EnterpriseWeb®

Learnings from "Dynamic APIs" Catalyst, TMF Live 2015

William Malyk, Chief System Architect

July 2015

1. Introduction

Our work on the Dynamic APIs Catalyst involved a number of TMForum assets. In particular, we used assets from Frameworx including the Digital REST APIs, TR 244 Information Framework Enhancements to Support ZOOM, TR 238 MANO Interface Requirements and the TR 235 Policy Architecture and Information model. The use-case below outlines their specific utilizations in our catalyst.

Use Case – Cross Ecosystem/Domain Interoperability

1. BSS (Comverse) exposes product billing policies via Open Digital REST API*
2. MANO (CloudNFV) exposes catalog of product specifications via Open Digital REST API*
    a. CloudNFV flexibly represents industry standards interfaces as entities in a graph-connected info model
    b. CloudNFV supports rapid on-boarding of VNFs & declarative composition of Network Services
    c. CloudNFV enables products to be declaratively composed from Network Services mapped to TR 244 classes** / TR 238 interfaces*** / TR 235 models**** and linked to BSS (Comverse) billing policies
3. OSS (DGIT) fetches product specifications from MANO via Open Digital REST API*
4. OSS (DGIT) submits orders with customer/product details to MANO via Open Digital REST API*
5. MANO (CloudNFV) fulfills order, orchestrates Network Service, instantiate Virtual Network Functions
    a. CloudNFV associates product billing policy to customer with BSS through the TMF Open Digital REST API*
    b. CloudNFV instantiates related VNFs through TMF TR 244 classes** / TMF TR 238 interfaces***
    c. CloudNFV deploys DPI monitoring (Qosmos) to endpoints
    d. CloudNFV reports service state to OSS through the TMF Open Digital REST API (TMF622)*
6. MANO (CloudNFV) manages lifecycle operations (scale-up, down)
    a. CloudNFV monitors Network State directly through TMF TR 244 classes** / TMF TR 238 interfaces***
    b. CloudNFV balances Network Services as required through TMF TR 244 classes** / TMF TR 238 interfaces***
    c. CloudNFV reports billing event to BSS as they occur through the TMF Open Digital REST API*

In the following sections we outline our experience, feedback and recommendations against each of the 4 main assets used.

EnterpriseWeb®

2 Open Digital REST APIs*

We leveraged and extended with metadata (re: types, constraints, operations, etc.), to enable "Dynamic" APIs based on the "EnterpriseWeb: Management and Operations Platform Metamodel " previously contributed - http://www.tmforum.org/community/communities/frameworx/contributions.aspx#

2.1 Experience

The catalyst work focused mainly on APIs found in the Product Ordering section of the Digital REST APIs.

In the context of the catalyst, we needed to demonstrate standards based exchange between the EnterpriseWeb MANO system, the DGIT Ordering system (OSS), and the Converse Billing system (BSS).

For the OSS-Orchestrator communication, both EnterpriseWeb and DGIT were able to implement the Product Ordering APIs, fetched from the API Hub on the TMForum website, and work independently on both sides of the interface. We were able to successfully make exchanges based solely on this work, without having to engage directly except to exchange security credentials and endpoint information.

This sufficed for standard system-to-system exchange, however it did not support product reconfiguration in a dynamic environment where partner Virtual Functions might be added, updated, upgraded or replaced. We found it necessary to extend the data model of the Open Digital REST APIs in order to enable flexible metadata exchange.

For the BSS-Orchestrator communication, our partner Comverse indicated there was no existing BSS/Billing equivalent to the Product Ordering API. We implemented a Billing API following the Open Digital REST APIs semantics for exchange between Comverse and EnterpriseWeb. As with the OSS interfaces, we enabled extensible metadata for these new BSS interfaces.

Comverse and EnterpriseWeb agreed on a Catalog-based approach to identifying billing packages, and a package items based approach for updating specific parameters in the billing. For example, we were able to specify an "IMS billing profile", and then modify items such as "number of users" dynamically from within EnterpriseWeb in response to service lifecycle events.

