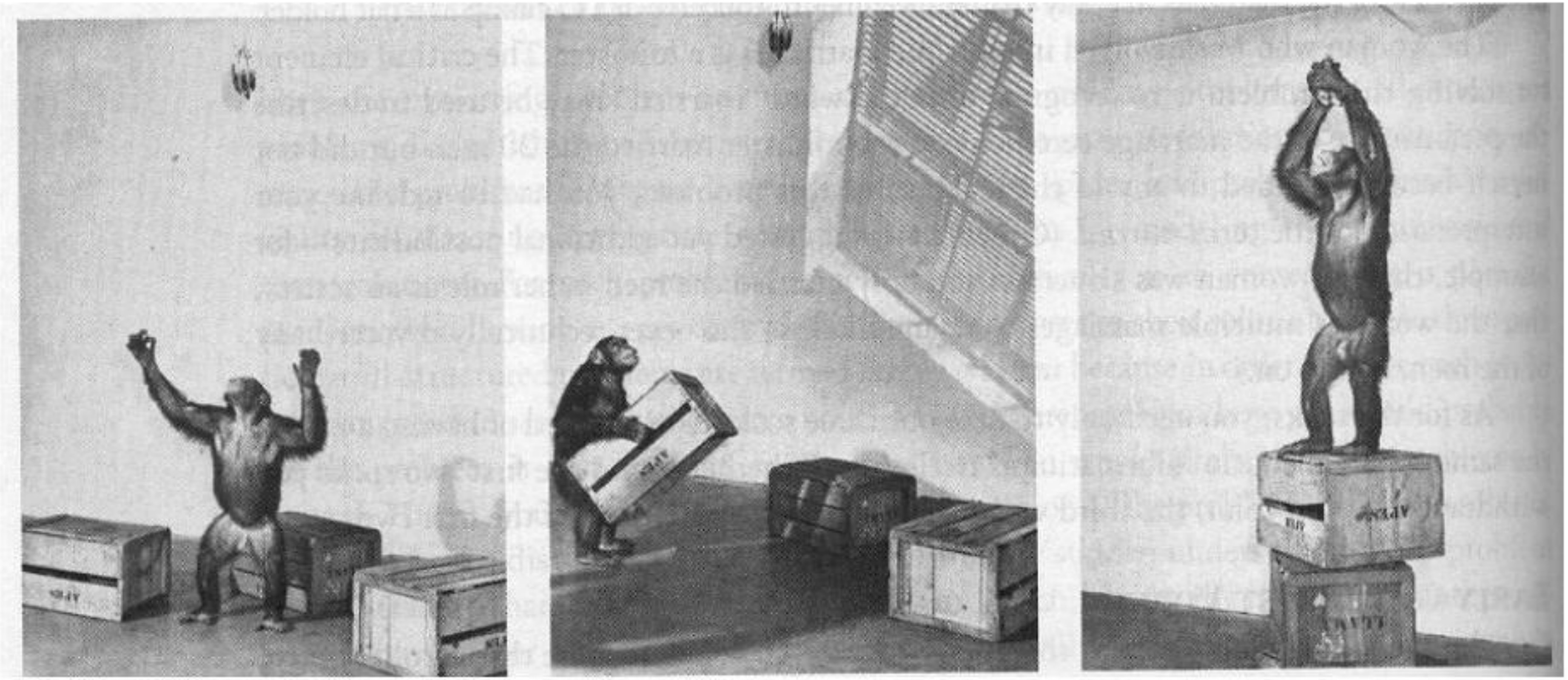
Anuket – Modelling and AI for Reference Models / Architecture



By: John Hartley (Anuket – Reference Model Contributor)

Ambitions

Discussion of Anuket / CNTT Ambition:

- Can we specify Reference Model / Architecture Formally ?
- Can we generate architecture design from specification ?

How could this be achieved ?

- For specification of Model / Architecture need a specification language
- For generation of design need “Network Designer”

What existing assets are available ?

- Existing languages / tools tend to be to low level (NetConf – the how rather than the what), provided a set of graphical conventions (ITU G.805), did not provide semantic constraints (general graphs & graph databases), Canonical JuJu does not provide sufficient Interface richness …
- Can represent a design as well as specify inputs and constraints into a design
- Did you lend themselves to AI / constraint based solutioning techniques

How hard is this ?

