Reverse engineering of data models – how and why

Jacqueline Beaulac, Ericsson

June 2019 DDF

Data models… and reverse-engineering

- Many ONAP projects have developed their own data models independently to fill their own needs.
- As the complexity of ONAP increases, there is a need not only to document these data models, but also to visualize them, compare them, find commonalities, create abstractions, and define common principles.
  - One path towards these goals is through reverse-engineering the data models
  - This session will present different considerations and techniques when reverse-engineering data models to a common notation, looking at specific examples.
- The intention is to start a discussion around the needs in the community and how to best fulfil them.

The data model “flora”

- Almost all projects have their own APIs and/or models, specific to their own domain
  - See ONAP R4+ Per-Project Models and API Specs
  - A number of different representations have been used, such as:
    - JSON
    - YAML
    - XML and/or XSD (XML schema)
    - TOSCA
    - Swagger documentation
    - PlantUML documentation
- Many of these models include similar, but not identical, entities
  - Ex. Resources and services on a conceptual level are used throughout the whole system, but each component has its own domain-specific definitions for them
- Very few common principles established so far
  - The Root Common IM is one attempt to begin to capture common concepts and properties, in this case through an inheritance tree
  - See https://wiki.onap.org/display/DW/Root

Case study: PNF software version

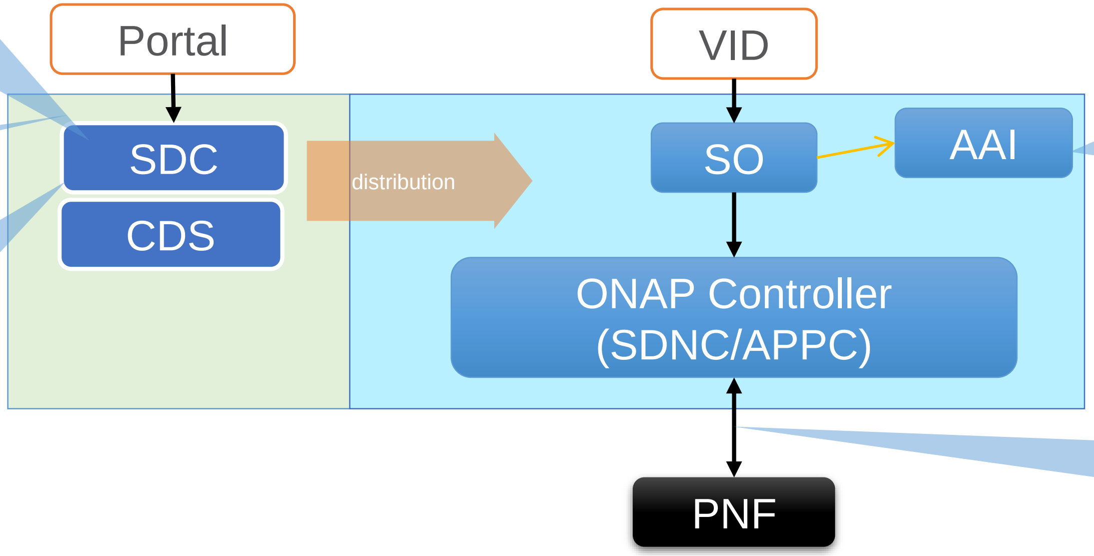
- In Dublin, PNF software version has been defined in various ONAP components
  - Slightly different definitions, not fully consistent
  - Not all are used by code

-- Software version list is added by SDC team and AAI team internally, **without any agreed IM/DM model team**.
-- SDC GUI (Resource composition GUI) allow operator manually to define the list of software version list. The softvare version list of internal descriptor shall be updated based on SDC GUI. But no component uses it today

AID DM in SDC internal datamodel
Generic_PNF.yml

```
node_types:
  org.openecomp.resource.abstract.nodes.PNF:
    derived_from: tosca.nodes.Root
    properties:
      software_versions
        type: list
```

-- Reserved for PNF SW UP UC, no components update it in Dublin



PNF Common IM (Agreed PNFD Model)

-- PNF Common IM does not include the sw version today.

A&AI DataStructure
element name="software-version
element name="sw-version"

PNF on-boarding DM(TOSCA node types of PNFD)

-- For on-boarding PNF package, software_versions list in internal descripter is empty because it is fully complied with ETSI(no software_versions in PNFD in ETSI).

