Architectural Implications of a Model Driven ONAP

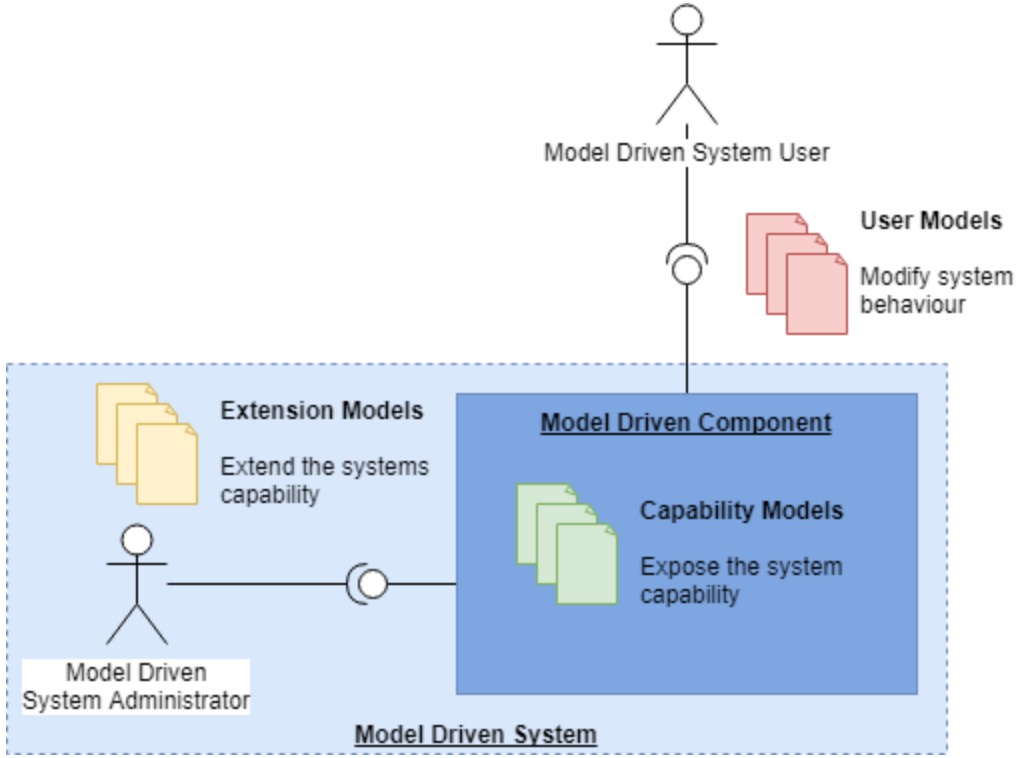
Aims

- Recap / expand on presentation in San Jose
  - From an architectural perspective, what do we mean by “model driven”?
- What level of “model-driven” do we want to achieve?
- Example use case
- Discussion

Model Driven Systems

Models can be defined at various stages in the lifecycle of a system

- Capability Models
  - Describe the capabilities and protocol of the interface exposed by the component / system
    - REST API
    - Event topic / queue
- Extension Models
  - Provide a means for system administrators to extend the behaviour of a component / system by adding models at runtime
    - Additional REST resources, event types
    - Are there examples of this in ONAP?
- User Models
  - Deployed by users of the system to modify the behaviour of the system
    - Service descriptors, YANG network resource models, blueprints



Capability Models and Interfaces

- Interfaces between services are contracts. Interfaces comprise of a number of aspects:
  - **Behavioural contract:** what happens when an operation on an API is invoked
  - **Protocol contract:** transport / communication protocol, physical encoding, security
  - **Model contract:** signature of the operations, data types etc supported by the API

Model Contract Concepts

Concept	Description	Example
Schema Language	Governs the syntax and semantics of elements and attributes	XSD, ASN.1, YANG, OpenAPI Specification, AAI Schema
Model	A specific model adhering to a language	RFCXXXX, Yang fragment, SOLXXX, AAI Model
Instance Data	Data which conforms to a model	Service Instance, configuration data

Capability models encompass the behavioural contract and the model contract

Contract Testing

Do you set your house on fire to test your smoke alarm? No, you test the contract it holds with your ears by using the testing button.