- Is problem tractable (ie not killed by combinatorial explosion)
- Test using a prototype

For specification of Model / Architecture need a specification language

Start with basic concepts:

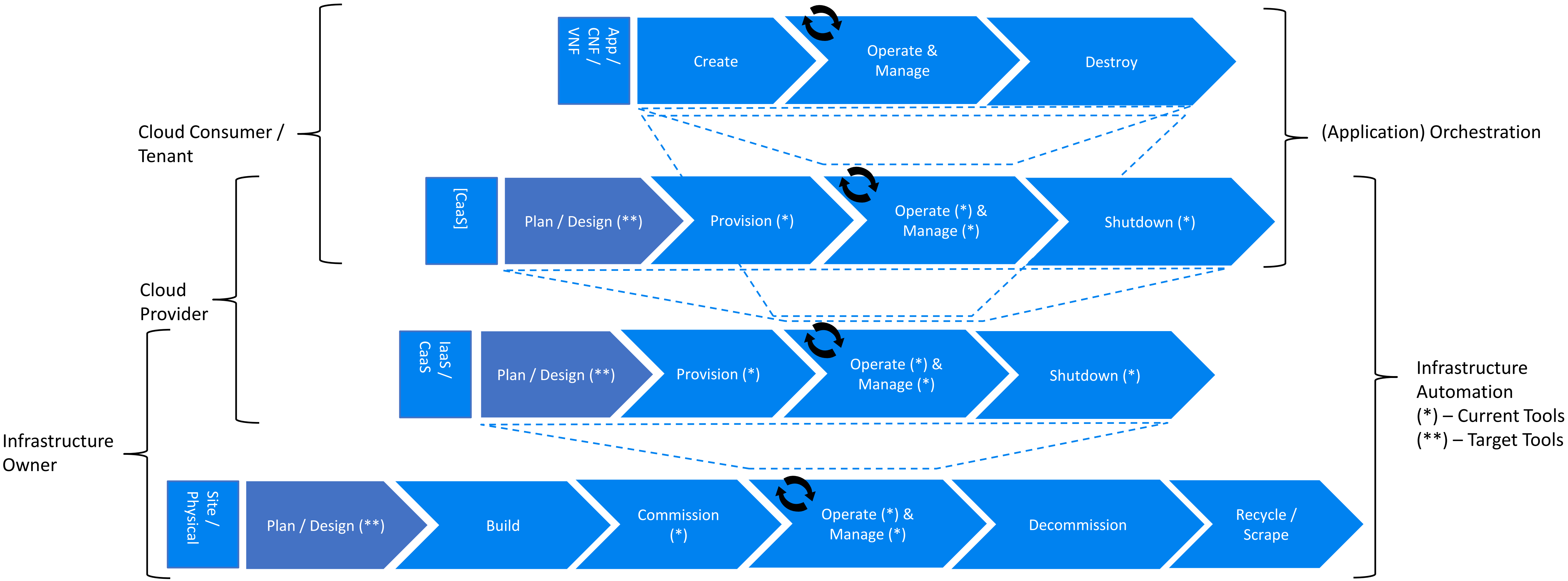
- Network – A Collection of Network Elements and Links
- Network Element – a node within graph
- Link – a graph edge, which connects the “Network Elements”
- Path – a sequence of “Network Elements” and “Links”, which could be standalone network or path through an existing network
- Interface – the specification for connecting the “Network Elements” and “Links “

Must support concept of layering and aggregation and exposure of network service:

- Physical Link -> Ethernet -> IP Network
- LAG group – made of a collection interfaces (could operate over distinct “Network Elements”
- Ethernet -> Ethernet VLAN