EnterpriseWeb uses the Dynamic APIs to flexibly communicate real-time product and service information to constantly synchronize the ecosystem; improving the business agility of the end-to-end solution.

2.2 Feedback

Working with the Product Ordering / OSS APIs was straightforward. They were well documented, clear and concise making them easy to implement. We were able to translate the lifecycles and payload specs (data-models) directly into system entities in our declarative modeling environment in order to automate the API exchange.

In general, working through the Open Digital REST APIs we observed a general lack of extensibility in these interfaces. This posed a problem in regards to managing lower-level details that are not made explicit in the API.

This was a good opportunity to leverage other TMForum assets including the Metamodel contributed by EnterpriseWeb. Specifically, in regards to the Catalyst, we found that two areas were prime candidates for expansion: the productRelationship, and the productCharacterists elements of the exchange payloads.

2.3 Recommendations

The following refer explicitly to the Product Ordering APIs, but a general review of the APIs suggests that these statements apply across the full set of Open Digital REST APIs.

2.3.1 Include metadata for productCharacteristics.

The productCharateristics array is found universally across the API. It is meant to communication features in the products being ordered/instantiated. In specific the existing specification calls for these characteristics to be exchanged in the following format:

```
"productCharacteristic":[
  {
    "name":"anotherCharacteristic",
    "value":"itsValue"
  }
],
```

This format is effectively a key-value exchange based on a-priori agreement.

The Open Digital REST API specification currently limits the dynamics of an integrated OSS/MANO stack; constraining Zero-touch Operations, Orchestration and Management (ZOOM), because interface extensions are implicitly hardcoded.

In the context of the Dynamic APIs Catalyst, an OSS such as DGIT would have needed to explicitly add rules around these values into their schemas including data types, labels, etc.

For the catalyst we used the following:

```
"productCharacteristic":[
{
  "id":"anotherCharacteristic",
  "value":"itsValue",
   "metadata":{
        “label”: “Name for display of the characteristic”,
        “type”: “xsd:String”,
        “constraints”: []
  }
 }
],
```

The addition of explicit metadata was in keeping with the Metamodel contributed by EnterpriseWeb, in that each ProductCharacteristic is effectively an instance of an assertion, with enough data to dictate its use. This was used explicitly to populate / generate user interfaces directly, to enforce business rules on the parameters in the remote system (at the point of OSS ordering), and could be used to dictate workflows on both sides of the exchange.

We would suggest such extensions universally across the set of Digital REST APIs.

Embedding metadata in the exchange empowers rich clients to exploit and adapt, affording the potential to realize in part or full the benefits of the Dynamic API concept.

2.3.2 Extend the productRelationship element with Policy

Like productCharateristics, the productRelationship element is also found universally across the API. It denotes relationships between a product and a fixed set of endpoint types including “any other kind of link that may be useful”. In other words, it depicts relationships against elements at large.

As mentioned productRelationship has a fixed set of types: “bundled”, “reliesOn”, “targets”. This is an opportunity similar to the one around productCharacteristics. Rather than what are effectively key-value pairs denoting a fixed type relationship, there would be value in extending these relationships to include constraints and other policies.

2.3.3 API Testing / Synthesis

In general, it would be an asset for the API Hub to include sample endpoints for testing each interface. If you could hyper-link through the API document it would be ideal, that way when working with an endpoint you could directly access a testing interface.

