Process Enactment with Traceability Support for NFV Systems

Omar Hassane[1], Sadaf Mustafiz[1], Ferhat Khendek[1](✉), and Maria Toeroe[2]

[1] ECE, Concordia University, Montreal, Canada
o_assane@encs.concordia.ca, {sadaf.mustafiz, ferhat.khendek}@concordia.ca
[2] Ericsson Inc., Montreal, Canada
maria.toeroe@ericsson.com

Abstract. The Network Functions Virtualization (NFV) paradigm is heading towards an evolution with the recent zero-touch automation initiative. In particular, automating the orchestration and management of network services (NS) could progress rapidly with the help of model-driven engineering methods and tools. We have earlier proposed an integrated process modelling and enactment environment, MAPLE, for NS management. In our approach, enactment is enabled by transformation chaining and megamodelling. In this paper, we present our extension, MAPLE-T, which incorporates traceability information generation and analysis support in MAPLE. MAPLE-T allows the generation of both local and global traceability information during the enactment of a process model (PM), all of which is retained in the megamodel. The megamodel enables end-to-end navigation of the source and target artifacts in the PM and thus allows advanced traceability analysis to be carried out. We applied MAPLE-T on a NS design process to demonstrate the application of the change impact analysis feature.

Keywords: Process enactment · Megamodelling · Traceability · Network Functions Virtualization (NFV)

1 Introduction

With the advent of 5G, the telecommunications industry is faced with opportunities and challenges which require rapid innovations. Traditional networks have a high dependence on proprietary hardware. Telecoms are moving from such networks to virtualized networks. Telecoms are leveraging the Network Functions Virtualization (NFV) paradigm which is a key enabler for 5G. NFV builds on cloud computing and virtualization technologies which enable the automation of the orchestration and the management of network services [10, 24].

We believe model-driven engineering (MDE) methods and tools can help with the automation. As a first step, we have earlier proposed an approach for explicitly modelling and enacting NFV processes and have applied our work to the NS design and management process [25–27]. Our NS Management Process Model is compliant with the NFV reference framework. MAPLE (MAGIC Process Modelling and Enactment Environment) provides an integrated environment for creating and enacting process models (PM) with the use of model transformation chains. Transformation chaining is the preferred technique for modelling the composition of different model transformations and orchestrating them. MAPLE supports the enactment of heterogeneous (cross-technology) transformation chains based on megamodels used for supporting model management, and on composition of transformations. Megamodels provide complex structures to link all relevant artifacts (models, transformations, and other metadata) forming a map for model management [15, 16]. We have built MAPLE in the Eclipse Papyrus environment [12], which is the modeling environment of choice of the European Telecommunications Standards Institute (ETSI) NFV [13].

Advanced support for discoverability and traceability have been identified as essential features in virtualizing network services [7]. Traceability support enables information recovery, origin tracking (for instance, backtracking from design to requirements artifacts), change impact analysis, change propagation, dependency visualization, and even defect detection and prediction [9, 33]. Traceability management can be effectively carried out with MDE methods and tools [29]. While NFV would greatly benefit from end-to-end traceability support, there has been very little done in this regard in this domain.

In this paper, we extend MAPLE with traceability support for NFV systems. We integrate means for local (transformation-level) and global (process model-level) traceability information generation and also provide the groundwork for change impact analysis. We apply our work in the NFV domain to the traceability analysis of the network service design process in order to assess the impact of changes in the source models. The vendor-provided virtualized network function (VNF) form the core of a network service, and any changes in the VNF descriptors (VNFD) can affect the target artifacts and the process itself. It would be highly beneficial in NFV systems to be able to assess the impact of a change and to provide feedback.

The rest of this paper is structured as follows: Sect. 2 gives a brief background on traceability in MDE. Section 3 proposes a model-driven process enactment approach with traceability support and presents our MAPLE-T environment. Section 4 describes an application of MAPLE-T in the NFV domain. Section 5 discusses related work and Sect. 6 concludes with some future work.

2 Background

Traceability is defined as *the degree to which a relationship can be established between two or more products of the development process, especially products having a predecessor-successor or master-subordinate relationship to one another [2].*

Traceability information in MDE can be classified as *generic* (no semantics retained with trace links) or *specific* (domain-dependent with semantically rich links) [4]. Traceability information can be represented as models conforming to an external metamodel (extra-model traceability), or as part of the traced models (intra-model traceability) thus requiring the metamodels of the traced models to be extended and polluted with trace information [32].

Traceability metamodelling can follow a *pure metamodel* approach or a *tag-based* approach [32]. In the first approach, all the required trace types along with their usage semantics are specified at the metamodel level making the traceability metamodel rigid to change and therefore hard to reuse in other projects. The trace tagging approach uses a general traceability metamodel which can be annotated with specific tags. This allows for more flexible traces that can be used in any project, but with weak usage semantics specified in the metamodel.

When referring to traceability at the model transformation level, the trace links are between the elements of the source and target artifacts associated with the transformation. A trace model is created for each transformation. This is referred to as *local traceability* or *traceability in the small*. However, the link between the different trace models across multiple model transformations (or a model transformation chain) needs to be created to produce *global traceability* (or *traceability in the large*) information. This enables end-to-end navigation throughout a chain of intermediately created trace models [6].

In our work, megamodels are used to build traceability support. Megamodelling is generally used for model management, to provide structures for handling and interrelating models [15, 16]. A megamodel may contain heterogeneous models, relations between them (e.g., transformations) and any other relevant metadata. It can be used to capture conformance links and also to enable compatibility checks.

3 MAPLE-T Approach

We have earlier proposed an integrated process modelling and enactment approach, MAPLE [25]. MAPLE supports process modelling with UML Activity Diagrams. The MAPLE enactment approach is based on model transformation chaining and megamodelling. We have used megamodels (MgM) to manage all the resources needed for the enactment. We begin with a process model (PM) and the repository of resources (metamodels, profiles, model instances, model transformations, programs). In MAPLE, the MgM was used as a resource repository that aggregates and links all these resources as well as their metadata. This was quite useful for gathering and managing the relevant information regarding the transformations that need to be enacted.

An initial megamodel (MgM) is automatically derived based on the resources registered. When the PM is registered in the MgM, it leads to the creation of a *weave model*. This *weave model* binds the PM and the MgM together. The PM is then mapped to a model transformation chain, with the help of the MgM and the *weave model*, and it can be executed using token-based semantics. The MgM is dynamically updated with the generated models during enactment. MAPLE is built on top of Eclipse Papyrus. For further details, the reader can refer to [25]. We propose an extension to our process enactment approach providing traceability support. Our goal is to go further and use the MgM for advanced traceability of model transformation chains. The MAPLE-T approach is shown in Fig. 1. Following the derivation of the megamodel (MgM) and the construction of the model transformation (MT) chain, the chain execution results in the generation of artifacts. During this execution, trace models will also be output (*traceability in the small* context) and will be retained in the MgM in order to build a global traceability map (*traceability in the large* context). The global traceability map can then be used for traceability analysis. As an add-on, the map can also be used as a basis for traceability visualization.