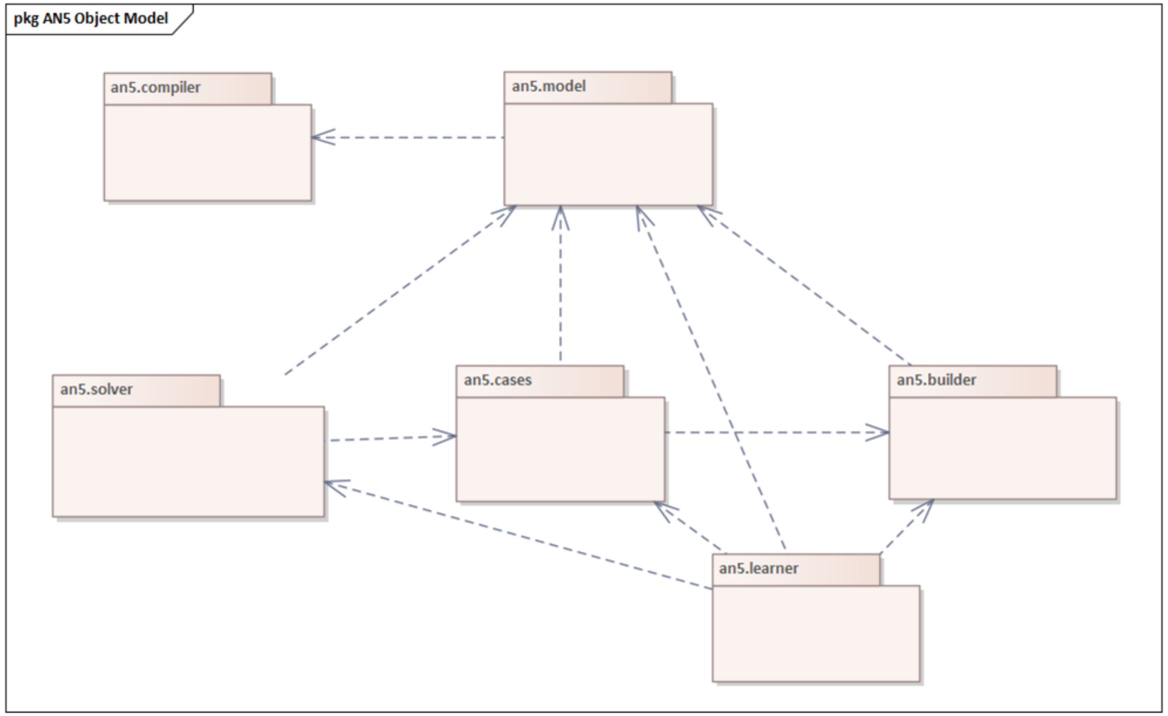
So ended up with an5 – **a n**ETWORK LANGUAGE WITH **5** CLASSES

What is the target for building specification ?

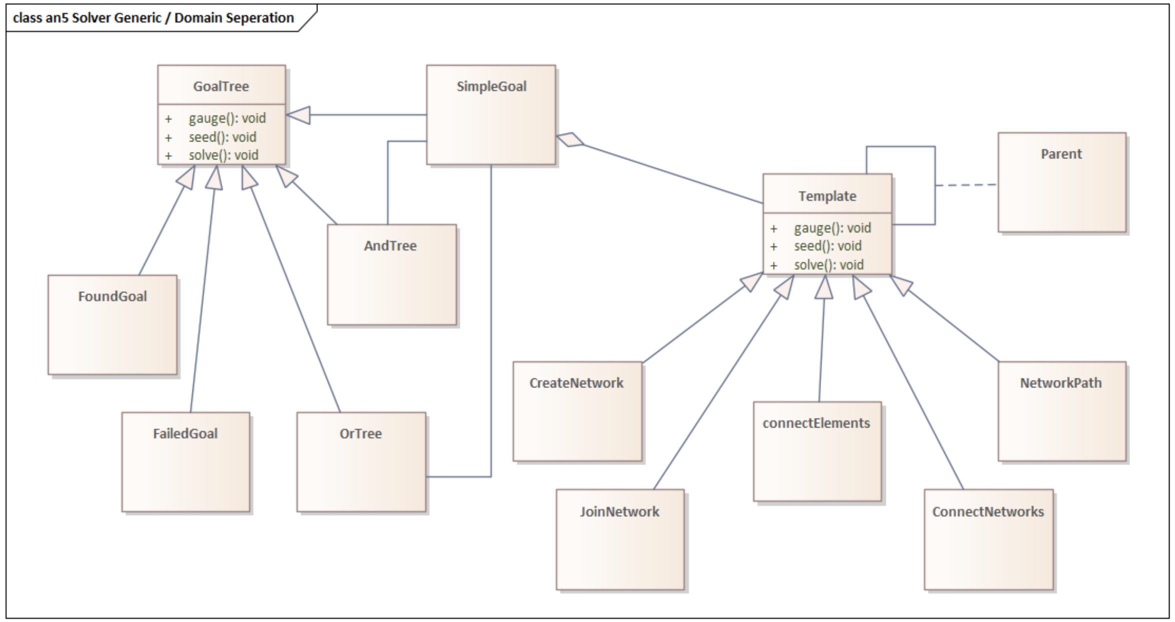


How hard is this ?