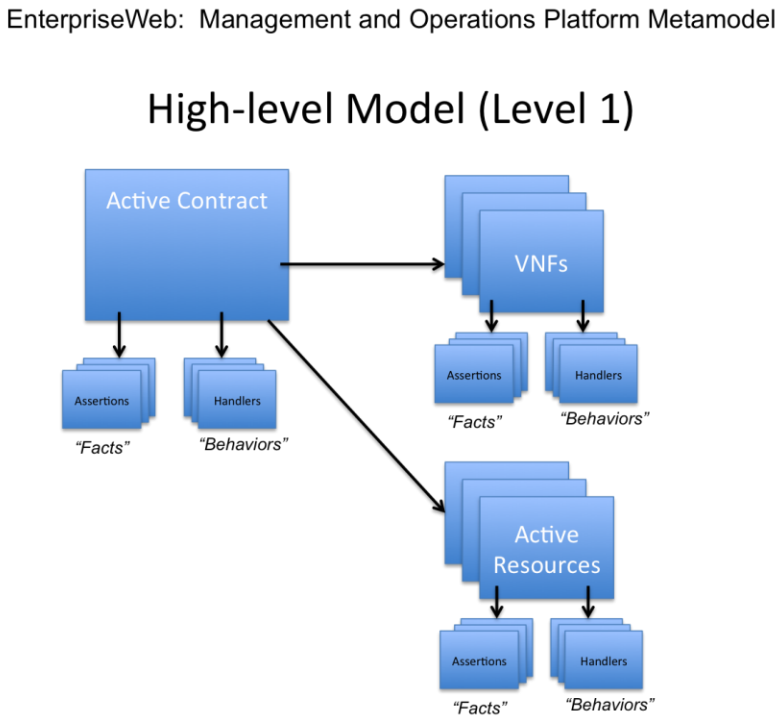
We'd also suggest the following tool as an asset for testing the Digital REST APIs, as well as REST interface testing in general: http://restclient.net/.

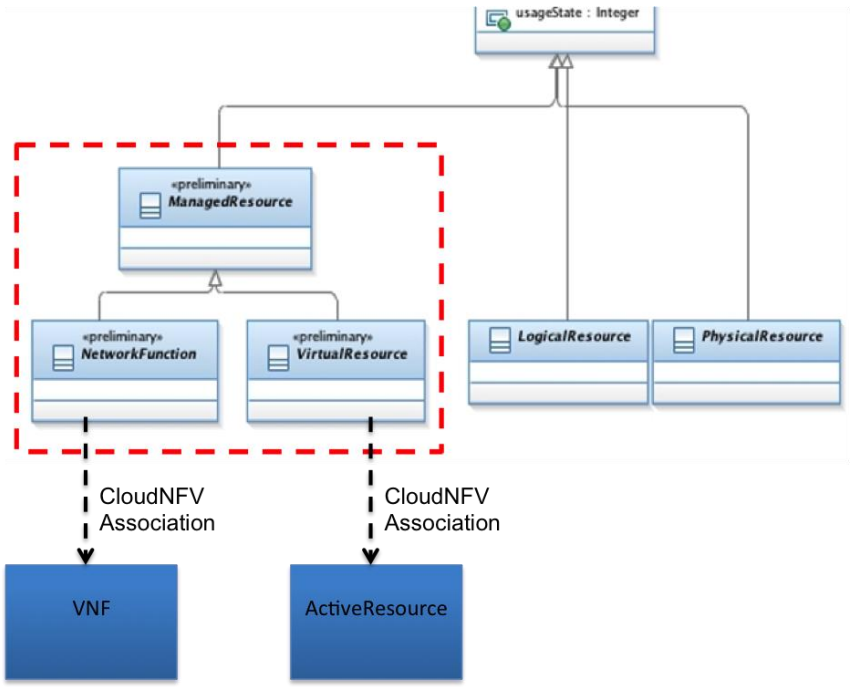
3. TMF TR 244** - Frameworx Enhancements to Support ZOOM

We mapped existing classes and extended them to support CloudNFV's dynamic, data-driven, policy-controlled orchestration (as per Metamodel contributed by EnterpriseWeb).

