A MODEL TRACEABILITY FRAMEWORK FOR NETWORK SERVICE MANAGEMENT

Omar Hassane

A thesis
in
The Department
of
Electrical & Computer Engineering

Presented in Partial Fulfillment of the Requirements
For the Degree of Master of Applied Science
Concordia University
Montréal, Québec, Canada

November 2020

Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Omar Hassane**

Entitled: **A Model Traceability Framework for Network Service Management**

and submitted in partial fulfillment of the requirements for the degree of

Master of Applied Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining commitee:

_____ Chair
Dr. Abdelwahab Hamou-Lhadj

_____ Examiner, External
Dr. Jamal Bentahar (CIISE)

_____ Examiner
Dr. Abdelwahab Hamou-Lhadj

_____ Examiner
Dr. Maria Toeroe

_____ Supervisor
Dr. Ferhat Khendek

_____ Supervisor
Dr. Sadaf Mustafiz

Approved _____
Dr. Rastko Selmic
Chair of Department or Graduate Program Director

_____ 2020 _____ _____
Dr. Mourad Debbabi
Dean, Gina Cody School of Engineering and Computer Science

Abstract

A Model Traceability Framework for Network Service Management

Omar Hassane

Automating the enactment of processes using model-driven methods and tools paves the way for streamlining or optimizing these processes. Establishing traceability in automated processes is instrumental in carrying out analysis of the process and the involved artifacts.

In this thesis, we propose a traceability information generation, visualization and analysis approach integrated with process modelling and enactment. A process model (PM) defined as an Activity Diagram has associated model transformations implementing the various activities and actions in the process. Enactment of the PM is carried out with the use of model transformation chaining in cooperation with model management means, in particular, megamodelling. We have incorporated both traceability in the small (at the model transformation level) and traceability in the large (at the PM level) in our approach. The traceability information is retained in the megamodel and forms the basis for traceability analysis of the enacted process. We have built a change impact analysis which allows the impact of a change in a model involved in the process to be assessed with the help of the derived megamodel.

We further extended our approach with the notion of *intents*. We propose the usage of intents at both the PM and model-transformation levels as part of our traceability information. We define intents as information representing the objective of the PM actions/activities and their implementations. Furthermore, we have incorporated traceability visualization support to visualize trace links relating models at different levels through the captured intents. The intent-enriched traceability information and the enhanced visualization enable semantically richer traceability analysis.

We applied our work to Network Service (NS) management in the context of the Network Functions Virtualization (NFV) paradigm.We believe automation of the orchestration and management of network services can progress rapidly with the help of model-driven engineering methods and tools. We applied our approach on a NS design process to analyze the impact of changing input models on output models as well as to show the benefits of intents not only in the context of this process, but also for the whole NS lifecycle management operations.

Our work is concretized in a tool, *MAPLE-T*, built as an Eclipse plugin. It extends MAPLE, an integrated process modelling and enactment environment.

Acknowledgments

I would like to greatly thank my supervisors, Dr. Ferhat Khendek and Dr. Sadaf Mustafiz for granting me the opportunity to pursue my thesis under their supervision. This thesis would not have been possible without their continuous support, patience and guidance.

I am deeply grateful to Dr. Maria Toeroe (Ericsson) for her support, knowledge and rigorous feedback that helped me carry out my research and complete my thesis.

I am very grateful to my colleagues in the MAGIC team for their friendship and support.

I would like to offer my warmest gratitude to my family for their love, support, and encouragement.

This work has been partially supported by Natural Sciences and Engineering Research Council of Canada (NSERC), Ericsson and Concordia University as part of the Industrial Research Chair in Model Based Software Management.

Contents

List of Figures

List of Tables

List of Acronyms

AD Activity Diagram

API Application Programming Interface

ATL Atlas Transformation Language

ATL EMFTVM ATL EMF Transformation Virtual Machine

CP Connection Point

EM Element Manager

EMF Eclipse Modelling Framework

ETSI European Telecommunications Standards Institute

FG Forwarding Graph

HOT Higher Order Transformation

M2M Model-to-Model

M2T Model-to-Text

MANO Management and Orchestration

MAPLE MAGIC Process Modelling and Enactment Environment

MAPLE-T MAGIC Process Modelling and Enactment Environment with Traceability Support

MDE Model-Driven Engineering

MgM Megamodel

MOF Meta-Object Facility

MT Model Transformation

NF Network Function

NFOntology Network Function Ontology

NFR Non-Functional Requirement

NFV Network Functions Virtualization

NFVI NFV Infrastructure

NFVO NFV Orchestrator

NS Network Service

NSD Network Service Descriptor

NSReq Network Service Requirements

OCL Object Constraint Language

OMG Object Management Group

OSS/BSS Operation Support System/ Business Support System

PM Process Model

QVT Query/View/Transform

SwImageDesc Software Image Descriptor

UML Unified Modeling Language

VDU Virtual Deployment Unit

VIM Virtualized Infrastructure Manager

VNF Virtual Network Function

VNFAD VNF Architecture Descriptor

VNFC VNF Component

VNFD VNF Descriptor

VNFFG VNF Forwarding Graph

VNFFGD VNFFG Descriptor

VNFM VNF Manager

XMI XML Metadata Interchange

Chapter 1

Introduction

In this chapter, we introduce the motivations, the target domain, context and contributions of this thesis.

1.1 Thesis Motivations

Model-driven engineering (MDE) is a paradigm for top-down software development. It promotes using models as first-class citizens in the software engineering process. Thus, models are no longer used only to describe software processes. They are now created and manipulated as part of these processes. Models are successively manipulated to achieve the final desired results. Furthermore, explicit modelling of processes not only allows for automation but also paves the way for streamlining or optimizing these processes. Process models can be used to not only represent but also execute workflows by leveraging MDE methods and tools.

However, in order to understand and analyze any process, it is not only essential to link its artifacts, but also generate and retain the traceability details at each step of the process. In fact, the traceability information collected can be the basis for various types of analysis or assessments (e.g., origin tracking, change impact analysis) of the enacted process [71] and also enables information recovery, change propagation, dependency visualization, and even defect detection and prediction [25,81]. Traceability management in software processes can be facilitated with the use of MDE enablers, such as model transformation chaining and megamodelling.

Generally, the analysis and diagnosis of a process is done in a manual and error-prone manner that requires a deep understanding of the process, its technicalities and interdependencies. Moreover, these interdependencies are generally implicit and are not captured in any step of the process. Thus, the same dependencies can be interpreted differently leading to confusing and contradicting analysis results.

In this thesis, our goal is to propose a method which automatically generates traceability information at both the transformation-level and process model-level. Thus, making the implicit links in the process model explicit. Based on that we provide the groundwork for change impact analysis.

While the generated traceability information proved to be useful in many ways, the relationships induced by them are generally shallow and not sufficient to gain deeper insights about the process. This knowledge can be obtained by augmenting the traceability information. We enhance the traceability information by augmenting the local and global traceability support with application-specific semantics captured from annotations of the process.

Additionally, when trace models and trace links become detailed and numerous, it becomes essential to be able to visualize the traceability information. Therefore, we further enriched our approach with the ability to visualize the semantically-enhanced traceability information.

1.2 Target Domain

The telecom industry has been moving from dedicated hardware /physical equipment network functions to virtualized network functions (VNF). This has been enabled by the virtualization technology and the cloud. The paradigm of Network Function Virtualization (NFV) [6,7] decouples network functions from the hardware infrastructure and allows for automated network service (NS) provisioning and management. Automating the management and orchestration of network services is one of the primary goals of network operators, but this comes with major challenges [27, 57]. Achieving full automation for NS management is among the main requirements of 5G. Zero-touch network and service management (ZSM), an industry specification group, was launched by the European Telecom Standards Institute (ETSI) to focus on full automation of the end-to-end management of network services to help with the 5G

deployment challenges [36].

Model-driven engineering (MDE) is a potential enabler for achieving automation in the NFV domain [22, 55]. Advanced support for discoverability and traceability have also been identified as essential features in virtualizing network services [20]. Furthermore, deeper knowledge of the NS design, deployment and management process and the collection of meaningful application-specific information about this process can lead to advanced analysis of NFV systems.

While NFV would greatly benefit from end-to-end traceability support, there has been very little done in this regard in this domain in the context of MDE.

Thus, we apply our work in the NFV domain for traceability analysis of the network service design process in order to assess the impact of changes in source models. The vendor-provided virtualized network function deployment templates form the core of the network service design process, and any changes in these templates can affect the target artifacts (mainly, the network service deployment template) 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.

Furthermore, our approach is applied to the NS design process to gain deep and useful insights into this design process through traceability analysis. We use the intent-enriched traceability analysis of the NS design process in order to investigate the dependencies between the input and output artifacts along with their design intents. This is achieved by leveraging the generated NS design traceability information and visualizing it. This turns out to be highly beneficial as we are able to investigate explicitly the entire NS design process and generate semantically richer analysis results that could not be achieved previously. These results can help in improving the diagnostic of potential problems observed at runtime.

1.3 Thesis Context

This work is part of a larger research program within the NSERC/Ericsson Industrial Research Chair on Model Based Software Management [1] related to the automation of Network Service (NS) design and management in the context of the Network Functions Virtualization (NFV) framework.

[1] `https://users.encs.concordia.ca/~magic/`

Prior to this work, a high-level model-based process for NS design, deployment and management [62], and a model-based approach for the design of network services [61] have been proposed. Furthermore, an approach has been proposed to model and enact process models (PM) and has been applied in the NFV domain particularly to the NS design process [59,60]. This approach has been concretized in a tool, MAPLE (**MA**GIC **P**rocess Mode**l**ling and **E**nactment Environment), which uses MDE methods to enact process models.

In this thesis we propose a model traceability framework for generating and analyzing traceability information in the context of process model enactment. We have applied our framework to the NS design process. We concretize our approach as an Eclipse plugin extending **MAPLE** and which is called **MAPLE-T**.

1.4 Thesis Contributions

The contributions of this thesis can be summarized as follows:

- An automatic model-driven traceability information generation approach at both the local (transformation-level) and global (process model-level) levels for enactable process models.

  Published in: [50].

- An easily extensible traceability analysis solution carried out on the basis of the generated traceability information. We have incorporated more advanced traceability analysis solutions, specifically to incorporate change impact analysis. The purpose of this is to determine how impactful a model or element is on the whole process at both the metamodel and model levels.

  Published in: [50].

- An approach for capturing application-specific information (intents) at both the process model (PM) and model-transformation levels (from their annotations) as part of our traceability information. We further use them to enable advanced application-specific traceability analysis solutions.

  Accepted for publication in: [51].

- A traceability visualization solution for visualizing traceability information relating models at different levels through the captured application-specific information.

  Accepted for publication in: [51].

- An Eclipse [39] plugin tool incorporating all the aforementioned solutions. The tool support is demonstrated with the use of the NS design PM modelled as a UML 2 activity diagram [64] in the Papyrus [35] environment.

  Published in: [49].

1.5 Thesis Organization

This thesis is organized into nine chapters. Chapter 2 introduces the background information related to Model-driven engineering , traceability and MAPLE. In Chapter 3, we present existing work related to our thesis. Chapter 4 presents the overall approach for integrating traceability support with process enactment. In Chapter 5, we discuss how the traceability information generation process is incorporated in our work. We follow up in Chapter 6 by presenting how traceability visualization and analysis is incorporated based on the generated traceability information. In Chapter 7, we present the MAPLE-T tool and its backend. In Chapter 8, we present the NS design case study in the context of NFV. We further discuss the results induced by the enabled traceability generation, visualization and analysis in two different applications. Finally, we sum up with a conclusion and future work in Chapter 9.

Chapter 2

Background

In this chapter, we present the model driven engineering (MDE) background information related to our work. Moreover, we discuss the concept of traceability in general and in the MDE context. Furthermore, we introduce the *MAGIC Process Modelling and Enactment Environment (MAPLE).*

2.1 Model Driven Engineering

The term Model-Driven Engineering (MDE) is typically used to describe a software development methodology in which abstract models of software systems are created and systematically transformed to concrete implementations [40]. Models are considered as first class citizens in the engineering process. The models are manipulated via transformations which form the backbone for automation in MDE.

In the remainder of this section, we give a brief overview of the main MDE concepts, particularly those that relate to the contributions of this thesis. First, we introduce the concepts of models and metamodels in Section 2.1.1. Model transformations are discussed in Section 2.1.2. Furthermore, we discuss how model transformations can be composed to form what is referred to as a transformation chain in Section 2.1.3. Since various heterogeneous models can be used as part of the MDE approach, there needs to be a way to properly manage them. This is called *model management* [19]. In Section 2.1.4 we discuss megamodelling as one of the main concepts used in model management. Finally, we introduce the concept of process modelling in Section 2.1.5.

2.1.1 Models and Metamodels

According to [37]: "A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system".

A model is composed of objects and associations (i.e. relations) between them. A model conforms to well-defined rules constructing what is referred to as a *metamodel*. The same way a model is defined as an abstraction of a system, we can define a metamodel as yet another abstraction, capturing properties of the model itself [26]. The relation between a model and its metamodel is referred to as a *conformance* relationship.

Metamodels are hugely beneficial in defining constructs of modelling languages. Metamodels are themselves represented as models and often conform to a metamodel referred to as a *meta-metamodel* which conforms to itself. *MOF* (Meta-Object Facility) [10] is the defacto metamodelling framework defined by the Object Management Group (OMG). One of its well-known implementations is EMF (Eclipse Modelling Framework) [2]. Normally EMF is not a strict implementation of MOF so as often than not it is considered a standalone metamodelling framework itself. The core EMF implementation includes a meta-metamodel called *Ecore*. Furthermore, EMF includes an API enabling easy manipulation of Ecore models.

The OMG also defined a useful modelling standard called the Unified Modeling Language (UML) [64]. Generally, UML has been always used for representation and design purposes. A major release (UML 2) introduced the ability to extend UML and its semantics by using the concept of *profiles*. A profile can be mapped to the concept of metamodel by extending the UML metamodel with meta-classes (stereotypes), tagged values (attributes for stereotypes), constraints and relations between them. However, a UML 2 profile should not be confused with a metamodel. A model can use various profiles at the same time, but it can conform to one and only one metamodel.

2.1.2 Model Transformations

Model transformations are generally categorized as Model-to-Model (M2M) or Model-to-Text (M2T). M2M transformations transform one or multiple source models to one or multiple target models. M2T transformations transform one or multiple source

models to text strings.

Transformations are generally implemented using special purpose programming languages such as ATL (Atlas Transformation Language) [5] or QVT (Query/ View/ Transform) [21]. They provide specific syntax and execution semantics for the purpose of simplifying the implementation and maintenance of model transformations [76].

ATL is a widely used M2M model transformation language. It is a hybrid model transformation language supporting both the declarative and imperative programming styles. ATL is rule-based building heavily on the *Object Constraint Language (OCL)* [63], but also provides built-in features that OCL is missing [26]. The main components of an ATL transformation are *rules* and *helpers*. Each rule transforms several input model elements into multiple output elements in the output models. The input and output models as well as their metamodels are generally specified in a field referred to as the *Launch Configuration*.

Rules are categorized into *matched* and *lazy* rules. Matched rules are called in a declarative manner. They automatically match well-defined source elements and generate a set of elements in the target models. Lazy rules on the other hand are explicitly invoked from other rules. A helper is a function that makes it possible to factorize and reuse a block of code in different parts of the ATL transformation [26].

A model transformation can itself be expressed as a model conforming to a transformation metamodel. This can enable advanced capabilities such as Higher Order Transformations (HOTs) [74]. They are defined as model transformations transforming an input transformation model to another transformation model [76].

2.1.3 Transformation Chaining

In order to achieve better maintainability, reusability and extensibility, model transformations can be integrated to form what is referred to as a *model transformation chain*. Model transformation chains are typically constructed following a pipeline architecture where the output of a transformation is the input of the subsequent one and so on [29]. Generally, a transformation chain can be composed of heterogeneous model transformations (i.e. implemented using different transformation languages) [45].

Transformation chaining is the preferred technique for modelling the orchestration of different model transformations [26]. Typically, model transformation chains are defined using orcherstration languages. These languages can model the chain as a set

of transformations to be executed sequentially. Furthermore, advanced features such as loops, conditions and nested compositions (nested model transformation chains) can be supported.

Model transformation chains are often defined using textual languages, such as ANT [33].

2.1.4 Megamodelling

Megamodelling is preeminently used in model management to avoid what we refer to as the 'meta-muddle'. A megamodel is defined as a model which contains other models as elements as well as the relationships between them [24]. A megamodel can contain metamodels, meta-metamodels, models, and transformations along with their interdependencies (such as, conformance or derivation). A megamodel can also be composed of other megamodels.

A megamodel can be used to ensure the consistency between the retained models. It can keep track of every change that can happen to a model and synchronize the related models accordingly [52]. Thus, conformance and compatibility checks can be enforced between the involved artifacts.

Among its various uses, a megamodel enables the reuse and construction of transformation compositions, allows for the creation and maintenance of traceability links and enables the construction of a repository of tools and languages.

Figure 1 shows an example of a simple megamodel composed of different kind of resources. rectangles represent model instances (Model1 and Model2), roundtangles represent metamodels to which these models conform to (MM1 and MM2), ellipses represent transformations/executables (ATLTransformation and JavaProgram) conforming to their corresponding metamodels (ATLMM and JavaMM). The conformance links within the megamodel are shown as dashed links and the transformation links are shown as straight arrows. Moreover, other artifacts are represented as circles in the megamodel (metadata files, unrecognized resources, etc).

2.1.5 Process Modelling and Enactment

According to [38], a *process model* is : "an abstract representation of a process architecture, design or definition... Process models may be used to assist in process

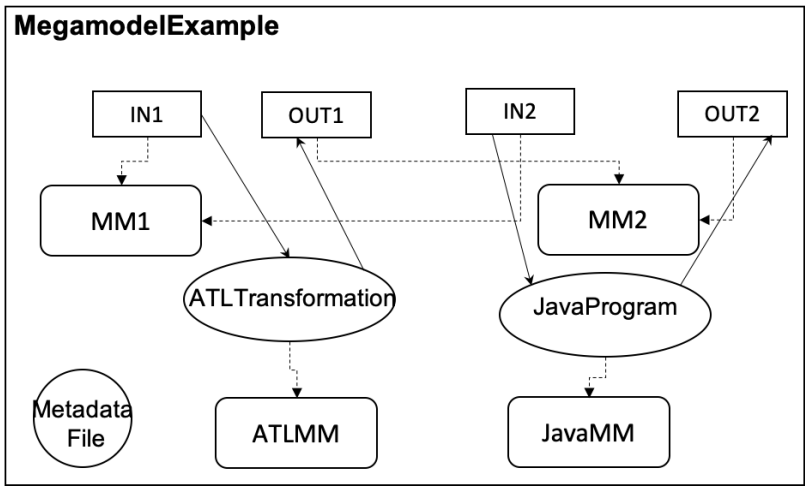

Figure 1: Simple Megamodel

analysis, to aid in process understanding, or to predict process behavior".

