Dominik Bork, Dimitris Karagiannis, Heinrich C. Mayr (Hrsg.): Modellierung 2020,
Lecture Notes in Informatics (LNI), Gesellschaft für Informatik, Bonn 2020 157

Modeling Low-Level Network Configurations for Analysis, Simulation and Testing

Marcel Schuster[1] Markus Germeier[2] Frank Hilken[3] Martin Gogolla[4] Karsten Sohr[5]

Abstract: In this paper, we present an approach to specifying a network configuration model based on the ISO/OSI reference model. The network configuration model is capable of modeling the lower layers of networks enabling several use cases: (a) analyze existing network configurations, e.g., to find configuration errors and identify which components contribute to the error; (b) simulate changes made to the configuration and predict their consequences; (c) serve as documentation for the network; and (d) visualize the network in order to better understand it and to make clear its structure to everyday users. These analysis techniques were applied to essential parts of a professional network center.

Keywords: UML and OCL model; Network configuration model; Network configuration analysis; Network configuration documentation.

1 Introduction

The administration of large-scale computer networks is tedious and error-prone. Since network documentation is often difficult to establish and maintain for administrators, a supporting methodology to systematically model and test a computer network is desirable. In particular, this applies to the lower layers of the well-known *Open Systems Interconnection* (OSI) model [IT94] as these layers deal with technical details where often administration errors occur.

Here, we address this problem with the help of UML and OCL modeling and the validation of UML models. We first define a UML/OCL model, which represents the frame for expressing the concepts of the two lowest layers of the OSI model. This allows us to represent network concepts, such as "network component", "interface", "interface link", and "Virtual LAN" (VLAN), by using UML class diagrams and complementing OCL constraints. The concrete description of a computer network is then expressed in form of a UML object diagram.

After defining our language for representing computer networks based on UML, we use the UML-based Specification Environment (USE) [GBR07, GHD17] for different validation tasks. First, we automatically read network descriptions for a real-world network (a university

[1] University of Bremen, Database Systems Group, Bremen, Germany, maschu@informatik.uni-bremen.de
[2] Center for Networks (ZFN), Bremen, Germany, zfn-leitung@uni-bremen.de
[3] University of Bremen, Database Systems Group, Bremen, Germany, fhilken@informatik.uni-bremen.de
[4] University of Bremen, Database Systems Group, Bremen, Germany, gogolla@informatik.uni-bremen.de
[5] Center for Computing and Communication Technologies (TZI), Bremen, Germany, sohr@tzi.de

housing center) and transform them into a corresponding UML object diagram. This diagram is subject to the validation tasks. If certain OCL constraints are violated, this gives an administrator hints that her network descriptions might be inconsistent. The ability to query the system state helps to identify the cause for the inconsistencies. For example, we found a concrete administration error in the network configuration. After reporting, this error was immediately eliminated by the administration of the housing center.

Our main contribution lies in the practical application of UML modeling and model validation in the area of network administration. In particular, the model concentrates on the lower layers 2 and below, which are seldom covered in other approaches. Based on this UML model we test concrete network configurations, which are represented as UML object diagrams. We then demonstrate the effectiveness of our approach with a real-world computer network at our university.

The rest of the paper is structured as follows. After presenting the technical background we explain our network configuration model for computer networks in Sect. 3. Section 4 describes the validation tasks in detail and Section 5 discusses related work. A discussion of the experience with our approach concludes the paper.

2 Background

2.1 Communication in Computer Networks

The OSI model is a communication model that describes the communication between two systems in seven layers [IT94], among them a *physical layer* and a *data link layer*. The data link layer provides mechanisms for detecting and correcting so-called frame errors. The "Ethernet" is a widely distributed and commonly used technology for LAN network communications [TW11]. A frequently used standard in corporate networks is IEEE 802.1Q [IE14a] which specifies virtual LANs (VLAN). VLANs provide the possibility for a logical separation of physical infrastructure for performance and security reasons. The implementation requires VLAN-aware network components and the main idea is to assign network interfaces to a VLAN by configuring a number. The number is called "VLAN-ID" and represents the membership of an interface to a certain VLAN. Figure 1 shows a switch interconnecting four hosts. Interface 1 and 2 belong to VLAN 10 and interface 3 and 4 to VLAN 20. Due to the VLAN memberships of the interfaces, only hosts A and B can communicate with each other on layer 2 (analogously hosts C and D).

