Optimization Service Design Framework

Gliffy Macro Error

An error occurred while rendering this diagram. Please contact your administrator.

- **Name:** Placement-via-HAS Copy Copy

Contents

Summary

The OOF plans to provide optimization capability as a service for ONAP R2 and beyond. OOF uses a typical optimization construct:

- Objective: Maximize/minimize a metric, measured by appropriate key performance indicators (KPIs)
- Technology and operating constraints, such as:
  - Parameter change limits (such as power)
  - Frequency of changes permitted
  - Number of parameters that can be changed simultaneously
  - Data latencies (typically in percentile)
  - DC compute, network, storage, energy capacity
  - Location based and time based energy cost

The objective metrics could be throughput (maximize), interference levels (minimize), accessibility/retainability (maximize), cost (minimize) etc. KPIs could be infrastructure utilization statistics provided by ONAP-MC.

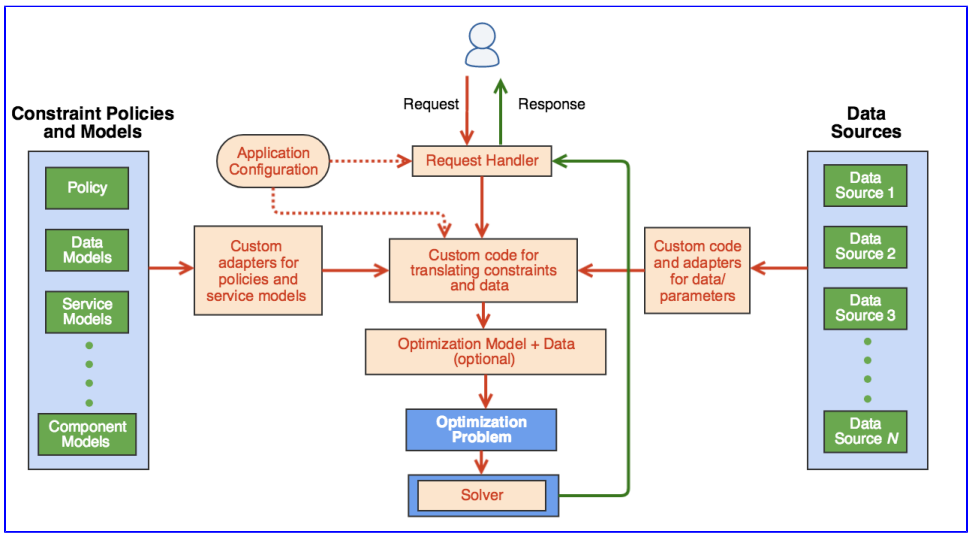
The OOF is developed based on the following core ideas:

1. Most optimization problems can be solved in a declarative manner using a high-level modeling language.
2. Recent advances in open source optimization platforms allow the solution process to be mostly solver-independent.
3. By leveraging the library of standard/global constraints, optimization models can be rapidly developed.
4. By developing a focused set of platform components, we can realize a policy-driven, declarative system that allows ONAP optimization applications be composed rapidly and managed easily
    a. Policy and data adapters
    b. Execution and management environment
    c. Curated "knowledge base" and recipes to provide information on typical optimization examples and how to use the OOF
5. More importantly, by providing a way to support both "traditional" optimization applications and model-driven applications, we can provide a choice for users to adapt the platform based on their business needs and skills/expertise.

The OOF aims to realize these via a set of initial applications for ONAP use cases that are being developed collaboratively across a broad team.

