Pydantify

Making Model-Driven Network Automation Pythonic

Students

Dominic Walther

Dejan Jovicic

Introduction: YANG is a data modelling language used to define data structures transmitted over either the NETCONF or RESTCONF protocol. Such models can be used to perform so-called Model Driven Network Automation.

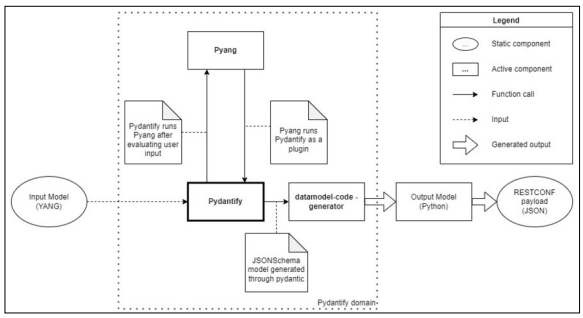
The goal of this project is to create a proof-of-concept to show how YANG models could be translated to Python data structures based on pydantic. These data structures can in turn be initialized with configuration values, serialized into a RESTCONF payload and sent to a network device, applying the configuration. If successful, this would facilitate configuring network devices through Python code without requiring the user to have prior knowledge of YANG.

Approach: We started by analysing the Python ecosystem surrounding YANG, including projects such as Pyang, pyangbind, yangson and Pyang-Pydantic, as well as pydantic and datamodel-code-generator. Our analysis revealed that most of the projects in the ecosystem have either been abandoned for years or are nowhere near robust or reliable enough to be used in a productive environment.

After some consideration, we settled on Pyang as our YANG parser and used the Pyang-Pydantic plugin as a starting point for our project. We then proceeded to gradually add features designed to make the generated Python models as intuitive and convenient to work with as possible.

Result: This resulted in the creation of Pydantify, a tool for translating YANG models into executable Python code, which can in turn generate valid RESTCONF payloads. On top of validating the concept as feasible, we successfully managed to implement a large section of the YANG specification, enabling some real-world models to be converted without issue.

Pydantify Component Overview
Own presentment



Input:
YANG sample model
Own presentment

```
module model {
  namespace 'http://hsr.ch/';
  prefix md;
  description 'A hello-world test model to validate the usablity of Pydantify.';

  container interface {
    description 'A simple container with 2 leaf nodes.';

    leaf address {
      type string;
      mandatory 'true';
      description 'A simple IPv4 address.';
    }

    leaf port {
      type uint16;
      mandatory 'true';
      description 'A simple port number.';
    }
  }
}
```

Output:
Pydantic model
Own presentment

```python
class AddressLeaf(BaseModel):
    __root__: str
    """A simple IPv4 address."""

class PortLeaf(BaseModel):
    __root__: Annotated[int, Field(ge=0, le=65535)]
    """A simple port number."""

class InterfaceContainer(BaseModel):
    """A simple container with 2 leaf nodes."""

    address: Annotated[AddressLeaf, Field(alias='model:address')]
    """A simple IPv4 address."""
    port: Annotated[PortLeaf, Field(alias='model:port')]
    """A simple port number."""

class Model(BaseModel):
    interface: Annotated[
        Optional[InterfaceContainer], Field(alias='model:interface')
    ] = None

if __name__ == "__main__":
    model = Model(interface=InterfaceContainer(address="192.168.1.1",port=23))
    restconf_payload = model.json(exclude_defaults=True, by_alias=True, indent=2)
    restconf_patch_request(url='...', user_pw_auth=('usr', 'pw'), data=restconf_payload)
```


Advisor
Urs Baumann

Subject Area
Software, Internet Technologies and Applications