Contract testing is a way to ensure that services (such as an API provider and a client) can communicate with each other. Without contract testing, the only way to know that services can communicate is by using expensive and brittle integration tests.

A contract is between a *consumer* (for example, a client that wants to receive some data) and a *provider* (for example, an API on a server that provides the data the client needs).

Proposal:

- **pick a component and API and PoC this approach**
- **Define principles to apply across the board**

https://docs.pact.io/

In the Zoo App (consumer) project

1. Start with your model

Imagine a model class that looks something like this. The attributes for a Alligator live on a remote server, and will need to be retrieved by an HTTP call to the Animal Service.

```
class Alligator
  attr_reader :name

  def initialize name
    @name = name
  end

  def == other
    other.is_a?(Alligator) && other.name == name
  end
end
```

2. Create a skeleton Animal Service client class

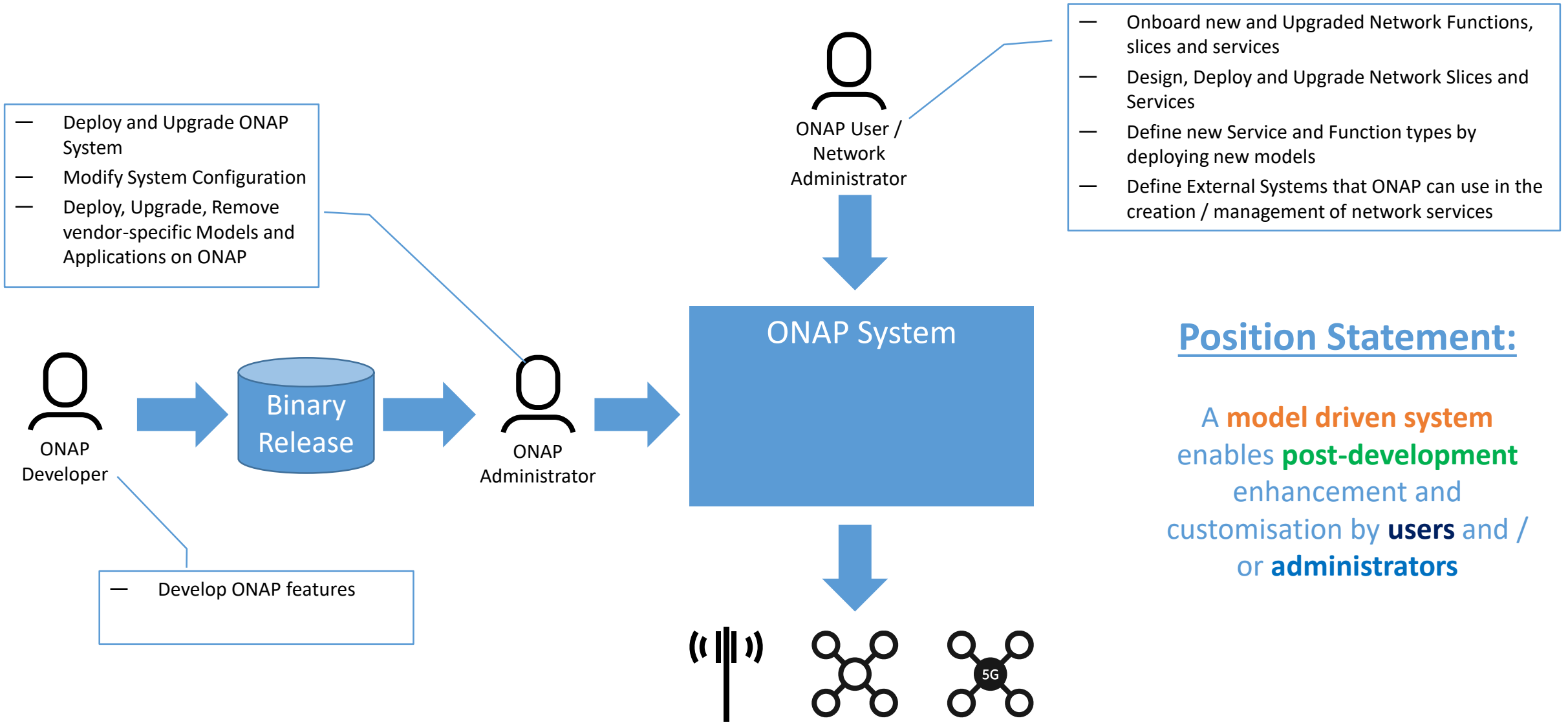
Perhaps we have an Animal Service client class that looks something like this (please excuse the use of httparty):

```
require 'httparty'

class AnimalServiceClient
  include HTTParty
  base_uri 'http://animal-service.com'

  def get_alligator
    # Yet to be implemented because we're doing Test First Development...
  end
end
```

