|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | INTERNATIONAL TELECOMMUNICATION UNION  **TELECOMMUNICATION STANDARDIZATION SECTOR**  STUDY PERIOD 2017-2020 | | Focus Group on Autonomous Networks (FG-AN) | |
| AN-I-xxx | |
| Original: English | |
| **Question(s):** | | N/A | Virtual, 3-5 November 2021 | |
| **INPUT DOCUMENT** | | | | |
| **Source:** | | FUTMinna, Nigeria | | |
| **Title:** | | Student team “WINEST” presentation – Build-a-thon – represent “controllers” (closed loops) in declarative fashion (intent). | | |
| **Contact:** | | Guda Blessed,  FUTMinna, Nigeria | | Email: [gudablessed@gmail.com](mailto:gudablessed@gmail.com) |
| **Contact:** | | Ibrahim Aliyu,  Chonnam National University,  South Korea | | Email: [ibal2010@yahoo.com](mailto:ibal2010@yahoo.com) |
| **Contact:** | | Dr James Agajo (Associate Professor)  PhD (Telecommunication and Computer Engineering)  Head of Department, Computer Engineering  (FUTMinna, Nigeria) | | Email: james.agajo@futminna.edu.ng |

|  |  |
| --- | --- |
| **Keywords:** | Build-a-thon, closed loop, controller, compose, intent |
| **Abstract:** | This contribution is a report on activities of team “WINEST” towards the Build-a-thon hosted by ITU FG AN in ITU AI/ML in 5G Challenge (2021). It will cover the progress of activity 1 to represent “controllers” (closed loops) in a declarative fashion (intent) and compose them. A proof of concept demo using an open-source orchestrator will be presented.  Demo video: https://drive.google.com/file/d/1lzV\_KWo4CvMKNytrUdJ9CJPTE-vGUBi4/view |

**1. Introduction**

As discussed in [Build-a-thon FG AN], the “WINEST” team registered for the Build-a-thon problem statement and worked to produce a Proof of Concept (PoC) demo. This contribution is a report from the build-a-thon activity from team “WINEST” done as part of the ITU AI/ML in 5G Challenge.

The clauses below will cover the progress of activity 1 to represent “controllers” (closed loops) in a declarative fashion (intent) and compose them.

**2. Problem statement**

* The high complexity of management of future networks [which includes new innovative services, network configurations] has led to requirements for autonomous behavior.
* To enable autonomous networks, we use closed loops or controllers.
* Based on the gap analysis [FGAN-I-126]:
  + How is intent-driven management applied for the coordination of Closed Loops?
  + For Zero-touch service operations, intents should be the sole means of representation requirements
  + Study of how to translate intents into decisions and actions?
* Main summary from the gap analysis. – related to the intent – as discussed with ETSI ZSM

1. Recipe-based composition of controllers (or network instances) and the ability to come up with potentially new recipes.

2. Selection of Mechanisms for monitoring, analyzing, and adapting – e.g. migration, acceleration – needs to be studied, keeping in mind the parallel evolution of the controller components and the underlays.

3. Continuous evolution, adaptation of closed loops and services, based on evolving slice-customer requirements.

**3. Approach taken by the team**

* Step-1: Literature survey: [JICTS-IBCL] [TOSCA Comparison] [FGAN-I-126] an TOSCA YAML (version 1.3) specification.
* Step-2: Study of opensource alternatives for orchestrator [xopera]
* Step-3: Derived the requirements for a simple controller
* Step-4: arrived at key questions to be answered

what are the attribute values of "Controllers"?

- src node and its attributes

- model and its attributes

- etc

* Step-5: how and what actions can be operated on the service models?

- creation, configuration, and deployment

* Step-6: investigate the features which can enable the steps in Controller lifecycle

- substitution

- node-filter.

* All of these were done under the guidance of our mentors in weekly meetings.

**4. High-level flow chart**

Figure 1 is a high-level flow chart of the activity, starting with the design of the controllers. An intent is written according to the design of the controllers. This high-level intent is parsed, and appropriate closed loops are set up to meet the goal of the intent.