In practice, process models and workflows can be described using UML 2.0 Activity Diagrams [65]. Activity diagrams support the modelling and synchronization of a set of behaviors using data and control flows. The data and control flows are represented using activity edges connecting activity nodes. Activity nodes can be a simple action (representing an atomic step within an activity) or an activity (representing an activity itself composed of actions and other activites).

A process model represented using UML 2.0 activity diagrams can contain:

- **Actions:** defines a task or several tasks related to input and output artifacts (e.g. models)
- **Object flows:** relations represented as links between the artifacts involved in an action.
- **Input/Output:** represent artifacts (models) that can be fed into or generated from the process model actions.
- **Control flows:** relations represented as links directing the ordering of the actions.

Figure 2 shows a simple process model expressed as a UML2 activity diagram. It has actions represented as rountangles, object nodes (input/output parameters) as rectangles, object flows as dashed arrows and control flows as filled arrows. Object

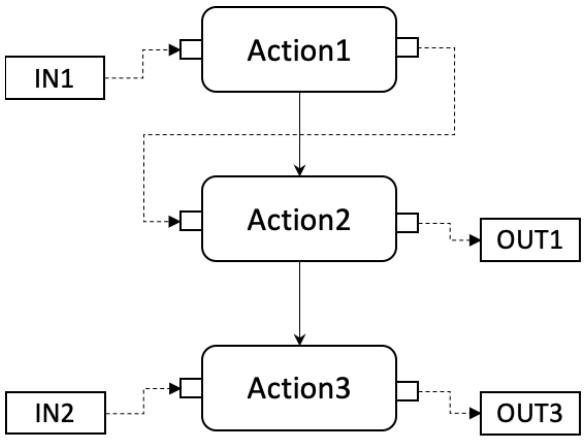

Figure 2: Simple Process Model

nodes can also be attached to a specific action (small rectangles attached to the action).

In our work, process models are modelled using Papyrus [35]. Papyrus is an Eclipse-based open source project which provides a modelling environment for creating UML 2 diagrams as well as other UML-based domain-specific languages (such as SysML [66] or UML-RT [65]).

Process models can be analyzed, validated and even *enacted*. The **enactment** of a process model refers to the act of executing the process model using what is refered to as *process agents*. A process agent is an entity responsible for executing the process model. It can be a person following a process script or a machine executing a program [38].

2.2 Traceability

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* [12].

Traceability information in MDE can be classified as *generic* or *specific* [15]. *Generic traceability* provides the ability to link MDE artifacts but does not provide detailed semantics for these relationships. The trace links are not tightly coupled to any domain, however, they are semantically weak. *Specific traceability*, on the other hand, is domain-dependent and provides a predefined and specific set of relationship

types relevant to the domain. This allows the generation of semantically richer and fine-grained traces.

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 [78]. *Intra-model traceability* stores traceability information in the traced models themselves. The metamodels of the traced models, hence, should be extended to include information related to traceability which are meaningless in the domain of the models. This often makes the intra-model traceability ineffective as it populates the models with traceability information that are meaningless in the domain of the models.

Extra-model traceability defines a separate trace model conforming to an external traceability metamodel. Elements of the traced models are linked via a trace link created within the external trace model.

Traceability metamodelling can follow a *pure metamodel* approach or a *tag-based* approach [78]. 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 [18].

2.3 MAPLE

MAPLE (MAGIC Process Modelling and Enactment Environment) [60] is an integrated environment for process modelling and enactment. Process modelling is supported with UML Activity Diagrams in Eclipse Papyrus. MAPLE supports enactment of process models with underlying heterogeneous (cross-technology) transformation chains.

Megamodels (MgM) have been used to manage all the resources needed for the enactment. This requires registering the process model (PM) as well as the repository of resources (metamodels, profiles, model instances, model transformations, programs, etc.). Relationships such as conformance links between all of these resources are included in the megamodel as well. Based on the derived megamodel and the process model, a model transformation (MT) chain is automatically created in MAPLE. This chain is then executed using token-based semantics leading to the generation of output artifacts. When an action is given a token, it is executed and the token is passed to the next action. The generated artifacts are added to the megamodel dynamically during enactment. MAPLE is built on top of the Eclipse Papyrus [35] environment.

Chapter 3

Related Work

In this chapter , we present an overview of approaches related to our work. Firstly, we discuss MDE approaches and projects related to supporting local, global and semantically rich traceability information generation. We follow up by discussing approaches supporting traceability visualization and analysis in the context of MDE.

3.1 Traceability Information Generation

There has been a lot of work done on traceability in MDE, and these are discussed and summarized in [11, 15, 44, 72, 81]. In the following, we discuss approaches that specifically address local, global and semantically rich traceability information generation in the context of model management, process enactment and model transformation chains.

3.1.1 Local Traceability

Guana et al. [47] propose *Chain Tracker*, a traceability generation environment for model transformation chains/compositions (particularly for Model-to-Model ATL transformations). Although they augment an ATL transformation chain with traceability information automatically, the collected traces are generated only at the transformation level and they are not interlinked globally.

Falleri et al. [68] introduce a framework to augment Kermeta [58] model transformations with traceability support. The transformations are manually augmented

with traceability support and no explicit link between distant models is captured within the generated traces.

Beyhl et al. [23] present a framework for retaining and maintaining traceability links between the artifacts within a hierarchical megamodel (a megamodel combining high-level artifacts along with low-level fine-grained artifacts). However, no support for linking distant artifacts using global traceability links has been mentioned in their approach.

Our first intention was to reuse the *Tracer Adder* Higher Order Transformation (HOT) [53] and the corresponding traceability metamodel, but it turns out that it is very basic and does not cover all of the possible traceability information we wish to retain. We have also looked into more advanced traceability support in ATL in the *ATL EMF Transformation Virtual Machine (ATL EMFTVM)* work [34,82]. However, this work does not support the ability to retain more granular tracing (e.g., with attribute-level elements traceability). Additionally, our first intention was to reuse the ATL EMFTVM engine for traceability generation support as a first potential solution for our traceability generation approach in MAPLE-T. However, we did not proceed in that direction as the EMFTVM engine is not straightforward to use for models which conform to UML2 profiles as their metamodels (all the metamodels are expressed as UML profiles in our case). It is only relatively easy to use for models conforming to Ecore metamodels (i.e. the EMFTVM is fundamentally built to be used with Ecore metamodels and its ease of usage depends on the complexity of these metamodels).

None of the work discussed above provide any support for process enactment with the use of megamodels.

3.1.2 Global Traceability

Fritzsche et al. [41, 42] and Jouault et al. [54] have proposed approaches similar to ours in terms of using model transformation chaining and/or model management with 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 and model-level traces 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. [80] extend UNiTI [77] (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 [13, 14], 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 local traceability metamodel in terms of the generated trace links. In our case, trace links are contained within model transformation rules. 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.

3.1.3 Semantically Rich Traceability Information

There has been work regarding process/transformation *intents* in the model-driven engineering community. Levi et al. [56] propose a framework for defining transformation intents from a pure MDE perspective. Moreover, they have defined an intent catalog in which a non-exhaustive list of transformation intents and their properties are defined and described. However in our case, intents are tightly related to the objectives of the implementer/modeller as labeled in the PM and the transformations. Also, there is no support for traceability in [56]. In MAPLE-T, intents are captured and integrated with the generated traceability information.

In the following, we discuss existing work related to rich traceability information generation.

Beatriz et al. [69] have proposed an automated and semantically-aware approach for capturing semantically-rich traceability relationships across different models. They further propose an architecture for model management task automation with the use

of semantically rich trace links. While they address semantically-rich traceability support and model management, their approach does not use model transformations and is not carried out in the context of process enactment or transformation chain execution. In MAPLE-T, intents are defined and captured at both PM and transformation levels in the traceability information.

Drivalos, Paige et al. [31, 67] have proposed an approach for creating and maintaining strongly typed and semantically rich model-to-model traceability links. They achieve this by defining a strongly typed application-specific traceability metamodel as well as a set of constraints to check the validity of the models. The traceability metamodel is tightly coupled to the application domain. The application-specific trace information is similar to our intent-augmented local trace model. In fact, they can define application-specific information as a rationale associated with a particular trace link. This rationale captures why the links exist. In our traceability information, the intents capture the purpose behind every traced link associated with every transformation rule and what its specific parameters are. However, we use a generic traceability metamodel which can be used to capture such traceability information in any domain. Also, no automatic semantically-rich traceability generation is supported in [31, 67].

Dick et al [30] have proposed an approach to augment traceability information with more semantics by adding textual rationals to traces. It makes use of a textual rationale expressing the rich semantics as well as propositional logic to enhance the traceability information. The work introduces advanced and new kind of analysis that are made possible due to the enhancing of the traceability information with semantics. This work is in the area of traceability in requirements engineering. No MDE techniques for the enactment nor automatic traceability information generation are supported.

3.2 Traceability Visualization

There exists various work on traceability visualization within the MDE community. van Amstel et al. [75] propose *TraceVis*, a tool for traceability information visualization in model transformation chains. However, there is no visualization filtering based on custom parameters in [75]. In MAPLE-T, we can focus on specific parameters

instead of going through a huge visualization graph.

Santiago et al. [70] propose *iTrace*, a tool which incorporates different kinds of visualizations at different granularity levels (transformations, rules, models and model elements). Although it shows various dashboards to visualize the trace models and the transformations, there is no support for semantically-rich and detailed links in their visualizations. In MAPLE-T, we label every link with application-specific semantics (intents) and their specific parameters. Moreover, we visualize explicitly the distant links between models. In [70], the visualization shows only local trace links for a given transformation/rule.

Pilgrim et al. [80] present a traceability visualization framework using the GEF3D [79] tool. It enables 3D visualization of diagrams related to the traced models involved in a transformation chain execution. However, there is no support for semantically rich traceability links in their visualization graph.

3.3 Traceability Analysis

There has been extensive work carried out on change impact analysis in the requirements engineering community [46, 72, 81]. However, these approaches are not in the context of process enactment and megamodelling techniques.

As mentioned in the previous section, van Amstel et al. [16] propose *TraceVis* [75], 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.

N. L. S. Fung et al. [43] 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 (which are cases or arguments guaranteeing that a system is operating as intended with a focus on security, safety, etc). However, it is not clear whether their megamodel has traceability extensions enabling navigation between artifacts at the global and local levels.

Table 1: Comparison of the approaches (supports (✓), does not support (✗) , unknown/unclear (_))

Approach	Traceability Information Generation			Traceability Visualization	Traceability Analysis
	Local	Global	Semantically Rich		
Guana et al. [47]	✓	✗	✗	✓	✗
Falleri et al. [68]	✓	✗	✗	✓	✗
ATL EMFTVM [34, 82]	✓	✗	✗	✗	✗
Fritzsche et al. [41, 42]	✓	✓	✗	✗	✗
Jouault et al. [54]	✓	✓	✗	✗	✗
Von Pilgrim et al. [80]	✓	✓	✗	✓	✗
MegaM@Rt2 [13,14]	✓	✓	✗	✗	_
Beyhl et al. [23]	✓	_	✗	✗	✗
Beatriz et al. [69]	✓	✓	✓	✗	✗
Drivalos, Paige et al. [31, 67]	✓	✗	✓	✗	✗
Van Amstel et al. [75]	✓	✗	✗	✓	Change Impact ✓
Santiago et al. [70]	✓	✗	✗	✓	✗
N. L. S. Fung et al. [43]	_	_	✗	✗	Change Impact ✓

3.4 Summary

In Table 1, we summarize and compare the related approaches using four criteria. The first criterion addresses support for traceability information generation. We further detail this criterion to see whether the approach supports local, global and semantically rich traceability information. The second criterion addresses whether the approaches support traceability visualization or not. Finally, we have also investigated whether each approach supports traceability analysis. As can be deduced from the comparison table, none of the related approaches satisfies all the three criteria, while our approach provides support for all of these features.

Chapter 4

Model Traceability Framework for Process Enactment: Big Picture

In this chapter, we present the overall picture of our work. In Section 4.1, we present the core concepts used in our approach. Particularly, we start by introducing the concept of intents which provides semantically richer traceability information. We then proceed by presenting the local and global traceability concepts as used in MAPLE-T. In Section 4.2, we introduce our model-driven traceability information generation, visualization and analysis approach. Our approach integrates the traceability information generation support with our megamodel-based process enactment. Furthermore, the generated traceability information are visualized and analyzed as part of our approach.

4.1 Concepts

4.1.1 Intents

The notion of *intent* in our work represents the objective of tasks at different levels in the process model: activities, actions, and the underlying model transformations (MTs). We classify intents into two categories:

- PM-level intent: represents the objective of an activity or action of the PM as well as the activity representing the PM itself.

- Transformation-level intent: represents the objective of a rule within a transformation (implementing the actions in the PM).

Each intent can be associated with multiple parameters representing further application-specific information. Also, since transformation rules can be composed of several helpers, lazy rules and imperative blocks (see Section 2.1.2) within their implementations, one needs to support capturing intents at a lower level by associating each intent with its corresponding *sub-intents*. These sub-intents represent the objectives of any nested logic within the transformation rules.

It should be noted that in this work we use the term *intent* with a meaning of *design or processing intent* and distinguish it clearly from the concept of *intent*, generally used to express *high level requirements on what should be done* in the networking domain in the context of intent-based networking [28, 48].

4.1.2 Local Traceability

As mentionned in Section 2.2, traceability is referred to as *local* when trace models are generated as a result of the transformation execution.

In MAPLE-T, we refer to the local trace models as `LTrace` models. This `LTrace` model contains the trace links for each in/out 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.

- 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 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.

Our `LTrace` metamodel was inspired from the EMFTVM trace metamodel defined in [82] (this work is discussed in Section 3.1.1). Our additional features are mainly

the "SourceElementAttribute" and "TargetElementAttribute" elements capturing further attribute-level traced information.

Additionally, in order to capture intents and their parameters in our trace models, it was necessary to extend the `LTrace` metamodel accordingly. An "intent" element has been added to the metamodel representing the intent of a `TracedRule`. Also, each intent can have sub-intents captured within the `TracedRule`. Figure 3 shows the `LTrace` metamodel illustrating the main elements mentionned above.

4.1.3 Global Traceability

The `LTrace` models are interlinked in the MgM to achieve global traceability. In MAPLE-T, the set `LTrace` models and the global links are refered to as the *GTrace*. The megamodel metamodel is extended to retain the GTrace elements.

The metamodel in Figure 4 shows the main elements of the MgM including elements constructing the GTrace. Every resource is associated with a *history* which represents a set of versions of the resource, that can be updated whenever the resource is being used or changed. Also, each resource has an *origin* that enables identification of the source it is coming from. It can be, unknown, user provided or derived (for instance, a transformation output). In addition to that, transformations are associated with a link to the set of their actual executions. Each *TransformationExecution* element contains a set of *TransformationConfiguration* elements which represent the actual transformation parameters given to the transformation launch configuration. Each TransformationExecution is associated with an `LTrace` model linking elements of the provided and the produced resources.

4.2 Overall Approach

Our work extends MAPLE's process enactment approach (introduced in Section 2.3) with traceability support. Our goal is to go further and use the MgM for advanced traceability of MT chains.

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.