- Have a Language => Need a Complier
- Designer A Network n=> Need an AI Network Designer Engine
- Should not encode the semantics of particular network types
- Should be able to leverage “knowledge” of a “good” design

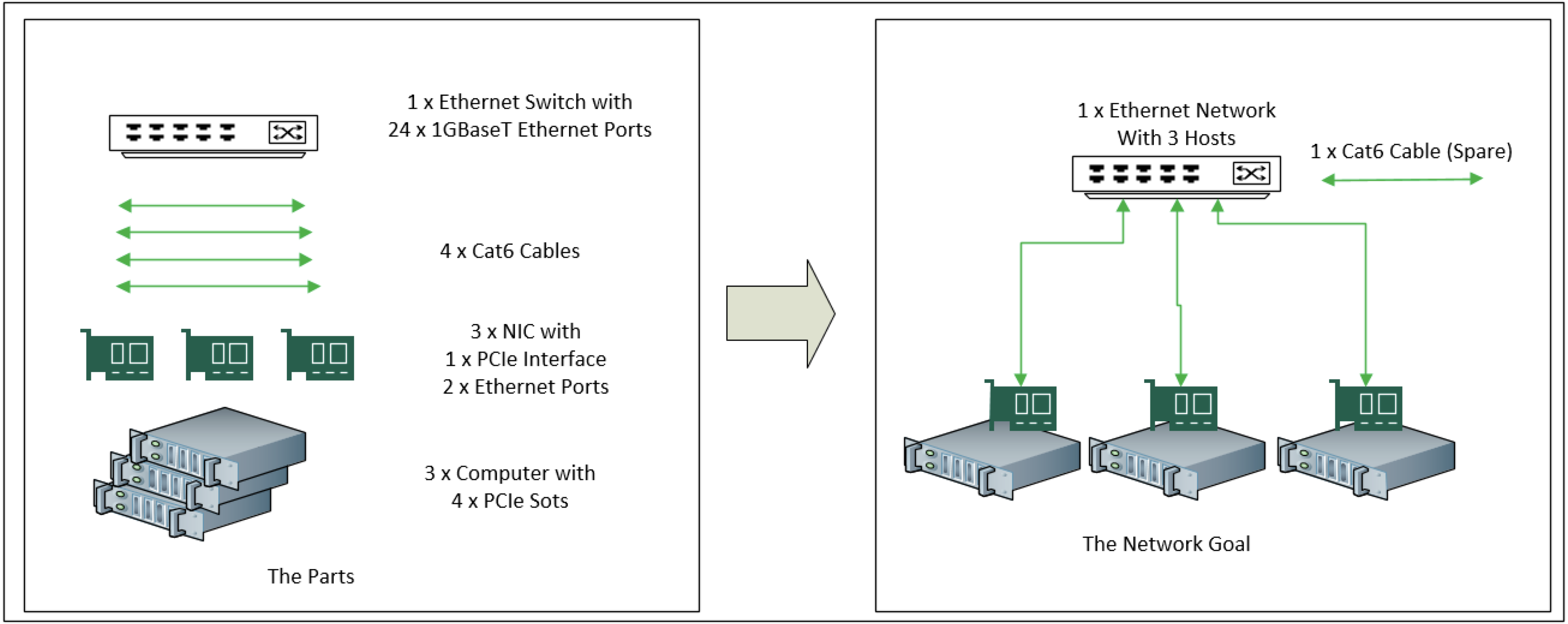


Separate the general AI solver from the network problem domain



- All network semantics are defined in an5 (with exception of Interface compatibility)
- Problem Solver uses general AND/OR tree solver with selectable methods:
  - Depth First, Breadth First, Bound, Score, Cost, A* …
- Solution Generators:
  - Create Network, Join Network, Connect Elements, Connect Networks, Network Path

Example Problem – for Prototype and Measure



Relationship to original "STRIPS" Constraint Solver is apparent, but Generator is significantly more complex…

The “AI” part

Specify:

- Inputs
  - A set of components:
    - Network
    - Network Element
    - Link
    - Path
    - Interface (Common, Needs, Provides)
  - An Initial Solver (Intent Handler)
    - ConnectElements - connect elements within network
    - CreateNetwork - create a new network from bucket of bits
    - JoinNetwork - add new elements into an existing network
    - NetworkPath - find path within an network
    - ConnectNetworks - join two network together
  - The network goal