In practice one can differentiate between "access interfaces", which can be member of exactly one VLAN, and "trunk interfaces", which can be member of multiple VLANs [TW11]. Every incoming frame to an interface is classified and assigned to a certain VLAN. Another commonly used Ethernet standard in corporate networks is IEEE 802.1AX [IE14b], which introduces the aggregation of physical links between network components. Parallel point-to-point connections between two components can be aggregated to one logical interconnection

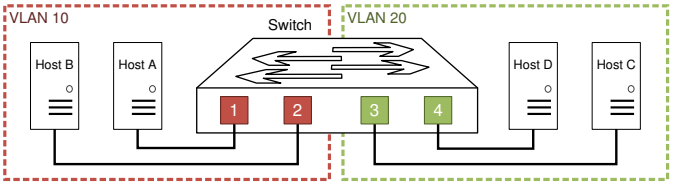

Fig. 1: Two VLANs on the ports of a switch.

called *Link Aggregation Group* (LAG) resulting in higher bandwidth and higher resilience due to redundancy. The link aggregation groups are not limited to interconnect only two network components. The according standard defines a so-called *Distributed Resilient Network Interconnect* (DRNI), which allows the aggregation of multi-homed connections resulting in a multi-chassis LAG (MC-LAG). For the operation of the individual components of the DRNI, an *Intra-Portal Link* (IPL) is mandatory, which is used for synchronization of status/configuration data.

These network techniques are some examples of commonly used concepts in industrial networks. Each one raises the complexity of network configurations in terms of sheer size and keeping track of the network structure in general. A modeling and analysis approach supporting network administration is desirable.

2.2 University of Bremen Housing Center

The housing center of the University of Bremen is the central data center which consolidates the IT infrastructure of all research groups. It is divided into two fire compartments, which have a total capacity of 4,000 rack-mounted servers. It is connected with a 80 Gbit/s bandwidth to the campus network.

The network design of the housing center follows the common data center network design consisting of three layers: *core layer*, *distribution layer* and *access layer*. The network components of the *core layer* build the heart of the network. They have high requirements regarding performance and resilience as a failure can lead to a complete outage of the housing center. The components of the *distribution layer* connect the core components to the access layer. The network components of the *access layer* are located in the two fire compartments of the housing center and provide the network access of the housed servers. The housing center uses components from Cisco Systems.

Despite not having many network components in the setup of the housing center, the individual configurations are complex and hard to maintain. This, for example, includes multiple VLANs for the separation of network traffic, LAG and MC-LAG setups for increasing the resilience of interconnections and diverse virtual routing instances (VRF) containing IP addresses and IP routes for an independent package routing. The configuration

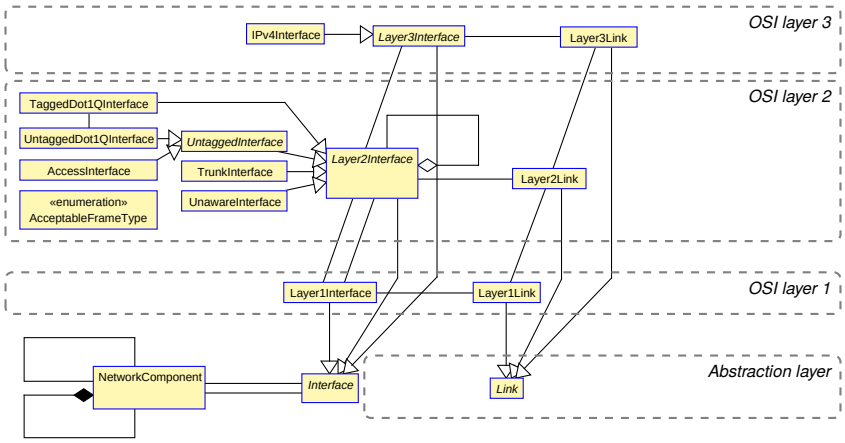

Fig. 2: Class diagram of the network configuration model.

files of all network components currently sum up to 32,500 lines for a proper interface configuration.