Figure 5 shows an example of a PM modelled as a UML 2.0 activity diagram

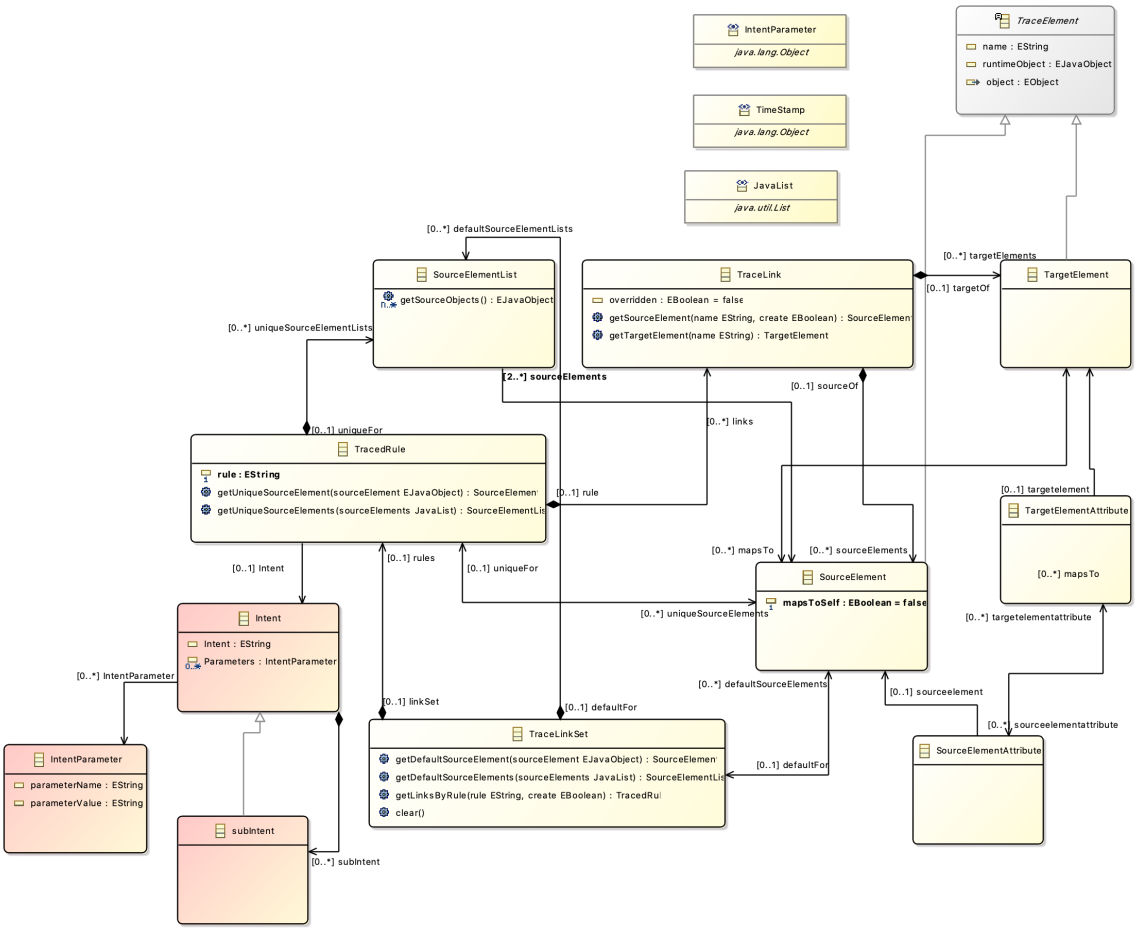

Figure 3: `LTrace` Metamodel

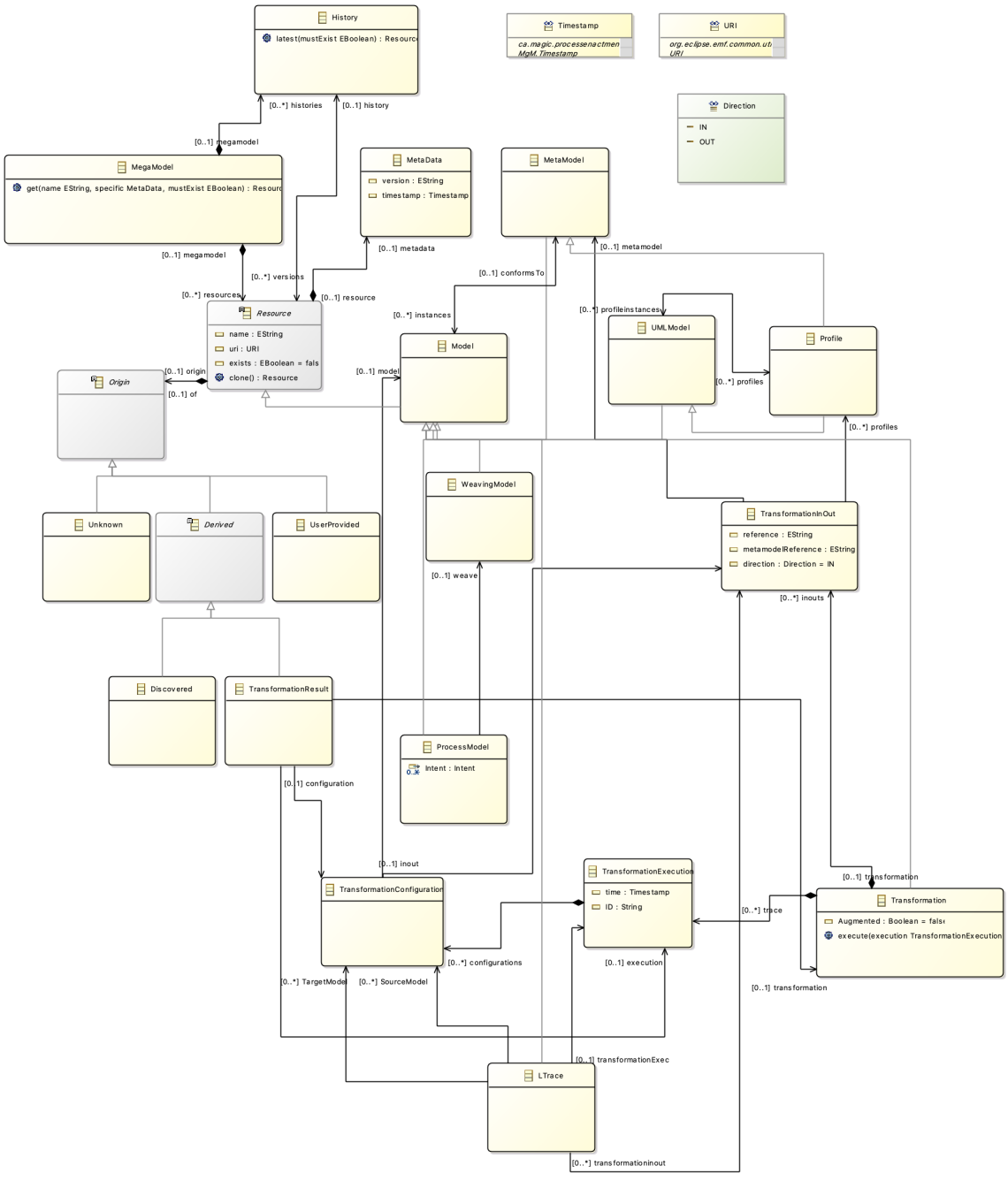

Figure 4: Megamodel Metamodel

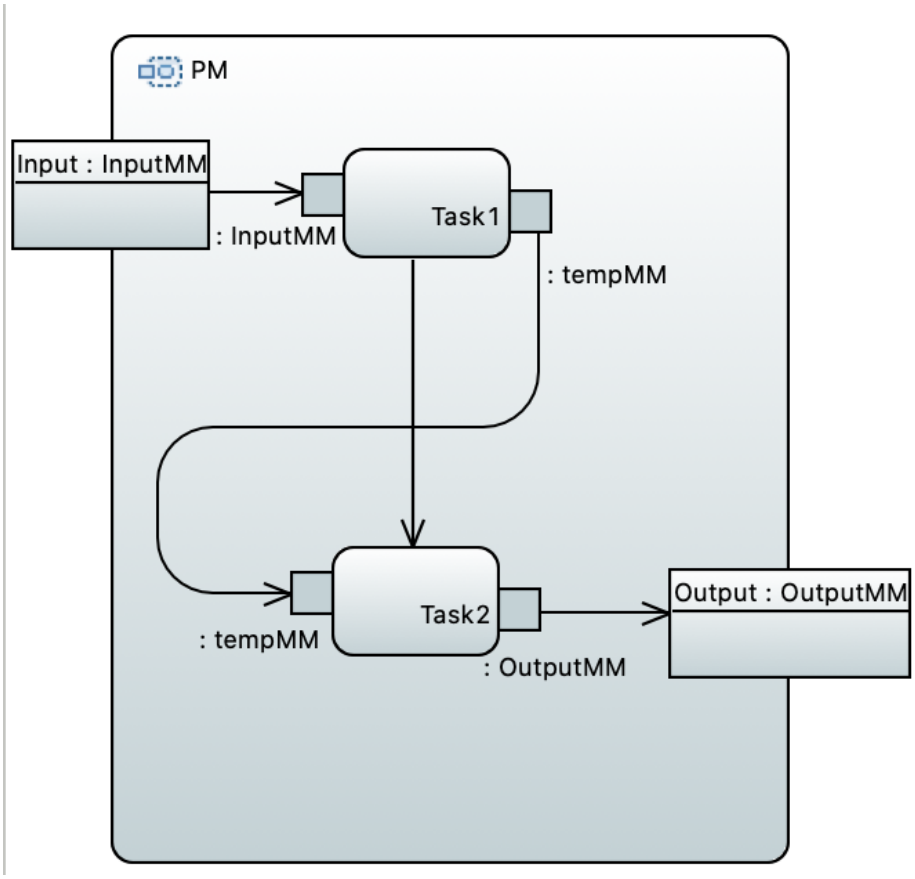

Figure 5: Simple PM Example

in the Papyrus environment. The figure shows two actions: "Task1" and "Task2". "Task1" takes a model named "Input" conforming to a UML profile ("InputMM") and its output is a model conforming to a profile called "tempMM". "Task2" takes a model of type "tempMM" as input and outputs a model of type "OutputMM".

Strictly speaking, the models do not actually "conform" to the profiles but rather to the UML metamodel. The conformance term is used here because each model has only one and only one profile applied to it [32].

In order to capture intents in the traceability information, the PM and its implementations need to be annotated first. This is done by the PM designer who labels every activity and action with their respective intent. These labels are generally added as comments within the PM and the transformations implementing it. These intents can be retrieved manually one by one or automatically as part of the traceability information gathered during the PM enactment.

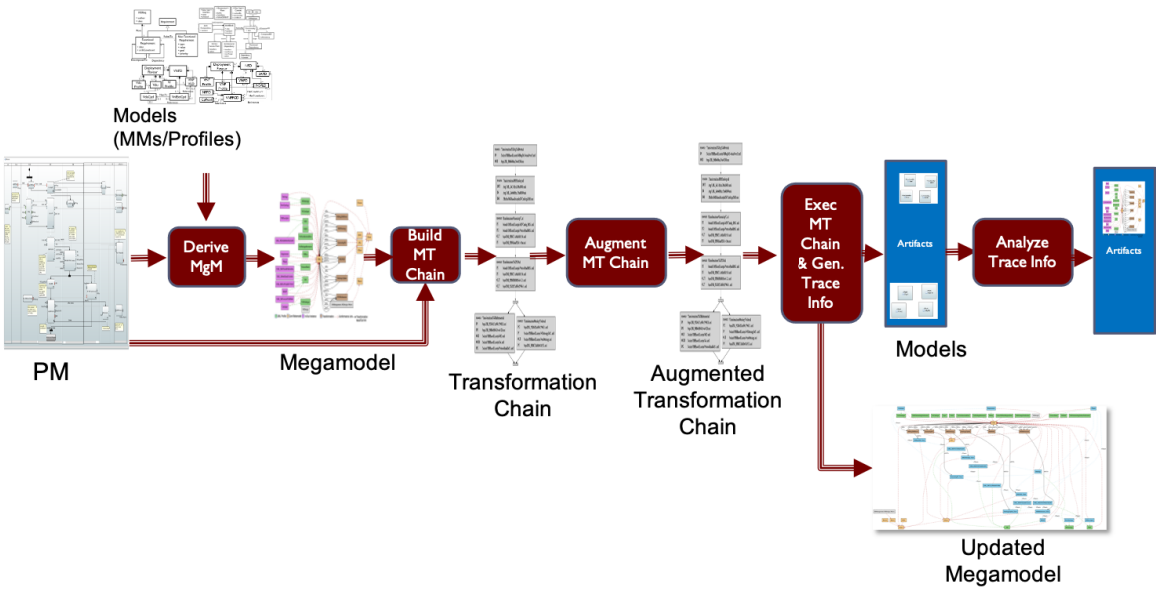

Figure 6: MAPLE-T Approach

Once the PM has been annotated, we can proceed with the PM enactment and the traceability information generation, visualization and analysis. The MAPLE-T approach is shown in Figure 6. Following the derivation of the megamodel (MgM) and the construction of the MT chain, the chain execution results in the generation of artifacts. During this execution, `LTrace` models will also be output and will be retained in the MgM in order to build a global traceability map (`GTrace`). The `GTrace` can then be used for traceability visualization and analysis.

In the following, we discuss every step and we highlight our traceability-related features in every one of them.

4.2.1 MgM Derivation

In MAPLE-T, 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, and hence MAPLE-T supports execution of cross-technology model transformation chains. Each transformation part of the MT chain can be implemented with a different language.

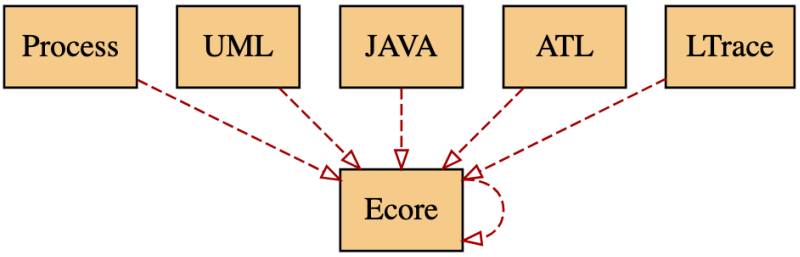

Figure 7: Base MgM

Moreover, languages can be model transformation languages (For instance, ATL or Epsilon) or general programming languages (For instance, Java or C) [59]. Due to this, deploying model management techniques is essential in MAPLE-T. As described in Section 2.3, megamodels have been used for this purpose.

The MgM is derived after registering the resources and the PM. First, a base MgM is loaded as part of the 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 Figure 7). Each trace model generated as a byproduct of a transformation execution conforms to this metamodel.

Figure 8 shows an example of the MgM after registering the three UML profiles (in green):"InputMM", "OutputMM" and "tempMM" to which the models depicted in Figure 5 conform to. The profiles conform to the UML metamodel. The red dashed links represent conformance links.

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. Furthermore, PM-level intents are added to the weave model and associated with their corresponding actions/activities.

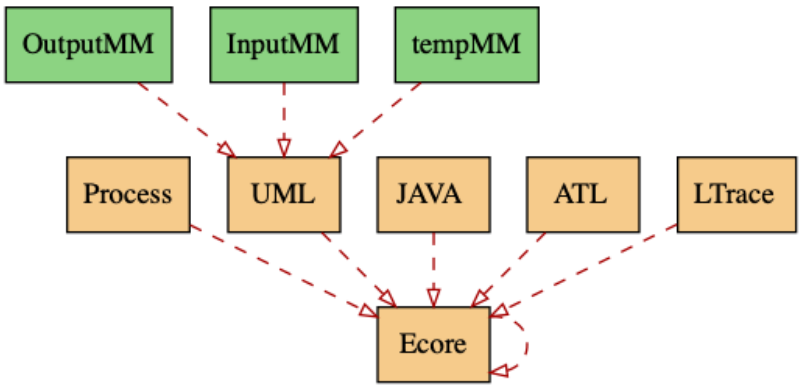

Figure 8: Example of an MgM after registering UML profiles

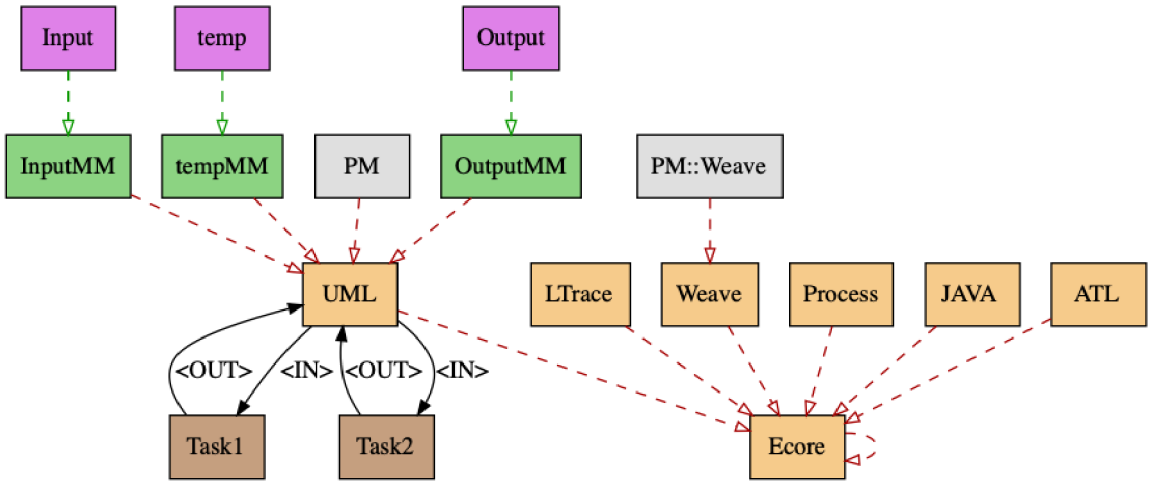

Figure 9: Example of an MgM after registering the PM in Figure 5

Figure 9 shows the MgM after registering the PM shown in Figure 5. The PM actions are registered as ATL transformations (shown in brown) conforming to the ATL metamodel. Moreover, the weave model and the PM are shown in grey and they respectively conform to the weave and UML metamodels. Also, the placeholders for the models conforming to the profiles are shown in purple.

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).

4.2.2 Transformation Chain Derivation

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 time-consuming in some applications.

4.2.3 Executing the Transformation Chain and Generating the Trace Models

Each of the transformation in the MT chain is first augmented such that `LTrace` models are generated as extra artifacts with the execution of each 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 `GTrace`.

This step is discussed in details in Chapter 5.

4.2.4 Traceability Visualization and Analysis

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. The exposed features allow the generated `LTrace` models and `GTrace` links to be parsed and manipulated, typically with the use of the captured intents that provide richer semantics for the analysis.

We have incorporated traceability analysis support, specifically to carry out *change impact analysis* as well as to generate a comprehensive and semantically-rich traceability map that provides in-depth knowledge about the process.

Also, in order to simplify the investigation and navigation of our semantically enhanced traceability information, we have incorporated traceability visualization support in MAPLE-T. Visualization is implemeted in MAPLE-T using automatically generated directed graphs in the DOT graph description language [1]. DOT graphs are essentially files with the "dot" extension. They provide a rich syntax aimed towards expressing graphs in a textual format. We generate the graphs as additional output of the enactment process. We use GraphViz [4] to process the generated graphs and render them in a graphical format.

This step is discussed in details in Chapter 6.

Chapter 5

Traceability Information Generation

In this chapter, we present how the traceability information generation support is built in MAPLE-T. In Section 5.1, we discuss how the PM and the transformations implementing it are first annotated with intents. In Section 5.2, we talk about the automatic transformation chain augmentation with traceability support. The augmented transformation chain is executed and as a result the desired output artifacts from the PM as well as the `LTrace` models are generated. Finally, we discuss how the MgM is updated and as a result the GTrace is constructed.

Figure 10 depicts the traceability information generation approach. The steps of the approach are discussed in details in the next sections.

5.1 PM and Transformations annotation

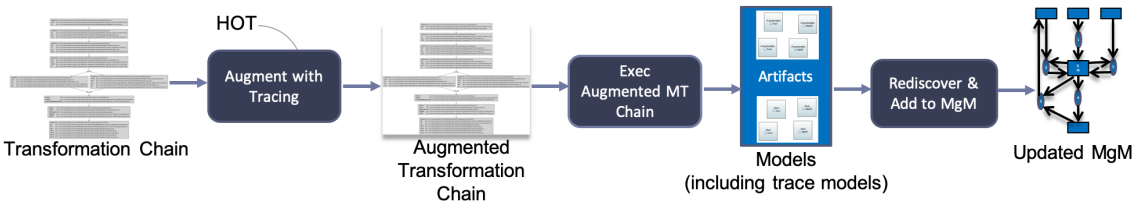

Figure 10: MAPLE-T Traceability Generation Approach

Before enacting the PM and therefore generating the traceability information, the PM and the transformations implementing it need to be annotated first as discussed in Section 4.2. This step is not necessary to generate the traceability information. However, if no annotation is performed, the generated traceability information will only retain trace links between the artifacts and no intents will be captured. Thus, advanced traceability features using the captured intents will not be applicable.

In order to efficiently label and parse intents in MAPLE-T, we provide the following labelling guidelines for both the PM and the transformations:

5.1.1 Annotating the PM

- Actions and activities in the PM should be labeled with *intents* in their *applied comment* fields. These fields represent text boxes where the user can enter custom comments related to the activity diagram elements [65].
- The activity representing the PM itself should be labeled as well, this way the PM can be reused/embedded in larger PMs.
- Only comments starting with the string "#intent" will be parsed by MAPLE-T to distinguish them from normal comments that start with the word "intent".

For instance, the two actions shown in Figure 5 are annotated in their *applied comment* fields with the intents "#intent: intent of task1" (as shown in Figure 11) and "#intent: intent of task2" (as shown in Figure 12) respectively. Moreover, the activity itself is annotated with the intent "#intent: intent of the PM" (as shown in Figure 13).

Note that the *intent* labels are just used as examples here to explain the approach; real applications would be using meaningful *intent* labels.