5G PnP REG VES (PnfRegistration) on

ONAP R5+ Common IM Clean

Attribute: softwareVersion → active SW

Case study: PNF identifier

- New A&AI schema adaptations discovered a discrepancy between PNFs and VNFs:
  - VNFs were identified via VNF-ID (UUID)
  - PNFs were identified via PNF-name = Correlation ID, rather than PNF-ID = UUID
  - Decided that the definitions should be aligned by changing A&AI to use PNF-ID as the PNF identifier – *a breaking change*
  - See https://wiki.onap.org/pages/viewpage.action?pageId=58232836
- Note that, in the PNF registration use case, PRH receives PNF-name in the registration event, not PNF-ID
  - After the change of AAI key, PRH needs to use the PNF-name and the search API to:
    1. search for the relevant PNF instance record in A&AI
    2. reads the PNF-ID from that instance
    3. use that PNF-ID for subsequent CRUD operations
  - See https://wiki.onap.org/display/DW/PNF+PnP+use-case+update

- Some take-aways from this:
  - Need for common principles on a high level – such as how to identify an object instance
  - Need to understand the flow of information through the system for certain key use cases

Case study: SDC to SO mapping

- When SDC distributes the service model to SO, the SO MariaDB is updated using a mapping between the parameters received from SDC (ex. in a TOSCA file) and those recognized by SO
  - See https://wiki.onap.org/display/DW/SO+%3A+How+it+works+between+API+and+BPMN
  - The mapping is close to 1:1 and relatively intuitive
  - But even this simple mapping is a potential source of misunderstanding
    - "Same same but different"…

Different receivers may have different needs

- Beginners and non-developer users need a static “snapshot” of each data model as it looks at release time
  - This view could be simplified to lift noteworthy aspects and promote understanding
- Modelling and Architecture sub-committees are interested in exploring commonalities and defining common principles
  - Comparison and alignment of data structures throughout the platform
  - Visualization of the data flow through the system
    - I.e. revealing how different components realize or use the “same” object
  - Providing guidance for developers working in ONAP, reducing the perceived complexity
- Developers need up-to-date views of their domain data model that **add value** compared to just looking at the “raw” data model files
  - Different complementary views can assist in understanding different aspects of the full data model
  - “Smart” editors can enable easier data model updates by presenting a higher-level view of the information

Data model visualization

- There are a number of possible techniques that can be used to visualize a data model, such as:
  - Visualization tools tailored for the specific data model
    - Ex. AAI Data Model Explorer exposes the AAI data model in a consolidated table format
  - Generation of diagrams expressed in a well-known format
    - Ex. AAI GraphGraph creates graph diagrams from the AAI data model
  - Generation of reusable models
    - UML is a standard format for expressing object models that can then be used in a variety of ways, such as code generation
    - UML stored as XMI files (XML using a standardized schema) can be generated and then imported by UML tools
- Some considerations:
  - Is the receiver an editor of the data model or just a viewer?
  - Is the receiver interested in the exact details of the data model or only an abstraction of it?
  - Does the receiver always want an up-to-date visualization of the data model, or can they use a periodically-generated visualization?
  - Is the receiver working with data models from several sources or only a single one?

Why the need to reverse-engineer? Why not just visualization tools?

- For many reasons, it may be necessary to create a view of a data model that cannot be provided “on the fly”, such as:
  - Need to provide a simplified view due to the size and/or complexity of the data model
  - Need to consolidate information when a data model consists of several disjoint sets of information (possibly in separate files or formats)
  - Need to expose and document high-level aspects of the data model that are not obvious from the model itself
  - Need to compare data models from different sources (components) in order to:
    - Find commonalities
    - Create abstractions
    - Define common principles
- Reverse-engineering of existing data models is a means to produce these kinds of views in an existing system
  - ONAP is already in R5 – we don’t have the luxury of starting from scratch

Different approaches to reverse-engineering a data model

- One-time reverse-engineering
  - Producing a one-time abstraction from a data model
  - The abstraction can subsequently be handled separately from the data model
- Continuous reverse-engineering
  - Continuously maintaining an abstracted view of a data model
  - The data model is considered master, and the abstraction is updated from it whenever the data model is changed
- Initial reverse-engineering followed by forward-engineering
  - Producing an initial view of the data model that subsequently is considered master for the data it contains
  - Updates to the model cause the data model to be updated
- Round-trip engineering
  - Continuously maintaining a 1:1 visual representation of a data model
  - Updates to either the original data model or its visualization should be synched, regardless of which has been updated