3 Network Configuration Model in UML/OCL

3.1 Major Challenges

Network administrators must address different challenges. One general challenge is the bottom-up configuration as networks are not configured as a whole. Each network component is configured individually and network administrators must make sure that the overall configuration is functional and secure so that no unwanted communication is possible. To track down errors, system tools like `ping` or `traceroute` are often used, which can help one roughly locate errors, but success is not guaranteed. In addition, sophisticated knowledge of the network is needed w.r.t the network components and wiring with their configurations consisting of VLANs, LAGs, IP addresses, subnets, IP routes, and firewalls.

Another challenge is the overview of logical topologies that evolve from the physical topology with its given configurations. With the technologies provided by Ethernet (e.g. VLAN and LAG) different network components can have different views on the network topology. Some components might not be reachable due to some VLAN restrictions and physical links might be aggregated to single logical LAGs. It is hard to keep track of these different network views. A documentation of the network is indispensable, yet difficult to create and maintain.

A further challenge are the differences between the theory of network standards and the practice. Companies often use a terminology for their products, which may be different

from standards; sometimes they even behave different than proposed. This makes it hard for administrators to understand the products and the comparison between different manufacturers. As an example, the access and trunk interfaces for the use of VLANs are very common but neither defined nor mentioned in IEEE 802.1Q. The standard allows a very flexible handling of VLANs at interfaces, which may not be useful or secure. This might be the reason why the simplified interface concepts are enforced in practice.

3.2 Network Configuration Model

To address the aforementioned challenges in network administration, a representation of static low-level configuration data of network components was established in UML and OCL. This model can help network administrators to analyze, simulate, document and visualize the details of a computer network. The model itself aligns to the OSI reference model and commonly used Ethernet standards like VLANs and link aggregation, which results in a homogeneous terminology and abstracts proprietary implementations. The advantage of focusing on static configuration data and neglect state data makes the model protocol independent, which leads to a wider applicability of the model. For the usage of the model, it makes no difference which proprietary equipment the network is built with and which implementations of common algorithms are used.

The basic idea of the network configuration model is that computer networks consist of `network components`, `interfaces` and `links`:

- **Network component:** A network component is an abstract representation of a physical or virtual device, which can be integrated into a computer network. It may represent something like a virtual machine, rack-mounted server, switch or router which is capable of dealing with VLANs and link aggregation. Network components do not belong to any layer.
- **Interface:** A network component has interfaces (or ports), which connect them to other network components. Interfaces can be physical or virtual as well and contain a layer-specific network address like a MAC address on layer 2 or an IP address on layer 3. Interfaces can be assigned to different layers on which they operate.
- **Link:** Interfaces of network components are interconnected by physical or virtual links. A link represents a bidirectional connection between exactly two interfaces. Two over a link interconnected interfaces are called *opposites* or *corresponding*.

Figure 2 shows an overview of the class diagram of the network configuration model, which consists of 16 classes, 14 associations, 48 invariants and 21 query operations. Unlike the OSI reference model, this model starts at the bottom with an *abstract layer* represented by the abstract classes `Interface` and `Link`, which combine common concepts from all layers. The specializations of these classes can be assigned specific layers, which are based on the OSI reference model. This results in layer-specific interfaces like the `Layer1Interface` with

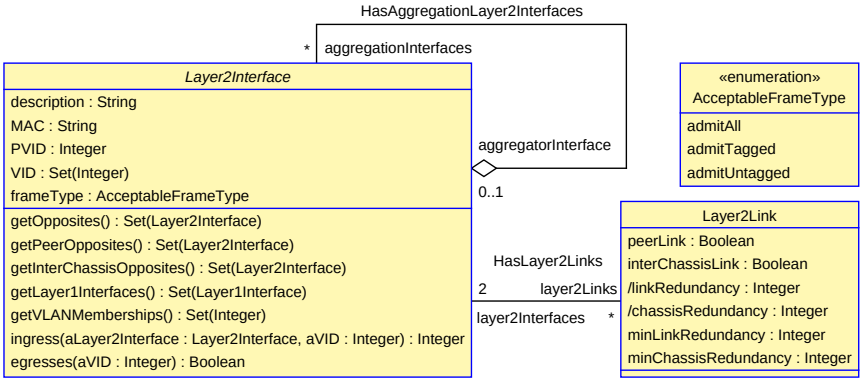

Fig. 3: Abstract class `Layer2Interface` with the class `Layer2Link` of the second layer.

a corresponding `Layer1Link`. The model represents the lower layers of the OSI reference model. In particular the focus is on the first two layers, which are fully implemented. Additionally, a rudimentary implementation of the third layer exists only focusing on the necessary parts for this project.