Rationale and Motivation

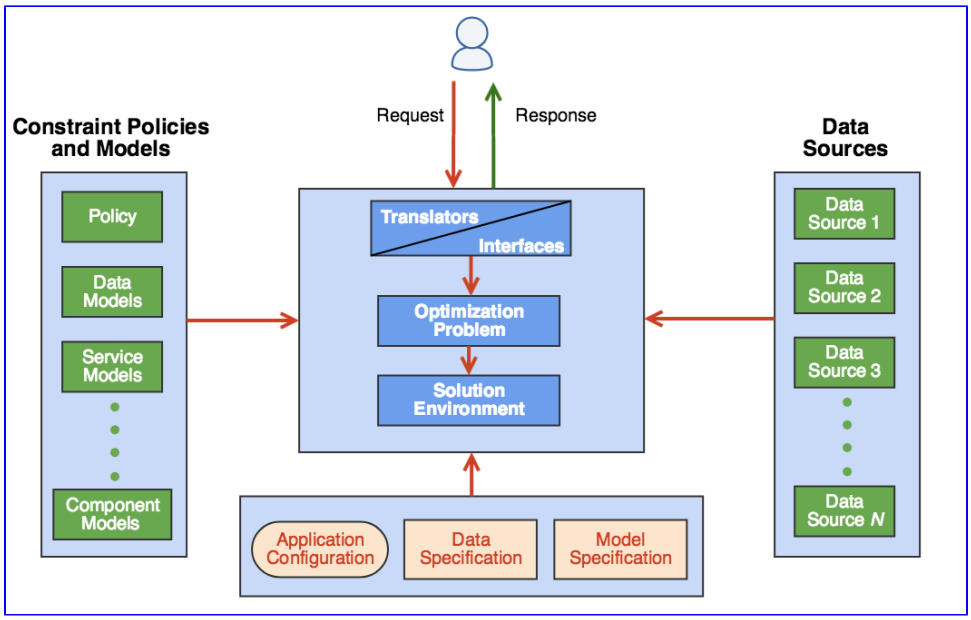
Traditional Optimization



Traditionally, optimization applications are tailor-made for specific requirements, and the process for developing an optimization application often involves substantial application-specific "custom code". Any changes in the problem (e.g. new optimization constraints, objectives, or data sources) requires development effort involving code changes in various components of the application. These changes can span aspects such as (a) Optimization Model Specification, (b) Request Handler, (c) Adapters for data and parameters, (d) Application Configuration, (e) Code in custom solver, etc., and involve long development cycles even for simple changes in requirements.

Declarative, Policy- and Model-Driven Architecture for Optimization Applications

The goal of the Optimization Framework is to drastically reduce the amount of such code changes by providing platform-level functionality.



Features and Utility of OOF

The main features of the ONAP Optimization Framework (OOF) are:

1. It provides a robust, scalable optimization framework for rapidly developing new optimization applications independent of how the underlying optimization modules are implemented.
2. It enables reusability of optimization engines, addresses problems arising due to different applications using custom optimization codes, adapter libraries, and configuration logic. This is achieved via a policy-driven configuration system, and a library of generalized optimizers that can be configured via policies.
3. OOF-based solutions can be quickly on-boarded onto ONAP, linked to various data collectors, databases, and microservices in ONAP/DCAE, and can dynamically be scaled at run-time.
4. Overall, the OOF eliminates software redundancy and inconsistencies arising from variations in quality and configurability of different optimizers. The unified approach of OOF reduces the overhead associated with managing different optimization applications.

Advantages of the Unified Approach of OOF

1. OOF is policy driven.
    a. OOF provides mechanisms to specify optimization constraints as policies that are configurable by service designers or operators. In contrast, legacy optimization applications include such information inside configuration files and sometimes in the code.
    b. Constraints and other policies are available for multiple uses, which encourages reusability. This reduces inconsistencies in constraints or policies across services, and helps reduce duplication of effort for common tasks.
2. OOF provides reusable, model-driven adapters for data sources and external systems
    a. Data formats and API calls are model-driven in OOF, so likely errors are identified very early in the request-response sequence. OOF provides adapters to different data sources (DCAE as well as external systems), which can be directly reused.
3. OOF is agnostic towards service, application, and optimization engine technology/language
    a. Optimizers that can be chained together in a technology and programming-language agnostic manner (e.g. general purpose mathematical solvers such as GLPK and CPLEX can co-exist with custom-implementations of algorithms). New optimization solutions can be composed by chaining existing optimizers (e.g. by linking placement, networking, and licensing optimizations).
4. OOF provides a dynamically scalable, fault-tolerant environment with resource pooling
    a. Runtime environment is based on ONAP/DCAE's Hadoop/Yarn technologies (and can be easily adapted to other cluster technologies such as kubernetes).
    b. OOF uses a queue-based system with independent ``workers'' processing optimization tasks. These workers can be dynamically scaled and jobs are picked up by the next available worker if a worker container fails.