The Engine:

- Search Algorithms
  - Non Domain Specific
  - Search / Solve Strategies
- Generate / Test
  - Domain Specific

  Domain Heuristics
  - Domain Heuristics
  - Domain Knowledge

The outcomes are surprising…

- So we need to constrain the intent to ensure it drives to our objective

Specifying the Goal/Intent ...
New Syntax makes Intent Clearer

Old Syntax (V0.1):

```
abstract class ethernet_lan extends network {
  @mandatory switch[] fabric;
  @mandatory computer[] hosts;
  object[] uses;
  service = { "ethernet", "(ethernet_vlan)*"};
}

abstract class ethernet_node extends element {
  @mandatory computer host;
  @mandatory switch ether;
  object[] uses;
 }
```

Intent Syntax (V0.2):

```
goal class ethernet_lan extends network {
  @mandatory switch[] fabric;
  @mandatory computer[] hosts;
  object[] uses;
  service = { "ethernet", "(ethernet_vlan)*" };
  handler = “an5.solve.an5.CreateNework”

  constraint class ethernet_node extends element {
    @mandatory computer host;
    @mandatory switch ether;
    object[] uses;
   }
}
```

an5 – components example …

```
interface pcie_interface {
  common = { "type=pcie", "width=(1|4|8|16)", "gen=([1-4]\\.*)"};
}

interface pcie_slot extends pcie_interface {
  needs = {"form=(card)?"};
  provides = {"form=slot"};
  binding = "slot-%I+1";
  string name;
}

class computer extends element exposes pcie_slot {
  reflects pcie_slot[] slot;
  string name;
}

interface rj45_plug {
  common = { "plug=rj45"};
}
```

```
interface base_t_sink extends rj45_plug {
  needs = { "cable=(cat([3-8].?))?",
          "gender=male", "media=copper"};
  provides = {"plug=rj45", "gender=female"};
}

interface ethernet_port_base_t extends base_t_sink {
  common = { "service=ethernet"};
  binding = "p-%I+1";
  string name,
       MAC;
}

interface ethernet_lag {
  common = { "service=(ethernet)+" }
  binding = "lag-%I";
  string name;
}
```

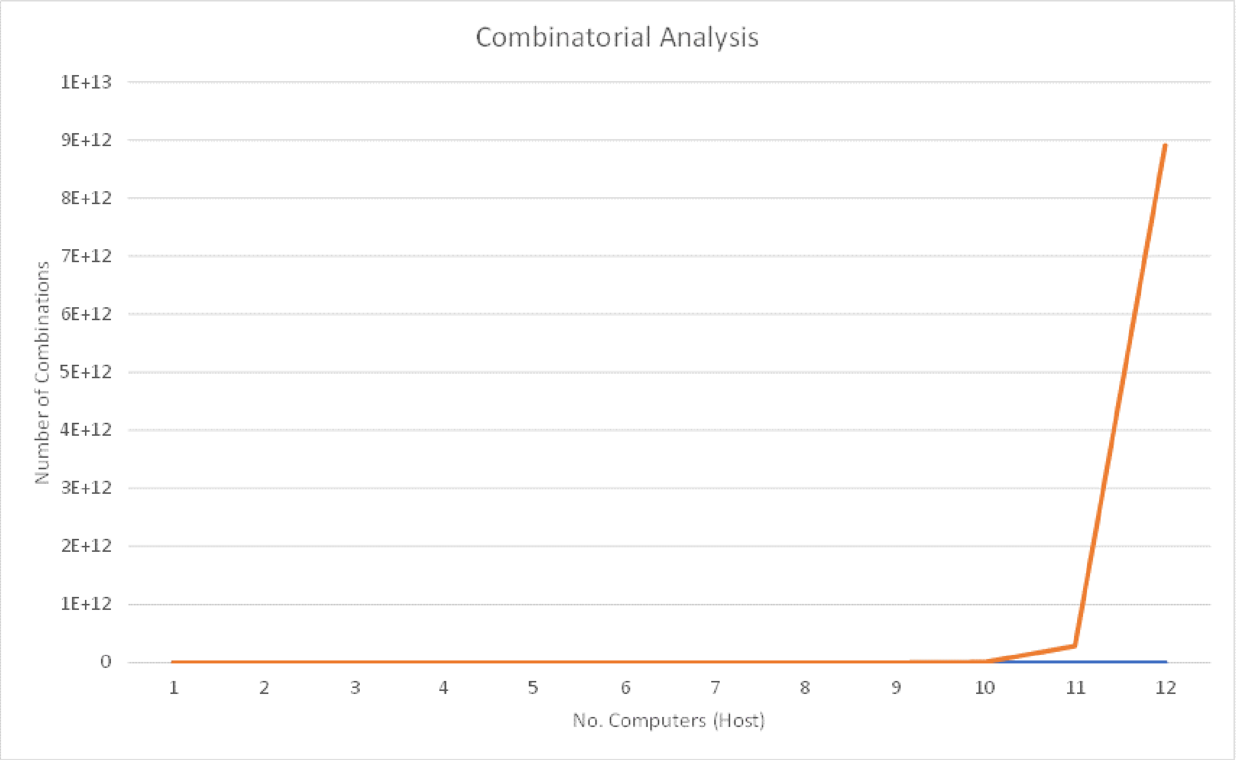
```
interface ethernet_vlan {
  common = { "service=(ethernet_vlan){0,4096}" }
  needs = { "service=ethernet" };
  binding = "vlan-%I";
  string name;
}

class ethernet_lag_link extends link exposes ethernet_lag {
  reflects ethernet_port_base_t[] ports;
}

class ethernet_vlan_link extends link exposes ethernet_lag,
ethernet_vlan {
  reflects ethernet_lag lags[];
  reflects ethernet_port_base_t ports[];
}

interface pcie_card extends pcie_interface {
  needs = {"form=(slot)?"};
  provides = {"form=card"};
}
```

