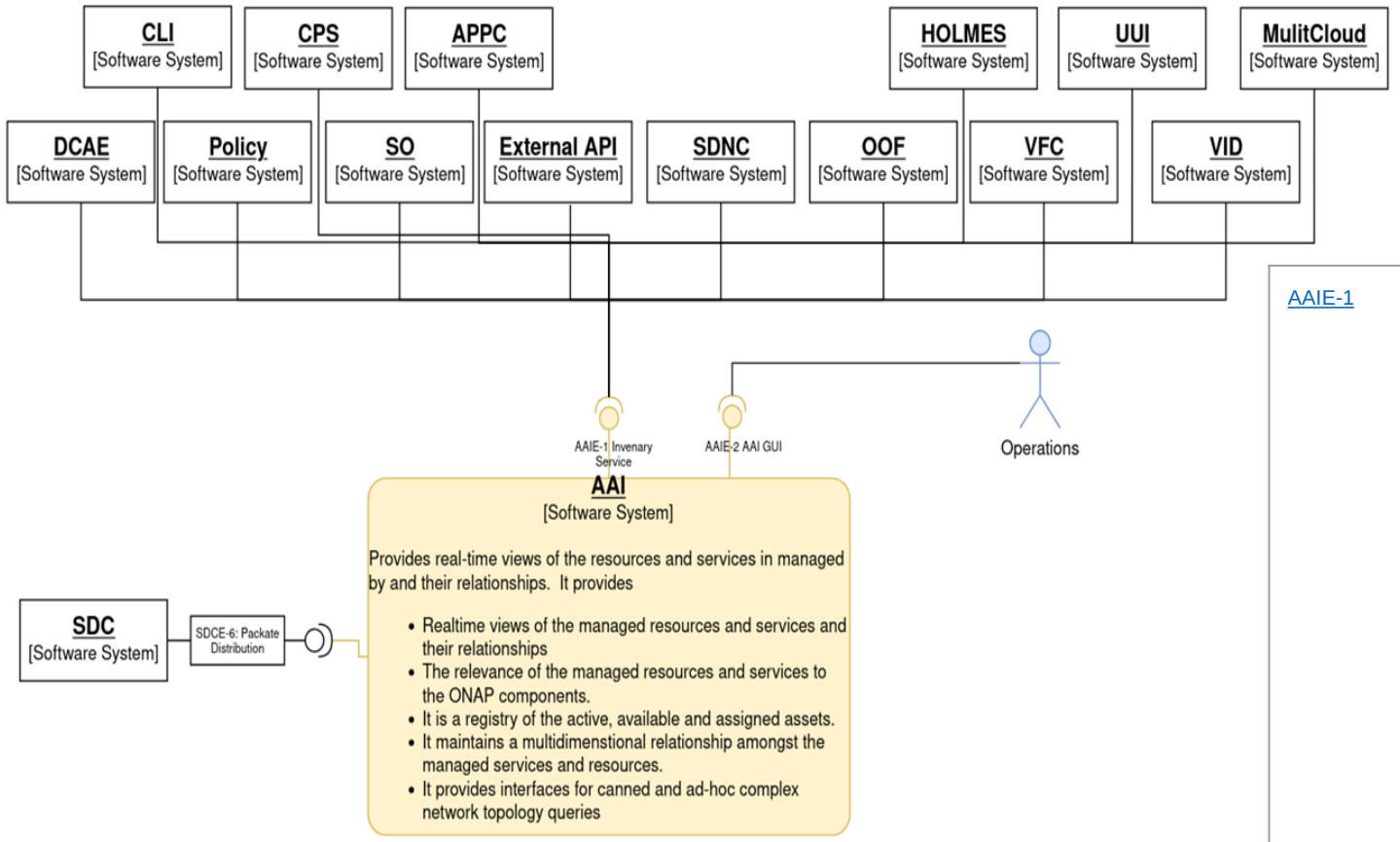
Library: contains general functionality, which may be imported and used in other modules

aai/aai-common	This holds the model, annotations and common modules used across the Resources and Traversal micro services.
aai/event-client	Event bus client library.
aai/logging-service	AAI common logging library.
aai/rest-client	Library for making REST calls.
aai/router-core	Library containing the core camel components for the data router.



AAI External Interaction & API

https://wiki.onap.org/display/DW/ARC+AAI+Component+Description+-+Jakarta-R10
https://docs.onap.org/en/elalto/submodules/aai/aai-common.git/docs/AAI%20REST%20API%20Documentation/AAIRESTAPI.html

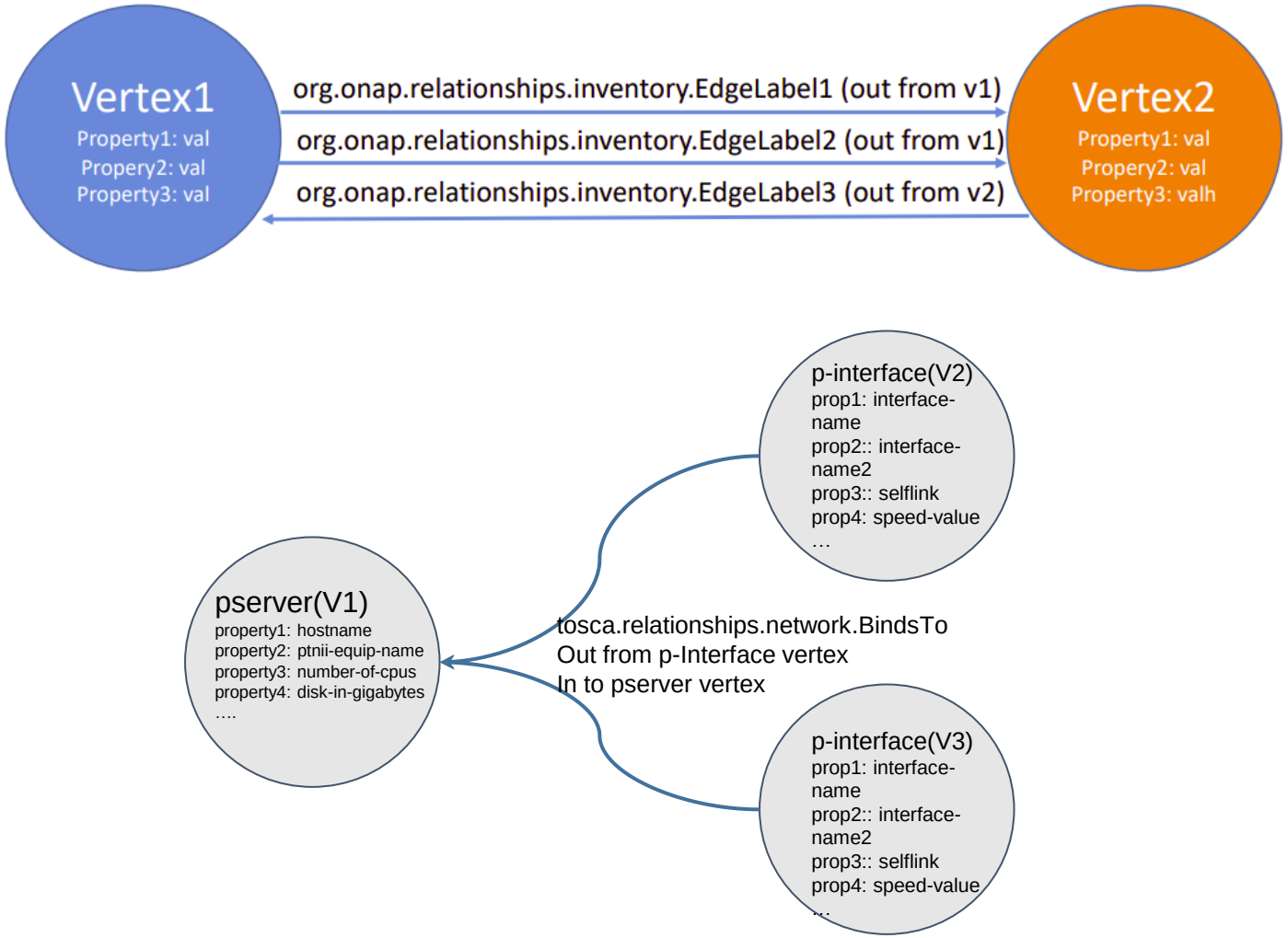


AAIE-1	Inventory Service Interface. CRUD & Queries	An interface to create, update, retrieve, query, delete the service, resources and relationship inventory information for: • Cloud infrastructure (including cloud infrastructure regions and availability zones, cloud infrastructure resources such as servers, storage, VLANs, cloud network technologies,...) • Paths and connectors • Business Customers and business customer relationship to managed assets • Service Instances (including the relationship to used resources) • Resource instances (including allotted resources) (virtual and physical, including VNFs, PNFs, Switches, VNF modules) • External systems (systems that ONAP connects to) • Images • Networks • Infrastructure tenants • Relationships • Interfaces • IP addresses • Configurations • Licenses • Site resources AAI provides multiple interfaces to use canned queries, dynamic queries (DSL), and CRUD operations (Resources)
AAIE-2	AAI Graphical User Interfaces	Provides the capability to view the inventory