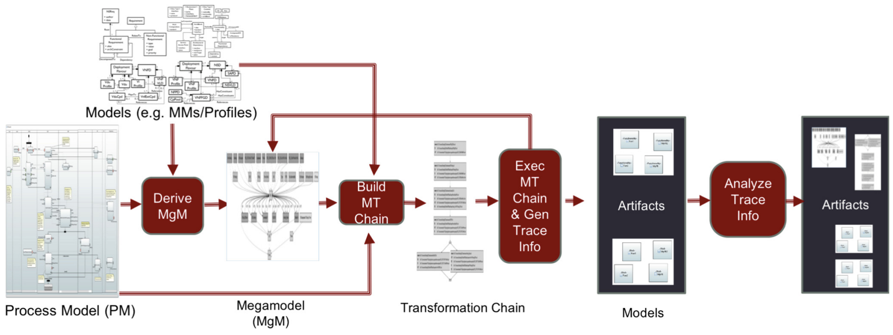


Fig. 1. MAPLE-T approach (Color figure online)

3.1 Traceability Support in MAPLE-T

Traceability support for process enactment can be incrementally built in three stages: (1) traceability information generation, (2) traceability analysis, and (3) traceability visualization. While traceability information generation is required to proceed with any analysis or visualization, the latter two stages can be carried out in parallel. This paper discusses the first two phases: the generation and analysis.

Traceability Generation. The first phase of the project involves building support for the generation of traceability information and global model management in MAPLE. Each of the transformations in the MT chain is first augmented such that trace models are generated as extra artifacts with the execution of each transformation. We refer to these local trace models as LTrace models.

During enactment, this augmented MT chain is executed, hence generating the output as defined in each transformation along with the LTrace model for every transformation. The generated artifacts, including the trace models, are dynamically added to the MgM during enactment. Each LTrace model in the MgM is connected to the relevant input and output models. In addition to the links between input and output models via the LTrace model, the links between the trace models are also saved in the MgM. The links retained are at the model-level as well as at the model-element level. This leads to the creation of the global traceability map within the MgM, which we refer to as the GTrace.

Traceability Analysis. Once we have a repository of traceability information and the global traceability map, the next step is to discover any *useful* trace information that can be analyzed for a given purpose. Whether a piece of trace information is *useful* is typically application-dependent. Discovering such trace links is possible when using the trace tagging method [32]. In the MgM, each LTrace model is associated with a corresponding *tag* representing the context of the traced transformation. The purpose behind this *tag* is to specialize our trace models with application-specific semantics. In MAPLE-T, the notion of *tags* has been incorporated at two levels: at transformation rule level and at transformation level.

3.2 MAPLE-T Functionalities

We describe here the main functionalities provided by MAPLE-T, corresponding to the red boxes in Fig. 1.

To enact a process model (PM), we need to start by creating the PM. We use the Eclipse Papyrus Activity Diagram environment to build PMs. In our work, the PM needs to comply with the ETSI Information Model for NFV [14] as well as the ETSI NFV Papyrus Guidelines [13]. The PM creation phase is out of the scope of this paper. For further details, the reader can refer to [25].

Deriving the MgM. In MAPLE, the actions in the PM are implemented with model transformations. A transformation involves several input and output models, possibly conforming to different metamodels that can be expressed using heterogeneous technologies. Moreover, a PM can be implemented with a heterogeneous set of languages (for instance, ATL, Epsilon, Java, or C), and hence MAPLE supports execution of cross-technology model transformation chains. Due to this, deploying model management techniques is essential in MAPLE. As described in [25], we use megamodels for this purpose. While megamodels have been very useful in MAPLE for managing resources and for enacting the PM, we wanted to go further and use the MgM for advanced traceability support. In MAPLE-T, we have introduced new traceability-related features in the MgM as well as new extensions with respect to the implementation of these features. To enable traceability at both levels (local and global), our MgM now supports storing the same resources with different versions over time, i.e. whenever they are being used or changed. The MgM also retains model instances per enactment. In addition to that, each transformation resource is now linked with a trace model - representing local traceability (LTrace models). This LTrace model contains the trace links for each input and output of the transformation execution and conforms to an LTrace metamodel which represents local traceability information elements both at the model element-level and at the attribute-level. The main elements in the LTrace are mentioned below and also shown in Fig. 2.

- TraceLinkSet: This represents the set of all the traced rules of a transformation execution as well as all the trace links linking input and output elements of the traced models.
- TracedRule: This represents the rule responsible for creating/transforming the traced output model element(s) from the corresponding traced input model element(s).
- TraceLink: This represents the set of input elements and their corresponding output elements within a rule.

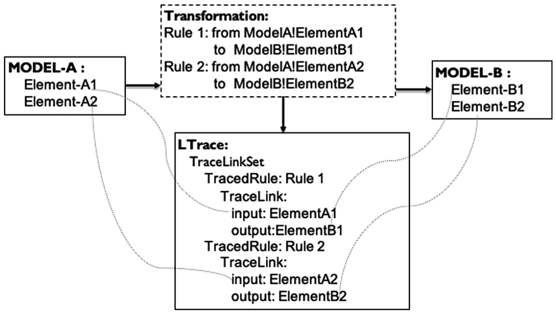


Fig. 2. LTrace structure

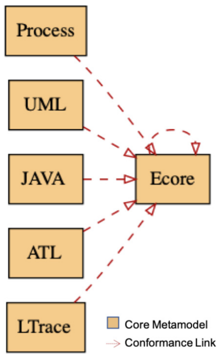


Fig. 3. Base MgM

The MgM is derived after registering the resources and the PM. First, a base MgM is loaded as part of MAPLE-T environment and consists of the metamodels of the built-in loaders (needed to load resources) and the pre-loaded meta-metamodels (e.g., Ecore) and their conformance links. This MgM is incrementally updated by registering the different resources which are part of the project (metamodels/profiles). This is carried out automatically by going through the project workspace (referred to as *workspace discovery*), and as a result an initial MgM is derived at this stage. A base LTrace metamodel conforming to the Ecore metamodel is also registered in the MgM (see Fig. 3). Each trace model generated as a byproduct of a transformation execution conforms to this metamodel.

As the next step, the MgM is refined by carrying out a *PM discovery*. This involves updating the MgM with new resources: the PM and the associated transformations. Since we wanted the MgM to be PM-agnostic, a *weave model* is automatically created behind the scenes whenever a PM is registered. The *weave model* binds all the necessary elements of the PM to their equivalent resources in the MgM.

At this point, the MgM holds all the essential resources which are required for enactment. During enactment, the LTrace models generated are added to the MgM which makes it possible to construct the GTrace (part of the MgM).

