

# A Model-Driven Process Enactment Approach for Network Service Design

Sadaf Mustafiz<sup>1</sup>, Navid Nazarzadeoghaz<sup>1</sup>, Guillaume Dupont<sup>1</sup>,  
Ferhat Khendek<sup>1</sup>(✉), and Maria Toeroe<sup>2</sup>

<sup>1</sup> ECE, Concordia University, Montreal, Canada  
{sadaf.mustafiz,ferhat.khendek}@concordia.ca,  
{n.nazarz,gdupont}@encs.concordia.ca

<sup>2</sup> Ericsson Inc., Montreal, Canada  
maria.toeroe@ericsson.com

**Abstract.** The development of the Network Functions Virtualisation (NFV) paradigm has made way for the rapid deployment and management of network services. The European Telecommunications Standards Institute (ETSI) has been actively defining the NFV framework, which includes functional blocks and artifacts at different levels of abstraction. As part of the artifacts, various deployment templates have been defined to drive the deployment and the management of network services (NS) and Virtual Networks Functions (VNFs). The design of an NS is a complex activity that aims at selecting appropriate VNFs, creating the VNF forwarding graph (VNFFG), and all the necessary templates for the NS deployment and management, on the basis of the tenant's requirements and existing VNFs. Automating the NS design activity as well as the NS management process itself is highly desirable and beneficial for NFV systems. Continuous deployment for NFV with model-driven orchestration means has been recently advocated.

In this paper, we propose a model-driven process for the design of network services which covers the automatic generation of the NS deployment template and the associated templates. The core of the process involves the decomposition of the NS requirements with the help of an ontology, and the selection of proper network functions based on a catalogue of existing VNFs. Moreover, we provide support for automated process execution with a model-driven process enactment approach. The process is modelled as a UML activity diagram. All the artifacts are models of defined metamodels. Enactment of the NS design process is carried out by mapping the process model to a model transformation chain, and executing the chain.

## 1 Introduction

Network Functions Virtualisation (NFV) is an emerging paradigm that builds on cloud computing and the virtualisation technology to eliminate the drawbacks of traditional physical network infrastructure and enables rapid provisioning of network services (NSs) [11, 18]. The use of NFV reduces capital and operating

expenses, since it does not require a wide range of network equipments to be deployed. The physical devices are remodelled into virtual entities implemented as software packages, referred to as Virtual Network Functions (VNFs) [18].

The NFV reference architectural framework standardized by ETSI [18] and adopted by TOSCA [29], defines various functional blocks playing different roles in the different phases of NS and VNF lifecycle management, from on-boarding to deployment and management. The ETSI standard specifies the NFV reference framework, its functional blocks, their roles, their interfaces, and some NS and VNF-related operational flows [17–19]. An NS and VNF deployment and management process is implied from these, however the workflow as such is not defined. Previously, we have proposed a model-based process for network service design and deployment [28]. The proposed workflow is compliant with the NFV reference framework. We had also proposed the network service design activity (which is outside the scope of the standard) as part of the process. NS design entails the generation of new NS deployment templates, namely NS Descriptors (NSD [17]), based on the tenant’s NS requirements and the provider’s VNF catalogue [19]. Network service requirements (NSReq) consist of functional and non-functional characteristics of a service requested by a tenant. Examples of NSs being requested include VoLTE or VoIP, for instance, with some specific non-functional characteristics.

This workflow is a first step towards the necessary automation of the NS design and deployment process for NFV systems. Automating NS management, in other words, automating the execution of the workflow or process for NS management without manual intervention is highly desirable in the NFV domain and remains a major challenge [10,27]. The application of model-driven engineering (MDE) methods and tools is essential to further such developments in the NFV domain [6]. MDE advocates the use of models as first class citizens in the engineering process. The models are manipulated with model transformations which form the backbone for automation in MDE. ETSI has recently released an information model for NFV [21]. Leveraging these models can substantially benefit the NFV systems by reducing their development and management efforts. Moreover, explicit modelling of the process not only allows the automation of the NS management process but also paves the way for streamlining or optimizing the process to ultimately speed up deployment time. Such a process model (PM) can potentially be mapped to model transformation chains hence enabling NS management and orchestration via model-driven process enactment [5,15,34].

We propose an approach for model-driven enactment of the NS design, deployment and management process. In this paper, we focus on applying our approach to the NS design activity only. We model the internal behavior of the NS design activity by outlining a set of actions that need to be taken to come up with a deployment template for network services (NSD). The enactment of the NS design process allows for automatic generation of the NSD. We adapt the Papyrus [14] environment to provide tool support for process enactment.

This paper is structured as follows: Sect. 2 gives a brief background on the NFV reference framework and the NFV artifacts. Section 3 proposes a process

model for NS design. Section 4 presents our enactment approach. In Sect. 5, we review the related work. Finally, Sect. 6 concludes with some future work.

## 2 Background

This section provides a brief introduction to network services and some of the artifacts and functional blocks in the NFV reference architecture as proposed in the ETSI standard [16–18].

As stated in [17], a *network service (NS)* is a composition of network functions (NF) arranged as a set of functions with unspecified connectivity between them or according to one or more forwarding graphs. ETSI defines the NF forwarding graph as a graph of logical links connecting NF nodes for the purpose of describing traffic flow between these network functions [16]. It is essentially the end-to-end sequence of NFs that packets traverse. *Virtualised Network Functions (VNFs)* are the building blocks of an NS in NFV. VNFs are software pieces which may have the same functionality as their corresponding physical network functions, *e.g.*, a virtual firewall (vFW) *vs.* a traditional firewall device. A VNF can be composed of multiple internal components (VNFC). The description of the deployment behaviour along with the non-functional characteristics of a VNFC is defined as a Virtual Deployment Unit (VDU) [16]. *Virtual links (VLs)* are used to connect VNFs to form a network topology. These are referred to as external VLs, whereas internal VLs are the links which connect VNFCs within a VNF. A *Connection Point (CP)* is the port that an NF exposes to connect to another NF component via VLs (similar to the ports in a physical network module, such as a switch). The connection point for an NS to link to the environment is defined as a *Service Access Point (SAP)*, every *VNF Forwarding Graph (VNFFG)* is associated with one or more pool(s) of connection points (CpPool). The sequence of connection points inside a VNFFG is referred to as the *Network Forwarding Path (NFP)* which is required when different traffic flows exist.