https://github.com/zebity/an5/blob/main/src/main/models/dc-bb.an5

Tractability - Combinatorial Analysis and Optimisation

Constraining Search Space:

- Generic Search Control - applicable to any search space problem and operate by providing alternate search algorithms including: Depth First Search, Breadth First Search, Branch and Bound, Score & Cost based Search
- Domain Optimised - where the search optimisation if based on applying some pruning based on the problem domain of the search
- Case Based - where search is optimised based on historical or statistical analysis of paths which have most likelihood of provide preferred or right solution. These are ones which leverage machine learning techniques



Intractable with blind search, beyond 8 hosts..

Domain Based Local Optimisation

- In observing the behaviour it was apparent that the AI Solver was doing things that would not be required in real world. In particular the generator was doing a great job of building all the combination of ways that you could plug the various compatible interfaces together. So for 1 Computer it would generate every next solution combination set of:
- C(h,n) = (h1-n1, h1-n2, h1-n3 ... h1-nX, h2-n1, h2-n2, h3-n3 ... h2-nX, ... hY-nX) where:
- C(h,n) - set of combination of hosts (h) and NICs (n) consisting of pairs:
- h1-n1 .. - host #1 with NIC #1, host #1 with NIC #2 etc
- In additional it was also generating combinations of multiple NIC cards within single host (as each host was defined as having 4 slots).
- In practice in the real world NICs and cables are treated as "commodity" devices and the result of assembling a path using any particular instance of an object of the same class of will result in the same outcome, irrespective of which instance I use.
- In simple terms this means that if I have 10 Cat6 cables available to plug from a NIC port to a switch port then there is no advantage to generating all the 10 NIC port to cable to switch port combinations, as the result be the same for each of the very large number of combinations. So instead you can just generate the first instance and make the cable un-available for further combinations.
- To test this I created a "Local Equivalent Removal" option within the Domain solver layer. The algorithm was "Local" as it operated solely within scope of a given interface bind request and removed cases of binds which have class template equivalency.
- If you do not prune the "Local Equivalents" the result is a huge number of essentially identical search trees.
- Practically the use of "Local Equivalent Removal" was to stop the combinatorial explosion in its tracks with each new depth step now having breadth of 1. So the Domain optimisation results in a reduction from 10 to power of 12 (=power(10,12) in Excel) to 1

Summary – Domain Based Optimisation make problem tractable

Anuket - Applicability

- Having Formal Specification would be valuable as it would lent itself to automatic generation of reference solutions (architectures)

- As there are different constraints and inputs, the solutions will vary based on the resources available

For more detailed information, please see my blog on this work …

https://www.linkedin.com/posts/john-hartley-28070421_an5-intelligent-network-design-activity-6792244208259485696-Dq4W?utm_source=share&utm_medium=member_desktop

Read at your leisure…