AAI Graph

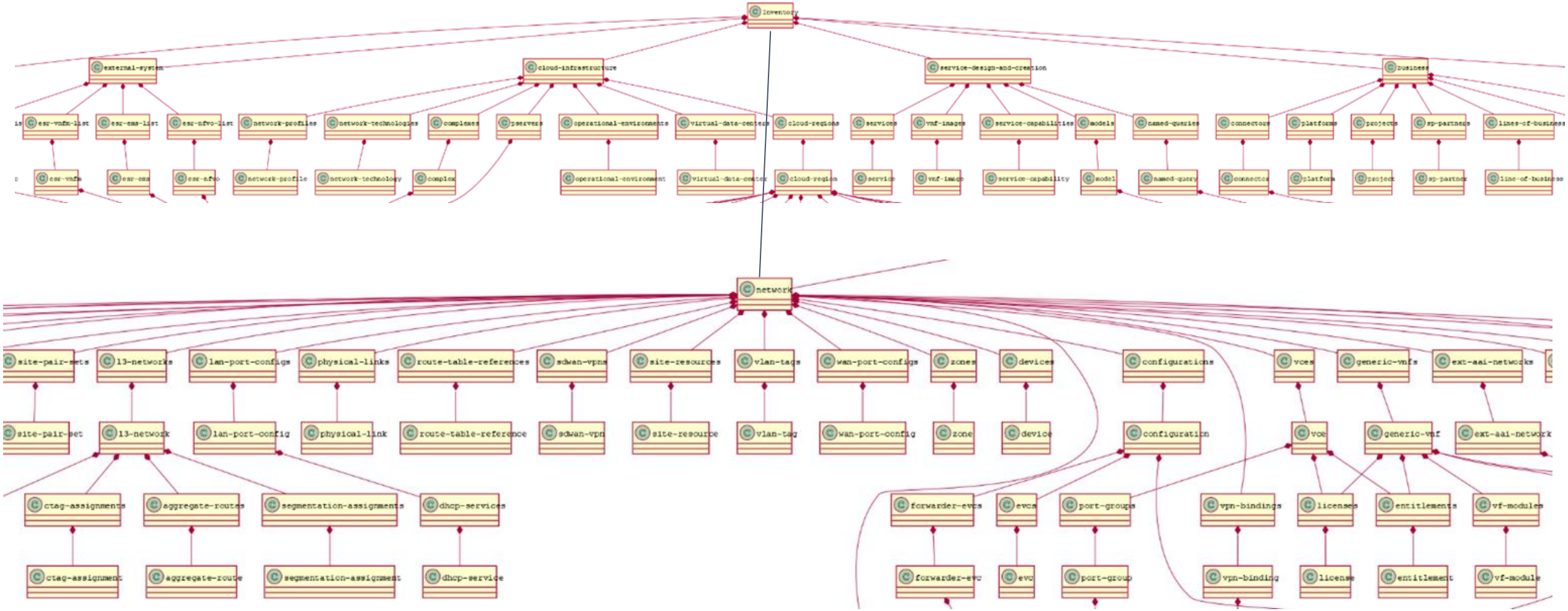
https://docs.janusgraph.org/schema/

- A&AI uses Janusgraph for persistence which is a property graph model, where a graph is a set of **vertices with edges** between them.
- A **vertex** is the fundamental unit of the graph and represents an object. Vertex can have properties to describe the object.
- An **edge** is a connection between two vertices that expresses a relationship between them. An edge can have a multiplicity, direction, and properties.
- In Onap, for each instance of a service or resource, related informations are stored in graph DB as a vertex alongwith its properties.
- Relationship between nodes is created using Edge.
- For example, information and relationship between pserver and p-interfaces will be maintained as 3 vertices with 2 edges



AAI Data Model

https://wiki.onap.org/display/DW/AAI+OXM+Schema+UML
https://wiki.onap.org/download/attachments/76875673/CCVPN%20network%20NNI%20AAI%20Models.pdf?version=1&modificationDate=1579781948000&api=v2&download=true



5 top level elements
Usually hierarchy goes like this: Tenant/Customer → Service Subscription → Service Instance → Resources (Refer CCVPN example)

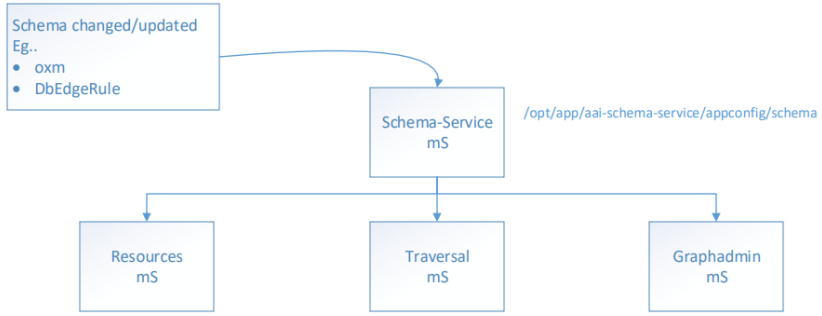
Core Components Insights

Schema service, Resources Service

Traversal, Model Loader, Babel

Core Components: Schema

What: Centralized hub for all A&AI schema needs. It **hold and load** all the schema (OXM and edge rules) at start-up and provide REST endpoints via GETs such as /aai/schema-service/{api-version} for consumer mS as shown below to **retrieve** schema



It generates XSD and POJO for AAI Clients.

Note: It usages Moven plugin jaxb2-maven-plugin for generate POJOs and exec-maven-plugin which call GenrateXsd class to generate XSD.

Note: Schema service are not used by end user directly, its used by other AAI mS or by AAI client.

Schema is defined using OXM and EdgeRule file:

1. **aai_oxm_v$.xml: Define resource/node entities including**
   - Resource Container
   - Resource element properties
   - Element metadata like container, namespace, uniqueprops etc.

Sample .oxm file

```
<java-type name="Pservers">
    <xml-properties>
        <xml-property name="description" value="Collection of compute hosts."/>
    </xml-properties>
    <xml-root-element name="pservers"/>
    <java-attributes>
        <xml-element container-type="java.util.ArrayList" java-attribute="pserver" name="pserver" type="inventory.aai.onap.org.v23.Pserver"/>
    </java-attributes>
    <xml-properties>
        <xml-property name="maximumDepth" value="0"/>
    </xml-properties>
</java-type>

<java-type name="Pserver">
    <xml-root-element name="pserver"/>
    <java-attributes>
        <xml-element java-attribute="hostname" name="hostname" required="true" type="java.lang.String" xml-key="true">
            <xml-properties>
                <xml-property name="description" value="Value from executing hostname on the compute node."/>
            </xml-properties>
        </xml-element>
        <xml-element java-attribute="ptniiEquipName" name="ptnii-equip-name" type="java.lang.String">
            <xml-properties>
                <xml-property name="description" value="PTNII name"/>
            </xml-properties>
        </xml-element>
        <xml-element java-attribute="numberOfCpus" name="number-of-cpus" type="java.lang.Integer">
            <xml-properties>
                <xml-property name="description" value="Number of cpus"/>
            </xml-properties>
        </xml-element>
        <xml-element java-attribute="diskInGigabytes" name="disk-in-gigabytes" type="java.lang.Integer">
            <xml-properties>
                <xml-property name="description" value="Disk size, in GBs"/>
            </xml-properties>
        </xml-element>
        <xml-element java-attribute="operationalStatus" name="operational-status" type="java.lang.String">
            <xml-properties>
                <xml-property name="description" value="Indicator for whether the resource is considered operational.  Valid values are in-service-path and out-of-service-path."/>
                <xml-property name="suggestibleOnSearch" value="true"/>
            </xml-properties>
        </xml-element>
        <xml-element java-attribute="relationshipList" name="relationship-list" type="inventory.aai.onap.org.v23.RelationshipList"/>
        <xml-element java-attribute="pInterfaces" name="p-interfaces" type="inventory.aai.onap.org.v23.PInterfaces"/>
        <xml-element java-attribute="lInterfaces" name="l-interfaces" type="inventory.aai.onap.org.v23.LInterfaces"/>
        <xml-element java-attribute="lagInterfaces" name="lag-interfaces" type="inventory.aai.onap.org.v23.LagInterfaces"/>
        <xml-element java-attribute="activities" name="activities" type="inventory.aai.onap.org.v23.Activities"/>
        <xml-element container-type="java.util.ArrayList" java-attribute="hpaCapacity" name="hpa-capacity" type="inventory.aai.onap.org.v23.HpaCapacity"/>
    </java-attributes>
    <xml-properties>
        <xml-property name="description" value="Compute host whose hostname must be unique and must exactly match what is sent as a relationship to a vserver."/>
        <xml-property name="nameProps" value="pserver-name2,fqdn"/>
        <xml-property name="indexedProps" value="hostname,pserver-id,pserver-name2,inv-status,fqdn,prov-status,ptnii-equip-name,data-owner,data-source,data-source-version"/>
        <xml-property name="searchable" value="hostname,pserver-name2,pserver-id,ipv4-oam-address,operational-status"/>
        <xml-property name="uniqueProps" value="hostname"/>
        <xml-property name="container" value="pservers"/>
        <xml-property name="namespace" value="cloud-infrastructure"/>
        <xml-property name="uriTemplate" value="/cloud-infrastructure/pservers/pserver/{hostname}"/>
        <xml-property name="requiredProps" value="hostname,in-maint"/>
    </xml-properties>
</java-type>
```

Core Components: Schema cont..

https://docs.onap.org/projects/onap-aai-aai-common/en/istanbul/platform/Getting%20Started/Edge_Rules.html#how-to-interpret-an-edge-rule

2. **DbEdgeRules_v$.json** – defines relationships between nodes

Multiplicity: One2One, One2Many, Many2One, Many2Many

One2One: On a node, there may be only one IN edge, and only one OUT edges.