During the lifetime of an NS, various artifacts at various levels of abstractions are used and produced. The deployment templates, referred to as **descriptors**, describe the deployment requirements, operational behaviour, and policies required by the NSs or VNFs. ETSI defines the Network Service Descriptor (NSD) as a deployment template which consists of information used by the NFV Orchestrator (NFVO) for lifecycle management of an NS [17]. Descriptors also exist for VNFs (VNFD) VLs (VLD), and VNFFGs (VNFFGD) among others. The constituent elements of an NSD are shown in Fig. 1.

*Catalogues* are defined in the NFV architecture which are part of the NFV data repositories. The **NS Catalogue** contains all the on-boarded NSDs, VNFFGDs, and VLDs. The **VNF Catalogue** contains all the on-boarded VNFDs.

Nested NS and Physical Network Functions (PNF) are outside the scope of this paper. In our process, the artifacts are all considered to be models (instances which conform to existing meta-models).

The main functional module in the architecture is the NFV Management and Orchestration (NFV-MANO), which is in charge of deployment, management,

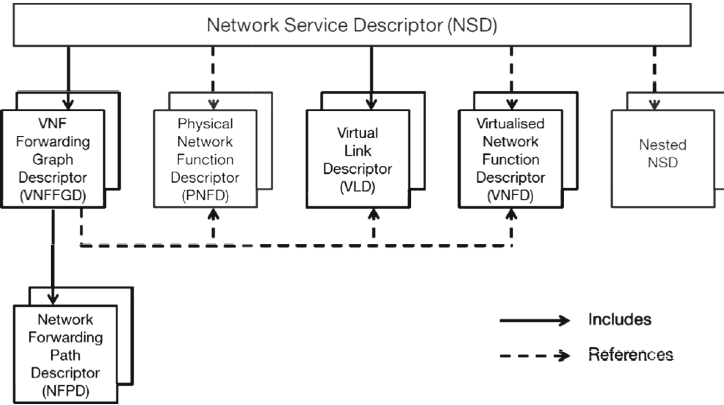


Fig. 1. NSD Overview (from ETSI NFV IFA014 [17])

and orchestration of NSs. The NS orchestration and lifecycle management which include onboarding and instantiation of NS is taken care of by the NFV Orchestrator (NFVO). NFV-MANO also includes managers which are responsible for the VNFs and the infrastructure, namely VNFM and VIM. Operations Support Systems and Business Support Systems (OSS/BSS) refer to the operator’s proprietary systems and management applications supporting their business. The OSS/BSS systems exchange a lot of information with NFV-MANO functional blocks to provide the desired network service. For details on all the functional blocks in the NFV framework, the reader can refer to [18].

### 3 Network Service Design

The NS design entails the definition of deployment templates (namely NSD, VNFFGD, and VLD). These descriptors include static information elements related to an NS. The NSD is used by the orchestrator as a template for instantiating the NS.

We propose a method for NS design by taking inspiration from [1]. A tenant may request a new NS by specifying the NS Requirements (NSReq), which consist of functional and non-functional requirements possibly with some initial decomposition targeting specific functions. There is a big gap between the information provided by the tenant and the network service to be deployed. The tenant has limited knowledge regarding the details of this target network service, and hence this gap needs to be filled. The knowledge to help in filling this gap comes from the various architectures and standards existing in the telecommunications and network service domain. It is essential for this knowledge to be captured and retained for use later when a new network service is required. In our approach, we propose to define and retain this knowledge in a Network Function Ontology (NF Ontology). With each new NS design, information about new architectures and functionalities is gained and this is used to enrich the ontology.

NFOntology captures standard network function (de)compositions as defined by different standardisation bodies such as 3GPP as well as knowledge and experience from previous decompositions, architectures and network service designers.

In our work, we assume that the OSS/BSS of an NFV provider gets the NSReq and generates the NSD based on the provider's VNF Catalogue. The NS design method involves the decomposition of the NSReq and the selection of proper network functions, e.g., VNFs (and/or PNFs) from the VNF Catalogue. The NSReq decomposition is guided by a NFOntology. The NFOntology captures the decomposition of network functionalities to some level of granularity where each functionality can be mapped onto some VNF provided functionality. When the decomposition reaches that level, VNFs from the VNF Catalogue are matched and selected to compose the network service. During this activity, the VNF forwarding graph descriptor (VNFFGD) and the virtual link descriptors (VLD) are also generated. The design phase also takes into account the non-functional requirements and refines the NSD accordingly by adding deployment flavours and associating VNF profiles to the NS. *It should be noted that our NS design method does not address the concept of nested NS as yet.*

### 3.1 NS Design Languages

As part of our process, we propose languages for modelling the NSReq and the NFOntology which are required inputs for the process. We also define a VNF Catalogue metamodel for modelling a catalogue containing VNF packages (defined by ETSI in [21]).

**NSReq and NFOntology.** We have defined an abstract syntax as well as a concrete syntax for the NSReq and NFOntology languages.

The NSReq contains the hierarchy of requirements for a network service according to the needs of the tenant. The metamodel of the NSReq language is shown in Fig. 2. As shown in the figure, an NSReq consists of the main functional requirement which is the highest level functionality of the network service. Each functional requirement (identified with a unique name in `FunctionalRequirement`) can be decomposed into lower-level functional requirements, and this builds a hierarchy of NS requirements. A functional requirement can be associated with various non-functional requirements (`Non-FunctionalRequirement`), such as availability, reliability, and throughput.

The proposed NFOntology language for NFV is an extended variant of a feature diagram [24]. The metamodel is shown in Fig. 3. The ontology language has two main components: Functionalities and Architectural Blocks. The *Functionality* part of the Ontology is modelled as a variant of a feature diagram. The *ArchitecturalBlock* part has specific syntax and semantics in addition. Essentially, the NFOntology is a hierarchy of (unique) functionalities in the network service domain where a functionality can have zero to many decompositions. As in feature diagrams, decomposition relationships between a functionality and lower-level functionalities can be categorized as: *mandatory*, *optional*, *alternative*, and *OR*. A functionality can be dependent on another functionality.