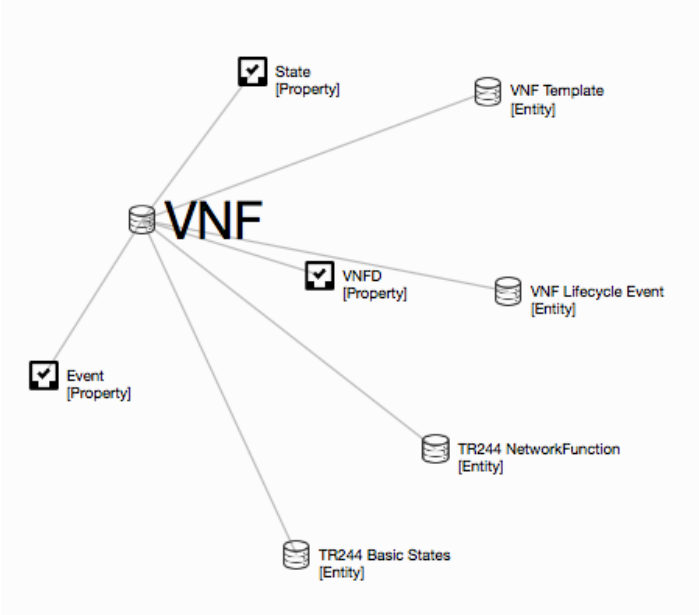
3.1 Experience

As part of the CloudNFV Catalysts, EnterpriseWeb has generally followed the ZOOM principles in our work with policies, with entities that generally align with classes/objects from its information model. For the Dynamic API catalyst, we found that the TR 244 offered us a compatible set of classes against which to map to the EntepriseWeb abstractions, as described in the Metamodel contributed by EnterpriseWeb.





Inside the EnterpriseWeb platform we were able to model this relationship directly



Mapping the TR 244 classes to our entities made it possible to reference TMForum terms directly within our system when modeling the Digital REST APIs used to exchange messages with the OSS and BSS components provided by DGIT and Comverse respectively. This meant that payloads exchanged for these APIs were directly available, or in other words, with a bit of additional mapping Catalogs were automatically populated based on our existing assets.

3.2 Feedback

We found that the new classes introduced by the TR 244 facilitated mapping to the EnterpriseWeb's MANO abstractions.

ManagedResource gave us a good anchor for our mappings. As it is effectively the root class of all Resources available for control, we were able to focus our interfaces on that one object, and derive appropriately downward. We found that VirualResource as a first-class citizen, mapping indirectly to PhysicalResource, was an exact match for our ActiveResource abstraction.

We found that the treatment of graphs in the TR 244 was also closer to our own. The expression of graphs in raw terms (DirectedEdge, Vertex, RootEntity) was appropriate and expressive enough, that you can then layer on the meaning for specific contexts – i.e. you could create a specific instance for a ConnectionGraph vs a ForwardingGraph. For EnterpriseWeb, this was natural – we use graphs as our native means of representation, and then derive specific graphs as views in all cases.

We also support the generalization of Catalogs as contained within TR244. Rather than a fixed set of redundant catalog types and associated class hierarchies, the new Catalog, CatalogItem and CatalogItemDetails set provide a generic and reusable set of abstractions from which to derive any specific catalog. We support a similar "Decorator" like pattern for generating the specific behaviours around types, and would encourage the use of similar structures wherever possible in Frameworx.

3.3 Recommendations

3.3.1 Graphs

The new classes for graphs have are well structured, but we believe the following should be considered:

- Rather than DirectedEdge, we'd recommend a more generic edge class, where it can be typed to be directed, bi-directional, or undirected.
- RootEntity seems convenient, but we think you'd increase flexibility by making this an attribute of particular Vertexes, making the graphs more general purpose.

3.3.2 Events

The newer event model classes are flexible as is, but we think would benefit from generally being treated in a way similar to that of Catalogs, where an explicit scheme for "decorating" types of events would be beneficial.

Also regarding Events, considering the relation of ZOOM to the Open Digital REST APIs, the TR 244 could make a natural bridge to their event models. At present the event models found in the Open Digital REST APIs are independent, using the TR 244 classes as a bridge point would allowing the mapping of references back to Frameworx objects within the definitions of those endpoints and payloads.

4. TMF TR 238*** - Fulfilling NFV MANO Interface Requirements

We mapped behaviors described for the MANO interfaces and extended them to support CloudNFV's dynamic, data-driven, policy-controlled orchestration (as per Metamodel contributed by EnterpriseWeb).