Building the Transformation Chain. The PM is given translational semantics by mapping it to a transformation chain. The chain is in essence a schedule with the required details (sequence of actions, transformations used, inputs and outputs of the transformations). This allows us to build a generic enacter, instead of having an enacter for each kind of PM. Having a generic enacter also leaves scope for integrating other formalisms for modelling the PM.

The translation from a PM to an MT chain is implemented using an ATL transformation which takes relevant inputs (including the MgM and the PM) and produces the target transformation chain.

This phase of the process has no traceability-related extensions. It would be possible to augment the transformation chain to build a chain with traceability support. The reason we did not proceed in that direction was to provide more flexibility and let the user enable or disable traceability within MAPLE-T during enactment. Otherwise, we would end up with a solution which always generates traceability information as a result of the enactment, which might not be always desirable, as generating trace information might be unnecessarily cumbersome and time-consuming in some applications.

Executing the Transformation Chain and Generating the Trace Models. In MAPLE-T, a PM is enacted by executing the underlying MT chain. Similar to UML Activity Diagrams, the generated chain is given token-based semantics. Therefore, the enacter developed is based on controlling the tokens and activating the actions when needed.

However, in order to support both local and global traceability, it was necessary to integrate means to generate local traces of the transformations chain execution. Additionally, these trace models are linked in the MgM to construct the global traceability map. The MgM also needs to be updated with these new model instances and their relationships.

Generating Trace Models. One of the issues we had to address when building a traceability solution with MAPLE-T was how to actually generate traceability information during enactment. One might consider a naive approach in which each model transformation implementing an action in the PM is refined manually with new traceability-related rule bindings or blocks of code. In such a case, each transformation will need to be manually modified to generate new target models for the trace information. This approach is clearly not ideal, as extensively refining every transformation manually results in a very cumbersome process that is in total opposition of our main vision, which is full automation. For this reason, we adopted an approach that augments the transformation chain automatically with traceability information (see Fig. 4). Similar to [22] we used the well-known concept of higher order transformation (HOT), and we defined an HOT to systematically enrich our transformations with traceability notions. The HOT takes the transformations parts of the chain and augments them, resulting in a new chain which has the same flow but with traceability-augmented transformations. Each transformation ends up having in addition to its original input/output parameters, a new target parameter representing the trace model to be generated - the LTrace model.

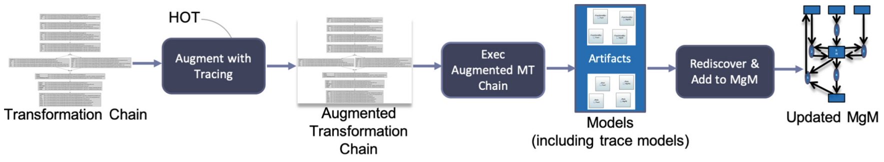


Fig. 4. MAPLE-T traceability generation approach

While MAPLE provides enactment support for a heterogeneous set of transformation languages (e.g., ATL, Java), MAPLE-T only supports implementations with ATL transformations at the moment. The HOT implementation augments transformation models conforming to the ATL metamodel. Also, the LTrace metamodel is built to represent trace models produced from the execution of the augmented ATL transformations.

Updating the MgM. During enactment, the MgM is updated on the fly with the augmented transformation executions and their corresponding input/output instances including the LTrace models. Once the enactment is done, the MgM is completely updated with all the newly generated artifacts. At this point, the MgM also provides a global traceability map, the GTrace. The set of global links as well as the local traces generated for each transformation form the basis for carrying out traceability analysis in MAPLE-T.

Analyzing Traceability Information. Following the generation of traceability information, traceability analysis can be carried out on the basis of the GTrace. For this purpose, we have incorporated means to analyze trace information within MAPLE-T which can be easily extended and adapted to the targeted application domain. We have built a core traceability analysis solution that exposes common traceability analysis features (via an API). The exposed features allow the generated LTrace models and GTrace links to be parsed and manipulated, typically with the use of the rule-level and transformation-level tags that provide richer semantics for the analysis.

We have incorporated traceability analysis support, specifically to carry out *change impact analysis* in MAPLE-T, which relies on the proposed traceability generation means. The change impact analysis is triggered by a request specifying the element or the model for which the change impact is to be analyzed. The purpose is to determine how impactful a model or an element is on the whole process (i.e., how impactful it is on the other involved models, model elements, and transformations) at both the metamodel and the model levels. The process starts first by filtering all the relevant information from the GTrace and LTrace models based on what was provided as input at the metamodel level. Based on this, we can conclude whether the input is impactful or impactless at the metamodel level. In case it is *impactless at the metamodel level*, then it is inferred to be *impactless at the model level* as well. In this case, it is concluded that the input model or model element is *impactless at both levels* and no further analysis is required. On the other hand, if the input model or model element turns out to be impactful at the metamodel level, then the decision is not as straightforward as in the previous case. MAPLE-T then continues to analyze the gathered traceability information (LTrace models and GTrace links) at the model level as well. As a result, the impact decision is further categorized into two outcomes.

– The input is *impactful at the model level*: This means that the provided input has been used in the enacted process and changing it requires re-enactment. Additionally, the solution collects the set of all the impacted resources (models, model elements, and transformations) and provides them as outputs of the change impact analysis along with the impact decision.

– The input is *impactless at the model level*: This means that although the type of the input model/element has an impact on the enacted process, the actual input model/element instance has never been used and has no impact on that specific enactment.

4 NFV Application

In this section, we use MAPLE-T to enact an NS design process and to carry out traceability analysis, in particular, a change impact analysis. The process is a subset of the NS Design and Deployment PM proposed earlier in [27].

A network service is a composition of network function(s) (NF) and/or other nested NSs to provide a desired functionality/behaviour (e.g. VoIP). An NF is a functional block identified by well-defined functional behaviour and external interfaces. NFs within an NS can be a physical NF (PNF) (e.g. a traditional firewall device) or a virtual network function (VNF) (e.g. a virtual firewall) decoupled from the infrastructure and implemented as software that can be deployed on a virtualized infrastructure. The different NFs/nested NSs within an NS are interconnected with one or more forwarding graphs (FG) that define the traffic flows between them.

The main goal behind the NS design process (proposed in [26]) is to automatically design an NS and generate an NS Descriptor (NSD) which is a template used for the deployment and management of the NS. The process starts by specifying the functional and non-functional characteristics of the NS as the NS requirements (NSReq). The functionalities in the NSReq are then decomposed with the help of an NF ontology (NFOntology) which represents a knowledge-base capturing known NF decompositions and their architectures. After decomposition to a certain level, VNFs are selected from a catalog (VNFCatalog) by matching the decomposed functionalities. The traffic flows in the NS are then defined with the design of the VNF FGs (VNFFGs) and the NS dimensioned according to the non-functional requirements. The NFOntology may be updated with new information from NSReq after a successful design, with the onboarding of new VNFs, and manually by an expert. A VNF is described by a VNFD which captures all its deployment characteristics. One main element within the VNF is a VNF component (VNFC) which represents an internal component of the VNF that provides part of the VNF functionality. A Virtual Deployment Unit (VDU) is the deployment template or descriptor of the VNFC and it is an element contained within the VNFD. The generated NSD is compliant to ETSI NFV definition and refers to the NS constituent descriptors including VNFDs and VNFFG descriptors (VNFFGDs). For a detailed description of the NS Design process, the reader can refer to [26]. Figure 5 presents the NS design PM.