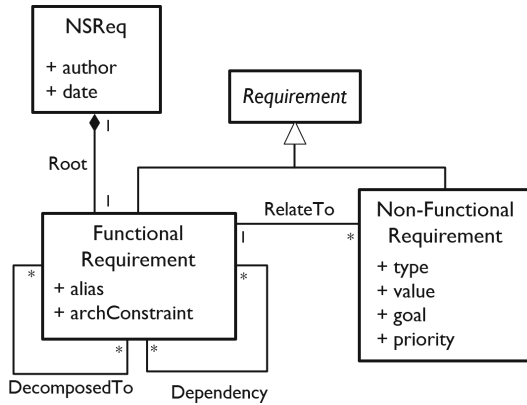


Fig. 2. NS requirements metamodel

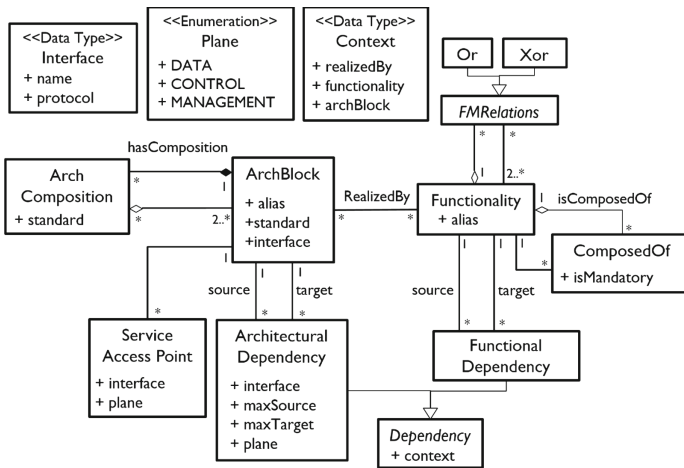


Fig. 3. Network function ontology metamodel

The architecture defines possible ways of realizing the functionalities with established architectures for network services. The architectural blocks in the ontology are unique blocks detailing specific architectural designs with well-defined interfaces and protocols. Dependency relationships may exist between architectural blocks. On the basis of the functionality and the architectural blocks, a good decomposition of the network service requirements can be achieved. For example, VoIP service is an essential NS nowadays. While capturing the details required to create such a service can be quite difficult, having such architectures like IMS (IP Multimedia Subsystem) to cover most of the requirements for providing VoIP can be very helpful. IMS has a well known architecture and well known functional components. There is a high possibility of finding VNFs which have been developed for implementing such architectures. IMS



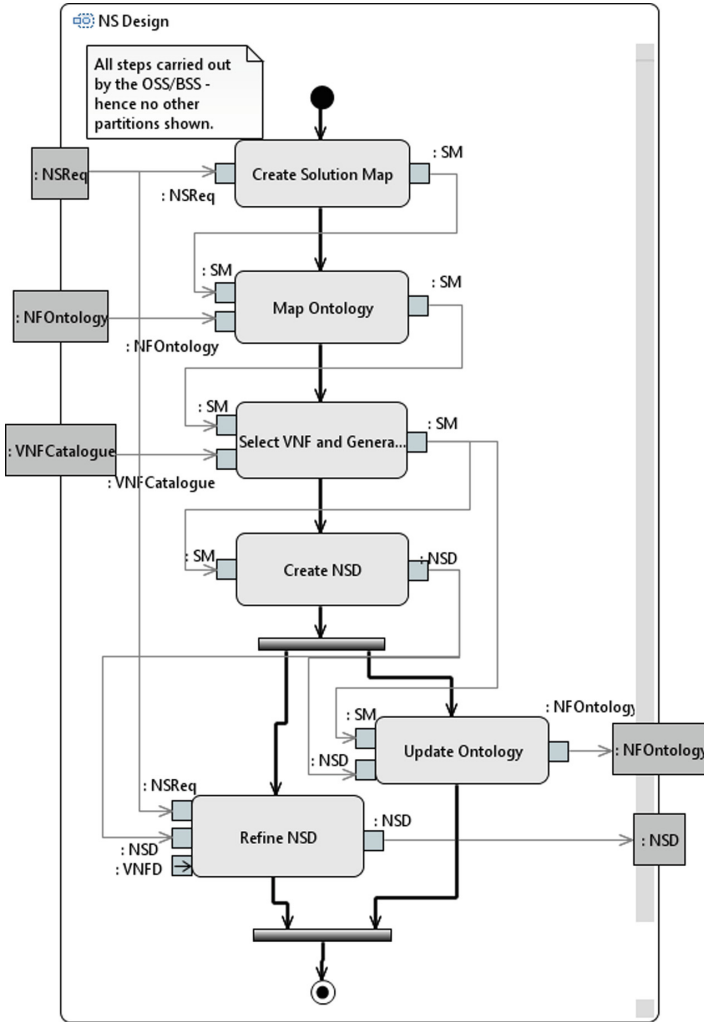


Fig. 6. NS design process model (PM)

due to space constraints). This is a combination of the `NSReq`, `NFOntology`, and `VNFD` metamodels.

The actions which are part of the NS Design PM are outlined here.

- **Create Solution Map.** This action takes as input an `NSReq` model (see Fig. 7) and initializes the `SolutionMap` with the content of the source model. The `SolutionMap` is an intermediate artifact created to aid in the NSD generation process.
- **Map Ontology.** This action takes as input the `SolutionMap` model created in the first step and an existing `NFOntology` model (see Fig. 8). For each functionality in the `SolutionMap`, the ontology is traversed to find any existing