Contributors

Overall Architecture/Design: AT&T, VMWare, Intel

Core OOF Components (Adapters, architecture, seed code): AT&T

Underlying optimization platform: Code developed by University of Melbourne and Monash University

Packaging and verification of OOF System: VMWare, NetCracker, AT&T

Policies and interpretation: Intel, AT&T

Homing and Allocation Service: AT&T, Intel

CI/CD and Test Coverage: NetCracker, AT&T

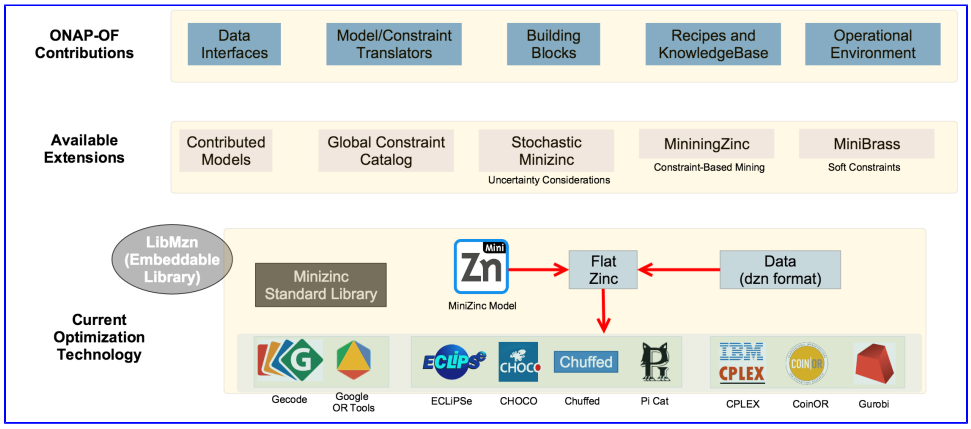
Modification of Adapters: Huawei, AT&T, VMWare

Functional Architecture

Technology Choices

Minizinc provides an open source constraint modeling language/platform for specifying optimization applications. It contains direct interfaces to COIN-OR CBC, Gurobi and IBM ILOG CPLEX. Additionally, many optimization projects support minizinc via FlatZinc interfaces. The Minizinc standard library provide s a subset of constraints form the global constraint catalogue as a high-level abstraction that have efficient algorithms implemented by several solvers.

Components of the Core Framework



An overview of the components of the core optimization framework. The OOF utilizes the open source project Minizinc, which has a solver-independent modeling language and has interfaces to various open source and commercial solvers. One of the additional benefits of this approach is that by developing a focused set of ONAP-related components, we can utilize ongoing advances in optimization technologies, as well as adapt other currently available extensions to Minizinc and related projects. The OOF project aims to build these components with a focus on minimal viable product for Beijing Release in order to support initial applications and use cases, with subsequent focus on expanding the platform.

Data Adapter Library

The OOF will provide a library of adapters for common ONAP systems. These can be directly used in data specification templates of the applications. In the initial release, these will include adapters to Policy, A&AI, Multi-Cloud, and SDC (additional "stretch goals" for this release include SDN-C, Microservice Bus). As new use cases are implemented, this library will be augmented by new adapters to other services.

Translation Modules

The OOF will provide modules for translating policies into constraints for the optimization environment. When an underlying minizinc model is used for optimization, it is translated into a minizinc constraint (either via a data specification template or directly from policy). For custom optimizers, these constraints will be translated to the input format expected by the optimizer via the data specification template. The Homing and Allocation Service (HAS; described in the next section) uses a custom optimization module and hence uses the data specification template approach.

Modeling Support

The OOF provides simple templating system through which users can specify links to different ONAP components, including policy, A&AI, SDC, etc.

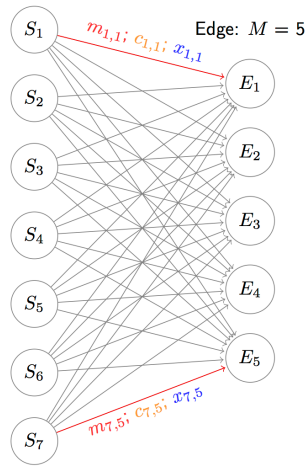
Execution Environment

The OOF execution environment contains the minizinc system, along with data/template rendering system that leverages the adapters to various systems. The flow of execution can be configured via the configuration file for the application and supports a choice of specific solver or invocations to custom /external solvers.