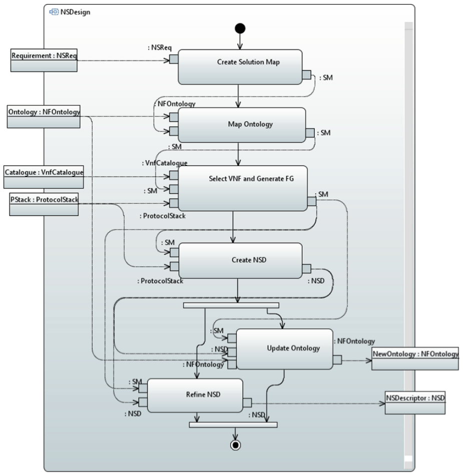
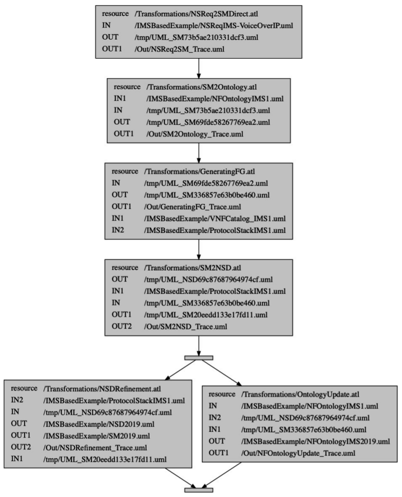


Fig. 5. NS design PM [25]



resource	/Transformations/NSReq2SMDirect.atl
IN	/IMSBasedExample/NSReqIMS-VoiceOverIP.uml
OUT	/tmp/UML_SM73b5ae210331dcf3.uml
OUT1	/Out/NSReq2SM_Trace.uml

resource	/Transformations/SM2Ontology.atl
IN1	/IMSBasedExample/NFOntologyIMS1.uml
IN	/tmp/UML_SM73b5ae210331dcf3.uml
OUT	/tmp/UML_SM69fde58267769ea2.uml
OUT1	/Out/SM2Ontology_Trace.uml

resource	/Transformations/GeneratingFG.atl
IN	/tmp/UML_SM69fde58267769ea2.uml
OUT	/tmp/UML_SM336857e63b0be460.uml
OUT1	/Out/GeneratingFG_Trace.uml
IN1	/IMSBasedExample/VNFCatalog_IMS1.uml
IN2	/IMSBasedExample/ProtocolStackIMS1.uml

resource	/Transformations/SM2NSD.atl
OUT	/tmp/UML_NSD69c87687964974cf.uml
IN1	/IMSBasedExample/ProtocolStackIMS1.uml
IN	/tmp/UML_SM336857e63b0be460.uml
OUT1	/tmp/UML_SM20eedd133e17fd11.uml
OUT2	/Out/SM2NSD_Trace.uml

resource	/Transformations/NSDRefinement.atl
IN2	/IMSBasedExample/ProtocolStackIMS1.uml
IN	/tmp/UML_NSD69c87687964974cf.uml
OUT	/IMSBasedExample/NSD2019.uml
OUT1	/IMSBasedExample/SM2019.uml
OUT2	/Out/NSDRefinement_Trace.uml
IN1	/tmp/UML_SM20eedd133e17fd11.uml

resource	/Transformations/OntologyUpdate.atl
IN	/IMSBasedExample/NFOntologyIMS1.uml
IN2	/tmp/UML_NSD69c87687964974cf.uml
IN1	/tmp/UML_SM336857e63b0be460.uml
OUT	/IMSBasedExample/NFOntologyIMS2019.uml
OUT1	/Out/NFOntologyUpdate_Trace.uml

Fig. 6. Augmented MT chain

4.1 Enactment and Traceability Generation with MAPLE-T

In order to enact the NS Design PM, we need to register all the needed resources/profiles (NSReq, NSD profiles, etc.). As a result, the base MgM (Fig. 3) is updated with all the registered UML profiles as well as conformance links. Figure 7 shows the initial MgM with UML profiles. Next, we need to register the PM which automatically registers all the underlying model transformations implementing the actions in the PM. Consequently, the MgM is updated (see Fig. 8) with the following: (1) a new resource representing the NS Design PM as a UML activity diagram conforming to the UML metamodel (shown in gray), (2) the ATL transformations conforming to the ATL metamodel (shown in brown), and also (3) the weave model containing the MgM and PM mappings (shown in gray). With this MgM, MAPLE-T has all the necessary resources to enact the PM, and therefore enable NS Design traceability generation and analysis.

Once all the model instances are specified, an initial transformation chain is built based on the NS design PM. This transformation chain is then augmented so that each transformation is able to generate LTrace model instances in addition to its original output model instance(s) (see Fig. 6).

The execution of this MT chain includes six augmented transformation executions. The first transformation starts by taking the NSReq model as input and generates an initial intermediate model as well as the LTrace model corresponding to that transformation execution. In the same way, the execution process continues according to the order defined in the MT chain. For each subsequent transformation execution, the

LTrace model is generated and the intermediate model incrementally refined until we end up with our desired models: NSD and updated NFOntology.

The MgM is updated during enactment with actual model instances (see Fig. 8). The LTrace model instances along with the global links interconnecting them are also added to the MgM. This results in the construction of our NS Design GTrace. The subset of the MgM representing the NS Design GTrace is shown in Fig. 9. LTrace models (e.g., NSReq2SM Trace, SM2NSD Trace) are shown in blue and their interconnections are shown with blue dashed links. Each LTrace model (output of a transformation) is linked with its corresponding model transformation with an object flow link (solid black line).