One2Many: means on a node, there may be only one IN edge, and many OUT edges.

Many2One: means on a node, there may be many IN edge, and only one OUT edges.

Many2Many: On a node, there may be many IN edge, and many OUT edges.

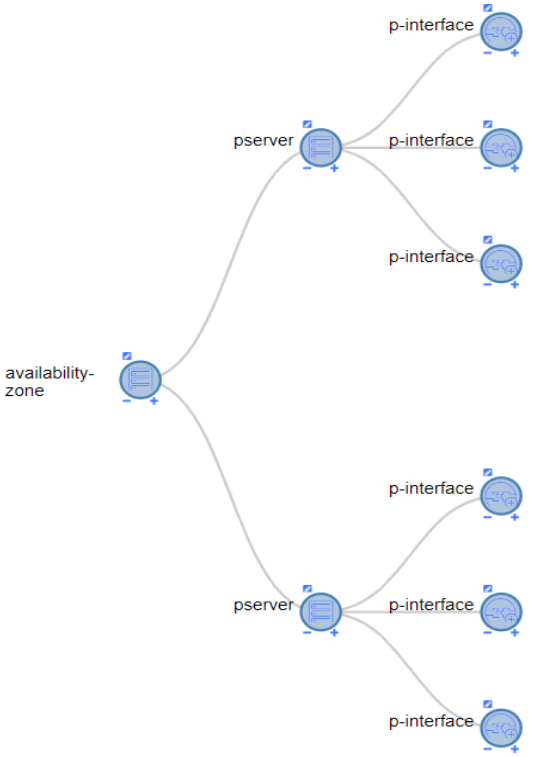
In this example, It has multiplicity of MANY2ONE. pserver gets the IN edge, so it may have many edges from p-interfaces. The p-interface gets the OUT edge, so it may have only one edge to pserver.

Direction: OUT, IN
OUT: Point the edge out from the “from node” and IN to “to Node”.
IN: Point the edge in to “from node” and out from “to Node”.
In diagram, Edge is point **out** from p-interface and **in** to pserver

Containsother-v : is like isparent.
Delete-v : to denote if there is anything to be delete along with

Sample EdgeRule File

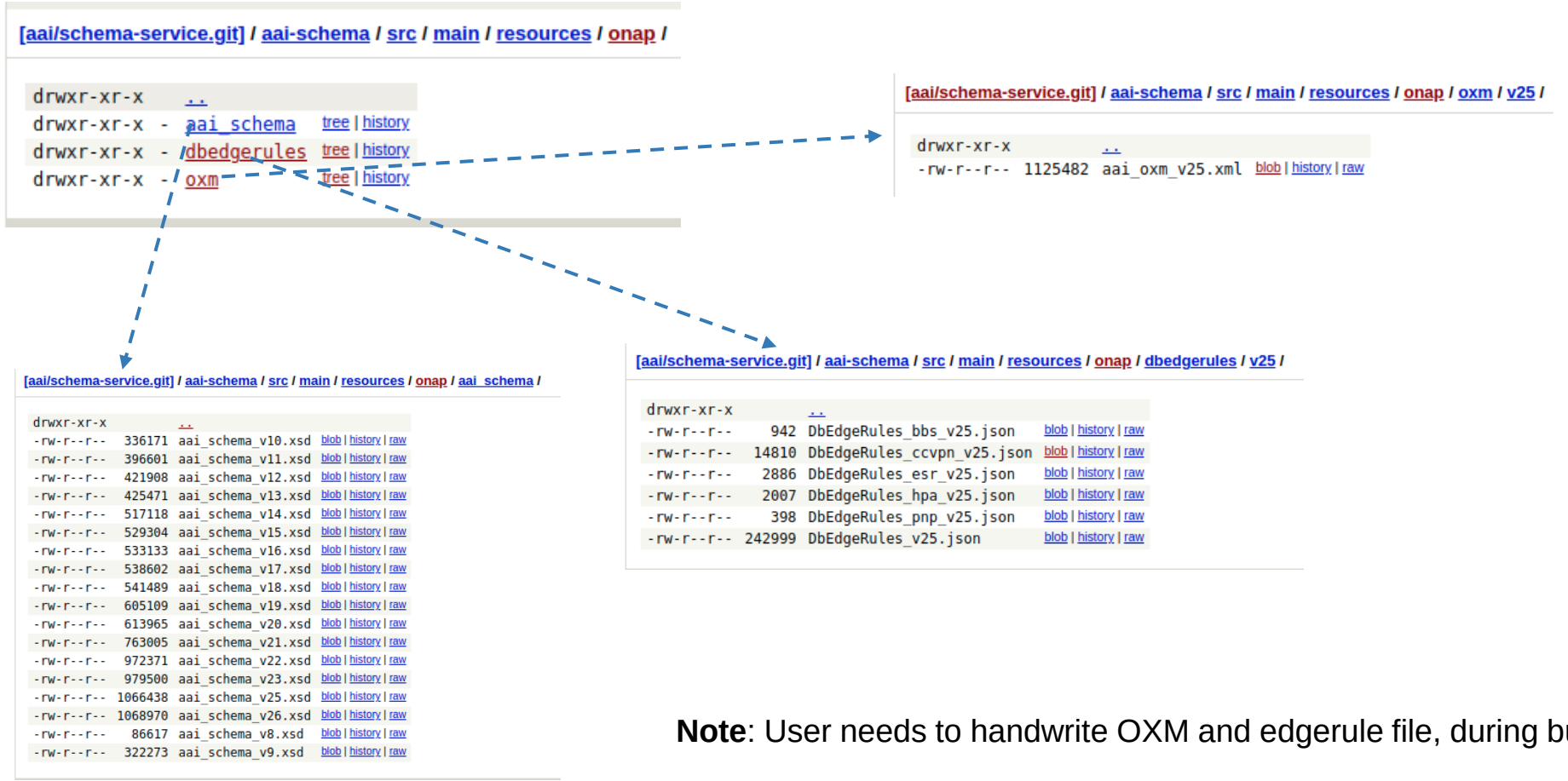
```
{
        "from": "p-interface",
        "to": "pserver",
        "label": "tosca.relationships.network.BindsTo
        "direction": "OUT",
        "multiplicity": "MANY2ONE",
        "contains-other-v": "!${direction}",
        "delete-other-v": "!${direction}",
        "prevent-delete": "NONE",
        "default": "true",
        "description": ""
},
{
        "from": "pserver",
        "to": "availability-zone",
        "label": "org.onap.relationships.inventory.Me
        "direction": "OUT",
        "multiplicity": "MANY2ONE",
        "contains-other-v": "NONE",
        "delete-other-v": "NONE",
        "prevent-delete": "!${direction}",
        "default": "true",
        "description": ""
},
```



Note: “from” and “to” do not imply direction. Think of them as more like “NodeA” and “NodeB”.)

Core Components: Schema - GIT Location

https://gerrit.onap.org/r/gitweb?p=aai/schema-service.git;a=tree;f=aai-schema/src/main/resources/onap;h=13228b31b9b9395d6382a8f19bc10f88bad0b309;hb=refs/heads/master



[aai/schema-service.git] / aai-schema / src / main / resources / onap /

drwxr-xr-x		..	
drwxr-xr-x	-	aai_schema	tree | history
drwxr-xr-x	-	dbedgerules	tree | history
drwxr-xr-x	-	oxm	tree | history

[aai/schema-service.git] / aai-schema / src / main / resources / onap / oxm / v25 /

drwxr-xr-x		..	
-rw-r--r--	1125482	aai_oxm_v25.xml	blob | history | raw

[aai/schema-service.git] / aai-schema / src / main / resources / onap / aai_schema /

drwxr-xr-x		..	
-rw-r--r--	336171	aai_schema_v10.xsd	blob | history | raw
-rw-r--r--	396601	aai_schema_v11.xsd	blob | history | raw
-rw-r--r--	421908	aai_schema_v12.xsd	blob | history | raw
-rw-r--r--	425471	aai_schema_v13.xsd	blob | history | raw
-rw-r--r--	517118	aai_schema_v14.xsd	blob | history | raw
-rw-r--r--	529304	aai_schema_v15.xsd	blob | history | raw
-rw-r--r--	533133	aai_schema_v16.xsd	blob | history | raw
-rw-r--r--	538602	aai_schema_v17.xsd	blob | history | raw
-rw-r--r--	541489	aai_schema_v18.xsd	blob | history | raw
-rw-r--r--	605109	aai_schema_v19.xsd	blob | history | raw
-rw-r--r--	613965	aai_schema_v20.xsd	blob | history | raw
-rw-r--r--	763005	aai_schema_v21.xsd	blob | history | raw
-rw-r--r--	972371	aai_schema_v22.xsd	blob | history | raw
-rw-r--r--	979500	aai_schema_v23.xsd	blob | history | raw
-rw-r--r--	1066438	aai_schema_v25.xsd	blob | history | raw
-rw-r--r--	1068970	aai_schema_v26.xsd	blob | history | raw
-rw-r--r--	86617	aai_schema_v8.xsd	blob | history | raw
-rw-r--r--	322273	aai_schema_v9.xsd	blob | history | raw

[aai/schema-service.git] / aai-schema / src / main / resources / onap / dbedgerules / v25 /

drwxr-xr-x		..	
-rw-r--r--	942	DbEdgeRules_bbs_v25.json	blob | history | raw
-rw-r--r--	14810	DbEdgeRules_ccvpn_v25.json	blob | history | raw
-rw-r--r--	2886	DbEdgeRules_esr_v25.json	blob | history | raw
-rw-r--r--	2007	DbEdgeRules_hpa_v25.json	blob | history | raw
-rw-r--r--	398	DbEdgeRules_pnp_v25.json	blob | history | raw
-rw-r--r--	242999	DbEdgeRules_v25.json	blob | history | raw

Note: User needs to handwrite OXM and edgerule file, during build XSD is autogenerated.