Simple Example: Budget Constrained Maximum Network Flow Problem

Problem Description

Router: $N = 7$



Edge: $M = 5$

Terms

x_{ij}	Utilization of link $S_i \to E_j$
c_{ij}	Cost/unit for using link $S_i \to E_j$
m_{ij}	Maximum bandwith of link $S_i \to E_j$
p_i	Maximum amount of traffic from node S_i
q_j	Maximum amount of traffic to node E_j
B	Budgeted funds

Objective

Maximize: $\sum_{j=1}^{M} \sum_{j=1}^{M} x_{ij}$

Constraints

$0 \leq x_{ij} \leq m_{ij} \quad i = 1, \ldots, N \quad j = 1, \ldots, M$ (flow limits)

$\sum_{i=1}^{N} x_{ij} \leq q_j \quad \forall j \in 1, \ldots, M$ (node capacity)

$\sum_{j=1}^{M} x_{ij} \leq p_i \quad \forall i \in 1, \ldots, N$ (node capacity)

$\sum_{j=1}^{M} \sum_{j=1}^{M} c_{ij} x_{ij} \leq B$ (budget)

Illustration of the budget-constrained max-flow problem. Given a set of starting nodes and destination nodes, the objective is to calculate the maximum amount of "flow" across the nodes, subject to capacity constraints of nodes, constraints on the capacity of the connections (edges), as well as costs associated with utilization of each connection (edge). The objective is to maximize the flow subject to the budget constraint.

This model can be composed from different components, each developed by contrbutors with different expertise and roles. The development can happen at different times (initial problem definition, service design time, run-time, and continuous improvement stages). Contributors can have different roles and expertise. Examples include:

1. An optimization modeler (with expertise on modeling and mathematical aspects of optimization)
2. A software developer with knowledge of data modeling and model/data templates
3. A developer with expertise in creating policy models in the context of specific business domains
4. An Ops team member with knowledge of applicable policies for a set of services and applications and knowledge on run-time configuration

An optimization model can therefore evolve from a mathematical concept to a computer program to a policy-driven, dynamically re-configurable application through the following stages:

1. An optimization modeler identifies the key concepts, comes up with a mathematical model, creates an optimization model, and tests with a small example dataset (a prototype corresponding to the model shown in the problem description above.
2. Then, a developer with an understanding of the API requirements, basic understanding of the model and data sources/adapters links the model to a data template (template file shown below). The developer needs a basic understanding of key variables of a model such as nodes, bandwidths, capacities in this example. While most adapters will be available from OSDF library, additional adapters/libraries can be developed and can be contributed back to the ODSF library.
3. A service designer, with basic understanding of key model concepts (key variables of a model such as nodes, bandwidths, capacities in this example) can create policy models (or use/extend existing constraint policy models from the OSDF library). In this example, this introduces a new constraint reflecting the maximum amount of flow that can go through a single network link, in order to reduce service disruption risks.
4. On top of this, an Ops person can enable or configure an applicable run-time operational constraint policy. In this example, this introduces a more stringent budget constraint (from all allowed budget to only 80% of the budget).

Thus, a new service can be created by extending existing optimization models, policies, and adapters, and using them as building blocks/ingredients.

Minizinc Model

MiniZinc model for the example application (budget constrained network flow optimization model). This model can be composed from different components, each independently contributed by contrbutors with different expertise and roles, as described above.

```
int: N;  % input nodes

int: M;  % output nodes

int: maxbw; % max bandwidth (for convenience)

float: budget;

set of int: inNodes = 1..N;
set of int: outNodes = 1..M;

array[inNodes] of int: inCap;  % capacities for input nodes

array[outNodes] of int: outCap;  % capacity for output nodes
array[inNodes, outNodes] of int: bw;  % max bandwidth of link

array[inNodes, outNodes] of float: cost;  % unit cost for the link

array[inNodes, outNodes] of var 0..maxbw: x;  % amount through this link


constraint forall (i in inNodes) (sum (j in outNodes) (x[i,j]) <= inCap[i]);
constraint forall (j in outNodes) (sum (i in inNodes) (x[i,j]) <= outCap[j]);
constraint forall (i in inNodes, j in outNodes) (x[i,j] <= bw[i,j]);

constraint sum (i in inNodes, j in outNodes) (x[i,j] * cost[i,j]) <= budget;
```