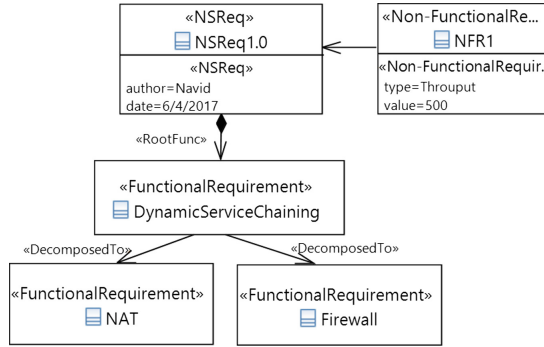


Fig. 7. NSReq model

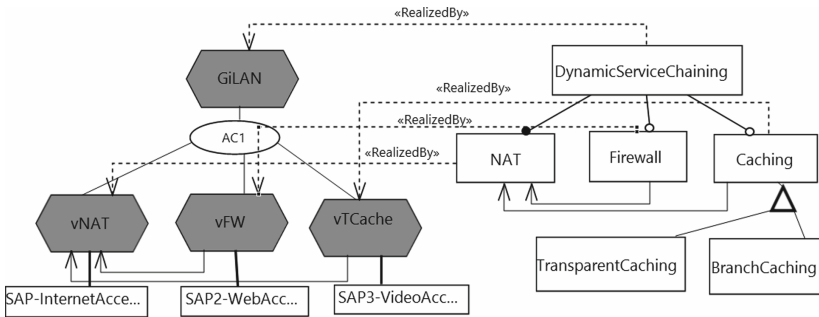
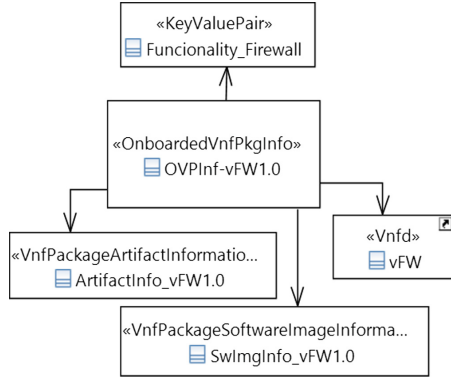


Fig. 8. NFOntology model

knowledge about its composition and dependencies. When a match is found in the ontology, all the details not available in the **SolutionMap** are added, including the architectural blocks and their dependencies. Unmatched functionalities, architectural blocks, and dependency relationships are tagged in the **SolutionMap**, and may be used to enrich the ontology later.

- **Select VNF and Generate FG.** This action takes as input the refined **SolutionMap** and the **VNFCatalogue** (see Fig. 9). With the **SolutionMap** as a guide, a proper set of VNFs is selected for creating the NS. The functionalities of the VNFs found in the catalogue are matched with the architectural blocks and the functionalities in the **SolutionMap**. The VNFD (see Fig. 10) of each of the selected VNFs are added to the **SolutionMap**. Next, the proper combination of a set of functionalities is derived, leading to the combination of a set of architectural blocks, and ultimately to the combination of a set of VNFs which fulfill the NS requirements. For this purpose, an initial forwarding graph (FG) is created which contains the VNFDs, their sequence, and the details of the interfaces and service access points (SAP). This FG contains only dependencies but no real virtual links. The created FG becomes part of the refined **SolutionMap**.



**Fig. 9.** Part of VNFCatalogue model

- **Create NSD.** In this action, an NSD model is created and initialized based on the **SolutionMap**. VNFDs associated with the VNFs part of the FG are added to the NSD. The Virtual Link (VL) and Service Access Point (SAP) descriptors are created from the FG. Pre-defined types (MESH, TREE, LINE) are used to define a VL type. Finally, the VNFFGD deployment template (VNFFGD) is created which includes the Connection Point(CP) pool(s) and the Network Forwarding Path(s) (NFP). Till this point, only the functional aspects of the requirements have been mapped to the NSD.
- **Refine NSD.** This action involves addressing the non-functional requirements in the **NSReq** and adding to the NSD the relevant details, such as the deployment flavours and the VNF Profiles. The non-functional requirements are available in the **SolutionMap** and so it is used as input here. The other input is the VNFDs for the VNFs selected from the **VNFCatalogue** earlier in the process (see Fig. 10). NS-specific VNF Profiles are defined for the VNFDs at this stage. This step completes the NSD generation process. This NSD (see Fig. 11) then can be sent to the NFVO for onboarding.
- **Update Ontology.** Once an NSD has been successfully generated, the ontology is enriched if applicable. If the **NSReq** includes functionalities and/or their decompositions which do not exist in the ontology yet, i.e. those that were marked as unmatched, these elements are added to the **NFOntology** as new functionalities based on the **SolutionMap**.
- **Set Thresholds.** This action involves creating initial threshold models, **NSCapacityThreshold** and **NSPerformanceThreshold**, based on the **SolutionMap** to define the capacity and performance related thresholds of an NS.

Each action in the PM is mapped to a model transformation written in the ATL transformation language.

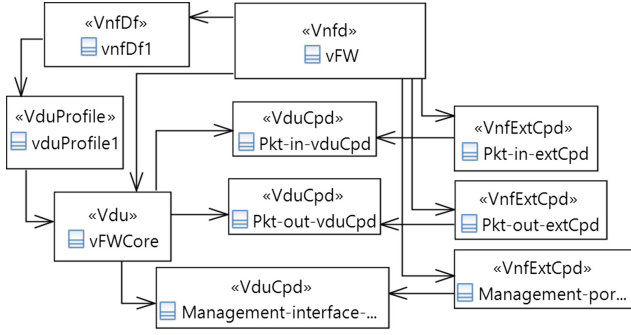


Fig. 10. VNFD model of a firewall

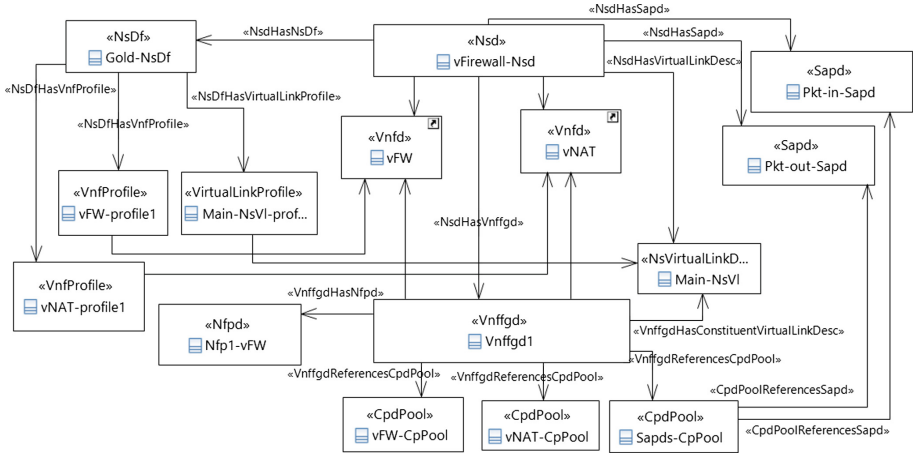


Fig. 11. Generated NSD model

## 4 Process Enactment

In this section, we present our approach for model-driven enactment of the NS design process. Our goal is to provide tool support for process execution by integrating enactment means with the Papyrus Activity Diagram environment leading to an integrated environment for process modelling and enactment.