5.1.2 Annotating the Transformations

In some cases, almost all the semantics are hidden within the transformations implementing the PM. To be able to retrieve the semantics (particularly, the *intents*), these transformations need to be well-labelled with such information. The designer, i.e. the

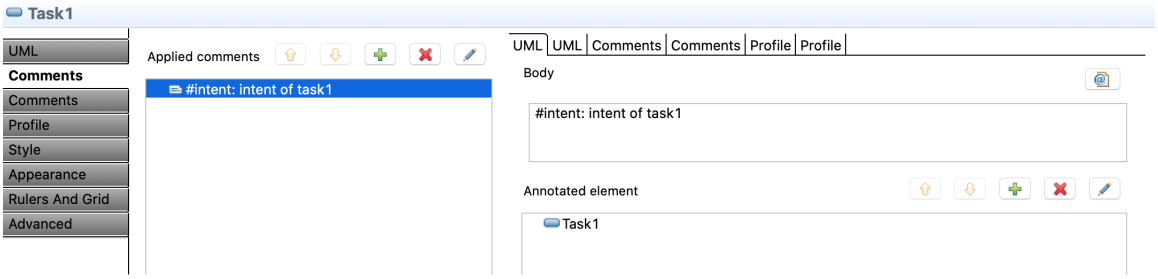

Figure 11: Applied comment field of the "Task1" action

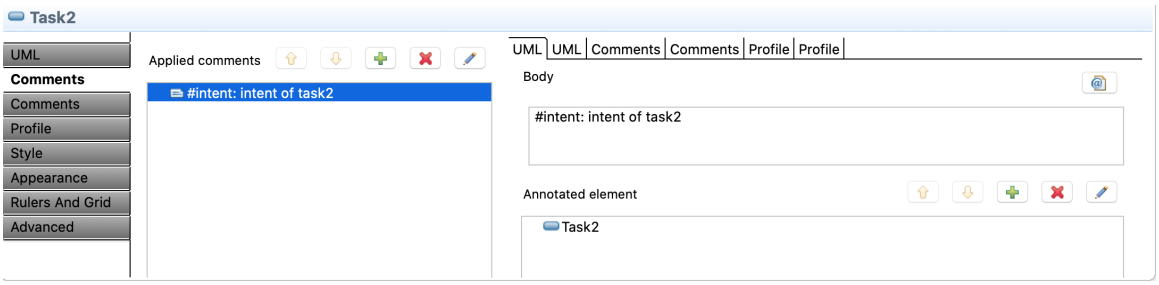

Figure 12: Applied Comment Field of the "Task2" Action

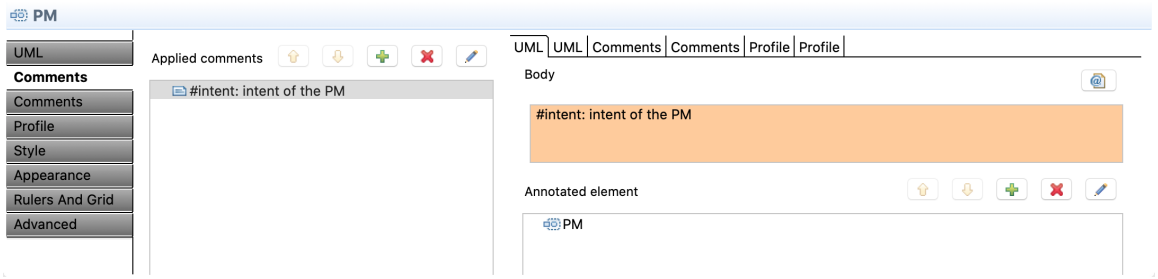

Figure 13: Applied Comment Field of the "PM" Activity

expert, is expected to label the transformation rules with meaningful intents, sub-intents and intent-specific parameters. Every label should be defined as a comment right before the rule definition.

These labels are parsed automatically by MAPLE-T by mapping the *#intent* label to the *intent* element in the `LTrace`. In other words, whenever a rule, a helper or even any imperative part of the transformation is labeled with a comment starting with "#intent" or "#sub-intent", MAPLE-T will automatically parse and retrieve this information. Any other label will be ignored and not parsed by MAPLE-T.

Figure 14 shows an example of an ATL transformation consisting of one rule

```
-- #sub-intent: MapE
helper def : helperMapE() : Type = …

--#Intent : DeriveBfromA, Parameters: [P1,P2]
rule Rule1 {
    from
        source: ModelA!Element1

    to
        target: ModelB!Element2
    do{
         -- some processing
        Element1 <- P1*Element2 + P2
        -- ...
        -- some processing

        thisModule.helperMapE();
     }
}
```

Figure 14: Annotated Transformation Example

named "Rule1" annotated with the intent "DeriveBfromA". It has also a helper named "helperMapE" with its intent "MapE".

5.2 Transformation Chain Augmentation and Execution

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 (see Section 2.3). 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 as byproducts of each transformation execution part of the chain and to explicitly link these trace models in the MgM to get the global traceability map. The MgM also needs to be updated with these new model instances and their relationships.

5.2.1 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 would 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 Figure 10). Similar to [53], we defined an ATL higher order transformation (HOT) to systematically enrich our transformations with traceability notions. The HOT takes the transformations part of the chain and augments them, resulting in a new chain which has the same flow but with traceability-augmented transformations.

The augmentation of transformations is similar to the concept of instrumentation in software engineering. Instrumentation is the process of adding informations to a program [73]. The additional information are generally pieces of code for collecting data about the program execution.

Figure 15 shows an example `LTrace` model (conforming to the metamodel shown in Figure 3 corresponding to the transformation shown in Figure 14.

When the transformation is executed, the `LTrace` is generated containing all the links between elements of source and target models (ModelA and ModelB in this example) as well as the application-specific intents of the traced rule and the sub-intents if any. In this example, we capture the intent of the rule "DeriveBfromA" and its specific parameters as well as a sub-intent "MapE" of a helper function called within the rule.

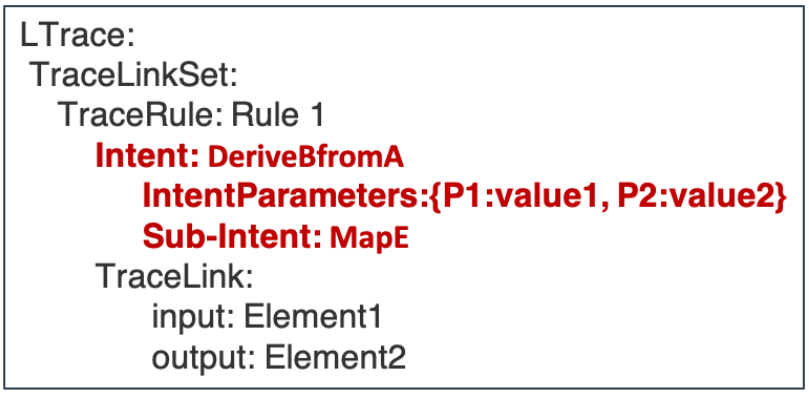

Figure 15: Sample LTrace Model

5.2.2 Megamodel Update and Global Traceability Construction

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, `GTrace`. The set of global links as well as the local traces generated for each transformation form the basis for carrying out traceability visualization and analysis in MAPLE-T.

Figure 16 shows the megamodel after enactment of the simple PM shown in Figure 5. The added input/output models as well as the `LTrace` models are shown in blue. The GTrace is the collection of all the `LTraces` (shown in blue) and the links connecting them (shown in green).

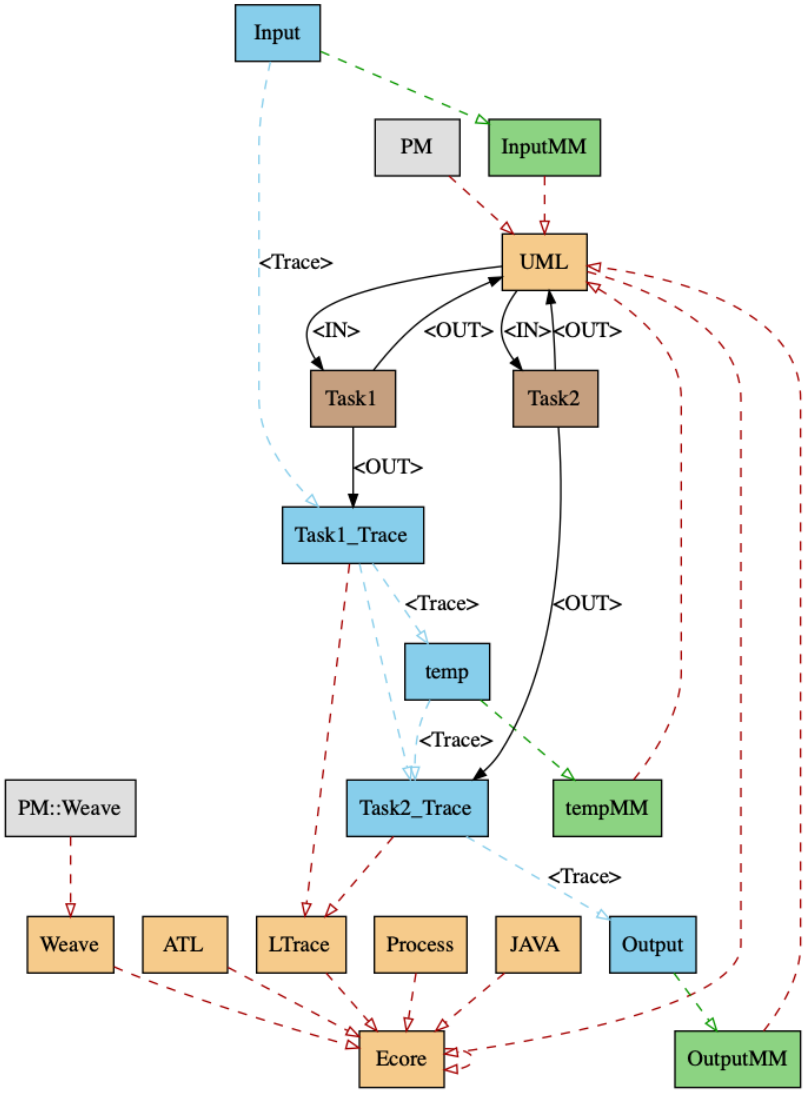

Figure 16: Example of an MgM after enacting the PM in Figure 5

Chapter 6

Traceability Visualization and Analysis

In this chapter, we present our traceability visualization and analysis approach enabled by the generated traceability information. In Section 6.1, we introduce our visualization approach which allows us to visualize relevant and useful information before and after enacting the PM. In Section 6.2, we discuss our traceability analysis approach which incorporates a change impact analysis approach as well as a semantically richer traceability analysis approach based on the captured intents.

6.1 Traceability Visualization

Visualization in MAPLE-T is incorporated in two phases of our approach. We use the visualization capabilities to visualize the relations between the PM-level intents and the components of the PM (pre-enactment) as well as the the relations between the transformation-level intents and their parameters, the trace links and the input/output artifacts of the PM (post-enactment).

6.1.1 Pre-enactment Visualization Graph

Once the PM is registered in the MgM, we generate a visualization graph associating every PM-level intent to its corresponding action and/or activity because at this stage intents are captured in the weave model and hence cannot be easily visualized. Figure 17 shows the structure of the automatically generated DOT graph. It

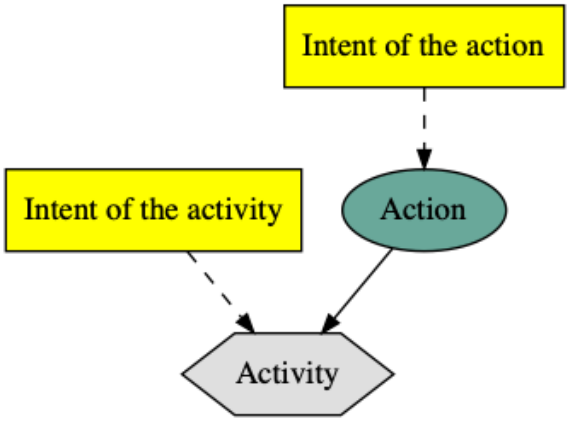

Figure 17: DOT Graph Structure for PM-level Intents

consists of rectangles representing the intents, ellipses representing the PM actions, hexagon representing the PM activity, dashed arrows associating every intent to its corresponding element and solid links linking actions to their related activity.

6.1.2 Post-enactment Visualization Graph

Once the enactment is finished and the `LTrace` and `GTrace` models are generated, we automatically create a traceability visualization graph showing every trace link between models and their elements as well as the intents/sub-intents and their parameters. Figure 18 shows the structure of such a graph. In orange , we represent every input/output model element. Dashed red arrows represent the links between the models or model elements and are labelled with the intents/sub-intents and their parameters. As shown in Figure 19, the graph can be further filtered and narrowed to focus only on a small number of elements and highlight the intent-specific parameters as well as the sub-intent for clearer visualization.

6.2 Traceability Analysis

We have incorporated traceability analysis support, specifically to carry out *change impact analysis* and *intent-based analysis* in MAPLE-T which relies on the proposed traceability generation means.

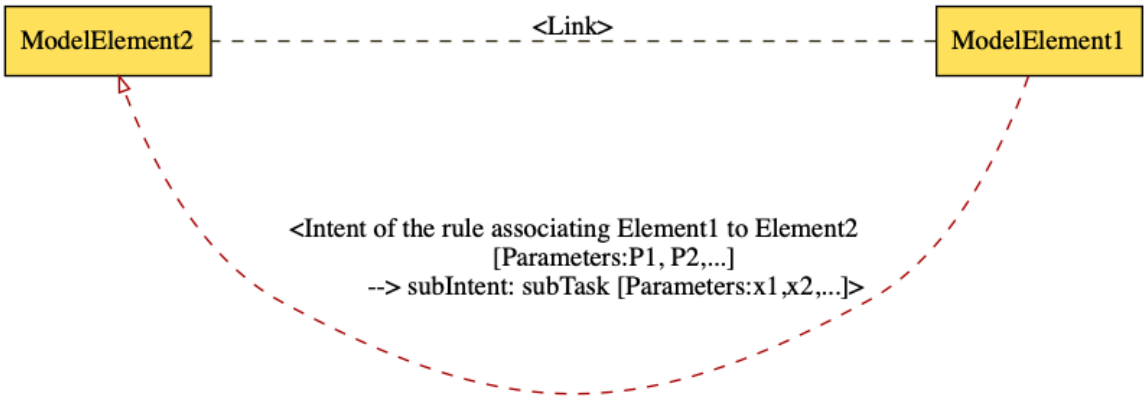

Figure 18: DOT Graph Structure for Intent-rich Traceability Graph

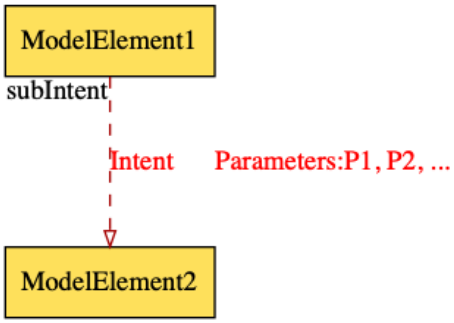

Figure 19: DOT Graph Structure for a filtered intent-rich traceability graph

6.2.1 Change Impact Analysis

Change impact analysis is defined as"identifying the potential consequences of a change, or estimating what needs to be modified to accomplish a change" [17]. The purpose of our change impact analysis approach is to determine how impactful a model or 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 model levels. It is triggered by a request specifying the element or the model for which the change impact is to be analyzed. 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 that, we can conclude whether the input is impactful (i.e, changing it can impact other artifacts in the process) or impactless (i.e, changing it has no impact on other artifacts in the process) 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, the input is concluded to

be *impactless at both levels* and no further analysis is required. On the other hand, if the input 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.

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

6.2.2 Traceability Analysis with Intents

With the augmentation of our traceability information generation with intents, application-specific traceability analysis can be enabled in MAPLE-T. As shown in Figure 20, our traceability analysis starts by analyzing every input element of the PM based on the generated traceability information. This means that we are able to explicitly link every input element with its related output element along with every *intent* and intent-specific parameter that have been captured throughout the process. The purpose of this is to generate a comprehensive and semantically-rich traceability map that provides in-depth knowledge about the process. This gives us an extensive view of the interdepencies within the process, and most importantly, reveals how artifacts are linked by *intents* and for what purpose. Also, we are able to identify every labeled intent-specific parameter used with each *intent*. The next step of the analysis is to filter this set of information based on custom parameters that the user specifies. This enables narrowing the analysis results down to information that the user is interesed in to investigate or to understand the process.

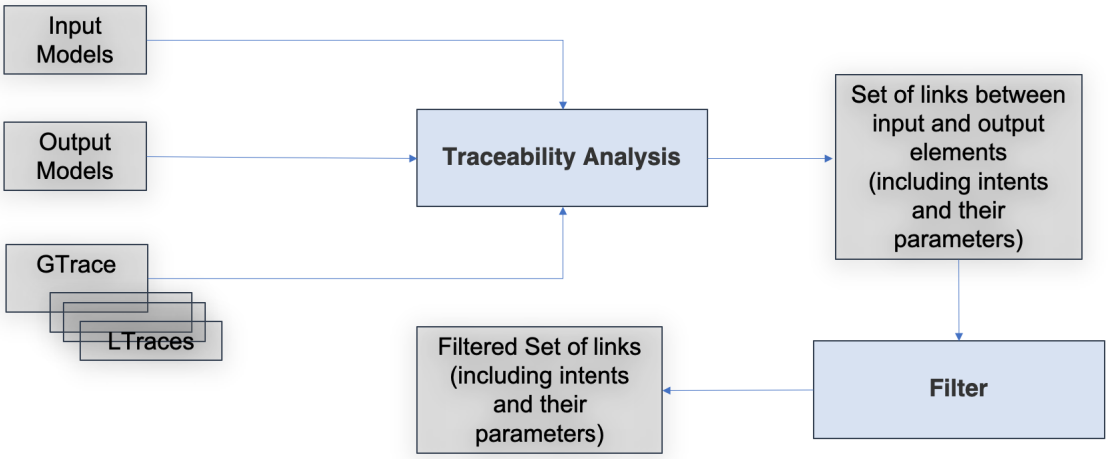

Figure 20: Intent-Based Traceability Analysis Process

Chapter 7

Tool Support

In this chapter we discuss the tool support for our approach developed as an Eclipse plugin (an entity enabling the customization of the Eclipse environment). We present existing MAPLE tool support in Section 7.1. We follow up in Section 7.2 by presenting the tool extensions developed exclusively in MAPLE-T.

The architecture of MAPLE-T is shown in Figure 21. The core functionalities are represented with rountangles (rounded-corner rectangles). Some of these functionalities provide an extension point (black circle) for specializations (shown with rectangles).

7.1 MAPLE Components

In this section we introduce existing components developed as part of MAPLE prior to our work. The **Megamodel Manager** (in orange in Figure 21) module deals with the megamodel (registering and management of resources). It is composed of the megamodel (Ecore model) itself and a manager which provides extensive interfaces to use megamodels within plugins. It binds megamodels to directories, manages the additional files that can be created (e.g., the weave , intermediate models, etc) and handles the megamodel user interface. The **Weave Engine** (in yellow) creates the weave model which maps elements of the PM to their equivalents in the MgM. The weave model keeps the MgM PM-agnostic.