Core Components: Resource mS

AAI Resources Micro Service providing CRUD REST APIs for inventory resources.

Schema-driven APIs: Create Rest Endpoints for operate on resources, from OXM file which is obtained from shema service.

Example:

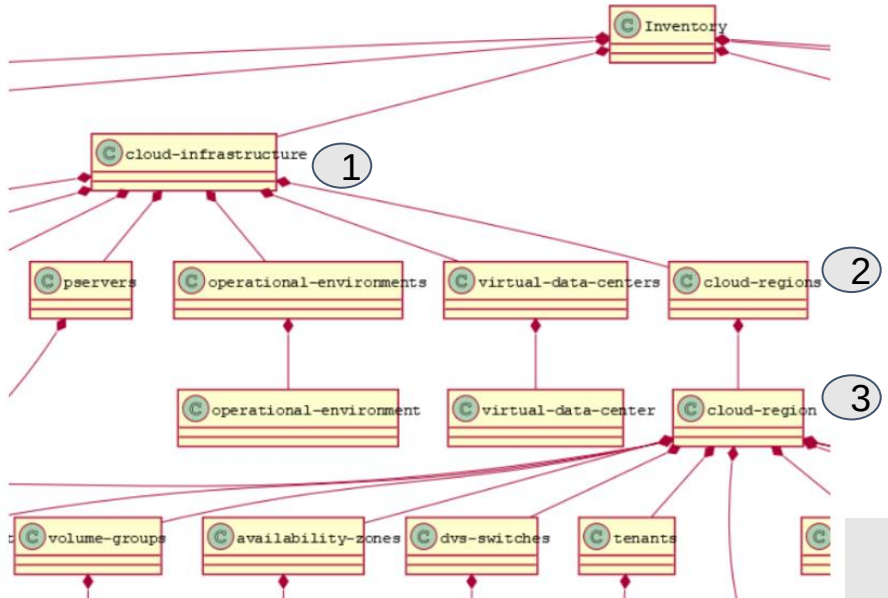
java-types defined in the OXM file define the REST endpoints - for example, the java-type “CloudRegion” in aai-common/aai-schema/src/main/resources/oxm/aai_oxm_v26.xml maps to **/aai/v26/cloud-infrastructure/cloud-regions/cloud-region**.

1 2 3

```
1
<java-type name="CloudInfrastructure">
    <xml-properties>
        <xml-property name="description" value="Namespace for cloud infrastructure."/>
    </xml-properties>
    <xml-root-element name="cloud-infrastructure"/>
    <java-attributes>
        <xml-element java-attribute="complexes" name="complexes" type="inventory.aai.onap.org.v26.Complexes"/>
        <xml-element java-attribute="cloudRegions" name="cloud-regions" type="inventory.aai.onap.org.v26.CloudRegions"/>  2
        <xml-element java-attribute="networkProfiles" name="network-profiles" type="inventory.aai.onap.org.v26.NetworkProfiles"/>
        <xml-element java-attribute="pservers" name="pservers" type="inventory.aai.onap.org.v26.Pservers"/>
        <xml-element java-attribute="endpoints" name="endpoints" type="inventory.aai.onap.org.v26.Endpoints"/>
        <xml-element java-attribute="virtualDataCenters" name="virtual-data-centers" type="inventory.aai.onap.org.v26.VirtualDataCenters"/>
        <xml-element java-attribute="operationalEnvironments" name="operational-environments" type="inventory.aai.onap.org.v26.OperationalEnvironments"/>
        <xml-element java-attribute="geoRegions" name="geo-regions" type="inventory.aai.onap.org.v26.GeoRegions"/>
        <xml-element java-attribute="networkTechnologies" name="network-technologies" type="inventory.aai.onap.org.v26.NetworkTechnologies"/>
    </java-attributes>
</java-type>

<java-type name="CloudRegions">  2
    <xml-root-element name="cloud-regions"/>
    <java-attributes>
        <xml-element container-type="java.util.ArrayList" java-attribute="cloudRegion" name="cloud-region" type="inventory.aai.onap.org.v26.CloudRegion"/>  3
    </java-attributes>
    <xml-properties>
        <xml-property name="maximumDepth" value="0"/>
    </xml-properties>
</java-type>

<java-type name="CloudRegion">  3
    <xml-root-element name="cloud-region"/>
    <java-attributes>
        <xml-element java-attribute="cloudOwner" name="cloud-owner" required="true" type="java.lang.String" xml-key="true">
            <xml-properties>
                <xml-property name="description" value="Identifies the vendor and cloud name. First part of composite key should be formatted as vendor-cloudname"/>
            </xml-properties>
        </xml-element>
```



Core Components: Resource mS - Create Resource

- Create Physical Server (no relationship)

  URI: PUT https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers/pserver/pserver-test

  Body:

```
{
  "hostname": "pserver-test",
  "in-maint": true
}
```

- Create Physical Server (w/relationship to complex)

  URI: PUT https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers/pserver/pserver-test

  Body:

```
{
  "hostname": "pserver-test",
  "in-maint": true,
  "relationship-list": {
    "relationship": [
      {
        "related-to": "complex",
        "relationship-label": "org.onap.relationships.inventory.LocatedIn",
        "related-link": "/aai/v23/cloud-infrastructure/complexes/complex/test-location-id",
        "relationship-data": [
          {
            "relationship-key": "complex.physical-location-id",
            "relationship-value": "test-location-id"
          }
        ]
      }
    ]
  }
}
```

Core Components: Resource mS (GET/Delete Op.)

https://docs.onap.org/en/elalto/submodules/aai/aai-common.git/docs/AAI%20REST%20API%20Documentation/AAIRESTAPI.html#aai-resources-

- Simple GET (specific physical server)

  URI: GET https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers/pserver/pserver-test?format=simple

- Simple GET (all physical servers)

  URI: GET https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers?format=simple

- Simple GET (physical servers having in-maint=false)

  URI: GET https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers?**in-maint=false**&format=simple

- Simple GET (physical server and directly connected nodes)

  URI: GET https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers /pserver/pserver-test?**format=resource&depth=1**

- Simple GET (physical server and all related subtrees)

  URI: GET https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers /pserver/pserver-test?**format=resource&depth=all**

- Simple GET (all physical servers w/o relationships)

  URI: GET https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers?format=simple**&node-only=true**

- Simple DELETE (specific physical server)

  URI: DELETE https://{{hostname}}:{{port}}/aai/{{version}}/cloud-infrastructure/pservers/pserver/pserver-test?**resource-version=12345**

Resource APIs support multiple formats of output results, for different needs.

- **Simple**: simplified format with node-type, graph vertex id, pathed url, object properties, and directly related objects in the graph are all returned.
- **depth**: A **depth** query parameter to get result of that label of connected nodes.
- **node-only**: query for nodes information without relationship.

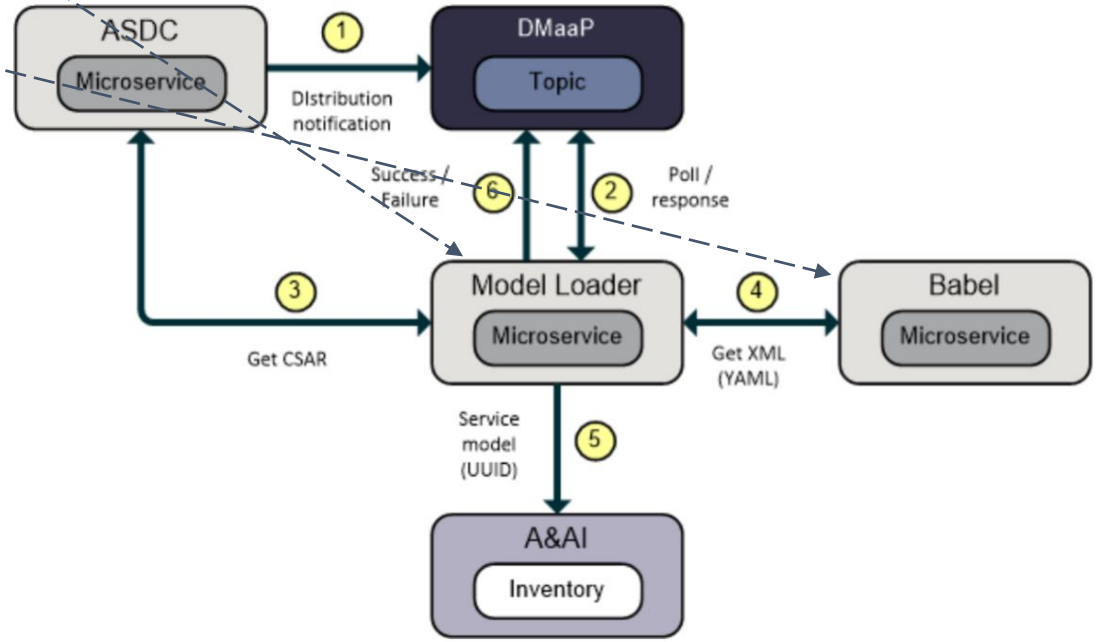
Core Component - Model Loader & Babel

Model Loader mS:

- ❏ mS that facilitates the **distribution and ingestion of new service and resource model from SDC to AAI**
- ❏ Uses Babel and Resources to load models into the A&AI graph database

Babel mS:

- ❏ mS that parses TOSCA YAML files extracted from a CSAR payload
- ❏ Provides a secured endpoint accepting HTTP POST requests
- ❏ Base64-encoded SDC CSAR data is POSTed to the Babel microservice in a JSON payload
- ❏ The JSON body is validated, and the encoded CSAR is extracted
- ❏ The CSAR content is transformed to XML files containing AAI Model Data that is returned to the requestor



1. A notification is posted on the distribution topic
2. The Model Loader polls the topic for new TOSCA CSAR distributions
3. The Model Loader GETs the new CSAR using the ASDC distribution client
4. The Model Loader POSTs the CSAR to the Babel service for transformation. The Babel service returns XML back to the Model Loader
5. The Model Loader GETs the model from A&AI Resources using its UUID. If the model already exists then model ingestion is skipped. **Otherwise the model is ingested using a REST PUT to the A&AI Resoures mS**
6. The Model Loader returns an event indicating the success/failure status of the model deployment

Core Component - Data Storage

https://janusgraph.org/
https://tinkerpop.apache.org/docs/3.6.0/reference/

Note: For Inventory Data storage, AAI usages open source Janus Graph DB.