The first layer of the model correlates to the first layer of the OSI reference model. Due to the strong relationship of the first layer and the physical medium, both can be seen as one in context of this UML model. Because of this simplification, objects of the class `Layer1Link` can be seen as a physical medium like a copper twisted-pair cable. Analogously, objects of `Layer1Interface` correlate to physical cable connectors.

In contrast to the straightforward layer 1, the second layer of the UML model is much more complex and correlates to the data link layer of the OSI reference model (from Fig. 3). The class `Layer2Interface` represents a logical (or virtual) interface capable of dealing with technologies like VLAN and link aggregation so that objects of this class can be seen as Ethernet interfaces. Similarly, objects of the class `Layer2Link` are logical (or virtual) links interconnecting Ethernet interfaces. Due to the virtual characteristic, multiple layer-2 links can be associated with an Ethernet interface depending on the number of VLAN memberships.

Beyond a description and a MAC as a layer-specific network address, the class `Layer2Interface` defines three more attributes: `PVID`, `VID` and `frameType` (see Fig. 3). With the help of these attributes, it is possible to represent a VLAN configuration compatible to the standard IEEE 802.1Q. Different from the standard, the VLAN-IDs are assigned to the interfaces to define the membership and not the other way around (as it is done in practice). To help network administrators with an easier reference to the practice, some specializations of layer-2 interfaces were implemented like `AccessInterface` or `TrunkInterface`. These specializations determine a certain allocation of the aforementioned attributes for an easier mapping. An `AccessInterface`, for example, is member of one VLAN and is able to receive

and send frames without a VLAN-tag. This results in a valid assignment of the `PVID` and an empty `VID` set with the default `frameType` configuration. The other specialization classes can be mapped to a specific attribute setting analogously.

To represent link aggregations as specified in standard IEEE 802.1AX a structural solution was chosen. With the help of the reflexive aggregation `HasAggregationLayer2Interfaces`, it is possible to aggregate layer-2 interfaces. Based on the IEEE standard, the aggregate of the whole-part relationship is called "aggregator interface", whereas the component is called "aggregation interface". An aggregator interface can be associated with multiple network components to represent a multi-chassis LAG (like a DRNI). The presence of a LAG or MC-LAG accommodates strong structural requirements. For example, aggregator interfaces can only be interconnected to other aggregator interfaces to ensure the increased resilience of the logical link. In case of MC-LAGs, "Peer Links" (like an IPL) are required between the associated network components to enable synchronization of status information, which are modeled via attributes on layer-2 links (see Fig. 3).

Two derived attributes were established in the class `Layer2Link` to visualize the resilience of the logical link in the context of link aggregation:

- `/linkRedundancy` computes on how many layer-1 links this logical link depends. Any value greater than 1 represents link redundancy where every but one physical link can fail without affecting the availability of the logical link.
- `/chassisRedundancy` calculates the chassis redundancy, which is relevant when using MC-LAGs, whose interfaces are distributed across multiple network components. Any value greater than 1 represents redundancy so that network components on each side of the logical link can fail without breaking the connection.

The UML model also contains multiple derived associations. They are useful for the inspection of large system states as they help visualizing implicit information. Figure 4 shows a simple system state as an object diagram. This system state consists of two network components named `NC-A` and `NC-B`, which are physically interconnected. The resulting connection is represented by a layer-1 link, which is associated to two layer-1 interfaces. The existence of this layer-1 connection leads to the existence of the derived association called `D_Layer1Connection` in the context of network components. This association connects two network components when they are actually interconnected on the first layer. A modeler can then hide objects of the first layer without losing information.

Furthermore, the system state in Fig. 4 contains two interconnected access interfaces, which are member of the VLAN 1. They are based on the existing layer-1 objects and represent a logical connection between the two network components. The derived association `D_DependsOnLayer1Link` associates objects of class `Layer2Link` to objects of class `Layer1Link` to represent on which physical link the logical link is based on. The existence of

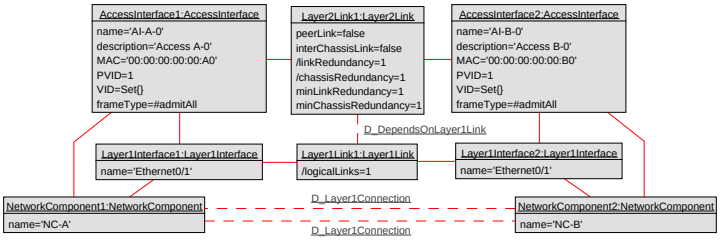

Fig. 4: Valid state: components *NC-A* and *NC-B* interconnected on different layers.

a derived layer-1 link is mandatory and can be inferred by the associations of the neighbored objects.