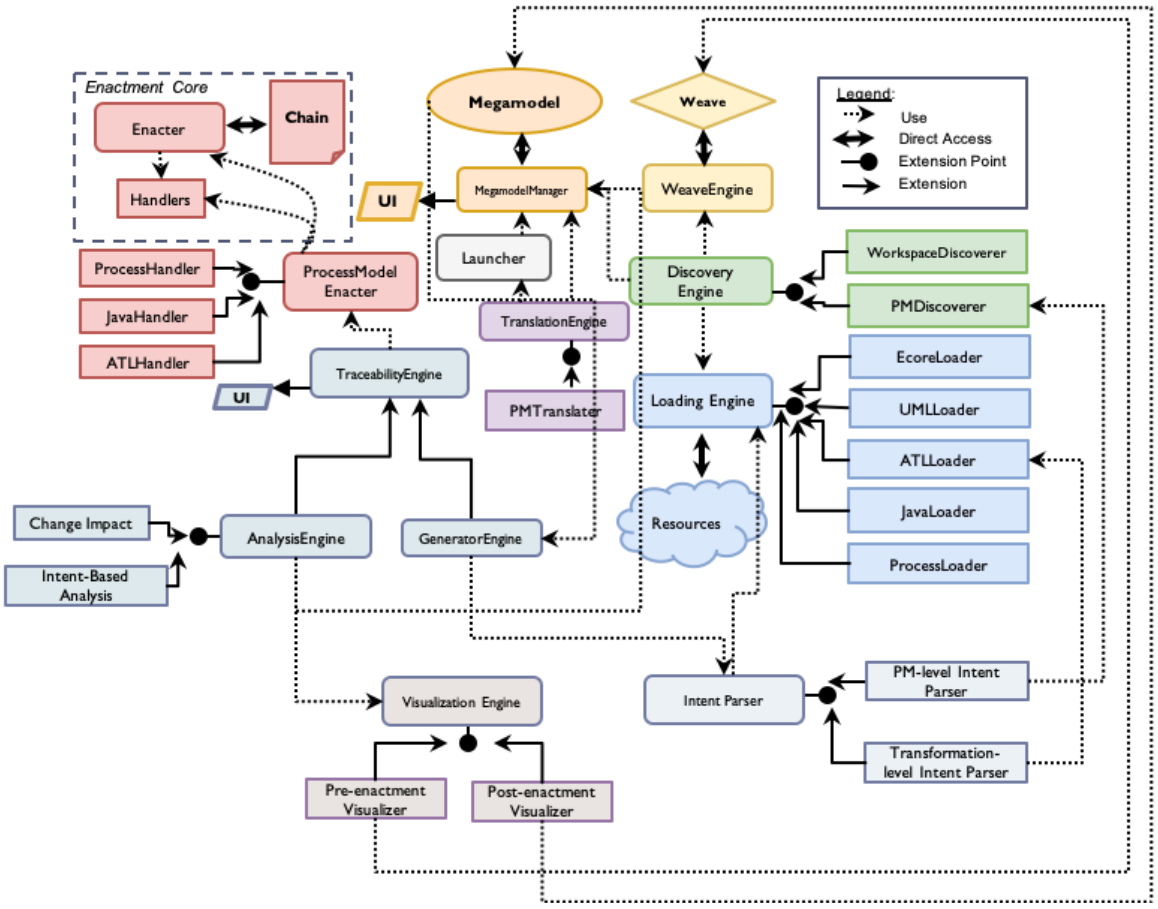

Figure 21: Backend Architecture

The **Loading Engine** (in blue) is used for loading, reading resources and extracting relevant information from them. It provides a generic extension point, that is for the system to be able to read a model, one should provide the associated loader extension, which will be automatically used by the engine. For instance, *UMLLoader* is used to read UML models, and ATLLoader is used to read ATL resources. The **Discovery Engine** (in green) is responsible for registering resources into the MgM. It is extended with specific discoverers via the exposed extension point. Two discoverer extensions are defined: *workspace discoverer* responsible for registering any non-process model resources (for example, UML profiles) , and *process model discoverer* used for discovering a PM.

The **Translation Engine** (in purple) is responsible for reading the PM and extracting its semantics. The **Enacter** (in red) handles the PM execution by configuring and launching the processes (transformations, executables, etc) it holds. It is composed of a generic and project independent enacter which is extended by the **ProcessModelEnacter** which makes use of handler interfaces representing interfaces to the different transformation engines. The **Launcher** (in gray) component serves as an interface (as a *launch configuration mechanism* in Eclipse) between the enactment components and the user.

7.2 MAPLE-T Components

Prior to building MAPLE-T, we have investigated and tested MAPLE's functionalities to identify what is supported and what is not. There were several issues encountered in the MAPLE core functionality. Some of the issues along with the fixes are listed below.

(i). When enacting a PM composed of multiple activities, we realized that each activity was being enacted individually in parallel. This created an incoherence during the enactment process. *Solution:* The MAPLE codebase needed to be modified to fix the issue so that during enactment multiple activities are merged in one transformation chain and enacted according to the PM flow (in sequence or in parallel).

(ii). The mapping between elements of the root PM and the composed activities was not functioning in the correct way. The sub-activities are defined as a call behaviour action - a separate UML activity diagram - which is part of the PM. *Solution:* The MAPLE codebase needed to be changed so that each input/output of the root PM is mapped to its corresponding input/output (activity parameter nodes of the call behaviour action) in the activities within.

We have also introduced new features in MAPLE. Mainly, we have added support to allow enactment of a heterogeneous model transformation chain.

These extensions are discussed in : [60].

In the following, we discuss components developed as part of MAPLE-T.

7.2.1 Traceability Engine

The **Traceability Engine** provides abstract features to handle the generation and analysis of traceability information during enactment. It provides an Application Programming Interface (API) for augmenting the transformation chain using a Higher-Order Transformation (HOT) and for manipulating the generated local traces and the global traceability map. It also enables user interactions through an abstract user interface.

7.2.2 Generator Engine

The **Generator Engine** represents the core component behind traceability information generation. It consists of the augmenter which is a higher-order transformation that upgrades the transformation chain with traceability features. It also provides means for the megamodel manager to access and manipulate the generated local traces (`LTrace` 4.1.2 models) in order to update the megamodel and construct the `GTrace` 4.1.3.

As shown in Figure 22, before proceeding with augmenting our transformations, we need to ensure that the transformations we provide to the HOT are passed as models (not as model transformations). Thus, we first convert each ATL model transformation into an ATL transformation model (XMI model) using the injector/extractor interfaces provided by the ATL API. Afterwards, our ATL HOT takes the XMI transformation models and augments them. In other words, it extends the transformations with extra rule bindings and expressions for the added traceability support. The XMI models are then serialized back into ATL model transformations, resulting in a new chain composed of traceability-augmented transformations. Each augmented transformation ends up having in addition to its original input/output parameters, a new target parameter (`LTrace` model) representing the trace model to be generated holding the local trace links for each in/out of the transformation execution. These trace models (`LTrace` models) are conforming to our defined `LTrace` metamodel which represents local traceability information elements (i.e., trace information at the element/attribute level).

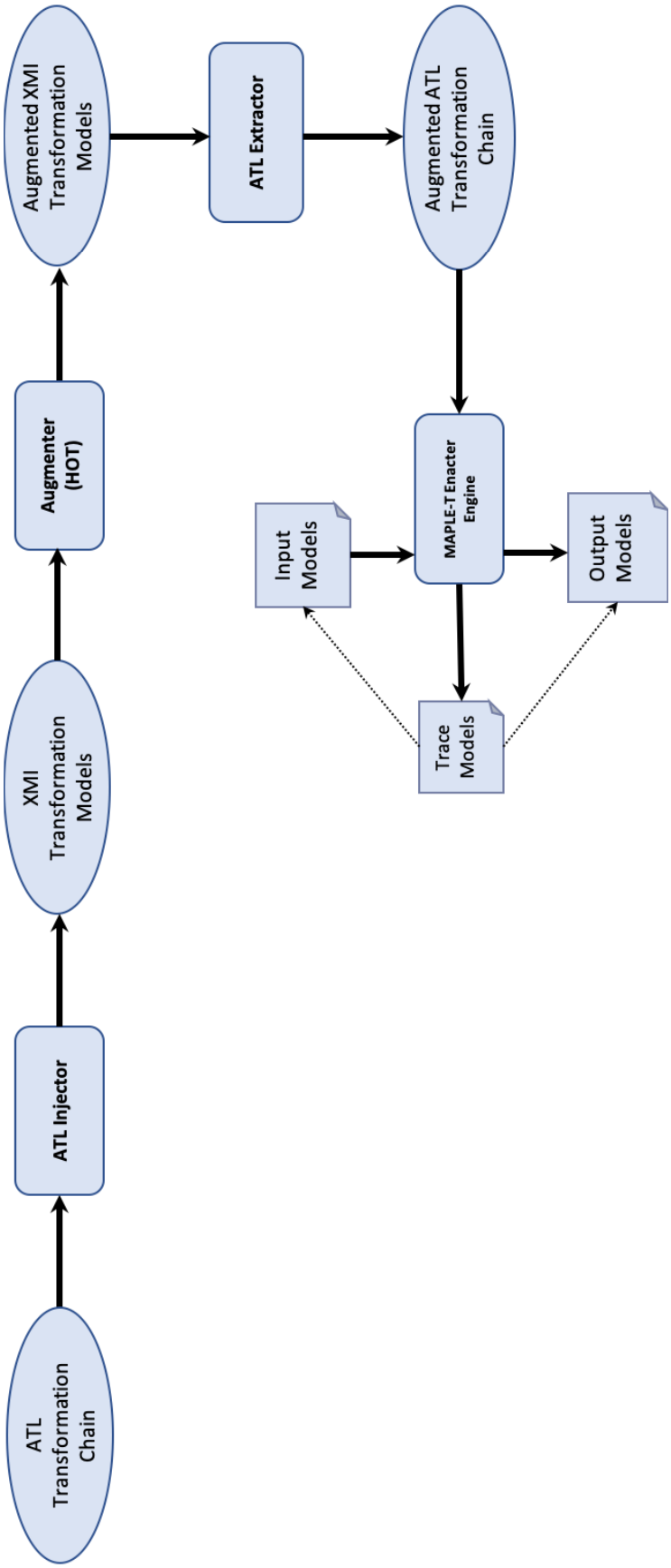

Figure 22: ATL Transformation Chain Augmentation Process

7.2.3 Analysis Engine

The **Analysis Engine** incorporates means to analyze the generated traceability information. It provides an abstract API which can be used to collect and manipulate this information for analysis for a specific application. For this purpose, it exposes an extension point to allow customization of the API features to application-specific features. Thus, MAPLE-T can be extended with different analysis engines for multiple applications and domains. For instance, a custom *change impact analysis engine* is built as an extension of the abstract engine. Furthermore, an *intent-based analysis engine* extends the generic analysis engine by using the parsed intents.

7.2.4 MAPLE-T Visualization-related Components

The **Visualization Engine** provides an abstract interface for mapping resources from the MgM or the Weave to DOT graphs.

In order to make the visualization backend more modular and easily reusable, we have built two separate extensions of the visualization engine. The first one is to visualize pre-enactment graphs and the second one is to visualize post-enactment graphs.

Pre-Enactment Visualizer

The **Pre-enactment Visualizer** is a custom extension of the abstract visualization interface. It generates a DOT graph from elements contained in the weave model. Particularly, it maps the PM actions/activities and the PM-level intent elements contained in the weave to their corresponding graph nodes in the DOT graph.

Post-Enactment Visualizer

Similarly, the **Post-enactment Visualizer** is an extension of the abstract visualization interface. It maps elements of the GTrace to nodes and edges in the DOT graph.

We have added these visualization extensions

7.2.5 Intent Parser Components

The **Intent Parser** is responsible for reading resources in order to retrieve intents. It is developed to parse any string starting with the string "#intent" using regular expression techniques to match string patterns. This parser is extended by two custom parsers. A **PM-level intent parser** responsible for parsing the PM-level intents when the PM is loaded to the MgM and a **Transformation-level intent parser** responsible for parsing the transformation-level intents when transformations are read by the loading engine.

These extensions are added in order to make the intent parser engine easily extensible and modular.

Chapter 8

Case Study: Network Service Design

In this chapter, we present the case study used to validate our work. In Section 8.1, we give an overview of the Network Functions Virtualization (NFV) framework which is the domain of application of our case study and we introduce network services (NSs). We follow up in Section 8.2 by introducing the NS Design method proposed in [62] to which we apply our approach. In Section 8.3, we illustrate the process of generating traceability information in MAPLE-T as part of the enactment of the NS Design process. In Section 8.4, we discuss the change impact analysis solution enabled by MAPLE-T applied to the NS design process. Particularly, we discuss how changing some input models impacts the output models in different scenarios. In Section 8.5, we introduce our second application concerning further diagnosis of the NS design process. We present the enhanced traceability information with intents leading to richer traceability analysis results for the NS design.

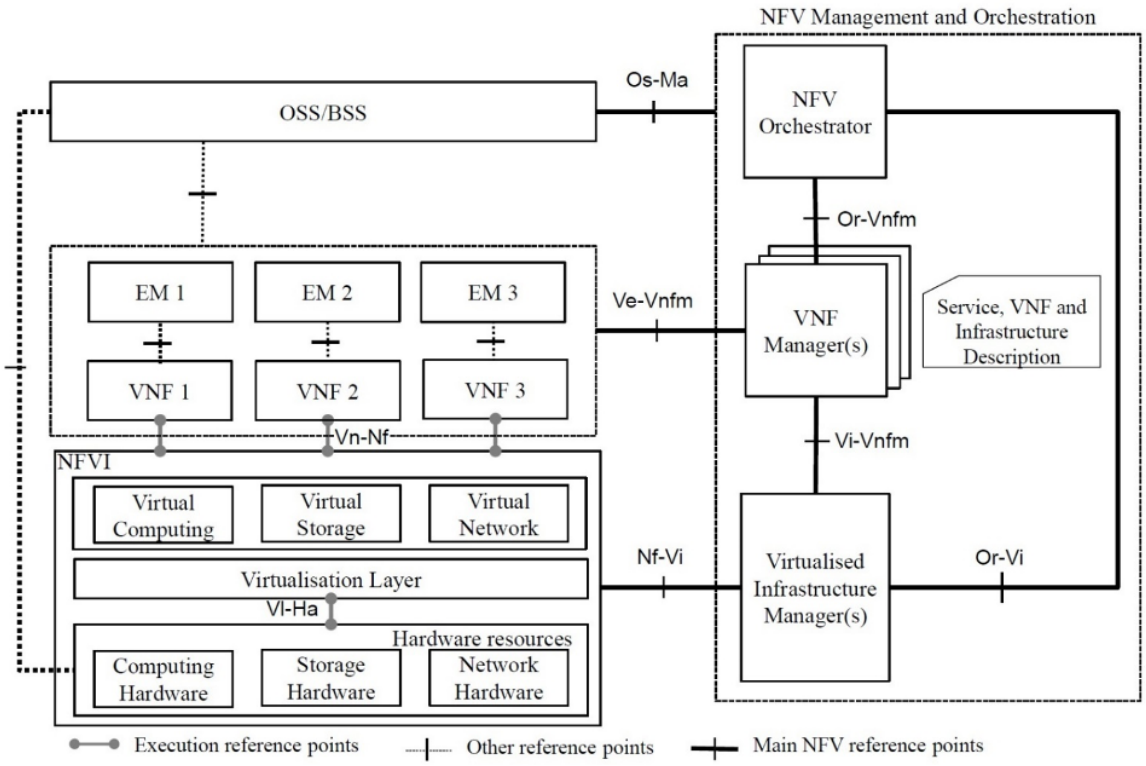

Figure 23: NFV Architectural Framework (ETSI GS NFV 002 [7])

8.1 Network Function Virtualization and Network Services

8.1.1 Network Functions Virtualization

The telecom industry has been moving from dedicated hardware /physical equipment network functions (NFs) to virtualized network functions (VNF). This has been enabled by the virtualization technology and the cloud. Network Function Virtualization (NFV) is a framework proposed and standardized by the European Telecommunications Standards Institute (ETSI) [3] for the management and orchestration of Network Services (NSs) by means of virtualization technology. NFV decouples the software implementations of the NFs from the infrastructure [8].

The NFV architectural framework proposed by ETSI is composed of four main parts, the NFV Management and orchestration (NFV-MANO), VNFs and Element Management (EM), the NFV Infrastructure (NFVI) and the Operation Support System/Business Support System (OSS/BSS) [7] as shown in Figure 23.

NFV Management and orchestration (NFV-MANO) is responsible for managing the infrastructure, allocating required resources to the VNFs and NSs, and orchestrating them [8]. It consists of three functional blocks:

- NFV Orchestrator (NFVO): responsible for the NS lifecycle management and the NFVI resources orchestration.
- VNF Manager (VNFM): responsible for the lifecycle management of VNF instances.
- Virtualized Infrastructure Manager (VIM): responsible for managing the NFVI compute, storage and network resources.

VNFs and Element Managers (EMs) A virtualized NF (VNF) is an NF deployed on NFV infrastructure virtual resources. An Element Manager (EM) is an entity responsible for the management of the VNF functionality including the security, configuration, fault management, etc.

NFV Infrastructure (NFVI) NFVI represents all the hardware and software components providing the infrastructure resources on top of which VNFs are deployed and executed. The NFVI is composed of: virtualized resources, hardware resources and the virtualization layer [7].

Operation Support System/Business Support System (OSS/BSS) The OSS/BSS is an entity encompassing the NS operator's proprietary system. It is outside the scope of the NFV architecture framework. It provides functions dealing with the operator's business and operation.

8.1.2 Network Services

A network service (NS) is a composition of network function(s) (NF) and/or other nested NSs to provide a desired (composite) functionality/behaviour. An NF can be physical (e.g. a traditional firewall device) or virtual (VNF) (e.g. a virtual firewall) and has a functional operation/behavior and well-defined external interfaces, i.e. connection points (CP). The different NFs/nested NSs within an NS are interconnected

with one or more forwarding graphs (FG) that define the traffic flows between them.

Figure 24 shows an example of an NS composed of three VNFs.

8.2 Network Service Design Process

The main goal behind the NS design process (proposed in [61]) is to automatically design an NS and generate an NS Descriptor (`NSD`) which is a template used by the NFV-MANO (see Section 8.1.1) for the deployment and management of the NS. The process starts by specifying the functional and non-functional characteristics of the NS using the NS requirements(i.e., intents in the networking domain) (`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. `NFOntology` may be updated with new information from `NSReq` after a successful design, with the onboarding of new VNFs, or manually by an expert.