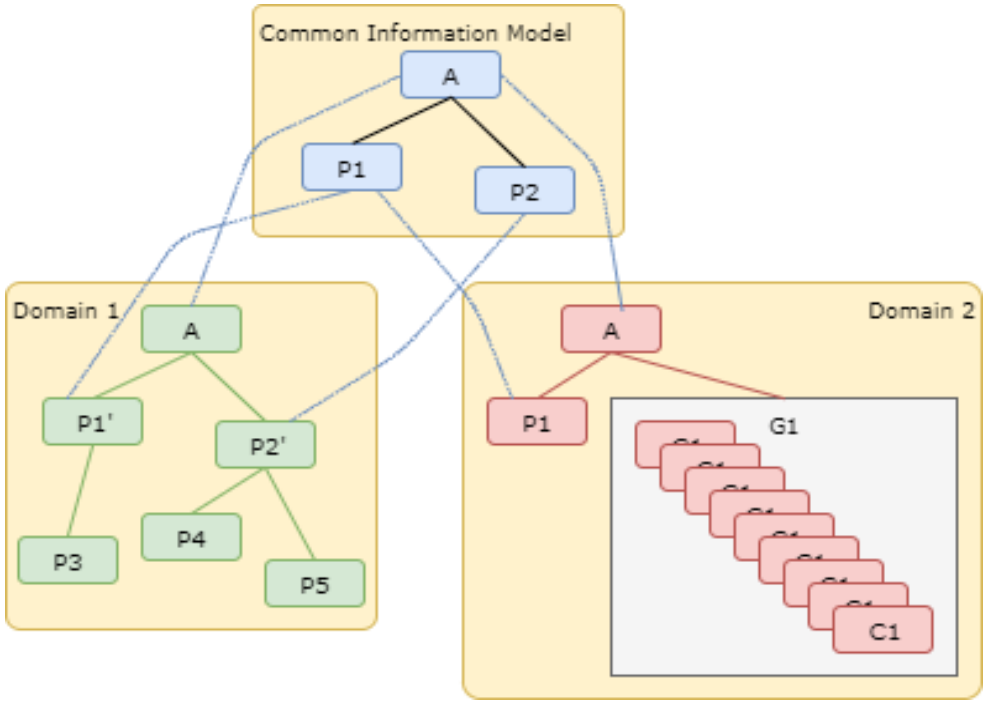
ONAP Actors



Position Statement:

A **model driven system** enables **post-development** enhancement and customisation by **users** and / or **administrators**

Domain Driven Design and Information Modelling



As the domain being modelled grows in scope and complexity it becomes progressively harder to align on a single unified model

Domain Driven Design divides a large system into Bounded Contexts, each of which can have its own unified model

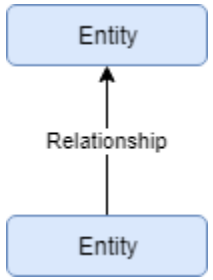
Each Bounded Context could have unrelated concepts but also related concepts such as a managed function or a service.