% Example of a constraint policy pushed by an Ops person at run-time (by "enabling" and/or configuring a policy)
% The OOF will inject the translated policy into the MiniZinc model for subsequent requests to this service

```
% another "stringent" service-specific policy

constraint sum (i in inNodes, j in outNodes) (x[i,j] * cost[i,j]) <= 0.8 * budget;
```

% Example of a constraint policy specified and/or pushed by a service designer at design time
% The OOF will inject the translated policy into the MiniZinc model for all requests to this service

```
% each link cannot have more than 20% of traffic from a customer

var flow = sum (i in inNodes, j in outNodes) (x[i,j]);
constraint forall (i in inNodes, j in outNodes) (x[i,j] <= 0.2 * flow);

solve maximize sum (i in inNodes, j in outNodes) (x[i,j]);
```

Minizinc Data Template

Data file used in the for the example application. The file format is dzn (Minizinc data format) and the file uses the widely used jinja2 templating for Python, with support for OOF to objects such as "input" (the input API request), SDC (a dummy object that provides information on cost per unit network utilization for each network edge), and AAI (another dummy object that provides network capacities of nodes, and also bandwith for links among different nodes). This data template is rendered into a data file (dzn format), which, together with the model file defines a complete optimization problem.

```
% Relevant calls to APIs
{% inNodes, outNodes, budget = input.get("inNodes", "outNodes", "budget") %}
{% inCap, outCap = AAI.getCapacities(inNodes, outNodes) %};
{% bw = AAI.getBandwidthMatrix(inNodes, outNodes) %};
{% cost = SDC.getNetworkCostMatrix(inNodes, outNodes) %};

N = {{ len(inNodes) }};
M = {{ len(outNodes) }};
maxbw = {{ max(max(bw)) }};
budget = {{ budget }};

inCap = {{ inCap }};
outCap = {{ outCap }};

bw = {{ mzn.toMatrix(bw) }}; % writes it out as minizinc matrix
cost = {{ mzn.toMatrix(cost) }};
```

Minizinc Data File

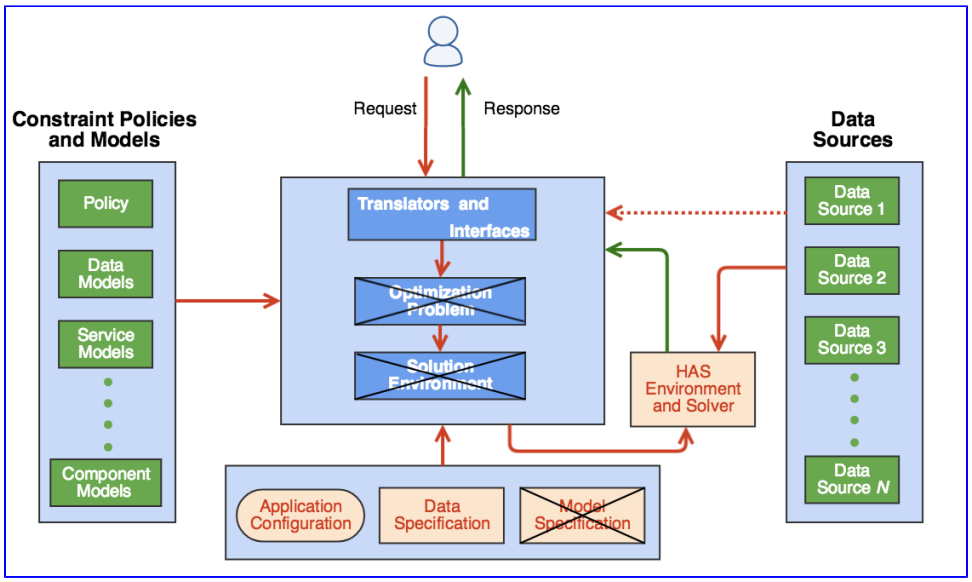
Rendered Minizinc Data File (from Template)

```
N = 5;
M = 4;
maxbw = 20;
budget = 50;

inCap = [10, 5, 0, 4, 20];

outCap = [10, 0, 5, 4];

bw = [| 10,  5,  0,  0
      |  2,  4, 10,  0
      |  4,  4, 10,  0
      |  2,  0,  0,  5
      |  0,  0,  0,  1 |];

cost = [|  1,  1, 10, 20
        | 90, 90, 90, 90
        |  2,  1,  1,  1
        |  2, 10, 10,  1
        |  9,  9,  9, 99.9 |];
```