A VNF is described by a VNF descriptor (`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 a part of its 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 with ETSI NFV definition and refers to the NS constituent descriptors

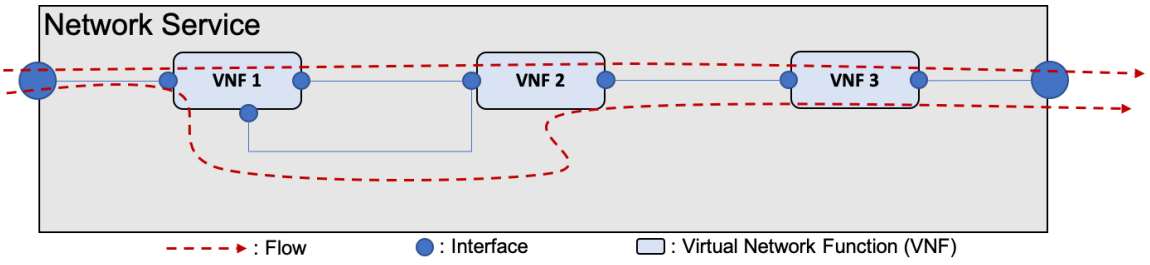

Figure 24: A Simple Network Service Example

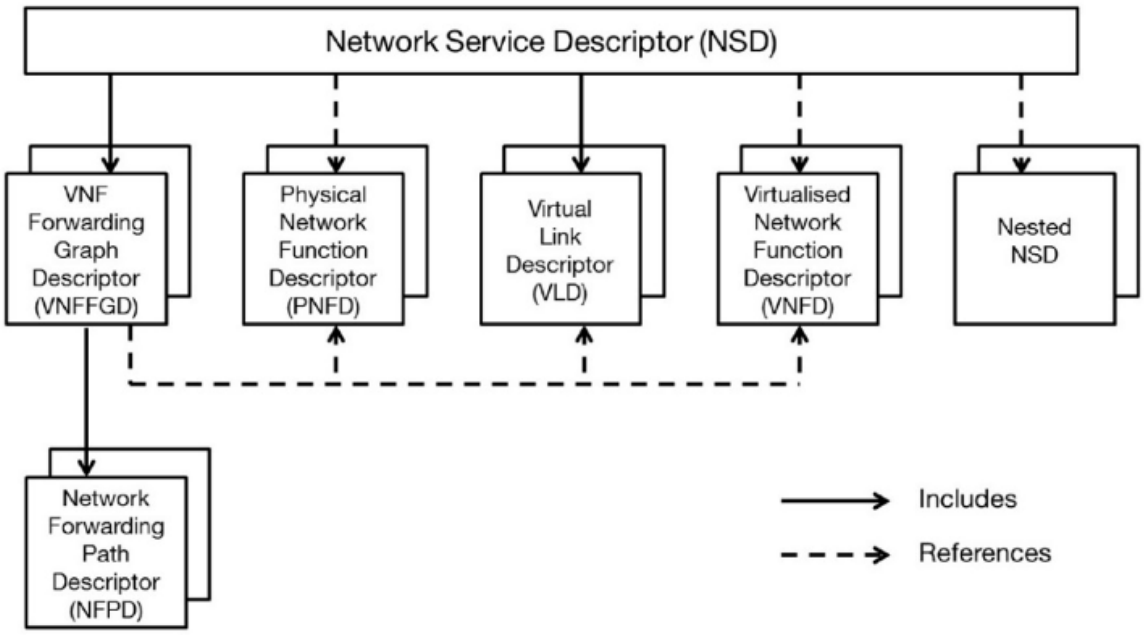

Figure 25: NSD Overview [9]

including VNFDs and VNFFG descriptors (`VNFFGDs`) as shown in Figure 25.

For the NS design process, the VNFD, provided by the VNF vendor to describe a VNF, has been extended with the VNF Architecture Descriptor (VNFAD) to provide information about the functionalities and the interfaces provided at the connection points to access these functionalities. This extension [61] is necessary for the NS design process for selecting appropriate VNFs and connecting them properly. The standard VNFD does not provide such information. This is part of a previous work [61].

.

As an example, let us consider the `NSReq` in Figure 26 with two functional requirements, Functionality1 and Functionality2, and two corresponding NFRs, T = 40 and T=60, respectively, indicating the throughput for each functionality delivered at the service interface.

After the design and dimensioning we may end up, for example, with an NS consisting of one VNF (VNF-A) realizing both functionalities as shown in Figure 26. VNF-A has four CPs providing two entry and two exit interfaces through which two flows are propagated. A flow related to a functionality can propagate within the NS through a sequence of VNF interfaces. Each flow is related to one functional requirement and its NFRs in the `NSReq`. The propagation flows within a VNF from an entry

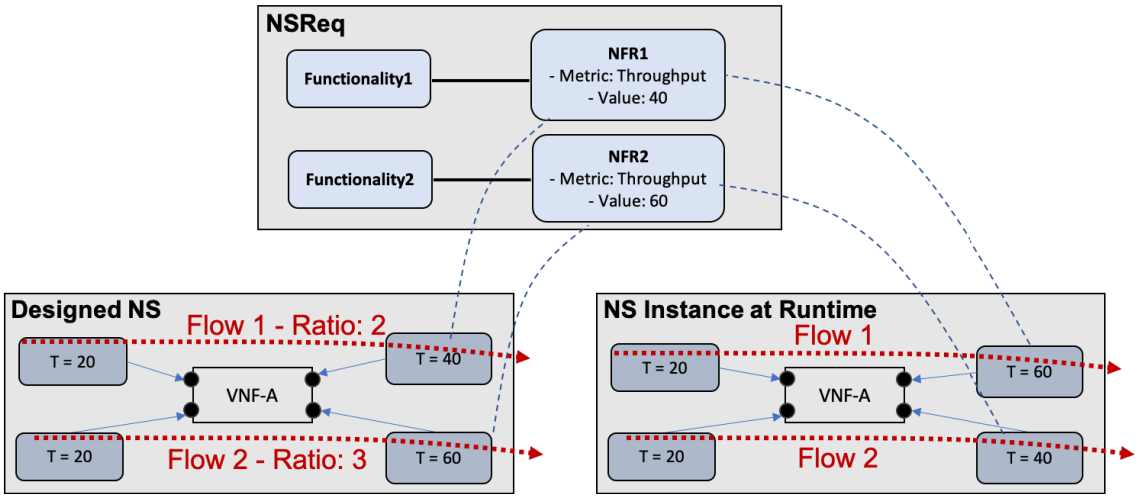

Figure 26: Network Service Design from `NSReq` Example

to an exit interface are defined in the VNFAD and characterized by a transformation ratio indicating the relation of the incoming and outgoing traffic of the flow. These ratio parameters are used in the NS design to meet the throughput requirements. In our example, a ratio of 2 was provided in the VNFAD and used for Flow 1 and a ratio of 3 was provided and used for Flow 2, to meet the required throughput at the exit interfaces. The designed NS is shown on the left hand side of Figure 26 where the throughputs and the design parameters like *ratio* are shown the way they have been used in the NS design process.

8.2.1 NS Design Process Model (PM)

Figure. 27 presents the NS design PM explicitly modelled in Papyrus as a UML2 Activity diagram. The PM is composed of six actions. The actions composing the PM are discussed here.

- **CreateSolutionMap**: This action takes as input the `NSReq` model and creates a *SolutionMap (SM)* model from it. The SM model is just an intermediate model to reduce the implementation complexity of the process.
- **SM2Ontology**: This action takes as input the *SM* model created in the first action as well as an *NFOntology* model and generates a new enriched *SM* model.

- **GeneratingFG**: This action takes as input the enriched *SM* model as well as a *VNFCatalogue* model and generates a new SM model enhanced with the information provided in the *VNFCatalogue*.

- **SM2NSD**: This action takes as input the enriched *SM* model as well as a *ProtocolStack* model (a model containing information about the TCP/IP protocol stack) and generates a generic *NSD* model as well as an *SM* model containing further details.

- **NSDRefinement**: This action takes as input the enriched *SM* model, the *ProtocolStack* model as well as the generic *NSD* model and generates a refined *NSD* model based on the non-functional requirements defined in the `NSReq` and captured in the *SM* model.

- **OntologyUpdate**: This action takes as input the enriched *SM* model, the *NFOntology* model as well as the generic *NSD* model and generates an updated *NFOntology* model.

8.3 NS Design Enactment and Traceability Information Generation

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 (Figure 7) is updated with all the registered UML profiles as well as the conformance links. Figure 28 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 Figure 29) 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).

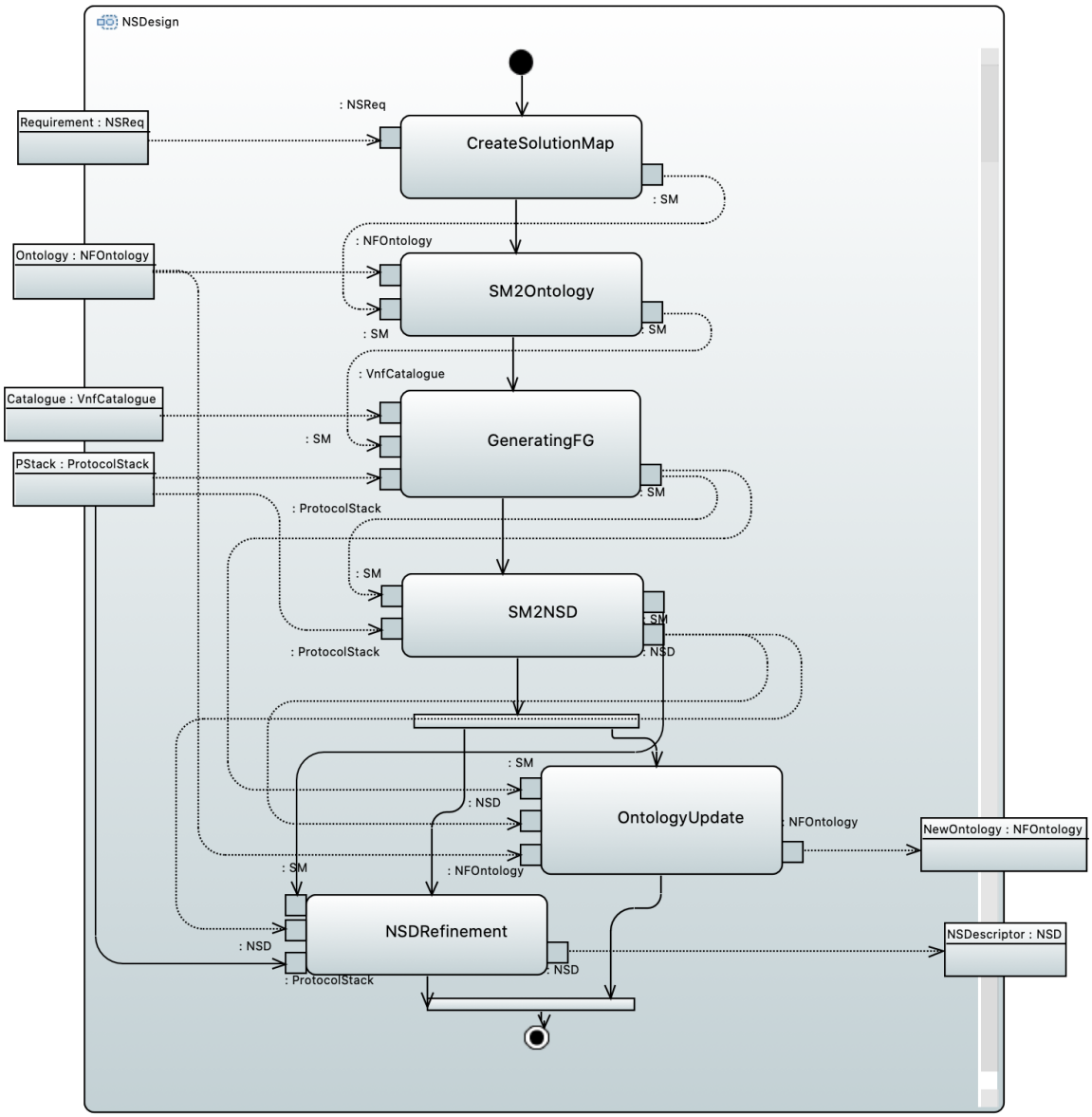

Figure 27: NS design PM [61]

With this MgM, MAPLE-T has all the necessary resources to enact the PM, and therefore enable NS Design traceability generation and analysis.

When the PM is registered, all the `intents` provided as comments in the NS design PM are retrieved and mapped to their corresponding actions in the weave model. Figure 30 illustrates the intent graph, which is automatically generated and which shows the captured PM-level intents corresponding to the actions and the activities of the NS design PM. As introduced in the previous section 6, the PM-level

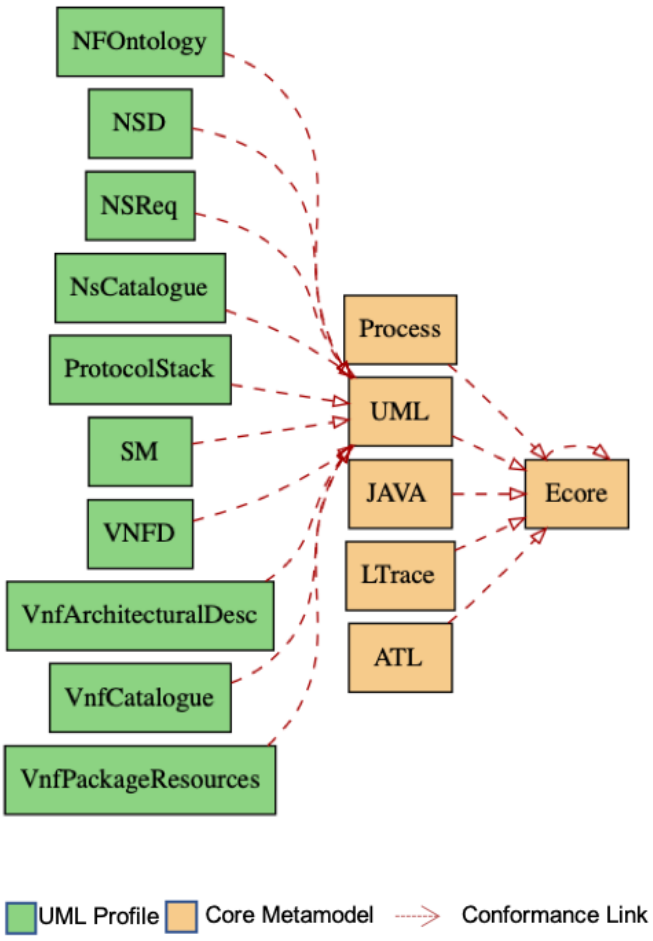

Figure 28: Initial NS Design MgM

intents are shown in yellow and are associated with actions (shown in green) in the PM as well as the activity (shown in gray) itself. For instance, the intent of the action "NSDRefinement" is "NFRBasedTailoring".

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 local trace (`LTrace`) model instances in addition to its original output model instance(s) (see Figure 31).

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

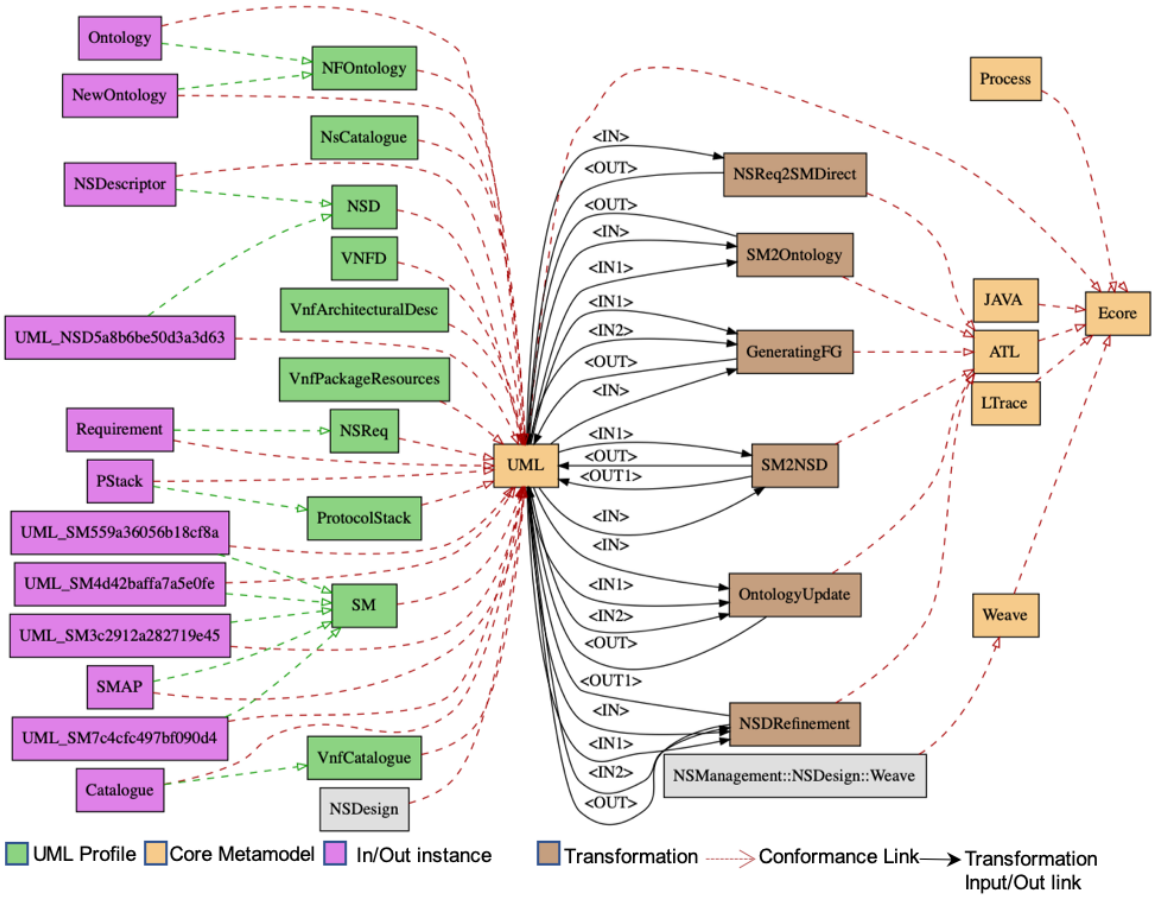

Figure 29: Updated NS Design MgM

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`.

Figure 32 shows a fragment of an `LTrace` corresponding to the execution of the last transformation in the NS design PM (NSDRefinement). It not only captures the links between source and target elements of that transformation at the model element/attribute level, but also holds intents, sub-intents and their parameters associated with the traced rule within the transformation. For example, the "Dimensioning" intent is captured within the TracedRule "calculateNumberOfInstances". It has the parameters: *VNF.Ratio* and *RequiredNumberofInstances*. It also has the sub-intent *getRealizedFunctionality* which is the intent of a helper rule (GetFunctionality) called within the TracedRule.