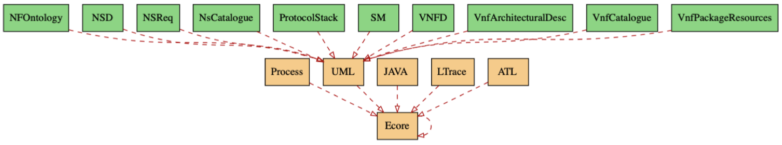


Fig. 7. Initial NS design MgM

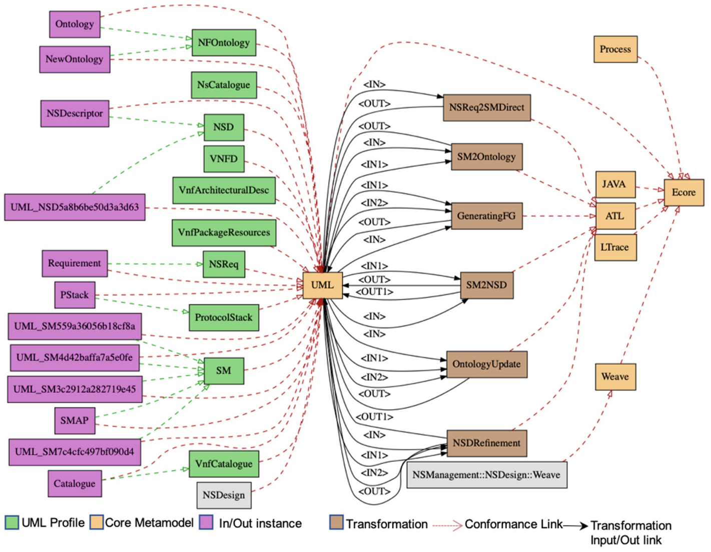


Fig. 8. Updated NS design MgM (Color figure online)

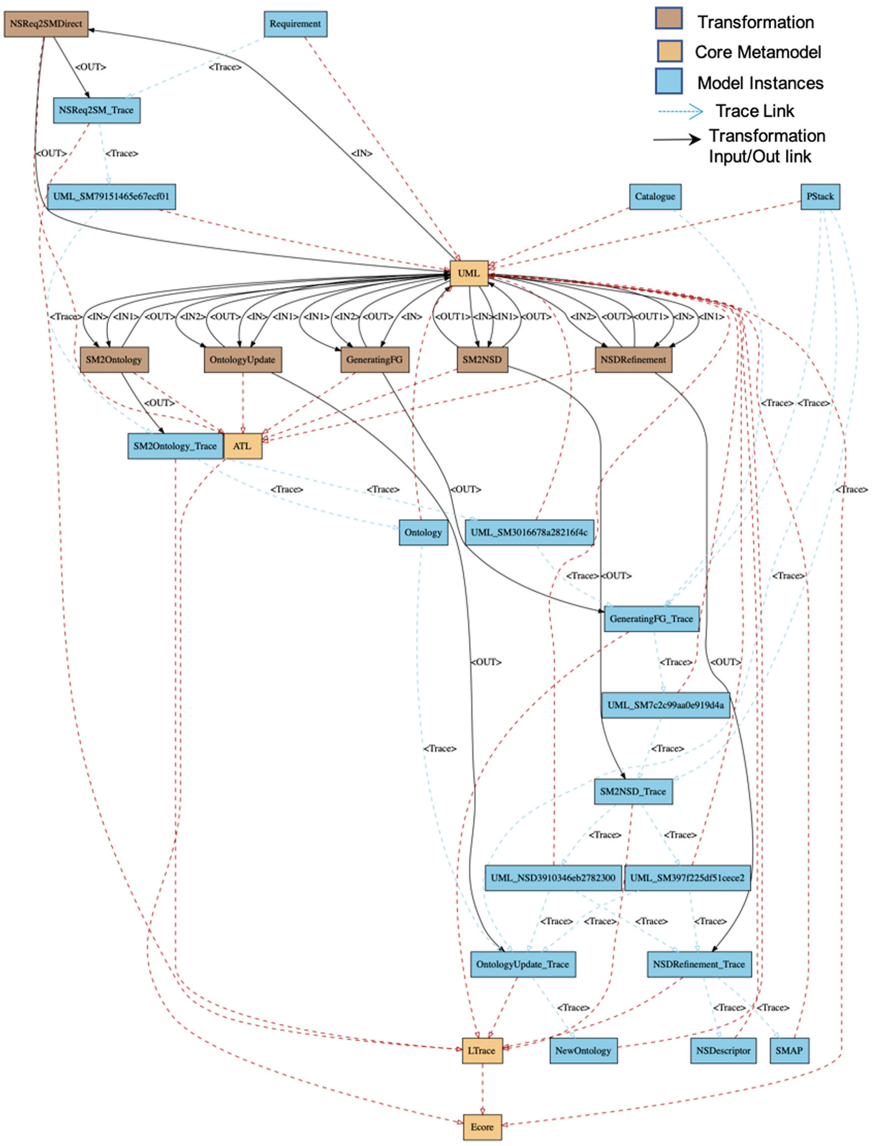


Fig. 9. NS design GTrace (Color figure online)

4.2 Traceability Analysis with MAPLE-T

Now that all the NS Design models are interlinked via LTrace models and GTrace links, we can automatically trace back and forth between all the involved source and target resources i.e.; NSReq, Ontology, the VNFCatalog and its constituent VNFDs as well as the resulting NSD and the updated Ontology. Each LTrace model enables navigation at the element level of adjacent models. Additionally, the GTrace enables navigation at the

PM level, which means that we can explicitly navigate between distant models as well. For example, we can directly trace back from the NSD (last element of the NS design process) to the NSReq (first element of the NS design process).

Because of the foundation set by the local and global traces, it is relatively straightforward to incorporate the *change impact analysis* in MAPLE-T. We can automatically figure out how changing an element of an input model (NSReq, VNFDs included in the

VNFCatalog, or the NFOntology) can impact the NS Design transformations and the target (e.g., NSD) models and their elements. Using MAPLE-T, the user selects the input element for which the change impact is to be determined. The user will then be provided with the result showing whether the selected element is impactful or not, and if applicable, a list of all the impacted transformations and models as well as their elements is provided.

Typically, the VNFPackage is provided by vendors and might be subject to change. In our case study, we focus specifically on the impact induced by changing a resource within the VNFPackage, mainly the VNFD. After an NS is deployed, a VNF vendor might point out that a parameter or set of parameters in a VNFD within the catalog is erroneous (not describing the VNF properly). In such a scenario, the decision on going about making a change in the design process and associated artifacts depends on whether considering the error the running NS instance still is behaving according to the requirements (NSReq) or not. With our traceability analysis, we can determine whether the erroneous parameters have an impact on the NS design process and therefore the generated NSD. This will allow us to evaluate if the running NS instance cannot meet the NSReq due to the error (e.g. if the erroneous VNFD parameters have an impact on the design and therefore on the generated NSD) or not, and whether the NS should be re-designed and re-deployed. In the rest of this section, we discuss both scenarios. In this analysis, we assume that our NS design approach, the NSReq and the NFOntology are correct and cannot be the source of errors.

Scenario 1: NS Instance is Behaving According to the Requirements (NSReq). In this scenario, the assumption is that the NS instance is running as expected according to the NSReq, no issues have been detected (yet). However, at some point in time, the VNF vendor indicates that a VNFD involved in the NS design was not correctly describing the VNF and its instance is used now within the running NS instance.

This implies that some VNFD parameters are erroneous and need to be changed. The decision of re-designing and re-deploying the NS depends on whether these parameters have an impact on the NSD.