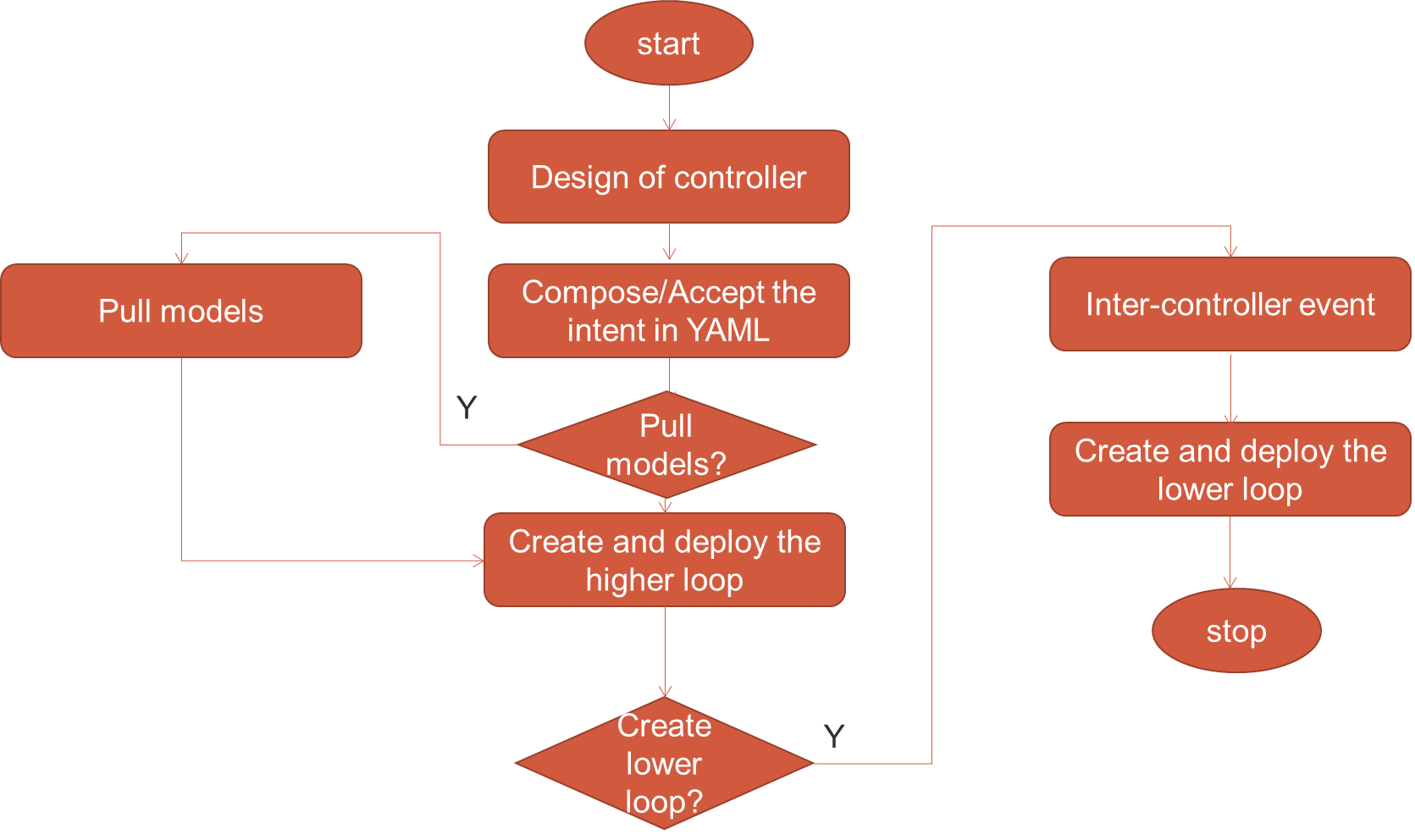
****

Figure 1: High-level flow chart of the activity.

The setup considered in our demo is shown in Figure 2. The intent is written in TOSCA YAML v1.3. The intent is to create a three-node closed loop comprising of a source, model, and sink nodes (corresponding to data collection, analysis and application). This intent is parsed by the xopera orchestrator for the deployment of the closed loop.