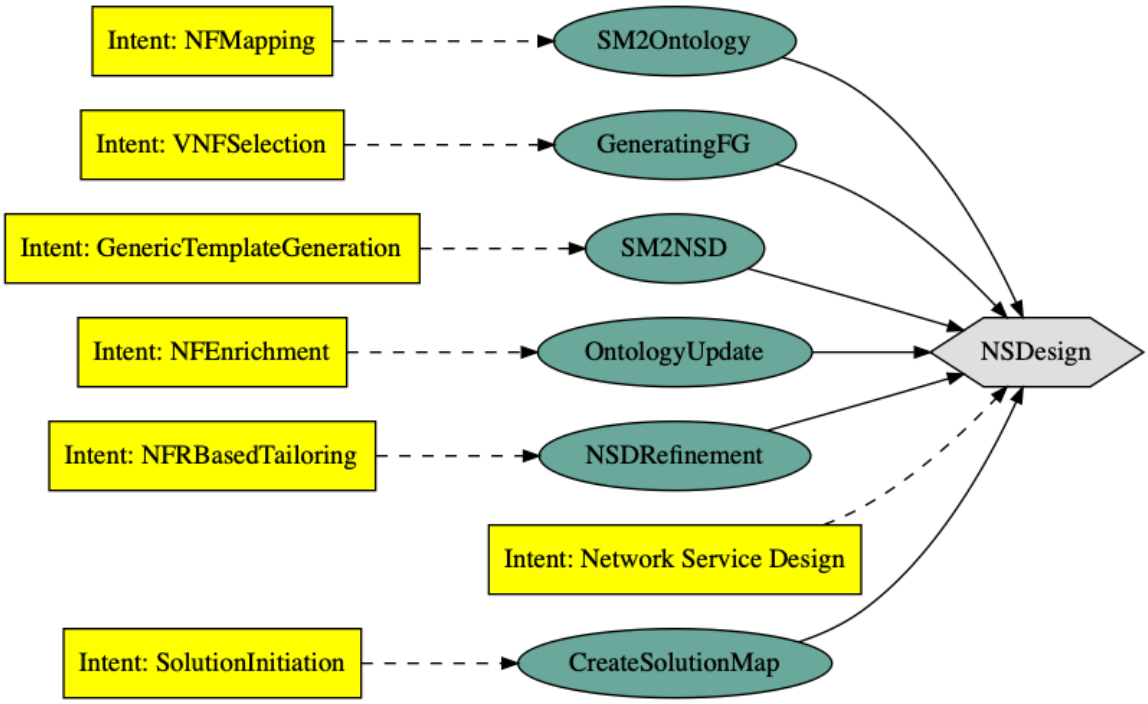

Figure 30: NS Design PM-level Intents

The MgM is updated during enactment with actual model instances (see Figure 29). 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 global traceability map (`GTrace`). The subset of the MgM representing the NS Design `GTrace` is shown in Figure 33. `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).

8.4 Change Impact Analysis for the Network Service Design

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

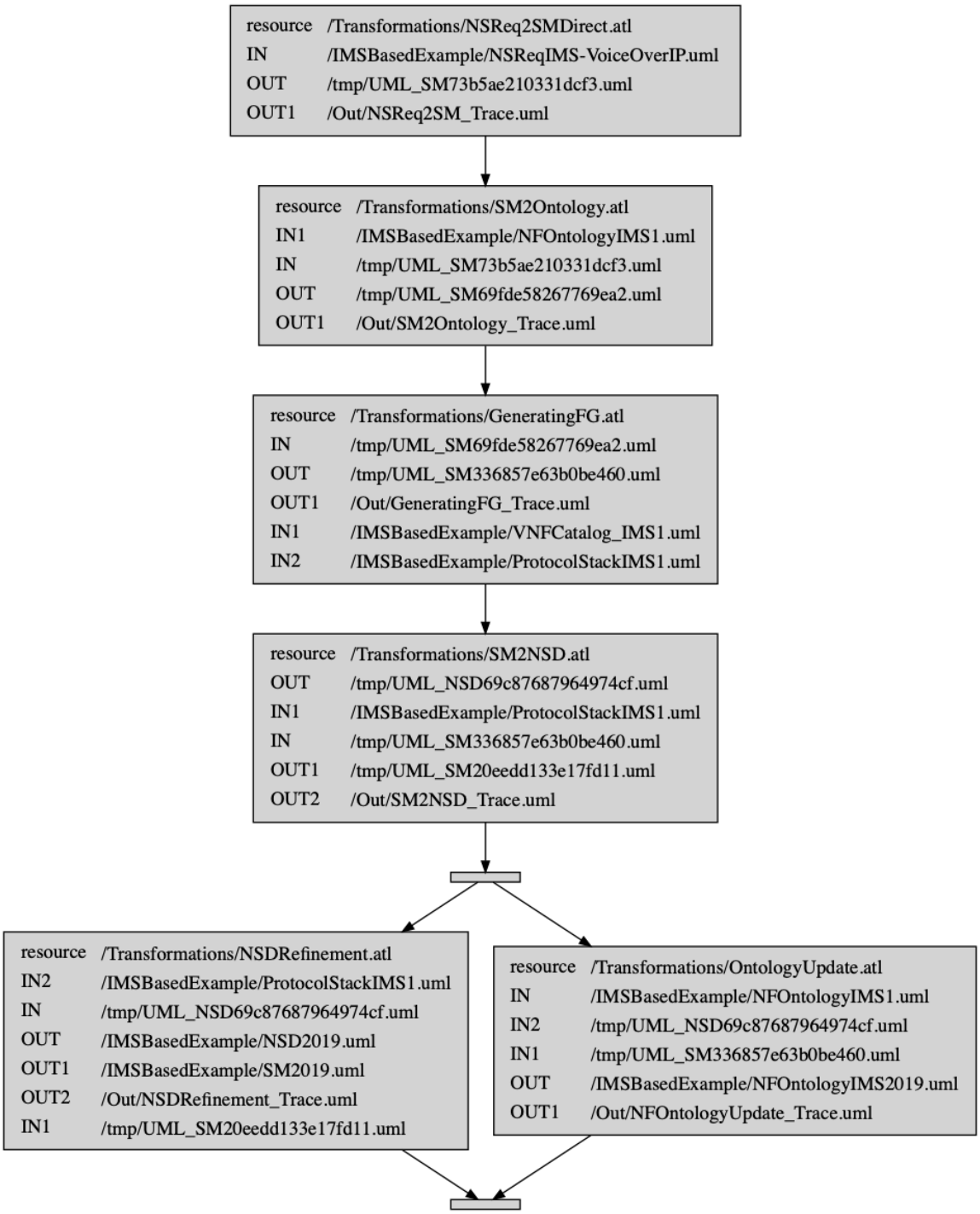

Figure 31: Augmented MT Chain

target resources (i.e; `NSReq`, `Ontology`, the `VNFCatalog` and its constituent `VNFD`s as well as the resulting `NSD` and the updated `Ontology`). Each `LTrace` model enables

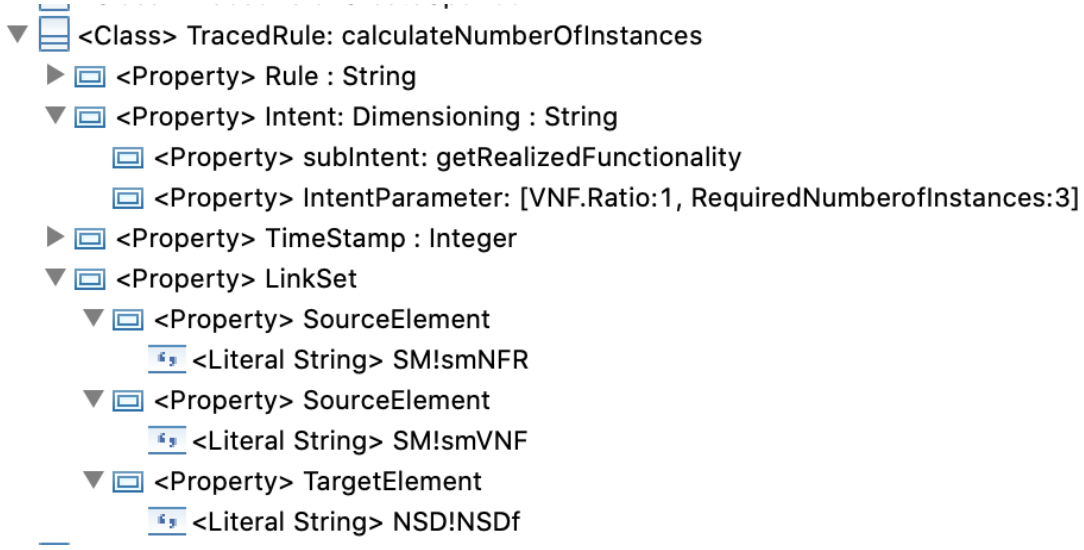

Figure 32: Augmented `LTrace`: NSD Refinement (Subset)

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 explicitly 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 *change impact analysis* in MAPLE-T. We can automatically figure out how changing an element of an input model (`NSReq`, `VNFD`s 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 desired 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 external vendors and might likely 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 are erroneous (not describing the VNF properly). In such a scenario, the decision on going about making a change in the process and associated

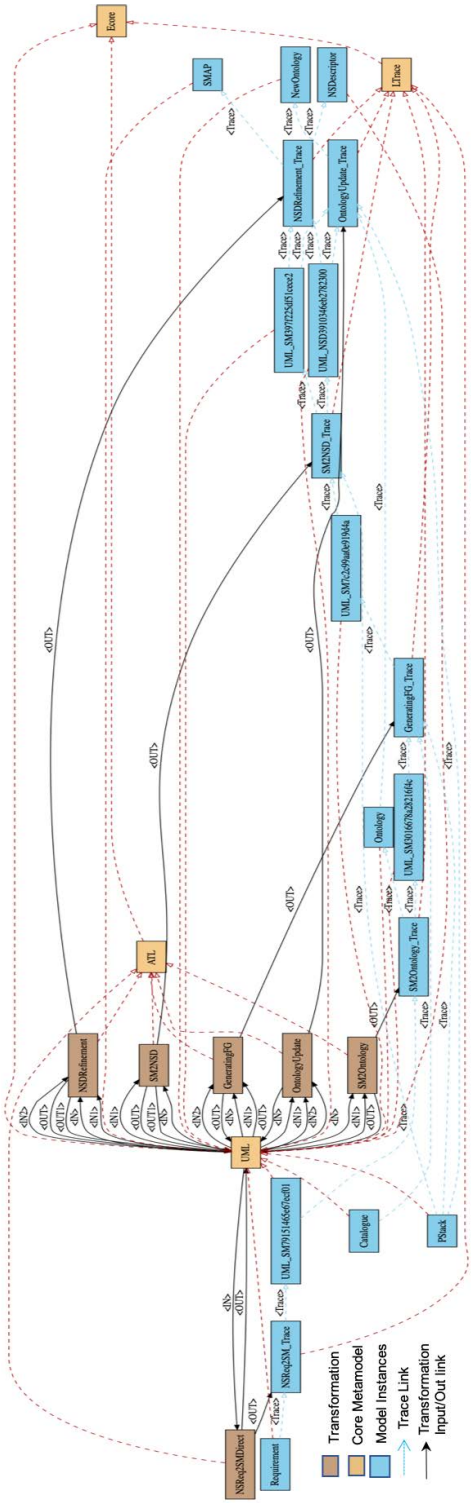

Figure 33: NS Design `GTrace`

artifacts depends on whether the running NS instance is successfully meeting the requirements (NSReq) or not. With our traceability analysis environment, we can determine whether the erroneous parameters have impact on the NS design process and therefore the generated NSD. This will allow us to know if a running NS instance is not meeting the NSReq due to an erroneous NSD (if the erroneous VNFD parameters have impact on the design) or not, and whether the NS should be re-designed and re-instantiated. In the rest of this section, we discuss both scenarios. In this analysis, we assume that our NS design approach, NSReq and NFOntology are correct and cannot be source of errors.

8.4.1 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, 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 Figure 37 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 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

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) "

Figure 34: 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"

Figure 35: 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) "

Figure 36: Impactful Instantiation Level Element

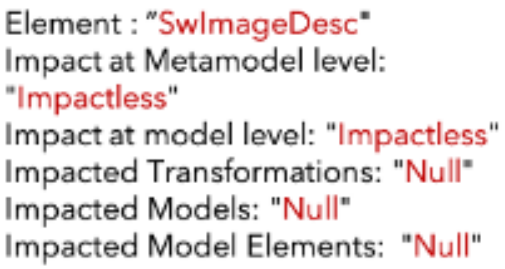

Figure 37: Impactless SwImageDesc Element

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 Figure 36). 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 Id of a `Vdu` element referenced in the `VNFD` is erroneous. As shown in Figure 35, the analysis of the change impact of the `Vdu` Id parameter on the NS Design process suggests that it is impactless. However, the reason may be that the `Vdu` with the incorrect Id was not selected because it did not meet the requirements. On the other hand, the correct Id 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.

8.4.2 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 Figure 37), we can conclude that the erroneous behaviour of the NS instance is not due to the erroneous `VNFD`. 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 consider the impact at the model level.

- Parameters are *impactful at the model level* (shown by the example in Figure 36):

  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 `VNFD`s). One needs to re-design the NS and re-deploy it.

- Parameters are *impactless at the model level*: As discussed in the first scenario (see Section 8.4.1) , 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 2.

Table 2: Summary of `VNFD` Change Impact Analysis Results

Running NS Instance / Impact Decision	**No problem has been detected**	**Problems have been detected**
Impactless at both metamodel and model levels	No re-design 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 pa-rameter error. NS needs to be re-designed
Impactful at meta-model 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.

8.5 Network Service Diagnosis

As mentioned in Section 8.1, the NFV Orchestrator is responsible for the NS lifecycle management and NFV infrastructure resource orchestration. The NS lifecycle management includes operations such as update, scaling and monitoring. Monitoring is performed based on predefined virtualized resource-related performance metrics to be monitored at an NS or a VNF level. The performance reports related to these metrics are collected from the infrastructure or VNF managers. Based on these reports, the orchestrator can scale out/in the NS when necessary, and also notify Operation Support System/Business Support System (OSS/BSS) with respect to predefined performance metrics such as throughput. In the case of problems with the reported performance metrics, OSS/BSS will need to investigate further.

Let us consider again the NS in Figure 26 and assume the NS is instantiated and deployed. Let us assume the NS instance (right hand side of Figure 26) is running, the throughputs are monitored at all interfaces and the values shown in the figure of the instance are reported to OSS/BSS. Based on these values, it can be inferred that the throughputs at the output interfaces do not meet the requirements (they do not satisfy the NFR1 and NFR2 defined in the `NSReq`). A violation of the `NSReq` has been detected. The problem may be due to the use of wrong parameters, wrong values for the parameters or wrong design decisions in the design process. The question is where to start the investigations. Knowing, for instance, the ratio relating the input interface to the output interface of a flow that has been used during the design of the NS can help and speed up these investigations. Indeed, the actual ratios of the VNF-A instance shown on the right hand side of Figure 26, i.e. calculated from the measured throughputs, are 3 and 2 for Flow 1 and Flow 2, respectively, instead of 2 and 3. One

can conclude that the ratios used in the design do not reflect the behavior of VNF-A, i.e. they are wrong in the VNFAD or the VNF instance is misbehaving with respect to these ratios. However, the ratios are not reflected among the attributes of the NS design. Even the throughputs are not associated with the VNF directly, but with the virtual links connected to the VNF.

Knowledge about the design process and the underlying implementation will allow to carry out a more meaningful and deeper investigation when an NS instance is not meeting its `NSReq`. With such knowledge, one can identify parameters hidden in the design process (e.g. Ratio) or design decisions that may impact the observed behavior (e.g. Throughput). This knowledge can be obtained with semantically-rich traceability information generated during enactment of the process.

Within the `GTrace`, every model associated with the process is explicitly interlinked to its related models with `LTrace` models and `GTrace` links. Using the incorporated visualization support, visualization graphs are generated showing every association between the NS design elements, the intents and their parameters. Based on this, investigation and analysis can be performed allowing for the explicit navigation back and forth between every `NSReq` element and its corresponding intermediate and target `NSD` elements. Furthermore, with the addition of intents at both levels, our analysis solution allows us to identify the intents/sub-intents of every action/transformation and rule executed by enacting the PM as well as to pinpoint application-specific parameters used within these intents. Such additional parameters can be used for a deeper analysis and diagnostic in case of problems observed at runtime.

An extensive traceability map is generated and enables rich traceability analysis applied on the NS design process. At this stage, a graph visualizing such a map is automatically generated. This visualization shows all the semantically-enhanced traceability information that one can navigate and analyze. Figure 38 shows a portion of the visualized map. It consists of `NSReq` elements such as FunctionalRequirement:Register, etc. It shows every intent and intent-specific parameter related to such elements. For example, FunctionalRequirement:Register element is related to NSD!NFPD[NFP-rg] through the "FlowDesign" intent. The NS design intent traceability map can be highly beneficial in different NFV contexts. For instance, one does not have to be an expert or dig into every transformation to know the intents of the

NS design process implementation. They can go through this map to identify and investigate the intents and their parameters. For easier investigations, this map of links can be further filtered based on user-specified parameters to focus only on specific elements. In this case, a filtered visualization graph is automatically generated at this stage as well. It highlights intents and their parameters according to the user-specified filter. Figure 39 shows a visualization of a filtered traceability map based on the `NSReq` element, NFR:Throughput. With this result, we can easily identify every intent and intent-specific parameters related to such an element. This analysis result helps us identify easily the application-specific parameters that can help in achieving deeper investigations in case a running NS instance is not meeting the required NFR:Throughput. Going back to the example shown in Figure 26, the intent "Dimensioning" has a "VNF Ratio" parameter, and is linked to NFR:Throughput. With this information, we know that the *ratio* has been used in the design and is linked to NFR:Throughput. If a problem with the throughputs has been detected at runtime, we now know the design parameters that may have an impact on this, the *ratio* parameter being one of them. The user can investigate the *ratios* of VNF-A as a starting point for the diagnosis of the throughput problem.

8.6 Summary

In this chapter, we presented the NFV network service design and management case study used in our work. We have demonstrated the capabilites of MAPLE-T applied to the NS design process. The enactment of the NS design process results in the generation of a network service descriptor in addition to `LTrace` models. The generated `LTrace` models are analyzed to enable a change impact analysis solution assessing the impact of changing input models (e.g., VNF catalog) on the NSD. Moreover, advanced network service diagnosis is enabled by using intents and their application-specific parameters.

Automating the management and orchestration of network services is one of the primary goals of network operators. Achieving full automation for NS management is among the main requirements of 5G. Our approach can be the basis for achieving full automation. The NS diagnosis, monitoring and other advanced management activities can be seamlessly automated with the help of our traceability generation and

Figure 38: Subset of the Generated `GTrace` Model

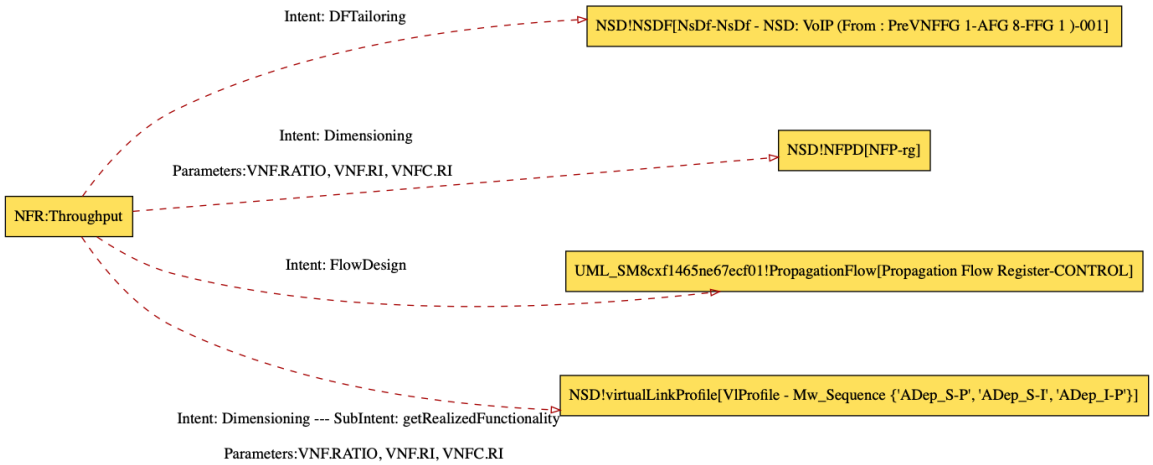

Figure 39: NS Monitoring and Diagnosis: Filtered Trace Links

analysis framework. For instance, instead of manually investigating network services, MAPLE-T can be employed to automatically provide analysis results allowing easier and efficient diagnosis.