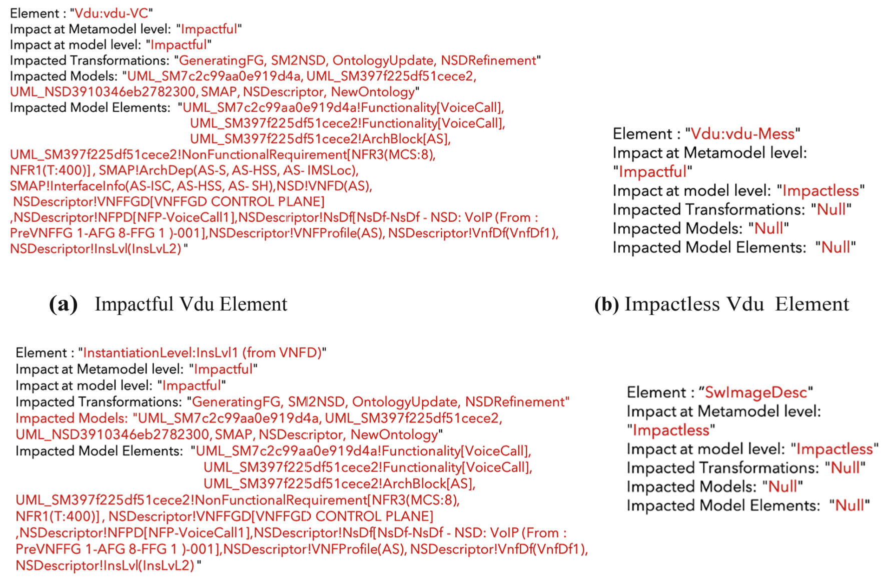
Parameters are Impactless at the Metamodel Level: In this case, since the erroneous parameters have no impact on the design and the NS instance is behaving as expected according to the NSReq, there is no action to take. For instance, the vendor might point out that the software image descriptor (SwImageDesc) used in the VNFD is erroneous. After analyzing the change impact of the SwImageDesc element on the NS Design process, it turns out that it is impactless as shown in Fig. 10(d) since it is never considered in the design process. Changing this element will never impact the generated NSD, and therefore there is no need to re-design or re-deploy the NS.

Parameters are Impactful at the Metamodel Level: In this case, the impact at the model level should be considered.

– Parameters are *impactful at the model level*: As opposed to the previous case, we need to consider re-designing the NS even though it is running as expected according to NSReq. In this case, the erroneous parameters were used to design the NS and therefore they are impactful. For example, the vendor might report that an Instantiation Level element (which specifies the number of instances of each VNFC within the VNF) within the VNFD is erroneous and needs to be corrected. The change impact analysis of this element finds it impactful (e.g., as shown in Fig. 10 (c)). This means that, while the NS instance shows no problem (yet), this does not preclude the possibility that the provisioning of VNFC instances is not done inefficiently (e.g. VNFC instances may be over-provisioned) and/or incorrectly (e.g. the parameter value may not have been used yet), and therefore the re-design of the NS needs to be considered in this case.
– Parameters are *impactless at the model level*: In this case, since the parameters are impactful at the metamodel level but not at the model level, it is not straightforward to conclude whether the re-enactment of the NS Design is needed or not. A new parameter value might make a previously impactless parameter impactful after the change. Using the generated traces to analyze the impact of such parameters might provide a *false negative* result, in the sense that the impact analysis will suggest that changing the parameter would be impactless, even though it is not the case. For instance, the vendor might indicate that the name of a Vdu element referenced in the VNFD is erroneous. As shown in Fig. 10(b), the analysis of the change impact of the Vdu name parameter on the NS Design process suggests that it is impactless. However, the reason may be that the Vdu with the incorrect name was not selected because it did not meet the requirements. On the other hand, the correct Vdu name might point to a VDU, which meets the requirements making the parameter impactful at the model level as well. In this case if we re-run the NS design enactment with the changed parameter value and generate new traces, our change impact analysis will suggest that this element is impactful. Thus, at this point, no conclusion can be made in this case from the analysis and it is better to re-enact the NS Design with the changed parameters.

Scenario 2: NS is Not Meeting the Requirements (NSReq). In this scenario the NS instance is not behaving as expected according to the NSReq. Similar to the previous scenario, the VNF vendor indicates that a provided VNFD is erroneous and requires changes. Using MAPLE-T, we can try to determine if the erroneous behaviour of the NS instance is due to the erroneous VNFD or not.

Parameters are Impactless at the Metamodel Level: Since the erroneous parameters of the VNFD are impactless (case shown in Fig. 10(d)), we can conclude that the erroneous behaviour of the NS instance is not due to the erroneous VNFD.



```
Element : "Vdu:vdu-VC"
Impact at Metamodel level: "Impactful"
Impact at model level: "Impactful"
Impacted Transformations: "GeneratingFG, SM2NSD, OntologyUpdate, NSDRefinement"
Impacted Models: "UML_SM7c2c99aa0e919d4a, UML_SM397f225df51cece2,
UML_NSD3910346eb2782300, SMAP, NSDescriptor, NewOntology"
Impacted Model Elements:  "UML_SM7c2c99aa0e919d4a!Functionality[VoiceCall],
                           UML_SM397f225df51cece2!Functionality[VoiceCall],
                           UML_SM397f225df51cece2!ArchBlock[AS],
UML_SM397f225df51cece2!NonFunctionalRequirement[NFR3(MCS:8),
NFR1(T:400)] , SMAP!ArchDep(AS-S, AS-HSS, AS- IMSLoc),
SMAP!InterfaceInfo(AS-ISC, AS-HSS, AS- SH),NSD!VNFD(AS),
 NSDescriptor!VNFFGD[VNFFGD CONTROL PLANE]
,NSDescriptor!NFPD[NFP-VoiceCall1],NSDescriptor!NsDf[NsDf-NsDf - NSD: VoIP (From :
PreVNFFG 1-AFG 8-FFG 1 )-001],NSDescriptor!VNFProfile(AS), NSDescriptor!VnfDf(VnfDf1),
NSDescriptor!InsLvl(InsLvL2) "
```

(a) Impactful Vdu Element

```
Element : "Vdu:vdu-Mess"
Impact at Metamodel level:
"Impactful"
Impact at model level: "Impactless"
Impacted Transformations: "Null"
Impacted Models: "Null"
Impacted Model Elements:  "Null"
```

(b) Impactless Vdu Element

```
Element : "InstantiationLevel:InsLvl1 (from VNFD)"
Impact at Metamodel level: "Impactful"
Impact at model level: "Impactful"
Impacted Transformations: "GeneratingFG, SM2NSD, OntologyUpdate, NSDRefinement"
Impacted Models: "UML_SM7c2c99aa0e919d4a, UML_SM397f225df51cece2,
UML_NSD3910346eb2782300, SMAP, NSDescriptor, NewOntology"
Impacted Model Elements:  "UML_SM7c2c99aa0e919d4a!Functionality[VoiceCall],
                           UML_SM397f225df51cece2!Functionality[VoiceCall],
                           UML_SM397f225df51cece2!ArchBlock[AS],
UML_SM397f225df51cece2!NonFunctionalRequirement[NFR3(MCS:8),
NFR1(T:400)] , NSDescriptor!VNFFGD[VNFFGD CONTROL PLANE]
,NSDescriptor!NFPD[NFP-VoiceCall1],NSDescriptor!NsDf[NsDf-NsDf - NSD: VoIP (From :
PreVNFFG 1-AFG 8-FFG 1 )-001],NSDescriptor!VNFProfile(AS), NSDescriptor!VnfDf(VnfDf1),
NSDescriptor!InsLvl(InsLvL2) "
```