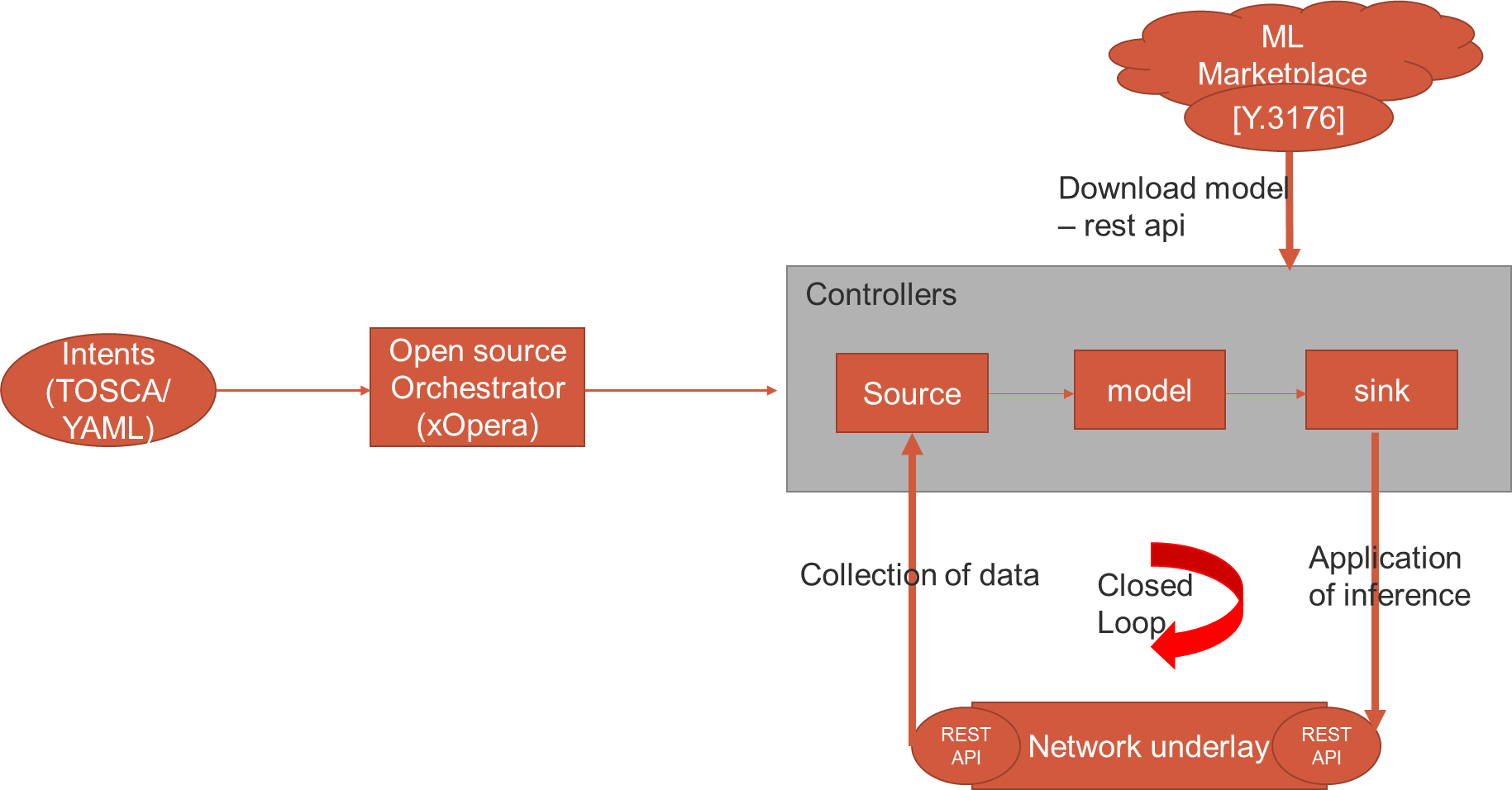


Figure 2: Setup design for creation & parsing of three Node Intent

**5.0 PoC: results**

An excerpt from the service model showing the definition of the model node is shown in Figure 3.0. The intent specifies the model node with attributes including the URL for pulling the ML model from a repository. Additional attributes like catalog ID, revision ID and solution ID may be used to identify the model.

**NOTE-** The additional attributes were added as per inputs from other teams. Further customizations may be added later.