JanusGraph: JanusGraph is a scalable graph database, optimized for storing and querying graphs, containing hundreds of **billions of vertices and edges** distributed across a **multi-machine cluster**. JanusGraph is built **upon several open source** like plugable storage backend, Index backends, Apache TinkerPop etc.

- Storage backends(Cassandra is used by AAI):
  - Apache Cassandra, Apache HBase, Google Cloud Bigtable, Oracle BerkeleyDB, Scylla
- Index Backends for Fast & Efficient traversal(Elasticsearch is used by AAI):
  - Elasticsearch, Apache Solr, Apache Lucene
- Apache Tinkerpop graph stack as API server and gremlin query language

Cassandra

Backend DB. scalable, distributed, NoSQL database

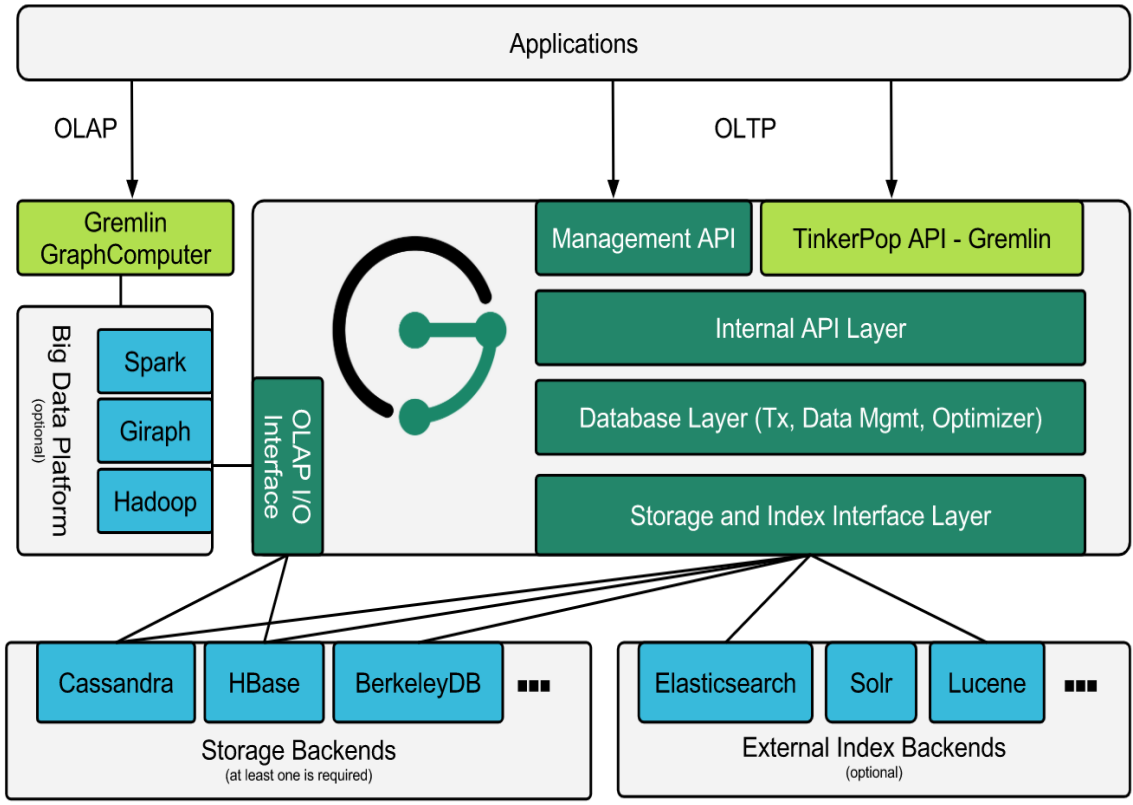
ElasticSearch for fast Indexing:

speed up and enable more complex queries

Apache TinkerPop

Apache TinkerPop is an open source Graph Computing Framework used by multiple graph DBs. It interacts with the Gremlin language commands to create, traverse, and manipulate graphs.

Gremlin
There are two parts to Gremlin: **the language and the server.**
Gremlin, the language, is a graph traversal language that are used to interact and query the graphs. Gremlin, the server, is a specification for a server that processes local or remote Gremlin language queries. An implementation of it is part of **Apache Tinkerpop**.



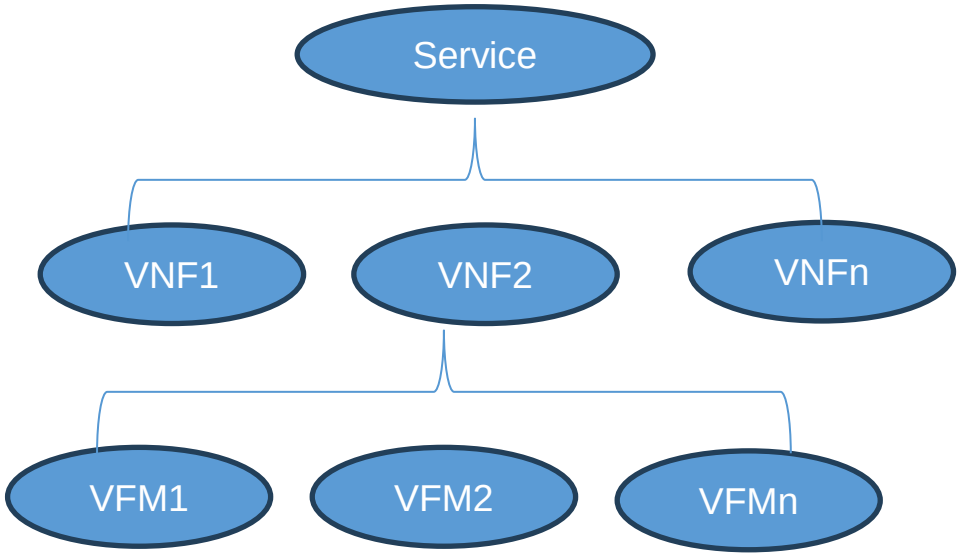
Janus Graph

Core Component - Traversal

https://docs.onap.org/en/elalto/submodules/aai/aai-common.git/docs/AAI%20REST%20API%20Documentation/customQueries.html
https://docs.onap.org/projects/onap-aai-aai-

AAI Traversal Micro Service providing REST APIs for traversal/search of inventory resources. It provides capabilities to do more advanced search for resources based on input queries than just GET API for specific resource and related nodes.

Traversal Example: A custom query that starts at a generic-vnf, follows the edge to the service-instance, then gets all connected vnfs and their vf-modules. It also return Tenet and Vservers



2 types of query APIs:

- Custom Query - It allows clients to develop and deploy stored queries to make processing complex services. It requires development for updates if any to existing query
- DSL - DSL is new feature delivered in Frankfurt that will allow users to specify ad-hoc queries that will make AAI more flexible and robust than previous versions. It required no development impact for new query.

Core Component - Traversal APIs - Custom & DSL query

https://docs.onap.org/projects/onap-aai-aai-common/en/istanbul/AAI%20REST%20API%20Documentation/customQueries.html
https://wiki.onap.org/display/DW/genericVnfs-fromPserver
https://wiki.onap.org/display/DW/AAIService+Custom+Query+in+SDN-C

Custom query example: To execute a custom query, a client will perform a PUT on the query API and include a payload indicating the starting node and the query to be run.

query genericVnfs-fromPserver allows a client to provide A&AI with a pserver hostname and retrieve the generic-vnfs related to it.
Input: pserver, Output: generic-vnfs

```
Custom Query API: PUT /aai/v26/query?format=resource
 API payload:
 {
 "start":["cloud-infrastructure/pservers?hostname={hostname}"],
 "query":"query/genericVnfs-fromPserver?nfNamingCode={nf-naming-code}"
 }
```

