

L'idea è quella di aggiungere un nuovo comportamento al sistema introducendo un valore booleano.

Perché farlo? Per limitare la scelta dell'utente nella selezione di alcuni parametri, ad esempio potremmo inserire un parametro simile a XSS_active: true.

Come farlo? Per prima cosa dobbiamo aggiungere e mappare il nuovo comportamento tramite mdsal generando le classi che saranno usate dal nemo engine.

La classe da cambiare coincide con nemo-object in particolare:

```
leaf property-value-type {  
  description  
    "The type of the property value.";  
  default string;  
  type enumeration {  
    enum string {  
      description  
        "An string-valued property.";  
    }  
    enum int {  
      description  
        "An integer-valued property.";  
    }  
    enum range {  
      description  
        "An integer-range property.";  
    }  
    enum boolean {  
      description  
        "A boolean property that can be use to specify active propperty.";  
    }  
  }  
}
```

Aggiungiamo quindi la definizione dei nostri valori booleani secondo lo standard yang. Effettuiamo un'operazione simile anche in nemo-template:

```
    type int64;  
  }  
  
  leaf order {  
    type uint32;  
  }  
}  
  
list boolean-value {  
  key "value order";  
  leaf value {  
    type boolean;  
  }  
  
  leaf order {  
    type uint32;  
  }  
}
```

Modificati i file di configurazione delle entità di nemo andiamo a modificare il comportamento del core di sistema:

nel file `LanguageStyle.jj` che viene utilizzato da `javacc` per creare un parser in automatico andiamo a definire le nuove entità sia per quanto riguarda le “istanziamenti dei modelli”, sia per la lettura dei parametri.

Fatto ciò, possiamo settare modelli che funzionano tramite valori booleani, ma nemo non conosce ancora come registrare il valore all’interno del template. Per registrare il valore si dovrebbe modificare tutto il core di sistema, ma se si vuole aggiungere il comportamento alla registrazione dei template si può banalmente modificare due classi:

- `UpdateTemplateInstanceLang.java`
- `UpdateTemplateInstance.java`

Rispettivamente si deve aggiungere un controllo per configurare i valori booleani nel template

```
98 for (String value : values.keySet()){
99     if (values.get(value).equals(NEMOConstants.string)){
100         StringValueBuilder stringValueBuilder = new StringValueBuilder();
101         stringValueBuilder.setKey(new StringValueKey(order, value))
102             .setValue(value)
103             .setOrder(order);
104         order++;
105         stringValues.add(stringValueBuilder.build());
106     }
107     if (values.get(value).equals(NEMOConstants.integer)){
108         IntValueBuilder intValueBuilder = new IntValueBuilder();
109         intValueBuilder.setKey(new IntValueKey(order, Long.parseLong(value)))
110             .setOrder(Long.parseLong(value))
111             .setOrder(order);
112         order++;
113         intValues.add(intValueBuilder.build());
114     }
115     if (values.get(value).equals(NEMOConstants.range)){
116         RangeValueBuilder rangeValueBuilder = new RangeValueBuilder();
117         String[] range = value.split(" ");
118         rangeValueBuilder.setMax(Long.parseLong(range[0])>Long.parseLong(range[1])?Long.parseLong(range[0]):Long.parseLong(range[1]));
119         rangeValueBuilder.setMin(Long.parseLong(range[0])<Long.parseLong(range[1])?Long.parseLong(range[0]):Long.parseLong(range[1]));
120         order++;
121         rangeValue = rangeValueBuilder.build();
122     }
123 }
```

E a verificare se i valori che sono stati aggiunti rispecchino le entità create in nemo, quindi se la proprietà X coincide con il template salvato in memoria:

```
98 List<org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.nemo.template.rev151201.template.instance.g
99 HashMap<ParameterName, org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.nemo.template.rev151201.t
100 if (instanceParameters != null){
101     for (org.opendaylight.yang.gen.v1.urn.opendaylight.params.xml.ns.yang.nemo.template.rev151201.template.instance.g
102         if (definitionMap.containsKey(parameter.getParameterName())){
103             if (definitionMap.get(parameter.getParameterName()).getIntValue()==0 &&
104                 !(parameter.getParameterValues().getIntValue()==null&&parameter.getParameterValues().getStringVal
105                 return "The value type should be string";
106             }
107             if (definitionMap.get(parameter.getParameterName()).getIntValue()==1 &&
108                 !(parameter.getParameterValues().getIntValue()!=null&&parameter.getParameterValues().getStringVal
109                 return "The value type should be int";
110             }
111             if (definitionMap.get(parameter.getParameterName()).getIntValue()==2 &&
112                 !(parameter.getParameterValues().getIntValue()==null&&parameter.getParameterValues().getStringVal
113                 return "The value type should be range";
114             }
115             else {
116                 instanceParameterMap.put(parameter.getParameterName(), parameter.getParameterValues());
117             }
118         }
119     }
120     else {
121         return "The parameter " + parameter.getParameterName().getValue() + " is not defined.";
122     }
123 }
```

Conclusione:

estendere in questo modo il core di nemo risulta inutile perché non fornisce nessun valore aggiunto, di fatto andando a limitare ancor più la sintassi e quindi andando contro le guideline del RFC che regolano gli intenti.

La conclusione logica è che per aumentare la flessibilità di nemo bisognerebbe utilizzare ciò che il linguaggio ci mette a disposizione, solo successivamente analizzare la sintassi nei moduli e tenere conto delle scelte per creare i file di configurazione generici che verranno tradotti successivamente.

Ho analizzato anche il funzionamento del parser puro, se si vuole modificare quello non ci sono problemi, le funzioni di richiamo sarebbero jjm0_0 per individuare il valore di reference di nemo, quindi bisognerebbe scrivere la parola boolean tramite la serie di funzioni a cascata richiamate e fornire il kind token salvato in nemo constant. Una volta riconosciuto il nome in nodemodel si deve salvare e configurare come lettura lo string id. Per quanto riguarda le property se si vuole operare manualmente si deve vedere se il valore letto coincida con un booleano nella funzione jjmove (il problema è sempre capire come scrivere il booleano, lo metto che deve essere uguale a true o a false in modo statico nella regex?) e successivamente la modifica avviene in modo analogo a quanto riportato sopra ma è da decidere. Secondo me non ha senso questa implementazione perché non aggiunge valore, oramai la sintassi ci permette di aggiungere etichette chiave valore da poter essere sfruttate nel manager che istanzia la VNF cioè che ne fa il rendering.