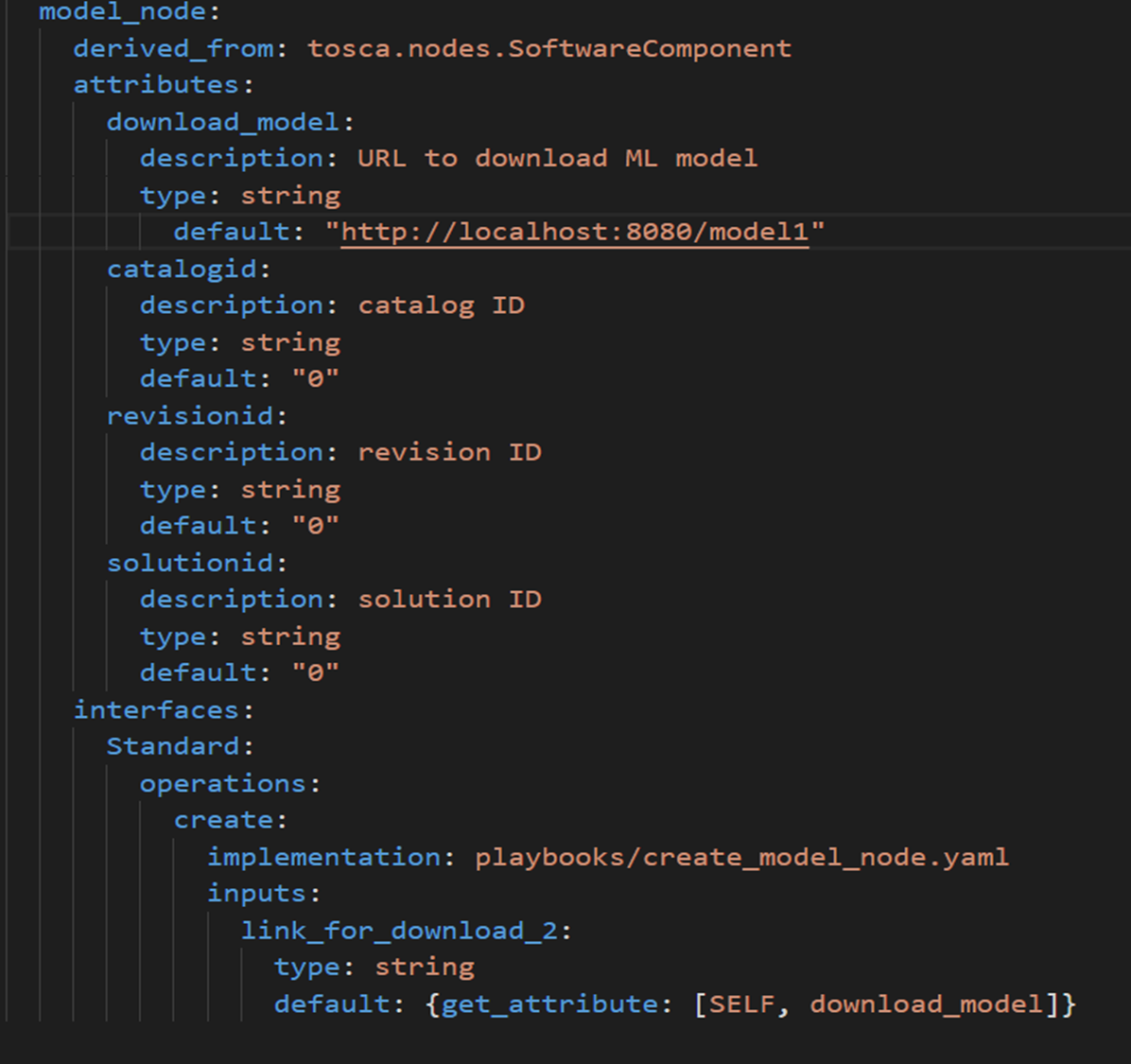
****

Figure 3:Creation & Parsing of Intent in YAML, [Model node] Declarative Intent in YAML format.

The outputs specified for the three nodes (source, model and sink) are shown in Figure 4.0. The outputs specified are the attributes of the nodes, which consist of the APIs.