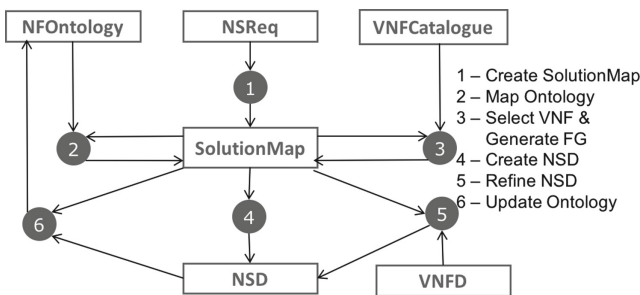
### 4.1 Enactment Approach

In our approach, process enactment is carried out with the use of transformation chain orchestration in combination with model management means. Transformation chaining is the preferred technique for modelling the orchestration of different model transformations [8]. Orchestration languages are used for the composition of the transformations in order to model the chain as sequential steps of transformations. Complex chains can incorporate conditional branches

and loops, and also can model composite chains (a chain including other transformation chains).

Model management approaches typically use megamodels which provide structures to avoid the so-called ‘meta-muddle’ [7]. A megamodel contains artifacts (which are models), relations between them (which may be transformations), and other relevant metadata. A megamodel can be seen as a map to find and link together all involved models. A megamodel forms a repository of models and transformations (and even tools). It can be used to enforce conformance and compatibility checks between the various models and transformations. It is also useful for reusing and composing transformations in transformation chains. The input and output models which are part of the PM are typed by metamodels residing in the megamodel. The transformation models that are associated with the actions in the PM are also known in the megamodel. In case of a transformation chain with a heterogeneous set of transformations, the megamodel helps in determining which transformation engine to use for the execution of the transformation. Figure 12 shows the visual representation of a simplified NS design megamodel (MgM).

We present here the approach we follow for process enactment to automatically execute the NS design process. We begin by mapping the Process Model (in essence a subset of the UML 2.0 Activity Diagram) to a model transformation chain. The transformation chain is generated using a higher order transformation [36] in a manner similar to [26]. An initial megamodel is automatically derived from the PM and then refined with further details if required. Orchestration of the transformation chain is carried out with the use of an orchestration engine. The workflow execution engine executes the chain of model transformations to generate the artifacts (the target models). The enactment approach is outlined in Fig. 13. We intend to apply the method to ultimately orchestrate the NS Management process presented in [28].



**Fig. 12.** NS design megamodel

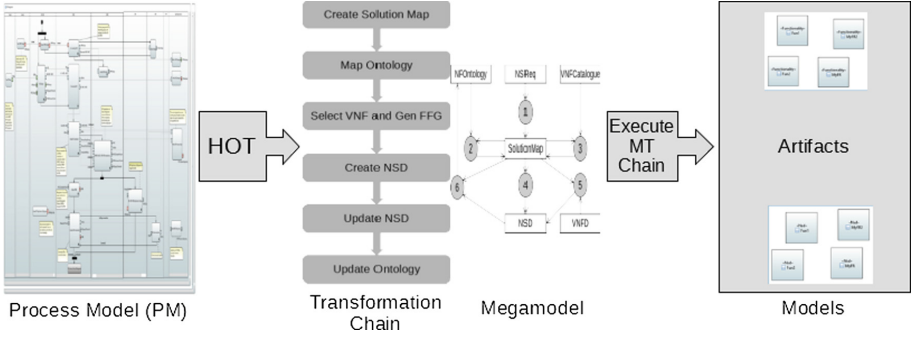


Fig. 13. Process enactment approach

4.2 Tool Support in Papyrus

We use Papyrus for both process modelling and enactment. Papyrus is NFV’s tool of choice. The NFV information model [21] released uses Papyrus as the modelling tool. We have extended the activity diagram environment in Papyrus to incorporate enactment means. The activity diagram contextual menu was adapted to include the *Enactment* option as shown in Fig. 14. The user has the option of choosing the transformation chain to execute (via run configurations) or the default chain gets executed.

The execution is carried out in the backend with a workflow execution engine. We use MoDISCO [9] for orchestrating the transformation chain. MoDISCO is a framework for model-driven reverse engineering which supports transformation

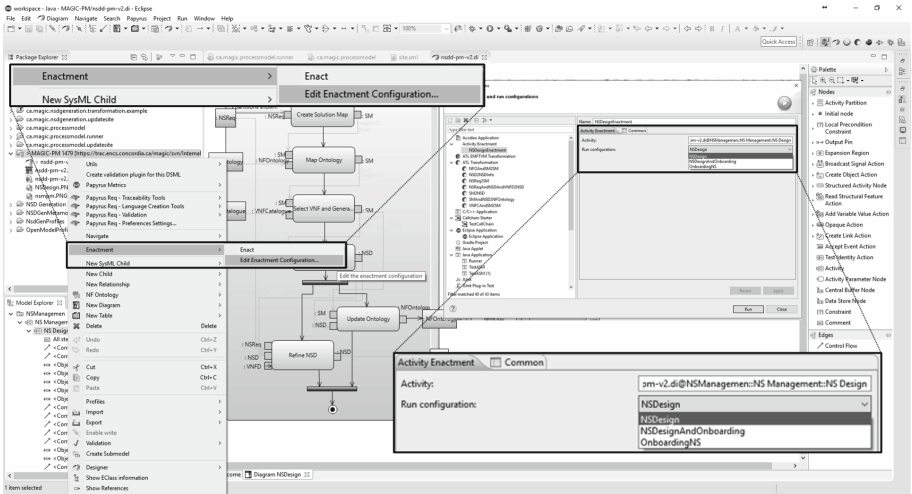


Fig. 14. Papyrus process enactment environment

chain execution along with automated discovery of artifacts. The MgM is created based on the Papyrus Activity Diagram. The launch configurations for the orchestration need to be defined prior to the execution. These can be generated from the information available in the PM and the MgM. Currently, the workflow is an ATL transformation chain. However with the use of the MgM, it will be possible to support execution of a chain of transformations in different languages. This is work in progress at the moment.