Output


```
{
    "results": [
        {
            "id": "124981392",
            "node-type": "generic-vnf",
            "url": "/aai/v16/network/generic-vnfs/generic-vnf/71a31568-a2a8-4505-a303-d3a486ff24e5",
            "properties": {
                "vnf-id": "71a31568-a2a8-4505-a303-d3a486ff24e5",
                "vnf-name": "US2929ORLFL0136UJDM06",
                "vnf-type": "DM",
                "service-id": "d7bb0a21-66f2-4e6d-87d9-9ef3ced63ae4",
                "prov-status": "PROV",
                "equipment-role": "EXAMPLE_SERVICE_SUBSCRIPTION",
                "orchestration-status": "Activated",
                "ipv4-oam-address": "12.80.1.17",
                "nm-lan-v6-address": "2001:1890:e00e:fffe::dc01",
                "vmemory-units": "",
                "vdisk-units": "",
                "in-maint": false,
                "is-closed-loop-disabled": false,
                "resource-version": "1518401336774",
                "nm-profile-name": "EXAMPLE_PROFILE_NAME"
            },
            "related-to": [
                {
                    "id": "280850472",
                    "relationship-label": "org.onap.relationships.inventory.ComposedOf",
                    "node-type": "service-instance",
                    "url": "/aai/v16/business/customers/customer/EXAMPLE_CUSTOMER/service-subscriptions/service-subscr
                },
                {
                    "id": "25850048",
                    "relationship-label": "tosca.relationships.HostedOn",
                    "node-type": "pserver",
                    "url": "/aai/v16/cloud-infrastructure/pservers/pserver/EXAMPLE_PSERVER"
                }
            ]
        },
        {
            "id": "141865152",
            "node-type": "generic-vnf",
            "url": "/aai/v16/network/generic-vnfs/generic-vnf/9efb0471-8f30-4f7f-9225-57778a2bcdef",
            "properties": {
```


DSL query example:
To execute a DSL query, a client will perform a PUT on the dsl endpoint and include a payload that includes the dsl query to be run
Example:

Starting with a customer object, traverse through service-subscriptions and service-instances, and traverse any edges to connector objects. Store the connectors and any connected virtual data centers and their generic vnfs, and return the set.

```
{
 "dsl" : " customer('global-customer-id','8310000058863-16102016-aai1539') > service-subscription > service-instance > connector* > virtual-data-center* > generic-vnf* "
 }
```

Custom query used by SDN-C AAI Service:
Sample custome-query request

```
<save plugin="org.openecomp.sdnc.sli.aai.AAIService"
                        resource="custom-query"
                        key="format = 'resource'"
                        force="true"
                        local-only="false"
                        pfx="tmp.AnAI-data.vnf">
                        <parameter name="start[0]" value="`'network/generic-vnfs?vnf-name=' + $tmp.vnf-name`" />
                        <parameter name="start_length" value="1" />
                        <parameter name="query" value="query/complex-fromVnf" />
</save>
```

AAI Installation

https://docs.onap.org/projects/onap-aai-aai-common/en/latest/platform/Getting%20Started/AAI_Developer_Environment_Setup.html

Development Environment Setup: https://docs.onap.org/projects/onap-aai-aai-common/en/latest/platform/Getting%20Started/AAI_Developer_Environment_Setup.html

1. Install single node janusgraph: This will start cassandra, elasticsearch and gremlin server(including janusgraph) process
2. Install haproxy
    - This will install HaProxy which act as gateway and listen on 8443.
    - Now All request can be made to 8443 and it will redirect to aai
3. Download AAI below services code: `aai-common schema-service resources traversal graphadmin logging-service`
4. Janus setup config to services: update config to resource, traversal and graphadmin in properties files
    - storage.backend=cassandra
    - storage.hostname=localhost
    - storage.cassandra.keyspace=onap
5. Build all above modules
6. Install the schema: Run graphadmin genteaster on the local instance which commit schema to db..
7. start Resource, traversal and graphadmin services

OOM based Setup: https://wiki.onap.org/pages/viewpage.action?pageId=35522241

Troubleshooting:

To verify that Janusgraph installed successfully, run below script which is part of janusgrah tar pkg.

```
./bin/janusgraph.sh status
```

The output should looks like this:

```
Gremlin-Server (org.apache.tinkerpop.gremlin.server.GremlinServer) is running with pid 9835
Elasticsearch (org.elasticsearch.bootstrap.Elasticsearch) is running with pid 9567
Cassandra (org.apache.cassandra.service.CassandraDaemon) is running with pid 9207
```

Swagger: https://wiki.onap.org/display/DW/AAI+REST+API+Documentation+-+Jakarta

Logs: `check logs for schema is committed successfully in file` **`graphadmin/logs/createschema/metrics.log`**

Schema driven Inventory

Xsd generation from OXM

Dynamic schema update :
In aai schema service, it generate xsd file from oxm.

At compile time, run GenrateXsd class to generate xsd file.
oxm file and output xsd file path is mentioned in POM.

File Location: schema-service---> aai-schema-gen ----> POM.xml

.**Exec Maven Plugin**

A plugin to allow execution of system and Java programs

Class to run: GenrateXsd

```
<version>1.1.1</version>
<executions>
    <execution>
        <id>autoGenerateXsd</id>
        <phase>process-classes</phase>
        <goals>
            <goal>java</goal>
        </goals>
        <configuration>
            <mainClass>org.onap.aai.schemagen.GenerateXsd</mainClass>
            <systemProperties>
                <systemProperty>
                    <key>gen_version</key>
                    <value>ALL</value>
                </systemProperty>
                <systemProperty>
                    <key>gen_type</key>
                    <value>XSD</value>
                </systemProperty>
                <systemProperty>
                    <key>schema.xsd.maxoccurs</key>
                    <value>${schema.xsd.maxoccurs}</value>
                </systemProperty>
                <systemProperty>
                    <key>yamlresponses_url</key>
                    <value></value>
                </systemProperty>
                <systemProperty>
                    <key>yamlresponses_label</key>
                    <value></value>
                </systemProperty>
                <systemProperty>
                    <key>schema.configuration.location</key>
                    <value>${schema.configuration.location}</value>
                </systemProperty>
                <systemProperty>
                    <key>schema.nodes.location</key>
```

Java class generation from schema(xsd file) in AAI

https://www.mojohaus.org/jaxb2-maven-plugin/Documentation/v2.2/example_xjc_basic.html

In schema service, generate java class from xsd file.
It uses `jaxb2-maven-plugin maven plugin to generate class from xsd files.`

POM File Location: schema-service---> aai-schema ----> POM.xml

```
maven plugin: jaxb2-maven-plugin (org.codehaus.mojo)
goals : xjc ,
```

```
Config:
<outputDirectory>${project.build.directory}/generated-sources</outputDirectory>
<sources>
   <source>${aai.xsd.source}</source>
</sources>
<xjbSources>
   <xjbSource>${aai.bindings.source}</xjbSource>
</xjbSources>
```

xsd Source:

```
<properties>
   <onap.nexus.url>https://nexus.onap.org</onap.nexus.url>
   <!-- Start of Default ONAP Schema Properties -->
   <aai.xsd.source>src/main/resources/onap/aai_schema</aai.xsd.source>
   <aai.bindings.source>src/main/xjb/bindings-onap.xjb</aai.bindings.source>
   <!-- End of Default ONAP Schema Properties -->
</properties>
```

```
<plugins>
    <plugin>
        <groupId>org.codehaus.mojo</groupId>
        <artifactId>jaxb2-maven-plugin</artifactId>
        <version>2.5.0</version>
        <executions>
            <execution>
                <id>xjc</id>
                <goals>
                    <goal>xjc</goal>
                </goals>
                <configuration>
                    <outputDirectory>${project.build.directory}/generated-sources</outputDirectory>
                    <sources>
                        <source>${aai.xsd.source}</source>
                    </sources>
                    <xjbSources>
                        <xjbSource>${aai.bindings.source}</xjbSource>
                    </xjbSources>
                    <xjcSourceExcludeFilters>
                        <filter
                                implementation="org.codehaus.mojo.jaxb2.shared.filters.pattern.PatternFileFilter">
                            <patterns>
                                <pattern>edgetagquery\.xsd</pattern>
                            </patterns>
                        </filter>
                    </xjcSourceExcludeFilters>
                    <extension>true</extension>
                    <arguments>
                        <argument>-Xannotate</argument>
                        <argument>-XtoString</argument>
                    </arguments>
```

Note: Usages of these java classes are not clear(it is not used by client OR resource service. maybe used by other aai mS)

AAI Dynamic APIs and Entities(Resource mS)

Dynamic APIs:

- Restcontroller for handle CRUD operation on resource.
- wildcard URLsupport for handle any resource by parsing the URL.
- Using resource name from url, generate/query graph DB for vertex with resource name.

Dynamic Entities:

- Dynamic entities class using DB schema from OXM file for each resource in oxm file.
- request body(xml/json) for a resource is marshal/unmarshal to dynamic entities.
- Usages EclipceLink JPA for Dynamic entities.