Chapter 9

Conclusion and Future Work

9.1 Conclusion

In this thesis, we presented a model-driven traceability information generation, visualization and analysis approach. Our approach provides support for automatic generation of local and global traceability information during process enactment. It starts with a process model (PM) and a set of resources (metamodels, profiles), which are all registered in a megamodel (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, the underlying transformations are executed and as a result the target models as well as the trace models (`LTrace` models) are generated. 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 (`GTrace`) which provides traceability information at the PM-level is also part of the MgM.

Furthermore, we have extended our traceability approach to capture *intents* representing the application-specific objective of every transformation rule implementing the PM as well as the higher-level intents of the PM actions/activities. Intents also are associated with application-specific parameters giving them deeper semantics. The megamodel captures all associated intents in a PM and its implementation. We have incorported multi-level traceability visualization support in our approach. Visualization graphs are automatically built as part of the enactment process. A post-enactment visualization graph is generated illustrating the associations between

the actions/activities of the PM and their intents. After enactment, a traceability map visualization is generated showing the links between all the models, model elements, the intents/sub-intents and their parameters. This visualization model can be further filtered based on custom parameters.

We have applied our approach to an NFV case study, namely to network service (NS) design, to carry out change impact analysis and enable richer traceability analysis (using intents). The goal of the change impact analysis solution 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.

The focus of the intent-enriched traceability analysis was mainly on how capturing intents (and their parameters) and visualizing them can further ease the interpretation and understanding of the NS design process. Particularly, it can help in achieving deeper understanding of how every NS requirement (`NSReq`) element is linked to its associated NS descriptor (`NSD`) elements and why. Moreover, the results of this analysis can be used to shed light on hidden NS design intent parameters. These parameters can be investigated for better and more accurate diagnosis in case a running network service instance is not satisfying the network service requirements.

We have implemented our approach as an Eclipse plugin called MAPLE-T built on top of MAPLE [59], an extensible environment which enables model-driven process enactment by interleaving transformation chaining and model management means.

9.2 Limitations and Future Work

9.2.1 Limitations

There are some limitations to the proposed approach presented in this thesis.

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 higher order transformation(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.

The captured intents are tightly coupled to how the PM and the transformations are annotated. There is no support for the validation or the verification of the

correctness of the labels provided by the PM designer. In case the annotations are semantically incorrect, the traces will retain incorrect intents.

The generated traceability visualization graphs are not scalable nor navigable when the number of traces is large. In case of a very complex PM, the graph becomes slow to process and hard to navigate.

9.2.2 Potential Future Work

Our traceability generation approach can be extended to support further transformation and general programming languages (e.g. Java, Epsilon, etc.). MAPLE-T can be extended to generate traceability information by enacting a PM implemented using various languages.

The traceability visualization can be extended to support advanced navigability. Responsive navigation and searching can make the generated visualization graphs more user friendly and interactive.

Bibliography

[1] Dot language. https://graphviz.org/doc/info/lang.html.

[2] Eclipse modeling framework (emf). https://www.eclipse.org/modeling/emf/.

[3] European Telecommunications Standards Institute (ETSI).

[4] Graphviz. https://graphviz.org/.

[5] ATL Project, 2006.

[6] Network Functions Virtualisation - Use Cases: ETSI GS NFV 001 V1.1.1, October 2013.

[7] Network Functions Virtualisation - Architectural Framework: ETSI GS NFV 002 V1.2.1, December 2014.

[8] Network Functions Virtualisation - Management and Orchestration: ETSI GS NFV-MAN 001 V1.1.1, December 2014.

[9] Network Functions Virtualisation; Management and Orchestration; Network Service Templates Specification: ETSI GS NFV-IFA 014 V2.1.1, October 2016.

[10] *OMG Meta Object Facility (MOF) Core Specification*, November 2016.

[11] D4.1: Foundations for Model Management and Traceability. Technical report, September 2017. MegaM@Rt2.

[12] ISO/IEC/IEEE International Standard - Systems and software engineering–Vocabulary. *ISO/IEC/IEEE 24765:2017(E)*, pages 1–541, Aug 2017.

[13] D4.3: Model and Traceability Management (MTM) Tool Set – Intermediate version. Technical report, November 2018. MegaM@Rt2.

[14] W. Afzal, H. Bruneliere, D. D. Ruscio, A. Sadovykh, S. Mazzini, E. Cariou, D. Truscan, J. Cabot, D. Field, L. Pomante, and P. Smrz. The MegaMart2 ECSEL project: Megamodelling at runtime - a scalable model-based framework for continuous development and runtime validation of complex systems. In *2017 Euromicro Conference on Digital System Design (DSD)*, pages 494–501, Aug 2017.

[15] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model traceability. *IBM Systems Journal*, 45(3):515–526, 2006.

[16] M.F. Amstel, van, A. Serebrenik, and M.G.J. Brand, van den. *Visualizing traceability in model transformation compositions*. Computer science reports. Technische Universiteit Eindhoven, 2011.

[17] Robert S. Arnold. *Software Change Impact Analysis*. IEEE Computer Society Press, Washington, DC, USA, 1996.

[18] Stefan Van Baelen and Bert Vanhoof. MARTES: Traceability Management Toolset D2.3. Technical report, September 2007. EUREKA - ITEA 04006.

[19] Mikaël Barbero, Marcos Didonet, Marcos Fabro, and Jean Bézivin. Traceability and provenance issues in global model management. 04 2007.

[20] Henrik Basilier, Marian Darula, and J Wilke. Virtualizing network services - the telecom cloud. *Ericsson Review*, 91:1–9, 2014.

[21] W Bast, Mariano Belaunde, X Blanc, K Duddy, C Griffin, Simon Helsen, Michael Lawley, M Murpree, S Reddy, and S Sendall. Mof qvt final adopted specification. *OMG document ptc/05-11-01*, 01 2005.

[22] Arthur Berezin. Utilizing Declarative Model-Driven TOSCA Orchestration for NFV. DZone, March 2017.

[23] Thomas Beyhl, Regina Hebig, and Holger Giese. A model management framework for maintaining traceability links. In *Software Engineering 2013 - Workshopband*, pages 453–457, 2013.

[24] Jean Bézivin, Frédéric Jouault, Peter Rosenthal, and Patrick Valduriez. Modeling in the large and modeling in the small. In *European Conference on Model Driven Architecture: Foundations and Applications*, MDAFA'03, pages 33–46. Springer-Verlag, 2005.

[25] Markus Borg, Per Runeson, and Anders Ardö. Recovering from a decade: A systematic mapping of information retrieval approaches to software traceability. *Empirical Softw. Engg.*, 19(6):1565–1616, December 2014.

[26] Marco Brambilla, Jordi Cabot, and Manuel Wimmer. *Model-Driven Software Engineering in Practice*. Morgan & Claypool Publishers, 1st edition, 2012.

[27] YuLing Chen, Yinghua Qin, Mark Lambe, and Wenjing Chu. Realizing network function virtualization management and orchestration with model-based open architecture. In *11th International Conference on Network and Service Management (CNSM '15)*, pages 410–418. IEEE, 2015.

[28] R. Cohen, K. Barabash, B. Rochwerger, L. Schour, D. Crisan, R. Birke, C. Minkenberg, M. Gusat, R. Recio, and V. Jain. An intent-based approach for network virtualization. In *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 42–50, 2013.

[29] Krzysztof Czarnecki. Generative programming: Methods, techniques, and applications tutorial abstract. In Cristina Gacek, editor, *Software Reuse: Methods, Techniques, and Tools*, pages 351–352, Berlin, Heidelberg, 2002. Springer Berlin Heidelberg.

[30] Jeremy Dick. Rich traceability. 01 2002.

[31] Nicholas Drivalos, Richard F. Paige, Kiran Jude Fernandes, and Dimitrios S. Kolovos. Towards rigorously defined model-to-model traceability. 2008.

[32] G. Dupont, S. Mustafiz, F. Khendek, and M. Toeroe. Building domain-specific modelling environments with papyrus: An experience report. In *2018 IEEE/ACM 10th International Workshop on Modelling in Software Engineering (MiSE)*, pages 49–56, 2018.

[33] Eclipse. Apache Ant. `https://ant.apache.org`. Accesed: 2019-01-29.

[34] Eclipse. ATL EMF Transformation Virtual Machine (ATL EMFTVM). `https://wiki.eclipse.org/ATL/EMFTVM`.

[35] Eclipse. Papyrus. `https://eclipse.org/papyrus/`. Accesed: 2018-12-01.

[36] ETSI. *Zero-touch Network and Service Management*, December 2017.

[37] Jean-Marie Favre. Towards a basic theory to model model driven engineering. In *3rd Workshop in Software Model Engineering, WiSME*, pages 262–271, 2004.

[38] Peter Feiler and Watts Humphrey. Software process development and enactment: Concepts and definitions. Technical Report CMU/SEI-92-TR-004, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1992.

[39] Eclipse Foundation. Eclipse webpage. `https://eclipse.org/`.

[40] R. France and B. Rumpe. Model-driven development of complex software: A research roadmap. In *Future of Software Engineering (FOSE '07)*, pages 37–54, 2007.

[41] M. Fritzsche, H. Brunelière, B. Vanhooff, Y. Berbers, F. Jouault, and W. Gilani. Applying megamodelling to model driven performance engineering. In *16th IEEE Engineering of Computer Based Systems, ECBS 2009*, pages 244–253, April 2009.

[42] M Fritzsche, J Johannes, S Zschaler, A Zherebtsov, and A Terekhov. Application of Tracing Techniques in Model-Driven Performance Engineering. In *4th ECMDA Traceability Workshop (2008)*, pages 1 – 10.

[43] Nick L. S. Fung, Sahar Kokaly, Alessio Di Sandro, Rick Salay, and Marsha Chechik. MMINT-A: A Tool for Automated Change Impact Assessment on Assurance Cases. In *Computer Safety, Reliability, and Security*, pages 60–70. Springer, 2018.

[44] I. Galvao and A. Goknil. Survey of traceability approaches in model-driven engineering. In *IEEE EDOC 2007*, pages 313–313, Oct 2007.

[45] Victor Guana Garces. *End-to-end Fine-grained Traceability Analysis in Model Transformations and Transformation Chains*. PhD thesis, University of Alberta, 2017.

[46] Arda Göknil, Ivan Ivanov, and Klaas van den Berg. Change impact analysis based on formalization of trace relations for requirements. In *ECMDA Traceability Workshop (ECMDA-TW)*, number 274, pages 59–75. SINTEF Report, 6 2008.

[47] Victor Guana and Eleni Stroulia. ChainTracker, a Model-Transformation Trace Analysis Tool for Code-Generation Environments. In *Theory and Practice of Model Transformations*, pages 146–153. Springer, 2014.

[48] Y. Han, J. Li, D. Hoang, J. Yoo, and J. W. Hong. An intent-based network virtualization platform for sdn. In *2016 12th International Conference on Network and Service Management (CNSM)*, pages 353–358, 2016.

[49] O. Hassane, S. Mustafiz, F. Khendek, and M. Toeroe. Maple-t: A tool for process enactment with traceability support. In *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, pages 759–763, Sep. 2019.

[50] Omar Hassane, Sadaf Mustafiz, Ferhat Khendek, and Maria Toeroe. Process enactment with traceability support for nfv systems. In *System Analysis and Modeling. Languages, Methods, and Tools for Industry 4.0*, pages 116–135, Cham, 2019. Springer International Publishing.

[51] Omar Hassane, Sadaf Mustafiz, Ferhat Khendek, and Maria Toeroe. A model traceability framework for network service management. In *System Analysis and Modeling. Languages, Methods, and Tools for AI-based Systems*, page Submitted for publication. Springer, 2020.

[52] Regina Hebig, Andreas Seibel, and Holger Giese. On the unification of megamodels. 2012.

[53] Frédéric Jouault. Loosely coupled traceability for ATL. In *ECMDA Workshop on Traceability*, pages 29–37, 2005.

[54] Frédéric Jouault, Bert Vanhooff, Hugo Bruneliere, Guillaume Doux, Yolande Berbers, and Jean Bézivin. Inter-DSL Coordination Support by Combining Megamodeling and Model Weaving. In *ACM 25th SAC 2010*, pages 2011–2018, March 2010.

[55] Amar Kapadia. https://www.aarnanetworks.com/single-post/2017/11/02/The-Magic-of-Model-Driven-Design-in-ONAP, November 2017.

[56] Levi Lúcio, Moussa Amrani, Juergen Dingel, Leen Lambers, Rick Salay, Gehan M. K. Selim, Eugene Syriani, and Manuel Wimmer. Model transformation intents and their properties. *Software & Systems Modeling*, 15(3):647–684, 2016.

[57] Rashid Mijumbi, Joan Serrat, Juan-Luis Gorricho, Steven Latre, Marinos Charalambides, and Diego Lopez. Management and orchestration challenges in network functions virtualization. *IEEE Communications Magazine*, 54(1):98–105, January 2016.

[58] Pierre-Alain Muller, Franck Fleurey, and Jean-Marc Jézéquel. Weaving executability into object-oriented meta-languages. In Lionel Briand and Clay Williams, editors, *Model Driven Engineering Languages and Systems*, pages 264–278, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[59] Sadaf Mustafiz, Guillaume Dupont, Ferhat Khendek, and Maria Toeroe. MAPLE: An integrated process modelling and enactment environment for nfv systems. In *14th European Conference on Modelling Foundations and Applications (ECMFA 2018), Toulouse, France, June 2018, Proceedings*, pages 164–178. Springer International Publishing, June 2018.

[60] Sadaf Mustafiz, Omar Hassane, Guillaume Dupont, Ferhat Khendek, and Maria Toeroe. Model-driven process modelling and enactment for nfv systems with MAPLE. In *International Journal on Software and Systems Modeling (SoSyM)*. Springer, February 2020.

[61] Sadaf Mustafiz, Navid Nazarzadeoghaz, Guillaume Dupont, Ferhat Khendek, and Maria Toeroe. A model-driven process enactment approach for network service design. In *SDL 2017: Model-Driven Engineering for Future Internet - 18th International SDL Forum*, volume 10567 of *LNCS*, pages 99–118. Springer, 2017.

[62] Sadaf Mustafiz, Francis Palma, Maria Toeroe, and Ferhat Khendek. A network service design and deployment process for NFV systems. In *15th IEEE Network Computing and Applications, NCA 2016*, pages 131–139. IEEE Computer Society, 2016.

[63] Object Management Group. *Object Constraint Language (OCL)*, February 2014.

[64] Object Management Group. *Unified Modeling Language (UML 2.5)*, March 2015.

[65] Object Management Group. *Unified Modeling Language (UML 2.5)*, March 2015.

[66] Object Management Group. *OMG Systems Modeling Language*, May 2017.

[67] Richard F. Paige, Nikolaos Drivalos, Dimitrios S. Kolovos, Kiran J. Fernandes, Christopher Power, Goran K. Olsen, and Steffen Zschaler. Rigorous identification and encoding of trace-links in model-driven engineering. *Software & Systems Modeling*, 10(4):469–487, 2011.

[68] Jean rÃ©my Falleri, Marianne Huchard, and ClÃ©mentine Nebut. C.: Towards a traceability framework for model transformations in kermeta. In *In: ECMDA-TW Workshop*, 2006.

[69] Beatriz A. Sanchez. Context-aware traceability across heterogeneous modelling environments. In *Proceedings of the 21st ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, MODELS '18, page 174–179, New York, NY, USA, 2018. Association for Computing Machinery.

[70] I. Santiago, J. M. Vara, V. de Castro, and E. Marcos. Visualizing traceability information with itrace. In *2014 9th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE)*, pages 1–11, April 2014.

[71] IváN Santiago, ÁLvaro JiméNez, Juan Manuel Vara, Valeria De Castro, VeróNica A. Bollati, and Esperanza Marcos. Model-driven engineering as a new landscape for traceability management: A systematic literature review. *Inf. Softw. Technol.*, 54(12):1340–1356, December 2012.

[72] Iván Santiago, Juan Manuel Vara, María Valeria de Castro, and Esperanza Marcos. Towards the effective use of traceability in model-driven engineering projects.

In *International Conference on Conceptual Modeling*, pages 429–437. Springer, 2013.

[73] C.U. Smith and L.G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley object technology series. Addison-Wesley, 2001.

[74] Massimo Tisi, Frédéric Jouault, Piero Fraternali, Stefano Ceri, and Jean Bézivin. *On the Use of Higher-Order Model Transformations*, pages 18–33. Springer, 2009.

[75] Marcel F. van Amstel, Mark G. J. van den Brand, and Alexander Serebrenik. Traceability Visualization in Model Transformations with TraceVis. In *Theory and Practice of Model Transformations*, pages 152–159. Springer, 2012.

[76] Bert Vanhooff. Loosely coupled transformation chains: How to enable transformation reuse with traceability information. Phd Thesis, Katholieke Universiteit Leuven, 2010.

[77] Bert Vanhooff, Dhouha Ayed, Stefan Van Baelen, Wouter Joosen, and Yolande Berbers. UniTI: A Unified Transformation Infrastructure. In *Model Driven Engineering Languages and Systems (MODELS 2007)*, pages 31–45. Springer, 2007.

[78] Bert Vanhooff, Stefan Van Baelen, Wouter Joosen, and Yolande Berbers. Traceability as input for model transformations. In *ECMDA Traceability Workshop (ECMDA-TW) 2007*, pages 37–46. SINTEF, 2007.

[79] Jens von Pilgrim, Kristian Duske, and Paul McIntosh. Eclipse gef3d: Bringing 3d to existing 2d editors. *Information Visualization*, 8(2):107–119, 2009.

[80] Jens von Pilgrim, Bert Vanhooff, Immo Schulz-Gerlach, and Yolande Berbers. Constructing and visualizing transformation chains. In Ina Schieferdecker and Alan Hartman, editors, *Model Driven Architecture – Foundations and Applications*, pages 17–32. Springer, 2008.

[81] Stefan Winkler and Jens Pilgrim. A survey of traceability in requirements engineering and model-driven development. *Softw. Syst. Model.*, 9(4):529–565, September 2010.

[82] Andrés Yie and Dennis Wagelaar. Advanced Traceability for ATL. In *1st International Workshop on Model Transformation with ATL (MtATL'09)*, 2009.