A Common Information Model defines the related concepts that tie the contexts together, while each context can evolve its own concepts independently

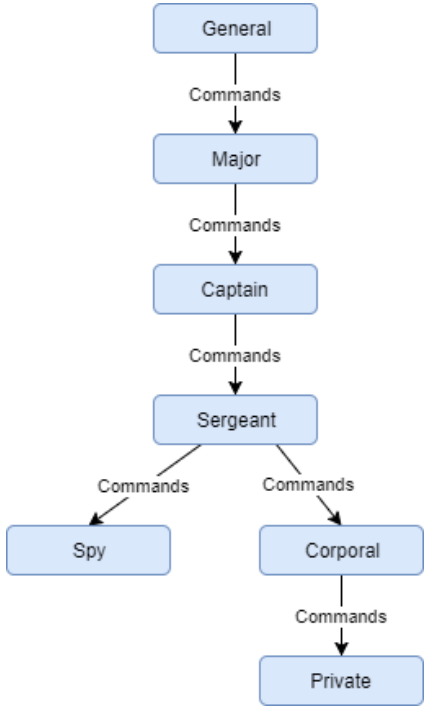
Each domain can select appropriate persistence technolog[y|ies] according to the characteristics of the data

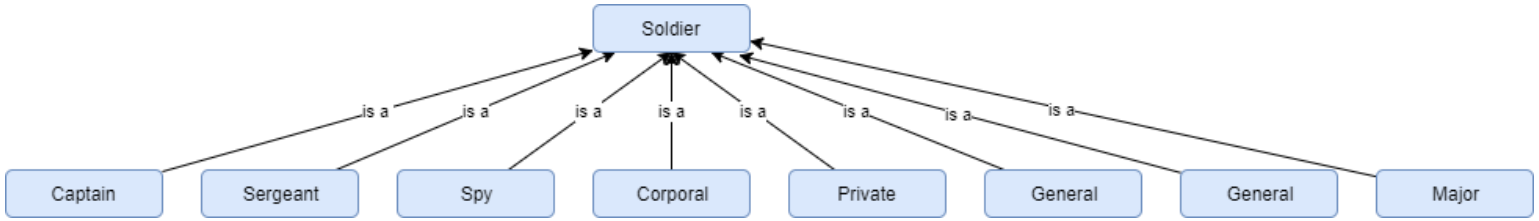
Modelling Principles

Generic Models

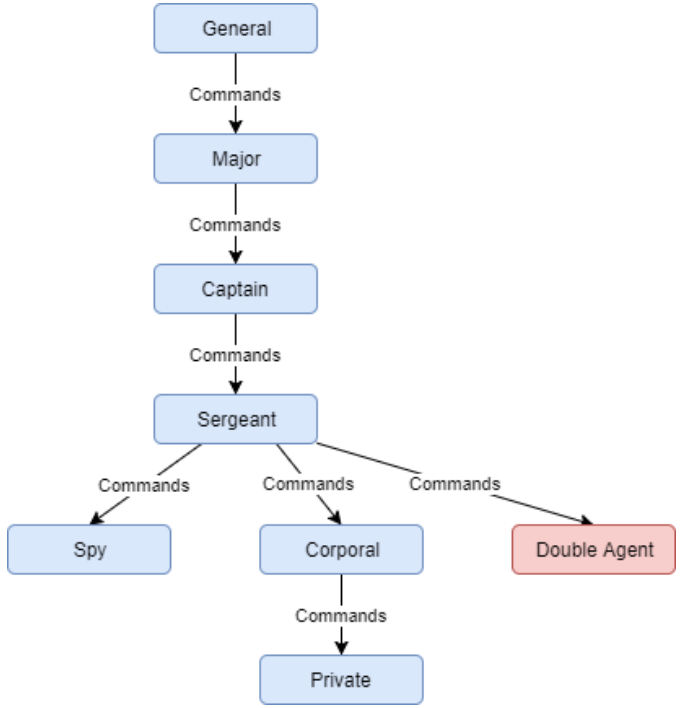


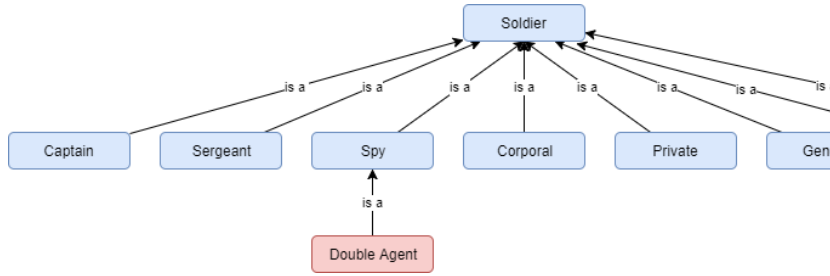
Domain Models





Extending Models





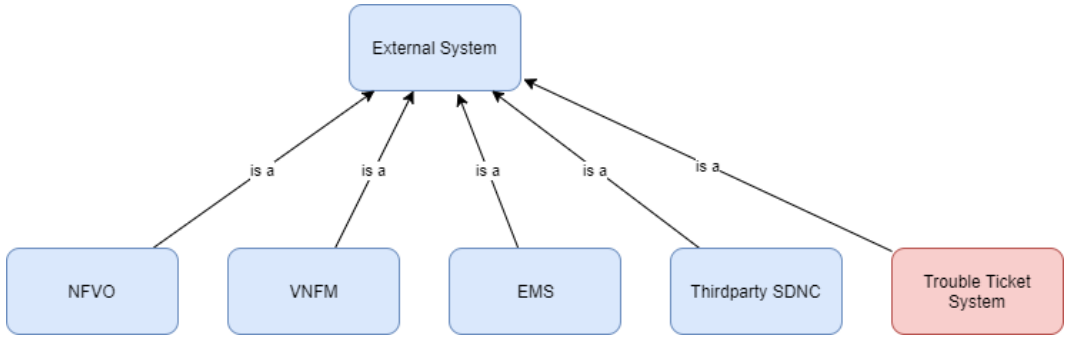
Model Driven Maturity Levels

- Level 0: not model driven
  - Core concepts are hard-coded everywhere, human-readable documentation and definitions exist to support developers
- Level 1: defined “data / information” model
  - Machine-readable common model that is shared / used by developers
- Level 2: extensible “data / information model”
  - Ability to deploy new models to modify the behaviour of the system at runtime

Model Extensibility – Use Case

- As an ONAP [User|Administrator] I want to create a Closed Loop with a policy based trigger to create a trouble ticket in the network operator's trouble ticketing system.
- To create the trouble ticket I need to represent the trouble ticketing system as an external system
- I also need to create a representation of the trouble ticket itself and relate this to the trouble ticketing system and the faulty NF
- Today this requires a code update to 2-3 files in AAI schema repository plus a rebuild
  - Is this the wanted position (dev-ops model)?