Abstraction vs. 1:1 visualization

- Reverse-engineering can employ various degrees of abstraction
- A high-level abstraction can diverge from the data model that inspired it, especially in order to lift particular aspects of interest
  - For example, the structure can be simplified and/or naming can be aligned, to make it easier to compare similar or related data models
  - The abstraction can then be updated when the data model is changed, or per release, or it can evolve semi-independently from the source data model
    - Creating the abstraction could be an entirely manual analysis activity, if no appropriate parsing tools exist
  - Once the abstraction has been created, it can be difficult to “reverse” the abstraction without manual work
    - Some information needed to reproduce the data model is often lost in the abstraction process
- A visualization on the other end of the scale, closer to a 1:1 mapping, is more restrictive – but at the same time opens different possibilities
  - For example, the structure, entity names and data types need to be preserved as-is, or shall be converted according to fixed mapping rules
  - The goal is to ensure that no information is lost in the view generation process
    - From an information perspective, the data model and its visual representation are kept completely equivalent
  - In theory it should then be possible to “re-reverse” the process and generate the data model from the visual representation
    - Updates to either the data model or its visualization can thus be synched, regardless of which has been updated

Example: Reverse-engineering VES data model into the Common Information Model

- See https://wiki.onap.org/display/DW/VES+7.1
- One-time manual reverse-engineering, resulting in the creation of a UML representation within the common IM
  - Created through inspection of the source data model files
- The resulting UML closely resembles the source data model

- A JSON to UML tool implemented as an Eclipse plugin was proposed to generate the model: https://github.com/SOM-Research/jsonSchema-to-uml
- However the translation created by this tool wasn't fully appropriate for use as part of the common IM
  - The resulting UML is not fully compatible with the IISOMI guidelines and tools used in the IM work
  - The tool produces a simple 1:1 translation of the JSON data types, not fully aligned with the concepts in the IM

- VES is currently treated as an independent information domain in the information model
  - For example, the information model does not currently include the relationship between the fields in the PNF Registration Event and the corresponding attributes in the AAI Pnf
- Further abstraction and/or associations with the rest of the information model are probably still needed

Example: Policy Framework models in UML and in TOSCA

- The Policy Framework uses a TOSCA data model based on TOSCA Simple Profile
  - As part of this work, the team created a UML visualization of the TOSCA objects included in Policy
  - Overview and UML model here: https://onap.readthedocs.io/en/latest/submodules/policy/parent.git/docs/architecture/tosca-policy-primer.html
- In parallel with the data model work, the team created a UML visualization of their data model
  - See https://onap.readthedocs.io/en/latest/submodules/policy/parent.git/docs/architecture/architecture.html#policy-framework-object-model
  - This was also a one-time manual reverse-engineering, with the goal of visualizing the data model
- These two UML models are radically different from the Policy model (still under discussion) currently included in the Common IM
  - See https://wiki.onap.org/display/DW/Policy+Draft
- How to reconcile the differences?
  - Can some abstractions be lifted from the UML models of Policy Framework and into the common IM?
  - Is there a need to revisit the modeling of Policy Framework and expand it along the lines of the Common IM proposal?

Example: Reverse-engineering AAI data model into UML using Eclipse/Papyrus

- See https://wiki.onap.org/display/DW/Reverse-engineering+AAI+data+model+to+Papyrus+information+model
- This initial abstraction is intended to be a *one-time reverse-engineering* used as input to the ONAP information model
- To simplify handling, this process creates a UML model using the same tools as the information model (Eclipse, Papyrus)
  - Step-by-step half-automated process using existing XML & UML support in Eclipse
  - A modeler needs to manually go through the steps in the Eclipse GUI
  - Possible to automate with a script…?
- Note that a manual process is needed to create useful Papyrus diagrams
  - AAI model is huge – not useful to have everything in one diagram
  - AAI uses RelationshipList as a generic linking class – need to convert to a more useful representation
  - AAI uses collection classes (ex. “Pnfs” as a collection for Pnf) – these don’t seem to add any information and could be filtered out
- More work to be done…

Discussion: Using a reverse-engineered data model as an input to a common ONAP information model

- The work to incorporate a reverse-engineering of any domain-specific model into a common information model is an analysis activity, requiring knowledge of the whole of ONAP and how different components interact
  - The IM is not organized per component – it is meant to be common for the whole platform
- The final information model should then make it easier to find *commonalities* between components by, for example:
  - Expressing all entity names according to a common convention
  - Expressing all data types according to a limit common set
  - Merging similar objects from different data models into common object definitions
- How to minimize the work needed to accomplish this…?

Discussion: What is the relation between a common ONAP IM and the individual domain data models?

- After a high-level common abstraction has been produced as a one-shot, it needs to be kept relevant
  - How (and when) can updates to the domain data models be brought into the common IM?
  - How to visualize the relationships between the entities in the common IM and the domain data models?

ONAP

OPEN NETWORK AUTOMATION PLATFORM