Graph DB Data Op:

- Tinkerpop gremlin client libs to communicate with janus graph DB.
- AAI internal code for handling DB request from rest request.
- Based on URL, first get resource and create/query vertex
- Then from dynamic entities class for this resource, get each properties and update in vertex peropties by using tinkerpop clint,

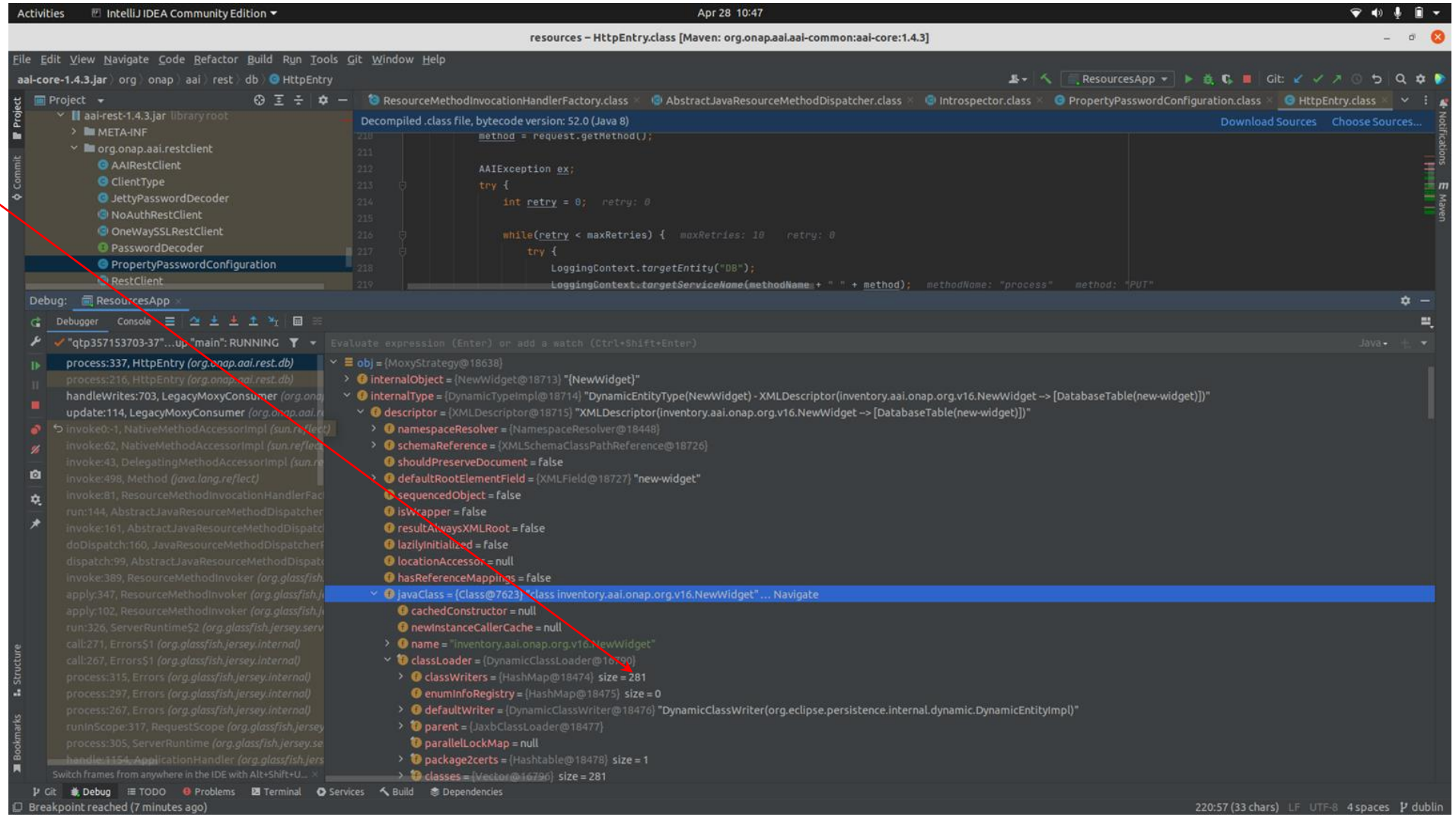
LegacyMoxyConsumer.java

```
@Controller
@Path("{version: v[1-9][0-9]*|latest}")
public class LegacyMoxyConsumer extends RESTAPI {
```

```
@PUT
@Path("/{uri: .+}")
@Consumes({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
@Produces({ MediaType.APPLICATION_JSON, MediaType.APPLICATION_XML })
public Response update (String content, @PathParam("version")String versionParam, @PathParam("uri") @Encoded String uri,
    MediaType mediaType = headers.getMediaType();
    return this.handleWrites(mediaType, HttpMethod.PUT, content, versionParam, uri, headers, info);
}
```

281 dynamic Entities Class based on OXM files

```
obj = request.getIntrospector();
```



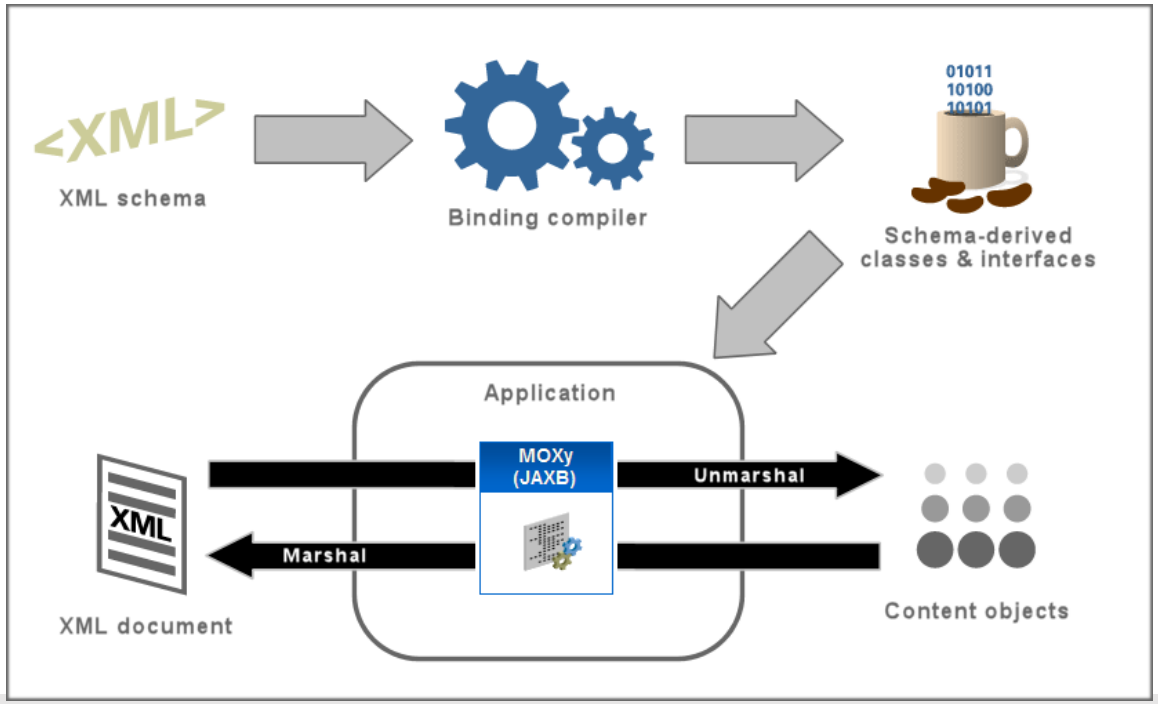
EclipseLink Moxy

https://www.eclipse.org/eclipselink/documentation/2.6/moxy/dynamic_jaxb004.htm

Eclise Moxy: generate dynamic entities class template from schema OXM file.

The MOXy component, supplied by EclipseLink, enables to efficiently bind Java classes to XML schemas.

When using EclipseLink MOXy as the JAXB provider, no metadata is required to convert your existing object model to XML.



Example 9-12 Sample XML Schema

```
<?xml version="1.0" encoding="US-ASCII"?>
<xml-bindings xmlns="http://www.eclipse.org/eclipselink/xsds/persistence/oxm" xmlns:xsi="http://www.w3.org/2001/XMLSchema-insta
    xmlns:xs="http://www.w3.org/2001/XMLSchema" package-name="example">

    <java-types>
        <java-type name="Customer">
            <xml-root-element name="customer"/>
            <java-attributes>
                <xml-element java-attribute="firstName" type="java.lang.String"/>
                <xml-element java-attribute="lastName" type="java.lang.String"/>
                <xml-element java-attribute="address" type="example.Address"/>
            </java-attributes>
        </java-type>

        <java-type name="Address">
            <java-attributes>
                <xml-element java-attribute="street" type="java.lang.String"/>
                <xml-element java-attribute="city" type="java.lang.String"/>
                <xml-element java-attribute="province" type="java.lang.String"/>
                <xml-element java-attribute="postalCode" type="java.lang.String"/>
            </java-attributes>
        </java-type>
    </java-types>

</xml-bindings>
```

Example 9-13 Sample Application Code

```
InputStream iStream = myClassLoader.getResourceAsStream("example/resources/eclipselink/eclipselink-oxm.xml");

Map<String, Object> properties = new HashMap<String, Object>();
properties.put(JAXBContextProperties.OXM_METADATA_SOURCE, iStream);

DynamicJAXBContext jaxbContext = DynamicJAXBContextFactory.createContextFromOXM(myClassLoader, properties);

DynamicEntity newCustomer = dContext.newDynamicEntity("example.Customer");
newCustomer.set("firstName", "George");
newCustomer.set("lastName", "Jones");

DynamicEntity newAddress = dContext.newDynamicEntity("example.Address");
newAddress.set("street", "227 Main St.");
newAddress.set("city", "Toronto");
newAddress.set("province", "Ontario");
newAddress.set("postalCode", "M5V1E6");

newCustomer.set("address", newAddress);

dContext.createMarshaller().marshal(newCustomer, System.out
```

Gremlin query language

What are the names of Gremlin's friends' friends?

- 1. Get the vertex with name "gremlin."
- 2. Traverse to the people that Gremlin knows.
- 3. Traverse to the people those people know.
- 4. Get those people's names.

```
g.V().has("name","gremlin").  out("knows").
out("knows").  values("name")
```

What are the names of the projects created by two friends?