(c) Impactful Instantiation Level Element

```
Element : "SwImageDesc"
Impact at Metamodel level:
"Impactless"
Impact at model level: "Impactless"
Impacted Transformations: "Null"
Impacted Models: "Null"
Impacted Model Elements:  "Null"
```

(d) Impactless SwImageDesc Element

Fig. 10. VNFD change impact results in MAPLE-T

It might be due to other NS management activities (instantiation, configuration, etc.), but the error did not originate from the VNFD parameters used in the design.

Parameters are Impactful at the Metamodel Level: Similar to the first scenario, we also consider the impact at the model level.

– Parameters are *impactful at the model level* (shown by the example in Fig. 10(c)): This means that the generated NSD is erroneous. Thus, we can infer that the misbehaviour is possibly due to the incorrectly-designed NS, which was due to input errors (in the VNFDs). One needs to re-design the NS and re-deploy it.
– Parameters are *impactless at the model level*: As discussed in the first scenario, this case is inconclusive. Even if the change impact analysis suggests that the parameters are impactless, we cannot know if this result is accurate or if it is a *false negative*. The only way we can determine this is to re-enact and generate new traces (but that is what we are trying to avoid in the first place).

A summary of the two scenarios, their different cases, and analysis results is shown in Table 1.

Table 1. Summary of VNFD change impact analysis results

Impact decision	Running NS instance	
	No problem has been detected	Problems have been detected
Impactless at both metamodel and model levels	No re-design is required	NS instance misbehaviour does not originate from the parameter error, no re-design is required
Impactful at both metamodel and model levels	NS needs to be re-designed (e.g., over-provisioning)	NS instance misbehaviour may originate from the parameter error. NS needs to be re-designed
Impactful at metamodel level and impactless at model level	Inconclusive, NS re-design needs to be considered	Inconclusive, NS re-design needs to be considered

In this section we considered only one NS and analyzed the impact of erroneous VNFDs on its design and the behavior of its instances. The same analysis applies similarly to all NSs in which the erroneous VNFDs are involved. Moreover, one may undertake the huge task of analyzing all designed NSs including NSs where the VNFDs are not involved as this could be the result of exclusion due to the erroneous VNFDs. This is along the same lines as reconsidering the design of any NS once a new VNF is made available, but this might be unrealistic.

5 Related Work

We have covered the state of the art on model-driven enactment support for NFV systems in [25]. Although there exists some work on model-based approaches in the NFV literature, to the best of our knowledge there is currently no published work on model-based traceability generation or/and change impact analysis for NFV systems.

In this section, we discuss some MDE approaches and projects related to process enactment, transformation chaining and model management with traceability generation and change impact analysis support.

5.1 Traceability Generation Support

There has been a lot of work done on traceability in MDE, and these are discussed and summarized in [1, 4, 19, 30, 33]. We only discuss here approaches that specifically address traceability generation and/or analysis in the context of model management, process enactment and model transformation chains.

Fritzsche et al. [16, 17] and Jouault et al. [23] have proposed approaches similar to ours in terms of using model transformation chaining and/or model management via megamodelling to enable traceability. The former combines both techniques and proposes automatic generation of trace models as byproducts of the execution of augmented ATL transformations. However, the generated trace models lack in details, since both the higher-order transformation and the corresponding traceability metamodel used are very basic and do not cover more granular trace information. While the latter work constructs model element-level traces (referred to as LTraces in our work) and model-level traces (links within the GTrace in our case) within the megamodel, no explicit support is provided with regards to process enactment nor automatic augmentation of transformation chains with traceability information.

von Pilgrim et al. [28] extend UNiTI [31] (an Eclipse-based tool to construct, reuse and execute transformation chains) with traceability generation support. Although they assume that the transformations explicitly generate trace models as target models, they do not mention anything about how the transformations are augmented (manually by the developer or automatically using a HOT).

In the MegaM@Rt2 ECSEL project [3], they attempt to use a traceability management approach with megamodels in order to handle and link runtime artifacts with their corresponding design artifacts. The generated trace models conform to a traceability metamodel which is much more generic than our LTrace metamodel in terms of the generated trace links. In our case, trace links are contained within model transformation rules (TracedRules). This gives us a more detailed view not only of what source and target elements are linked but also in which rule at the implementation level this trace link has been constructed. Moreover, to the best of our knowledge, no support for transformation chaining nor process enactment was proposed as part of their documents.

Beyhl et al. [8] presents a framework for retaining and maintaining traceability links between the artifacts within a hierarchical megamodel. However, no support for linking distant artifacts using global traceability links has been mentioned in their approach.

Other work exist which focuses solely on generating local traces as a result of transformation executions [5, 11, 21, 22, 34] and are not elaborated here.

5.2 Traceability Analysis Support

There has been extensive work carried out on change impact analysis in the requirements engineering community [20, 30, 33]. However, these approaches do not support process enactment and megamodelling techniques.

van Amstel et al. [5] propose TraceVis, a tool which uses traces to visualize the relationships between traced models. Using their generated traceability visualization, change impact analysis can be implicitly (manually) inferred from the visualization results, but no method or approach has been proposed to automatically analyze the change impact using the generated traces.

Fung et al. [18] presents MMINT-A, a tool built on top of a model management framework (MMINT) using megamodels, which identifies the impact of software system changes on their assurance cases. However, it is not clear whether their megamodel has traceability extensions enabling navigation between artifacts at the global and local levels.

6 Conclusion

In this paper, we presented MAPLE-T, a model-driven traceability information generation and analysis environment built on top of MAPLE [25], an extensible environment which enables model-driven process enactment by interleaving transformation chaining and model management means. MAPLE-T provides support for automatic generation of local and global traceability information during process enactment. Our approach starts with a PM and a set of resources (metamodels, profiles), which are all registered in an MgM. The PM is then mapped to a transformation chain with the help of the MgM. When process enactment begins, the transformation chain is augmented with traceability support on the fly. During enactment of the PM, MAPLE-T executes the underlying transformations and generates the target models as well as the trace models (transformation traces). Trace links are generated both at the model-level and at the model element-level. The generated artifacts are retained in the MgM. The global trace map (provides traceability information at the PM-level) is also part of the MgM.

We have applied our approach on an NFV case study, namely on the NS design, to carry out change impact analysis. The goal was to assess whether changes in the building blocks of a network service, the VNFs, have any impact on the process and the generated deployment templates. As future work, we intend to use MAPLE-T for traceability analysis of the NS design, deployment, and management process.