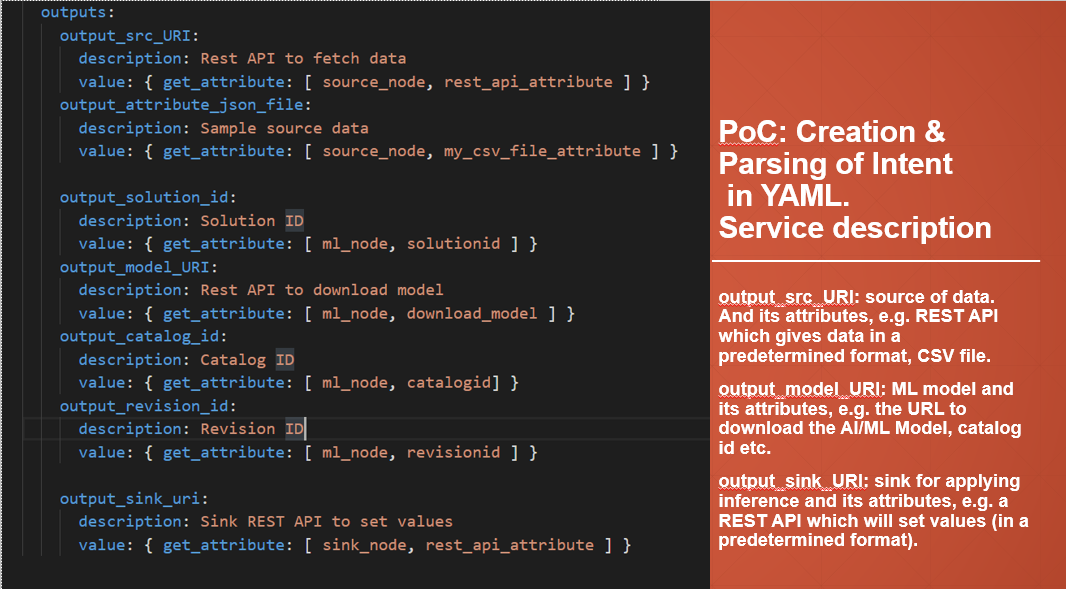


Figure 4: Creation and parsing of Intent in YAML

**5.1 Description of APIs.**

API Name: Source API

API Description: Returns current network velocity, volume and variety.

Direction: Network underlay to the controller

TABLE 1: Source API Description

|  |  |  |  |
| --- | --- | --- | --- |
| **Information Element** | **Type** | **Mandatory/Optional/Conditional** | **Description** |
| Data format | String | Optional | Specifies the data format for the network data. |

API Name: Model API

API Description: Returns specified ML model for analysis.

Direction: ML marketplace to the controller

TABLE 2: Model API Description

|  |  |  |  |
| --- | --- | --- | --- |
| **Information Element** | **Type** | **Mandatory/Optional/Conditional** | **Description** |
| CatalogID | String | Optional | Catalog ID specification of the ML model. |
| RevisionID | String | Optional | Revision number specification of the ML model. |
| SolutionID | String | Optional | Solution specification of the ML model. |

API Name: Sink API

API Description: Returns specified ML model for analysis.

Direction: controller to network underlay

TABLE 3: Sink API Description

|  |  |  |  |
| --- | --- | --- | --- |
| **Information Element** | **Type** | **Mandatory/Optional/Conditional** | **Description** |
| Network parameters | JSON | Mandatory | Network configuration parameters |

The parsing of the service model for the deployment of the three nodes is shown in Figure 5.0. The APIs in the intents are parsed into three JSON files. Three docker containers are created which use the APIs for data collection, analysis, and adaptation. A dummy data (based on the 3Vs (Velocity, Variety, and Volume)) and dummy h5 model are downloaded from corresponding repositories according to the specified URLs.