The NS Design PM (see Fig. 6) is mapped to a chain of ATL transformations (see in Fig. 15). In the NS design case, all underlying transformations have been modelled with the ATL transformation language. The MoDisco workflow for the NS design is shown in Fig. 16. *It should be noted that the chain and the megamodel does not include the transformation for initializing the threshold models (**Set Threshold** action in PM), which is currently work in progress.*

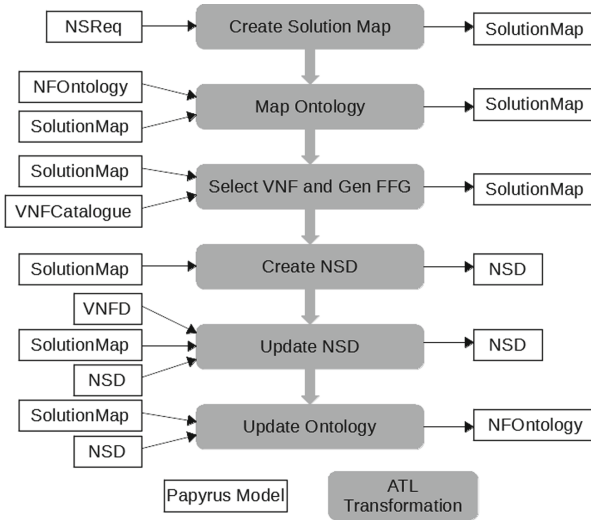


Fig. 15. NS design transformation chain

The process takes as input the `NSReq` (Fig. 7), `NFOntology` (Fig. 8), and the `VNFcatalogue` (Fig. 9). One of the `VNFD` models which is part of the input catalogue is shown in Fig. 10. The execution of the workflow generates the corresponding `NSD` (NS Descriptor) and updates the `NFOntology` model. For space reasons, we only show the `NSD` model here (see Fig. 11). Papyrus requires all metamodels to be mapped to Profiles to allow model instances to be created and to be used as source or target models of the ATL transformations. As per the NFV modelling guidelines, our models also comply with the Papyrus *Open-ModelProfile* [20].

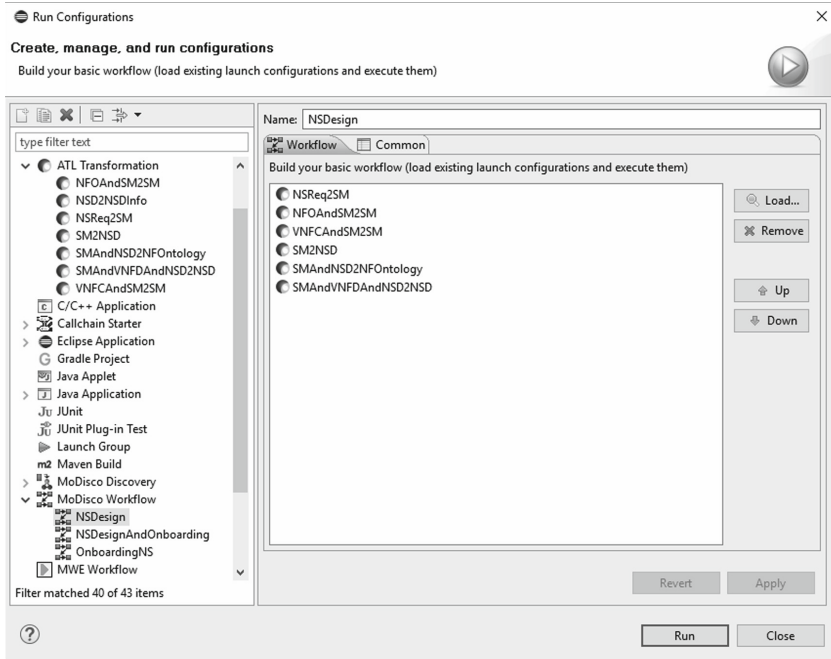


Fig. 16. NS design chain in MoDisco

## 5 Related Work

### 5.1 NS Design

While there exists work in the literature on service composition and decompositions, the notion of decomposing network service requirements for NFV systems has not been proposed to the best of our knowledge. We discuss here some related work on requirements and service (de)composition. Czarnecki et al. [13] presents an approach for carrying out staged configuration using specialization and multi-level configurations of cardinality-based feature models. At an abstract level, this is similar in concept to our NS requirements decomposition method. However, their work is applicable for feature models only and the decomposition technique is not automated. Web service composition is an area where extensive work has been done on decompositions of goals and functionalities [12, 32, 33]. These approaches however mostly deploy formal methods and are not model-driven in nature like our work.

Lin et al. [25] propose using an ontology as part of a requirements management process to capture design knowledge to help in concurrent engineering. Bartsch et al. [4] handles a component service replacement problem in the IT service domain with the help of ontologies. They do not address decomposition of user requirements.

As mentioned earlier, our work takes inspiration from the method proposed in [1]. In a similar manner, we carry out user requirements decomposition with the help of an ontology. However, our approach caters specifically towards network services. The NSD generation needs to take into account the various constituents of an NS which makes this a very complex method applicable for NFV systems.

In the NFV domain, Sahhaf et al. [35] consider different service compositions, i.e., VNFs arranged in different ways with different VNFFGs and VLs, and propose algorithms to select the optimal composition according to some criteria including resource demands, quality-of-service, and available infrastructure resources. This work focuses on VNF placement while our concern is the design of the network service. The *Oracle Communications Design Studio* [31] framework allows network services to be designed and also supports NS orchestration. However, the NS design requires to create various framework-specific NS constituent resources and they do not follow the ETSI specifications.

## 5.2 Process Enactment

We have covered the state of the art with regards to process modelling in the NFV domain in [28]. Process enactment is a widely adopted method in the business process modelling domain. Most of these work however do not follow a model-driven approach and/or do not provide support for model-driven enactment. Berezin [6] promotes using model-driven orchestration for NFV orchestration and talks about why this is a more robust method than business process workflows. BPMN-like workflows are in general implementations of specific task-oriented cases which are appropriate for immutable business processes as stated in [6]. In software defined environments which evolve rapidly, such workflows bring about difficulties and risks.