4.1 Experience

As a general platform for Cloud delivery, EnterpriseWeb crosses many of the interfaces described in TR 238. The Entity Provisioning (2.3.1) and Package Onboarding (2.3.4) interfaces have been focuses of our platform from the beginning of our efforts with the TMF (as demonstrated in previous Catalysts).

Also we have included many capabilities from the 2.4.1 Change Management, 2.4.3 Policy Information Exchange, Inventory (2.3.2) and Catalog Management (2.3.5) interfaces, as they are foundations for delivering the full-lifecycle of Network Services, which has always been our goal.

In the Dynamic APIs catalyst, we expanded the scope of these interfaces to include packages (and related entities – re: TF 244) involved in both BSS and OSS interchanges as realized through the Open Digital REST APIs. For this work we included the Ordering (2.3.3) interface, as well as expanded the scope of our current Inventory (2.3.2) and Catalog Management (2.3.5) interfaces.

4.2 Feedback

We found the TR 238 Interfaces document a valuable high-level map to our work in the MANO (ETSI) space. The Metamodel contributed by EnterpriseWeb contains our MANO abstractions in terms common to the MANO space at large, as a result we were able to use TR 238 as guidance for mapping to the new TR 244 classes as described previously in section 3.

4.3 Recommendations

We recommend that you consider giving the interfaces specified in the TR 238 explicit mappings to those found in the Open Digital REST APIs. At present, there is direct and obvious crossover in some areas (such as ordering), but the addition of a MANO specific crosswalk would be beneficial, and it would also help expose any gaps in the Open Digital REST APIs collection. This Catalyst identified gaps in those APIs around Billing, a similar identification of MANO specific gaps would help guide future extensions to the set.

5. TMF TR 235**** - Policy Model & Architecture

We mapped policy models to support CloudNFV's application model (as per Metamodel contributed by EnterpriseWeb).

5.1 Experience

TR 235 describes a policy model, with an associated Application Model left for future work (re: section 4.3.4). EnterpriseWeb provides such a generic application model at large, and when applied to TMF, implements an orchestration layer where models for policy plug in. The native policy model of EnterpriseWeb is very similar to that of Figure 5 in the document, with an "event-oriented" object related to rule definitions. EnterpriseWeb modeling is done using a graph-connected information model related Rich Entities (objects), as described in our previous Meta-model contribution.

Following this meta-model, in EnterpriseWeb the TR 235 PolicyRuleSpec class is mapped to a graph with conditions defined against any/all related entities and their attributes. This means that all data/meta-data from objects related (linked to) the policy context are reference-able, including information on VNFs, VNF templates, Resources, and even other relevant Policies. Further, since EnterpriseWeb also supports the mapping of other Frameworx resources such as the TR 244, this allows SLAs (and their metrics) to be described using those classes.

As described in the previous sections, onboarding TR 244 classes allowed us to plug into these existing policy abstractions, and automate the creation of system-to-system interfaces for both OSS and BSS integrations implemented exclusively through the use of policy models.

5.2 Feedback

We find that the TR235 provides an information model with a good decomposition and the right granularity to describe policy that can be flexibly implemented by a policy engine.

We agree with the general principles. A single rooted hierarchy, with all classes derived from either Entity, Value or MetaData, is convenient for us in our mappings, even though internally we are not hierarchy based. Similarly, we found the 'composition-first' nature of the Information Model supports an arbitrary assembly of Objects with Context, Policy and Event. Zooming in, we'd also say that the spectrum of policy "types" supported (ECA, Goal, Utility and Promise) are sufficient to capture the spectrum of policies one might want to implement from lower-level/specific/imperative up to high-level/intent based/declarative.

5.3 Recommendations

In keeping with the above recommendations, we see the extension of frameworks information model (TR 244) with concepts from the ZOOM as a positive. We'd recommend bringing these concepts as much as possible into the Digital REST APIs and other interfaces directly.