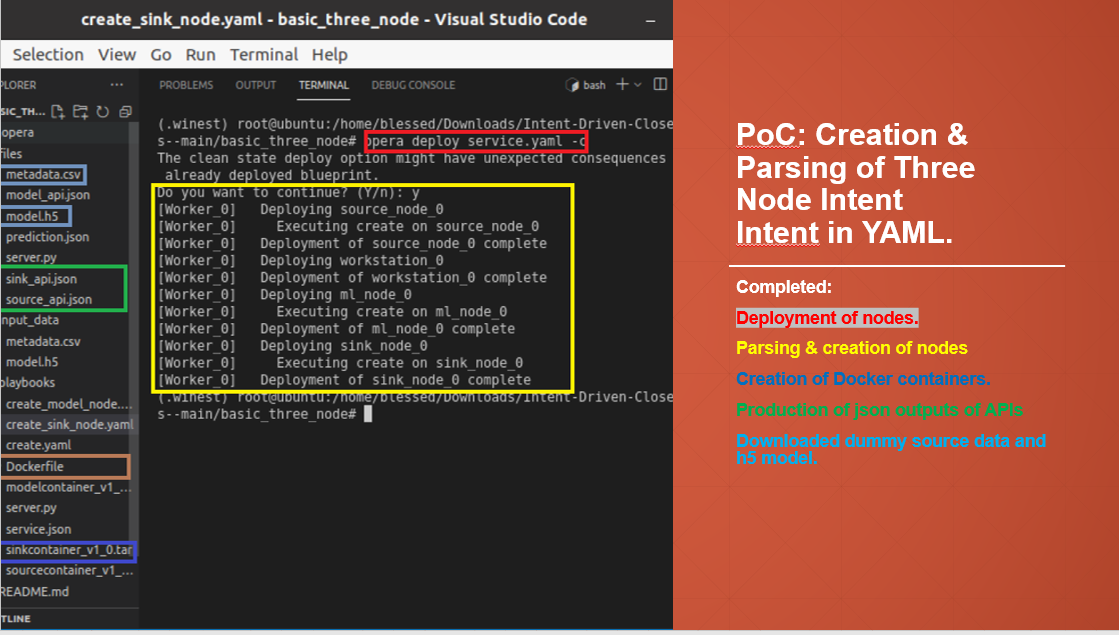


Figure 5: Deployment of the nodes

**5.1 Problems encountered.**

1. Abstraction of nodes allows the service template to select concrete nodes that best match the requirements of the abstract nodes during deployment. The concrete nodes can be provided in a repository known to the orchestrator. Abstract requirements can be achieved in TOSCA YAML using the ***node\_filter*** feature.

However, we found that this feature is not supported by xopera.

Similarly, we found that the substitution feature is also not supported by xopera.