There has been a lot of work on megamodelling [2, 7], transformation chaining [15, 30, 34, 38], and a combination of both [23, 37] in the MDE community. A few MDE-based continuous integration and deployment methods and tools have been proposed with cloud applications as the target domain [3, 22]. While model-based approaches exist in the NFV domain [10, 29], the application of such advanced MDE techniques is minimal for NFV systems.

## 6 Conclusion

The two main contributions of this paper are the following: (1) a high-level process for creating network service deployment templates (referred to as NSD by ETSI) based on requirements from the tenant, and (2) an integrated environment for automatically generating the NSD using model-driven process enactment means.

The process for NS design proposes the use of a network function ontology to decompose the NS requirements and to select appropriate VNFs for the NS (from a VNF catalogue or repository). Based on this decomposition and selection, deployment templates (NSD, VNFFGD, and VLD) are generated. The



generation of the descriptors after the receipt of a tenant's requirements is automated using a model-driven enactment approach. The process is modelled in Papyrus as a UML 2.0 Activity Diagram, and can be automatically executed within Papyrus by orchestrating the workflow. The Process Model is mapped to a chain of ATL transformations for this purpose. As part of the process, several domain-specific languages have been proposed to model the associated artifacts: NS Requirements (NSReq), Network Function Ontology (NFOntology), and VNF Catalogue. We have built modelling environments in Papyrus that allow users to create model instances. The NFV descriptors (NSD, VNF-FGD, and VLD) have meta-models defined by ETSI [21]. In our process, the generated models conform to the ETSI-defined meta-models.

This work sets the basis for the enactment of the entire NS design, deployment and management process. Each activity in the NS life-cycle involves a complex chain of tasks. We are currently working on modelling the internals of the other activities, such as *NS Instantiation* and *VNF Instantiation*, and writing model transformations. The entire Process Model can then be mapped on to a composite chain of transformations along with an extended mega-model to allow for automated deployment and management of network services.

As future work, we intend to extend our transformation chain orchestration means to support different transformation languages (such as QVT). Moreover, we plan on integrating further model management techniques (using the mega-model) with process enactment. This will allow us to enforce conformance and compatibility checks between the various models and transformations, and will also aid in providing end-to-end traceability support.

**Acknowledgment.** This work is partly funded by NSERC and Ericsson, and carried out within NSERC/Ericsson Industrial Research Chair in Model Based Software Management.

## References

1. Abbasipour, M., Sackmann, M., Khendek, F., Toeroe, M.: A model-based approach for user requirements decomposition and component selection. In: Bouabana-Tebibel, T., Rubin, S.H. (eds.) *Formalisms for Reuse and Systems Integration*. AISC, vol. 346, pp. 173–202. Springer, Cham (2015). doi:[10.1007/978-3-319-16577-6\\_8](https://doi.org/10.1007/978-3-319-16577-6_8)
2. Allilaire, F., Bézivin, J., Brunelière, H., Jouault, F.: Global Model Management in Eclipse GMT/AM3. In: *Eclipse Technology eXchange Workshop (eTX) - A ECOOP 2006 Satellite Event*. Nantes, France, July 2006
3. Artač, M., Borovšak, T., Di Nitto, E., Guerriero, M., Tamburri, D.A.: Model-driven continuous deployment for quality DevOps. In: *Proceedings of the 2nd International Workshop on Quality-Aware DevOps*. QUDOS 2016, pp. 40–41. ACM (2016)
4. Bartsch, C., Shwartz, L., Ward, C., Grabarnik, G., Bucu, M.J.: Decomposition of IT service processes and alternative service identification using ontologies. In: *NOMS 2008–2008 IEEE Network Operations and Management Symposium*, pp. 714–717, April 2008

5. Basciani, F., Ruscio, D., Iovino, L., Pierantonio, A.: Automated chaining of model transformations with incompatible metamodels. In: Dingel, J., Schulte, W., Ramos, I., Abrahão, S., Infran, E. (eds.) MODELS 2014. LNCS, vol. 8767, pp. 602–618. Springer, Cham (2014). doi:[10.1007/978-3-319-11653-2\\_37](https://doi.org/10.1007/978-3-319-11653-2_37)
6. Berezin, A.: Utilizing Declarative Model-Driven TOSCA Orchestration for NFV. DZone, March 2017. <https://dzone.com/articles/utilizing-declarative-model-driven-tosca-orchestration-for-nfv>
7. Bézivin, J., Jouault, F., Rosenthal, P., Valduriez, P.: Modeling in the large and modeling in the small. In: Aßmann, U., Aksit, M., Rensink, A. (eds.) MDAFA 2003-2004. LNCS, vol. 3599, pp. 33–46. Springer, Heidelberg (2005). doi:[10.1007/11538097\\_3](https://doi.org/10.1007/11538097_3)
8. Brambilla, M., Cabot, J., Wimmer, M.: Model-Driven Software Engineering in Practice, 1st edn. Morgan & Claypool Publishers, San Rafael (2012)
9. Brunelière, H., Cabot, J., Dupé, G., Madiot, F.: MoDisco: a model driven reverse engineering framework. *Inf. Softw. Technol.* **56**(8), 1012–1032 (2014)
10. Chen, Y., Qin, Y., Lambe, M., Chu, W.: Realizing network function virtualization management and orchestration with model-based open architecture. In: 11th International Conference on Network and Service Management (CNSM 2015), pp. 410–418. IEEE (2015)
11. Chiosi, M., Clarke, D., Willis, P., Reid, A., Feger, J., Bugenhagen, M., Khan, W., Fargano, M., Cui, C., Deng, H., et al.: Network functions virtualisation: an introduction, benefits, enablers, challenges and call for action. In: SDN and OpenFlow World Congress, pp. 22–24 (2012)
12. Chung, L., Ma, W., Cooper, K.: Requirements elicitation through model-driven evaluation of software components. In: Fifth International Conference on Commercial-off-the-Shelf (COTS)-Based Software Systems, pp. 1–10. IEEE (2006)
13. Czarnecki, K., Helsen, S., Eisenecker, U.W.: Staged configuration through specialization and multilevel configuration of feature models. *Software Process Improv. Pract.* **10**(2), 143–169 (2005)
14. Papyrus, 16 June 2017. <https://eclipse.org/papyrus/>
15. Etien, A., Aranega, V., Blanc, X., Paige, R.F.: Chaining model transformations. In: Proceedings of the 1st Workshop on the Analysis of Model Transformations. AMT 2012, pp. 9–14. ACM (2012)
16. ETSI: Network Functions Virtualisation (NFV); Terminology for Main Concepts in NFV: ETSI GS NFV 003 V1.2.1, December 2014
17. ETSI: Network Functions Virtualisation; Management and Orchestration; Network Service Templates Specification: ETSI GS NFV-IFA 014 V2.1.1, October 2016
18. ETSI: Network Functions Virtualisation; Management and Orchestration; Report on Architectural Options: ETSI GS NFV-IFA 009 V1.1.1, July 2016
19. ETSI: Network Functions Virtualisation; Management and Orchestration; VNF Packaging Specification: ETSI GS NFV-IFA 011 V2.1.1, October 2016
20. ETSI: Network Functions Virtualisation (NFV) Release 2; Information Modeling; Papyrus Guidelines: ETSI GR NFV-IFA 016 V2.1.1, March 2017
21. ETSI: Network Functions Virtualisation (NFV) Release 2; Management and Orchestration; Report on NFV Information Model: ETSI GR NFV-IFA 015 V2.1.1, January 2017
22. Ferry, N., Song, H., Rossini, A., Chauvel, F., Solberg, A.: CloudMF: applying MDE to tame the complexity of managing multi-cloud applications. In: 2014 IEEE/ACM 7th International Conference on Utility and Cloud Computing, pp. 269–277, December 2014