Initial, Representative Applications

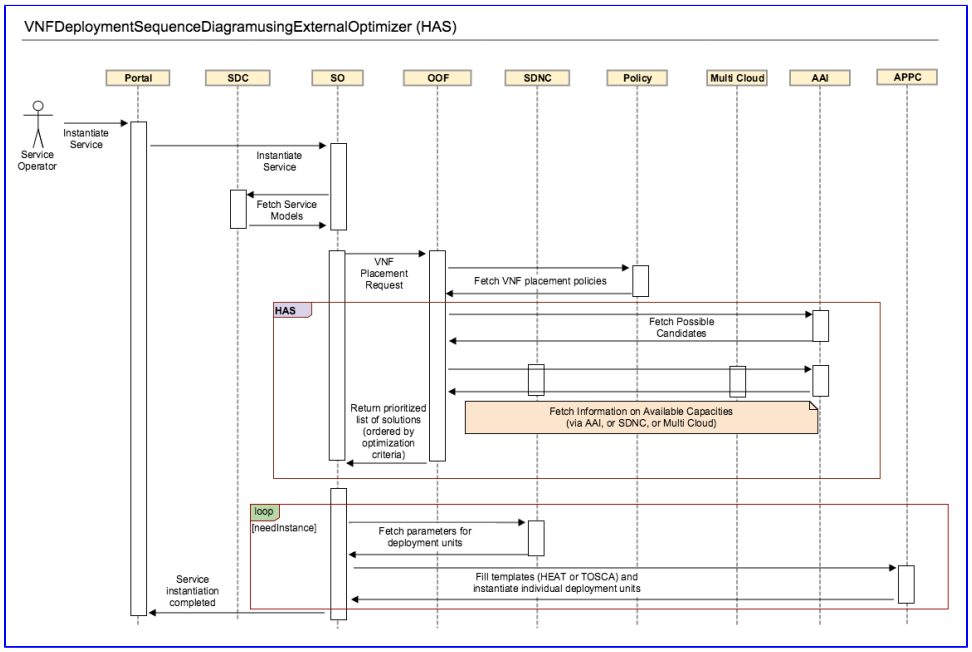
Homing and Allocation Service (HAS)

OF-HAS is a policy-driven placement optimizing service (or homing service) that allows ONAP to deploy services automatically across multiple sites and multiple clouds. It enables placement based on a wide variety of policy constraints including capacity, location, platform capabilities, and other service specific constraints. Given a set of service components (based on SO decomposition flows) and requirements for placing these components (driven by policies), HAS finds optimal resources (cloud regions or existing service instances) to home these service components such that it meets all the service requirements. HAS is architected as an extensible homing service that can accommodate a growing set of homing objectives, policy constraints, data sources and placement algorithms. More details on HAS flow and architecture can be found in the HAS Project Page.



The HAS application highlights how a custom optimization application can utilize the OOF. In this specific case, the HAS application provides on its own: (a) an execution environment, (b) an API specification, (c) adapters to data components such as A&AI and SO, (d) and a custom optimization algorithm /solver. The HAS application relies on OOF for policy retrieval and translation.

Sequence of steps involved in a placement request and solution process. In this particular case, the HAS application fetches the data from A&AI and applies constraints to identify the solution set.



Change Management Scheduling Optimization (CMSO)

This use case is meant to provide a basic skeleton for describing an example change management scheduling optimization problem. Typical CMSO applications involve a large range of constraints for describing the time constraints as well as complex inter-dependencies across different tasks and entities involved in the change management process. For this initial use case, we provide a small set of simple constraints via a simple but non-trivial application as a means to show how the OOF can be leveraged. As the broader CMSO use case evolves, we anticipate to make the application more representative.