Acknowledgement. The authors would like to thank Navid Nazarzadeoghaz for the discussions on the NFV application. This work is partly funded by the Natural Sciences and Engineering Research Council (NSERC) of Canada and Ericsson.

References

1. D4.1: Foundations for model management and traceability. Technical report, MegaM@Rt2, September 2017
2. ISO/IEC/IEEE International Standard - Systems and Software Engineering – Vocabulary. ISO/IEC/IEEE 24765:2017(E), pp. 1–541, August 2017
3. D4.3: Model and Traceability Management (MTM) Tool Set – Intermediate version. Technical report, MegaM@Rt2, November 2018
4. Aizenbud-Reshef, N., Nolan, B.T., Rubin, J., Shaham-Gafni, Y.: Model traceability. IBM Syst. J. **45**(3), 515–526 (2006)
5. van Amstel, M.F., van den Brand, M.G.J., Serebrenik, A.: Traceability visualization in model transformations with TraceVis. In: Hu, Z., de Lara, J. (eds.) ICMT 2012. LNCS, vol. 7307, pp. 152–159. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-30476-7_10
6. Baelen, S.V., Vanhoof, B.: MARTES: Traceability management toolset D2.3. Technical report, EUREKA - ITEA 04006, September 2007
7. Basilier, H., Darula, M., Wilke, J.: Virtualizing network services - the telecom cloud. Ericsson Technol. Rev. **91**, 1–9 (2014). https://www.ericsson.com/en/ericsson-technology-review/archive/2014/virtualizing-network-services—the-telecom-cloud
8. Beyhl, T., Hebig, R., Giese, H.: A model management framework for maintaining traceability links. In: Software Engineering 2013 – Workshopband, pp. 453–457 (2013)

9. Borg, M., Runeson, P., Ardö, A.: Recovering from a decade: a systematic mapping of information retrieval approaches to software traceability. Empir. Softw. Eng. **19**(6), 1565–1616 (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. Eclipse: ATL EMF Transformation Virtual Machine (ATL EMFTVM). https://wiki.eclipse.org/ATL/EMFTVM
12. Eclipse: Papyrus. https://eclipse.org/papyrus/
13. ETSI: Network Functions Virtualisation (NFV) Release 2; Information Modeling; Papyrus Guidelines: ETSI GR NFV-IFA 016 V2.1.1, March 2017
14. 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
15. Favre, J.M., Nguyen, T.: Towards a megamodel to model software evolution through transformations. Electron. Notes Theor. Comput. Sci. **127**(3), 59–74 (2005)
16. Fritzsche, M., Brunelière, H., Vanhooff, B., Berbers, Y., Jouault, F., Gilani, W.: Applying megamodelling to model driven performance engineering. In: 16th IEEE, ECBS 2009, pp. 244–253, April 2009
17. Fritzsche, M., Johannes, J., Zschaler, S., Zherebtsov, A., Terekhov, A.: Application of tracing techniques in model-driven performance engineering. In: 4th ECMDA Traceability Workshop, pp. 1–10 (2008)
18. Fung, N.L.S., Kokaly, S., Di Sandro, A., Salay, R., Chechik, M.: MMINT-A: a tool for automated change impact assessment on assurance cases. In: Gallina, B., Skavhaug, A., Schoitsch, E., Bitsch, F. (eds.) SAFECOMP 2018. LNCS, vol. 11094, pp. 60–70. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99229-7_7
19. Galvao, I., Goknil, A.: Survey of traceability approaches in model-driven engineering. In: IEEE EDOC 2007, p. 313, October 2007
20. Göknil, A., Ivanov, I., van den Berg, K.: Change impact analysis based on formalization of trace relations for requirements. In: ECMDA Traceability Workshop (ECMDA-TW), pp. 59–75. No. 274, SINTEF Report, June 2008
21. Guana, V., Stroulia, E.: ChainTracker, a model-transformation trace analysis tool for code-generation environments. In: Di Ruscio, D., Varró, D. (eds.) ICMT 2014. LNCS, vol. 8568, pp. 146–153. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08789-4_11
22. Jouault, F.: Loosely coupled traceability for ATL. In: ECMDA Workshop on Traceability, pp. 29–37 (2005)
23. Jouault, F., Vanhooff, B., Bruneliere, H., Doux, G., Berbers, Y., Bézivin, J.: Inter-DSL coordination support by combining megamodeling and model weaving. In: ACM 25th SAC 2010, pp. 2011–2018, March 2010
24. 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)
25. Mustafiz, S., Dupont, G., Khendek, F., Toeroe, M.: MAPLE: An integrated environment for process modelling and enactment for NFV systems. In: Pierantonio, A., Trujillo, S. (eds.) ECMFA 2018. LNCS, vol. 10890, pp. 164–178. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-92997-2_11
26. Mustafiz, S., Nazarzadeoghaz, N., Dupont, G., Khendek, F., Toeroe, M.: A model-driven process enactment approach for network service design. In: Csöndes, T., Kovács, G., Réthy, G. (eds.) SDL 2017. LNCS, vol. 10567, pp. 99–118. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68015-6_7

27. Mustafiz, S., Palma, F., Toeroe, M., Khendek, F.: A network service design and deployment process for NFV systems. In: 15th IEEE NCA 2016, pp. 131–139. IEEE Computer Society (2016)
28. von Pilgrim, J., Vanhooff, B., Schulz-Gerlach, I., Berbers, Y.: Constructing and visualizing transformation chains. In: Schieferdecker, I., Hartman, A. (eds.) ECMDA-FA 2008. LNCS, vol. 5095, pp. 17–32. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-69100-6_2
29. Santiago, I., Jiménez, A., Vara, J.M., De Castro, V., Bollati, V.A., Marcos, E.: Model-driven engineering as a new landscape for traceability management: A systematic literature review. Inf. Softw. Technol. **54**(12), 1340–1356 (2012)
30. Santiago, I., Vara, J.M., de Castro, M.V., Marcos, E.: Towards the effective use of traceability in model-driven engineering projects. In: Ng, W., Storey, V.C., Trujillo, J.C. (eds.) ER 2013. LNCS, vol. 8217, pp. 429–437. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-41924-9_35
31. Vanhooff, B., Ayed, D., Van 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). https://doi.org/10.1007/978-3-540-75209-7_3
32. Vanhooff, B., Van Baelen, S., Joosen, W., Berbers, Y.: Traceability as input for model transformations. In: ECMDA Traceability Workshop (ECMDA-TW), pp. 37–46. SINTEF (2007)
33. Winkler, S., Pilgrim, J.: A survey of traceability in requirements engineering and model-driven development. Softw. Syst. Model. **9**(4), 529–565 (2010)
34. Yie, A., Wagelaar, D.: Advanced traceability for ATL. In: 1st International Workshop on Model Transformation with ATL (MtATL 2009) (2009)