23. Fritzsche, M., Gilani, W.: Model transformation chains and model management for end-to-end performance decision support. In: Fernandes, J.M., Lämmel, R., Visser, J., Saraiva, J. (eds.) GTTSE 2009. LNCS, vol. 6491, pp. 345–363. Springer, Heidelberg (2011). doi:[10.1007/978-3-642-18023-1\\_9](https://doi.org/10.1007/978-3-642-18023-1_9)
24. Kang, K.C., Cohen, S.G., Hess, J.A., Novak, W.E., Peterson, A.S.: Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report CMU/SEI-90-TR-021, SEI, Carnegie Mellon University, November 1990
25. Lin, J., Fox, M.S., Bilgic, T.: A requirement ontology for engineering design. *Concurrent Eng.* **4**(3), 279–291 (1996)
26. Lúcio, L., Mustafiz, S., Denil, J., Vangheluwe, H., Jukss, M.: FTG+PM: an integrated framework for investigating model transformation chains. In: Khendek, F., Toeroe, M., Gherbi, A., Reed, R. (eds.) SDL 2013. LNCS, vol. 7916, pp. 182–202. Springer, Heidelberg (2013). doi:[10.1007/978-3-642-38911-5\\_11](https://doi.org/10.1007/978-3-642-38911-5_11)
27. Mijumbi, R., Serrat, J., Gorricho, J.L., Latre, S., Charalambides, M., Lopez, D.: Management and orchestration challenges in network functions virtualization. *IEEE Commun. Mag.* **54**(1), 98–105 (2016)
28. Mustafiz, S., Palma, F., Khendek, F., Toeroe, M.: A network service design and deployment process for NFV systems. In: IEEE NCA16: The 15th IEEE International Symposium on Network Computing and Applications, pp. 131–139. IEEE, October 2016
29. OASIS: TOSCA Simple Profile for Network Functions Virtualization (NFV) Version 1.0, March 2016. <http://docs.oasis-open.org/tosca/tosca-nfv/v1.0/tosca-nfv-v1.0.html>
30. Oldevik, J.: Transformation composition modelling framework. In: Kutvonen, L., Alonistioti, N. (eds.) DAIS 2005. LNCS, vol. 3543, pp. 108–114. Springer, Heidelberg (2005). doi:[10.1007/11498094\\_10](https://doi.org/10.1007/11498094_10)
31. Oracle: Oracle Communications Network Service Orchestration Solution Implementation Guide, Release 1.1. White Paper, July 2016. [https://docs.oracle.com/cd/E71075\\_01/doc.11/e65331/toc.htm](https://docs.oracle.com/cd/E71075_01/doc.11/e65331/toc.htm)
32. Oster, Z.J., Santhanam, G.R., Basu, S.: Decomposing the service composition problem. In: 8th IEEE European Conference on Web Services, pp. 163–170, December 2010
33. Oster, Z.J., Santhanam, G.R., Basu, S.: Identifying optimal composite services by decomposing the service composition problem. In: IEEE International Conference on Web Services. ICWS 2011, pp. 267–274. IEEE Computer Society (2011)
34. Rivera, J.E., Ruiz-Gonzalez, D., Lopez-Romero, F., Bautista, J., Vallecillo, A.: Orchestrating ATL model transformations. In: Proceedings of MtATL 2009, pp. 34–46. Nantes, France, July 2009
35. Sahhaf, S., Tavernier, W., Colle, D., Pickavet, M.: Network service chaining with efficient network function mapping based on service decompositions. In: 1st IEEE Conference on Network Softwarization (NetSoft), pp. 1–5, April 2015
36. Tisi, M., Jouault, F., Fraternali, P., Ceri, S., Béziniv, J.: On the use of higher-order model transformations. In: Paige, R.F., Hartman, A., Rensink, A. (eds.) ECMDA-FA 2009. LNCS, vol. 5562, pp. 18–33. Springer, Heidelberg (2009). doi:[10.1007/978-3-642-02674-4\\_3](https://doi.org/10.1007/978-3-642-02674-4_3)

37. Vanhooff, B., Ayed, D., Baelen, S., Joosen, W., Berbers, Y.: UniTI: a unified transformation infrastructure. In: Engels, G., Opdyke, B., Schmidt, D.C., Weil, F. (eds.) MODELS 2007. LNCS, vol. 4735, pp. 31–45. Springer, Heidelberg (2007). doi:[10.1007/978-3-540-75209-7\\_3](https://doi.org/10.1007/978-3-540-75209-7_3)
38. Wagelaar, D.: Blackbox composition of model transformations using domain-specific modelling languages. In: 1st European Workshop on Composition of Model Transformations (CMT), pp. 15–19 (2006)