  - OR: do we define a generic type and allow "free-form" data to be interpreted by the consuming service?
  - OR: would it be preferable to manage / deploy these model(s) as separate artefact(s)

https://wiki.onap.org/display/DW/AAI+Tutorial-Making+and+Testing+a+Schema+Change+-+Dublin



LINUX FOUNDATION COLLABORATIVE PROJECTS

ONAP
OPEN NETWORK AUTOMATION PLATFORM

Code Review / aai / schema-service.git / commitdiff

summary | shortlog | log | commit | commitdiff | review | tree
raw | patch | inline | side by side (parent: 85a4c19)

Add NFVO external-system in AAI 92/82892/4

```
author     udhaya chandran <udhayachandran.m@verizon.com>
           Thu, 21 Mar 2019 08:20:13 +0200 (11:50 +0530)
committer  udhaya chandran <udhayachandran.m@verizon.com>
           Thu, 21 Mar 2019 16:49:23 +0200 (20:19 +0530)

Change-Id: I71601180d47fe3de71ddbc07a4c54db7b398f520
Signed-off-by: udhaya Chandran <udhayachandran.m@verizon.com>
Issue-ID: AAI-2205
```

aai-schema/src/main/resources/onap/aai_schema/aai_schema_v16.xsd	patch | blob | history
aai-schema/src/main/resources/onap/dbedgerules/v16/DbEdgeRules_esr_v16.json	patch | blob | history
aai-schema/src/main/resources/onap/oxm/v16/aai_oxm_v16.xml	patch | blob | history

Next Steps …

- Receive feedback
- Align and formalise the language and concepts
- Amend the architecture principles to cover model driven aspects
- Investigate what additional information we need to capture in the ONAP architecture description
- Investigate what changes are required to ONAP to achieve our vision over time