This is shown in Figure 6

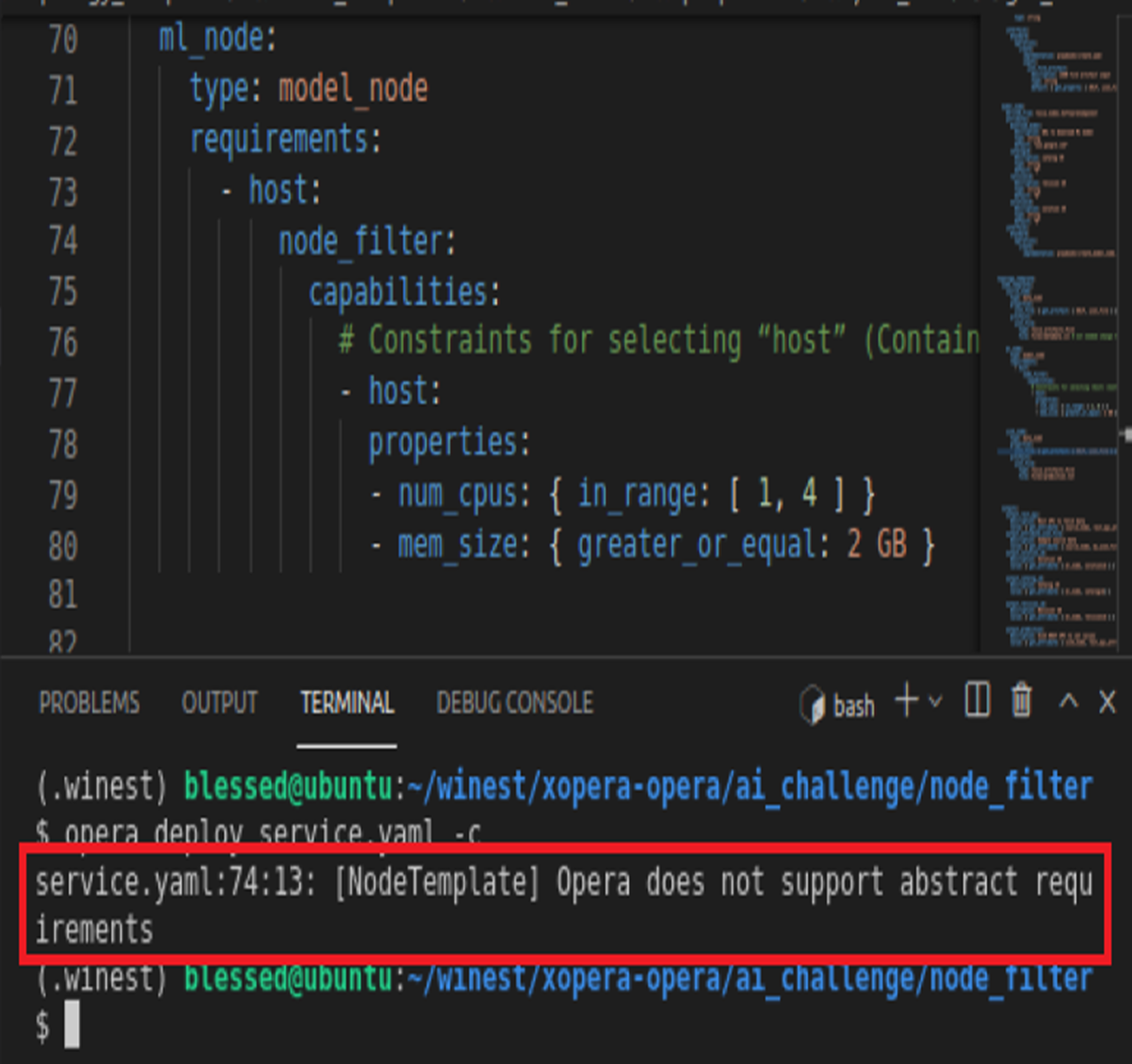


Figure 6: problems encountered with the node\_filter feature.

This shows that not all TOSCA features are supported by xopera. Table 1 shows a list of other TOSCA compliant open-source orchestrators.

Table 1.0: Open Source Orchestrators

|  |  |  |
| --- | --- | --- |
| **Orchestrator** | **Programming Language** | **Supported TOSCA version** |
| xOpera | Python | 1.3 |
| Yorc | Go | 1.1 |
| Tacker | Python | 1.1 |
| Unfuri | Python | 1.3 |
| Torch | Java | 1.0 |
| Turandot | Go | Unspecified |
| MSO4C | Python | Unspecified |
| Indigo | Java | 1.0 |

**NOTE-** However, according to our studies, a feature-based comparison of these orchestrators has not been done so far. This has been pointed out in our interactions with xopera.

**6. Future activities**

* Integration with the other teams to complete the whole pipeline of activities.
  + - json files are provided to team “automato” – [github]
    - Data pulling, model pulling and adaptation (via SINK) can be demonstrated after integration.

Management of containers hosting the nodes.

* + - Instantiation of container with Src node, model and SINK respectively.

Including more attributes in the nodes

* + - Data parameters in the SRC [e.g. 3xVs: velocity, variety and volume]
    - Model metadata [e.g. as per Y.3176]

SINK parameters [e.g. underlay specific APIs]

An extensive feature-based comparison of orchestrators may be useful to the community and may be contributed by the team.

**7. REFERENCES**

* [Build-a-thon FG AN] ITU-T FG AN-I-114-R1 “Proposal for a “Build-a-thon” for ITU AI/ML in 5G Challenge (second edition, 2021), aligned with FGAN WG3” <https://extranet.itu.int/sites/itu-t/focusgroups/an/input/FGAN-I-114-R1.docx>
* [Build-a-thon Challenge] ITU-T AI/ML in 5G Challenge problem statement “ITU-ML5G-PS-014: Build-a-thon(PoC) Network resource allocation for emergency management based on closed loop analysis” <https://challenge.aiforgood.itu.int/match/matchitem/45>
* [ETSI ZSM 001] “Zero-touch network and Service Management (ZSM); Requirements based on documented scenarios”
* [ETSI ZSM 002] “Zero-touch network and Service Management (ZSM); Reference Architecture”
* [ETSI ZSM 009-1] “Zero-touch network and Service Management (ZSM); Closed-Loop Automation; Part 1: Enablers”
* [ETSI ZSM 009-2] “Zero-Touch Network and Service Management (ZSM); Closed-loop automation; Solutions”
* [FGAN-I-126] ITU-T FG AN-I-126 “Standards gap analysis for AN”
* [FG AN use cases] ITU-T FG AN-I-113-R2 Working document for use cases for Autonomous Networks
* [FG AN arch] ITU-T FG AN-I-115-R1 Proposed initial draft of Architecture Framework for Autonomous Networks
* [github] [Full code and steps to run] available from <https://github.com/gblessed/Intent-Driven-Closed-Loops->
* [JICTS-IBCL] Gomes et al, “Intent-driven Closed Loops for Autonomous Networks”, available from https://journals.riverpublishers.com/index.php/JICTS/article/view/5829
* [TOSCA Comparison] Luzar A., Stanovnik S., Cankar M. (2020) Examination and Comparison of TOSCA Orchestration Tools. In: Muccini H. et al. (eds) Software Architecture. ECSA 2020. Communications in Computer and Information Science, vol 1269. Springer, Cham. <https://doi.org/10.1007/978-3-030-59155-7_19>
* [xopera] <https://github.com/xlab-si/xopera-opera> , <https://xlab-si.github.io/xopera-docs/introduction.html> ,
* [TOSCA Landscape] https://github.com/philippemerle/tosca-implementation-landscape