- 1. ...there exists some "a" who knows "b".
- 2. ...there exists some "a" who created "c".
- 3. ...there exists some "b" who created "c".
- 4. ...there exists some "c" created by 2 people.
- 5. Get the name of all matching "c" projects.

```
g.V().match(  as("a").out("knows").as("b"),
as("a").out("created").as("c"),  as("b").out("created").as("c"),
as("c").in("created").count().is(2)).    select("c").by("name")
```

Get the managers from Gremlin to the CEO in the hierarchy.

- 1. Get the vertex with the name "gremlin."
- 2. Traverse up the management chain...
- 3. ...until a person with the title of CEO is reached.
- 4. ...Get name of the managers in the path traversed.

```
g.V().has("name","gremlin").
  repeat(in("manages")).
    until(has("title","ceo")).
path().by("name")
```

Get the distribution of titles amongst Gremlin's collaborators.

- 1. Get the vertex with the name "gremlin" and label it "a."
- 2. Get Gremlin's created projects and then who created them...
- 3. ...that are not Gremlin.
- 4. Group count those collaborators by their titles.

```
g.V().has("name","gremlin").as("a").
  out("created").in("created").
    where(neq("a")).
groupCount().by("title")
```

Get a ranked list of relevant products for Gremlin to purchase.

- 1. Get the vertex with the name "gremlin."
- 2. Get the products Gremlin has purchased and save as "stash."
- 3. Who else bought those products and what else did they buy...
- 4. ...that Gremlin has not already purchased.
- 5. Group count the products and order by their relevance.

```
g.V().has("name","gremlin").
  out("bought").aggregate("stash").
  in("bought").out("bought").
    where(not(within("stash"))).
groupCount().order(local).by(values,desc)
```

TinkerPOP gremlin client in Java

Gremlin java class:

```
public Class GremlinTinkerPopExample{
 public void run(String name, String property){
  Graph graph = GraphFactory.open(...);
  GraphTraversalSource g =
traversal().withEmbedded(graph);

  double avg = g.V().has("name",name).
         out("knows").out("created").
         values(property).mean().next();

   System.out.printIn("Average rating :" +avg);
  }
 }
```

```
Graph graph = GraphFactory.open(...);
GraphTraversalSource g;
g = traversal().withEmbedded(graph);          //local OLTP
g = traversal().withRemote(DriverRemoteConnection.using("Localhost",8182));      //remote
g = traversal().withEmbedded((graph).withComputer(SparkGraphComputer.class);     //distributed OLAP
```

THANKS

AAI Client

Get xsd file from schema service and generate classes for each resource in schema file
Use:

```
maven plugin: jaxb2-maven-plugin (org.codehaus.mojo)
goals : xjc
```

Resource mS : XML to DB entities

https://wiki.eclipse.org/EclipseLink/Examples/SDO/JPA

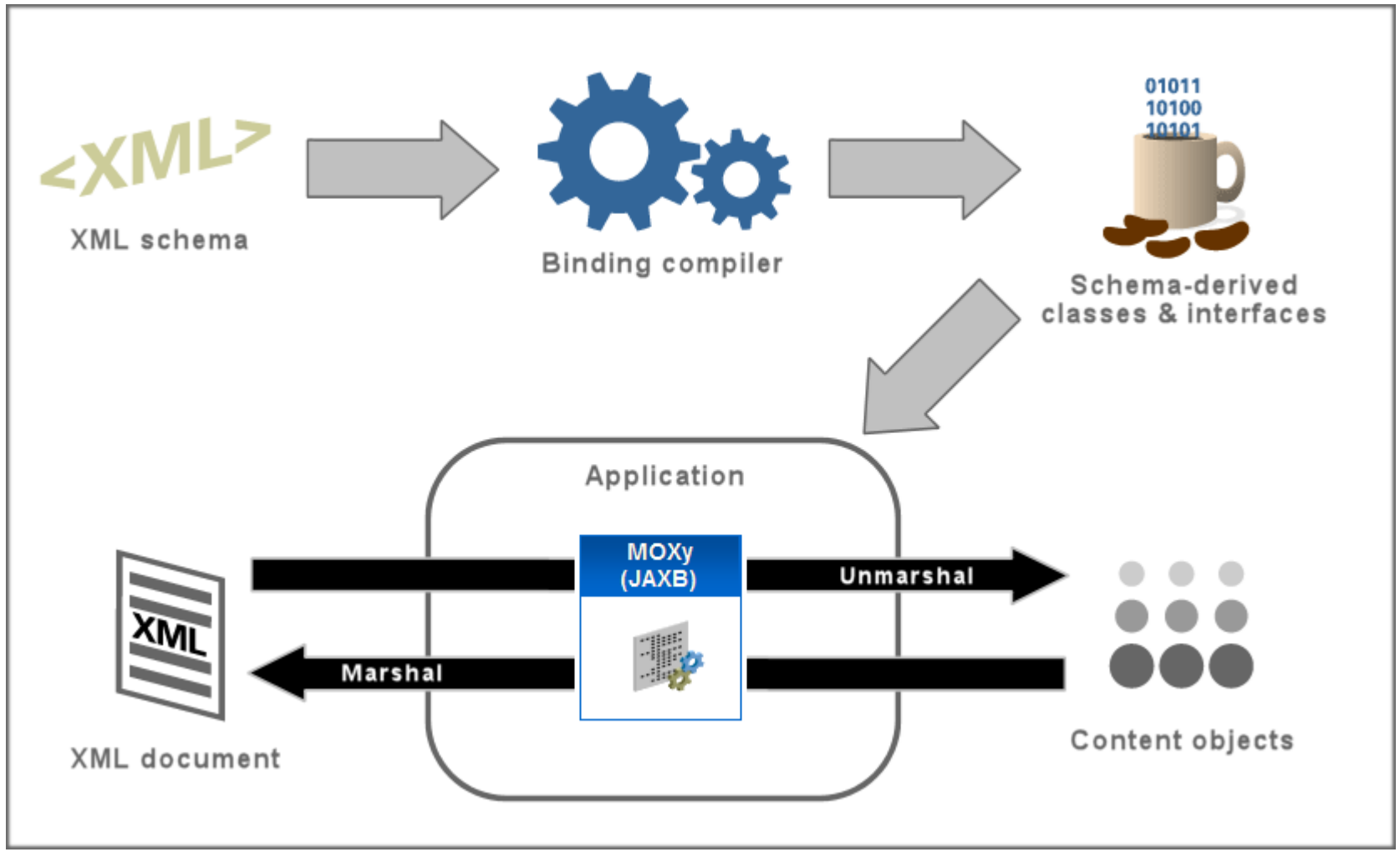
https://www.eclipse.org/eclipselink/api/2.3/org/eclipse/persistence/jaxb/dynamic/DynamicJAXBContextFactory.html

https://www.eclipse.org/eclipselink/documentation/2.6/moxy/dynamic_jaxb004.htm

Eclipse MOXy:

The MOXy component, supplied by EclipseLink, enables you to efficiently bind Java classes to XML schemas.

When using EclipseLink MOXy as the JAXB provider, no metadata is required to convert your existing object model to XML. You can supply metadata (using annotations or XML) if you want to fine-tune the XML representation.



Query Builder

Gremlin QueryBuilder

- Provides abstraction from the A&AI schema and from the underlying graph query language
- Converts the A&AI internal DSL to gremlin

Key Methods

- getVerticesByProperty(key,value) à has('key','value')
- getVerticesByProperty(key, MissingOptionalParameter value)
- getVerticesByProperty(key) à has('key')
- createEdgeTraversal(EdgeType, nodeType1, nodeType2)
- Find if edgerule exists between the nodeTypes. If yes, finds the direction for the traversal .out().has('aainode-type','nodeType2') or .in().has('aai-node-type','nodeType2')
- createEdgeTraversalWithLabels(EdgeType, nodeType1, nodeType2, List<> labels) .out('edgelabel').has('aai-node-type','nodeType2') or .in('edgelabel').has('aai-node-type','nodeType2')

Implemented constructs

- Union, where, has, hasNot, select, or , store, cap, unfold, dedup, emit, repeat, both, tree, by, path

Query Builder Internals

{ "start" : ["cloud-infrastructure/cloud-regions/cloud-region/{cloud-owner}/{cloud-region-id}"], "query" : "query/availabilityZoneAndComplex-fromCloudRegion" }

• builder.union(builder.newInstance().createEdgeTraversal(EdgeType.TREE, 'cloud-region', 'availability-zone').store('x'),builder.newInstance().createEdgeTraversal(EdgeType.COUSIN, 'cloud- region', 'complex').store('x')).cap('x').unfold().dedup()

• g.V(vertices).has('aai-node-type', 'cloud-region').has('cloud-owner', 'onap').has('cloud-region-id', 'ONAP25').union(__.in('org.onap.relationships.inventory.BelongsTo').has('aai-node-type', 'availability-zone').store('x'),__.out('org.onap.relationships.inventory.LocatedIn').has('aai-node-type', 'complex').store('x')).cap('x').unfold().dedup()

• GraphStep([],vertex), HasStep([aai-node-type.eq(cloud-region)]), StoreStep(x), HasStep([aai-node-type.eq(service-subscription)]), StoreStep(x), StoreStep(x), VertexStep(OUT,[usesL3Network],vertex), HasStep([aai-node-type.eq(l3-network)]), StoreStep(x), VertexStep(IN,[uses],vertex), HasStep([aai-node-type.eq(cloud-region)]), StoreStep(x), SideEffectCapStep([x]), UnfoldStep]

• CQ2Gremlin : API that will help you verify the gremlin for your query

Resource mS Dynamic POJOs

https://www.eclipse.org/eclipselink/documentation/2.5/moxy/dynamic_jaxb003.htm