Other Major ONAP-OOF Use Cases

In addition to Homing and Allocation Service (HAS) and Change Management Scheduling Optimization (CMSO), ongoing efforts focus on the following use cases:

OOF and ONAP Multicloud (ONAP-MC) for 5G-RAN

ONAP-MC currently provides infrastructure statistics and infrastructure fault notification/remediation as services. For ONAP R1, the MC service encompass detailed infrastructure utilization statistics across various subsystems compute, network, storage and energy. For ONAP R2 and beyond, this service plans to provide aggregate infrastructure utilization statistics at a multi-cloud instance level, clusters within a multi-cloud instance level etc.

In the context of 5G, OOF and ONAP-MC working together with other ONAP components can address several challenges such as

- Optimized SON dynamic spectrum allocation
- White space and unlicensed spectrum use
- Optimized VNF placement across distributed DCs for network slicing
- Slice optimization
- Macro and micro cell interplays
- Energy optimization
- Zero touch new carrier integration

Additional details of this use case are available at the Project Page for 5G-RAN Deployment Use Case

Higher Order Control Loop: VNF and Service scaling across multiple cloud instances

For Day 1 deployment of ONAP and going forward, dynamic scaling of VNF/service instances would entail the following:

- Horizontal scale out of the VNF within the existing cloud instance (VNF scaling) provided there is enough capacity within the cloud instance
- Horizontal scale out of the VNF to a new cloud instance (Service scaling) by creating a new VNF instance and other appropriate actions

The metrics needed for assessing the available capacity and the target cloud instance to migrate to (if needed) are:

- Infrastructure Metrics/KPIs from Multi Cloud
- Application Metrics/KPIs from DCAE

While some of these steps could arguably be provided by other ONAP components, the OOF is well positioned to provide a more holistic solution as part of homing and related optimization services. Ongoing work focuses on design this solution as simple API calls, so that individual components of this optimization application can be extracted and ingested into relevant ONAP framework or application components.

Release Planning

1. Amsterdam Release: OOF did not participate in the Amsterdam Release
2. Beijing Release:
    a. This is the first release. Details on the commitments, user stories, etc., are available at the Beijing Release Page
    b. Seed code from AT&T was onboarded by mid-January, 2018

Improvement of OOF by ingesting advances from open source efforts and new code

This process will merge new code being designed and developed collaboratively as part of the OOF project. The Beijing release epics and user stories capture the steps in reaching the target OOF platform for Beijing release.

Deployment Process

Linkages to different components and API's

Unit Tests and Code Coverage

Containerization and Deployment

Integration Tests

Links to Relevant Resources

Optimization and Minizinc Related Resources

1. Minizinc provides an open source constraint modeling language/platform for specifying optimization applications. It contains direct interfaces to COIN-OR CBC, Gurobi and IBM ILOG CPLEX. Additionally, many optimization projects support minizinc via FlatZinc interfaces.

2. The Minizinc standard library provides a subset of constraints form the global constraint catalogue as a high-level abstraction that have efficient algorithms implemented by several solvers.
3. The Minizinc team has developed courses on optimization and using Minizinc on Coursera. The Basic Modeling for Discrete Optimization and Advanced Modeling for Discrete Optimization courses provide an in-depth introduction to constraint modeling using MiniZinc.

Links to major ONAP components

1. ONAP Stack
    a. ONAP Installation on Vanilla OpenStack
    b. OOM for deployment and management of core ONAP stack. Installation instructions for Kubernetes: ONAP on Kubernetes
2. Policy system
3. Active and Available Inventory (AAI)
4. Service Orchestrator Project
5. Multi VIM/Cloud Project

Important Links

1. Håkan Kjellerstrand's page on Optimization Tools

Resources related to open issues

1. Questions on policy resolution. How do we manage a set of policies that are specified in a distributed manner. For example, if the VNF vendor specifies some policies, the site administrator provides corresponding optional "site-wide" policies, and an operator optionally provides over-ride policies. For initial releases, we anticipate that all relevant policies are in the same name-space for the application (which will mean that policies have to be replicated for each scenario, instead of hierarchically infer at run-time).

Feature Roadmap/Wishlist

Ingestion of new advances from open source optimization efforts

1. Explore aspects of uncertainty and robustness for optimization applications. For example, how do we develop an optimization solution for a future point of time, assuming that events between now and then can impact the validity or cost associated with the solution.
2. Explore how to develop optimization solutions when the input data has uncertainty (e.g. lagged data or aggregate summaries that we can get from some systems such as Multicloud).