**Appendix 1: Steps to repeat the demo**

1. Clone the repo https://github.com/gblessed/Intent-Driven-Closed-Loops-
2. Install xopera [https://github.com/xlab-si/xopera-opera], docker for ubuntu[https://docs.docker.com/engine/install/ubuntu/] and docker module in python3 [*pip install docker*]
3. Change all directory paths in the *create.yaml, create-model-node.yaml and create-sink-node.yaml* files to the correct paths in your storage location. [see Figure I]
4. Login as root user [sudo -s]
5. Activate the virtual enviroment where the xopera is installed
6. Deploy service template: *opera deploy service.yaml -c* [see Figure II]
7. To view outputs: *opera outputs* [see Figure III]
8. End the deployment: *opera undeploy*

Text

Description automatically generated

Figure I: modification of directory paths.

Text

Description automatically generated

Figure II: screenshot of successful deployment

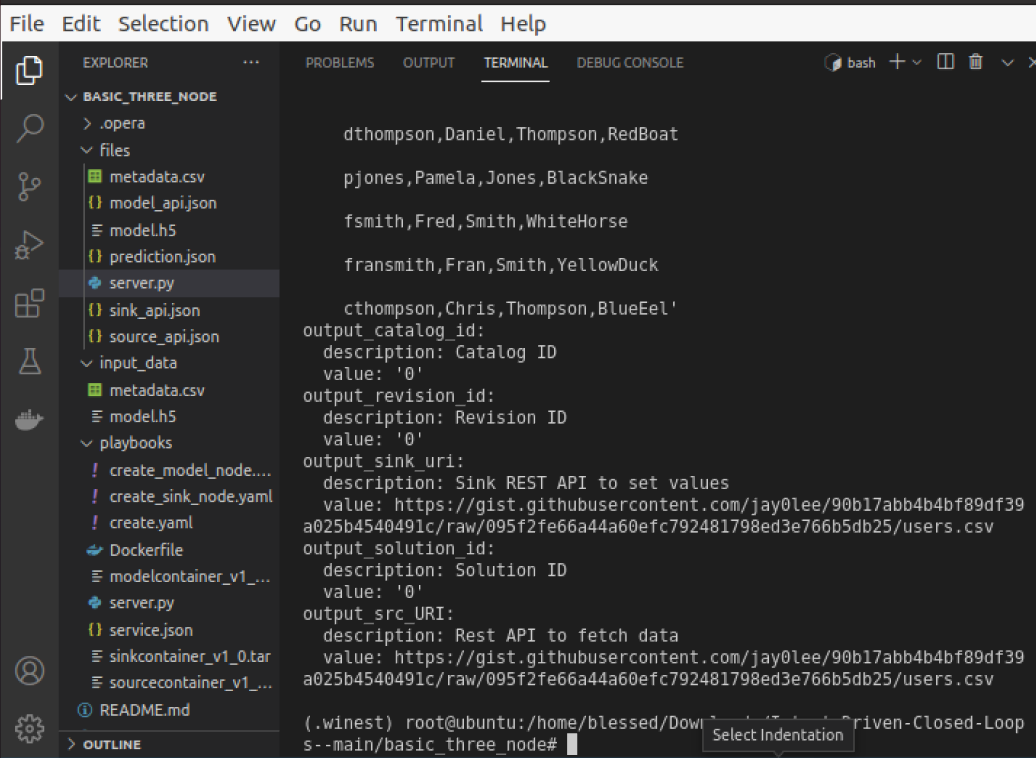


Figure III: outputs of deployed nodes

**Appendix 2: Questions asked by judges during FG-AN round**

1. What are the abstract languages and expressions used in addition to TOSCA?

2. Do a comparison with other languages – ONAP/SDL, FRINX/workflows (+GUI).

3. How to incorporate feedback from the underlay NFs in the process of design? How to come up with new “recipes”?

(1) top-down (from the orchestrator to the underlay): design by GUI, TOSCA editor - considering the feedback and resource status from the underlay. (2) bottom-up: analysis-based recommendation of new closed loops (e.g. from underlay networks to the orchestrator).

4. Add a comparison with other tools or options other than xopera?