4 Analysis of the Housing Center

4.1 Basic Analysis Process

The first step of the housing center analysis is to have a valid representation as a USE system state. Due to the infrastructure complexity, the focus lies on the described core network components with two servers modeled as an example. Nevertheless, the complexity of the pursued system state is high resulting from the complex configuration of individual network components.

To avoid to go through the network configurations and manually construct the USE system state, a Java parser was implemented for an automatic transformation. Usually the Cisco network components are configured using specific configuration commands via an SSH interface. The overall configuration can be displayed and exported using particular commands. Besides this configuration, the parser expects an export of the Cisco Discovery Protocol (CDP) information for each network component. The CDP is a proprietary implementation of the Link Layer Discovery Protocol (LLDP), which helps to identify neighbored network components. This information helps to infer the physical cabling among the considered network components and create the layer-specific interconnections.

Fig. 5: Processing the configuration files and input for USE.

An overview of the parsing process is shown in Fig. 5. In our case, the parser processes 14 files of configuration commands and CDP information. In total, they consist of about 32,500 lines. The parsing process itself only takes less than 1 second and outputs a so-called "SOIL file" containing OCL-like commands to build a USE system state. The SOIL file contains about 6,500 lines of statements for object and link creation and setting attribute

values. With this SOIL file and the class diagram, USE establishes the system state of the network.

Figure 6 shows the object and link count of the fully loaded system state of the housing center in USE. The object diagram consists of 13 network components, 712 (14+288+68+268+50+2+22) interfaces with 180 `Link` objects of different layers in addition to 1,760 regular and 857 derived links. About one third of the total links are derived showing their importance as these links are implicitly derived from various constraints. At this point, the system state satisfies only 45 of the 48 invariants. A detailed analysis of the three failed invariants is discussed in the next section.

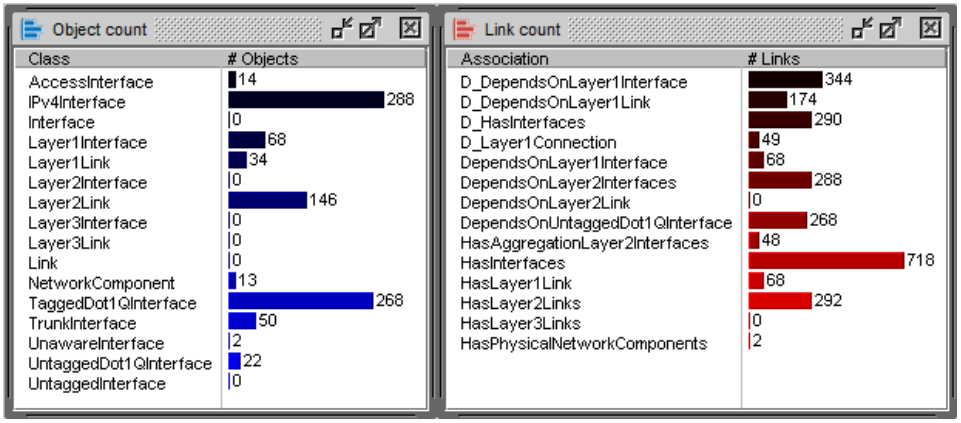

Fig. 6: Object and link count of the housing center system state in USE.

A network administrator now has many options to use the system state for her needs. It is possible to visualize the system state in USE to obtain a rough overview of the individual objects. Due to the quantity of objects and links, it is not advisable to display everything at once. USE provides features to show specific objects by name, type or OCL query and discover their neighbored objects by path length. Using this technique, a network administrator can display, for example, only interfaces and links of a certain layer providing a layer-specific investigation of the system state. The embedded derived associations help to keep track of the overall context, even if not every object is visible.

A network administrator can also query information with OCL. She can, for example, query specific data from a network component to obtain a tabular configuration overview. Even Cisco-specific analysis commands, like `show vlan brief` or `show interface trunk`, can be rebuilt in OCL. Using its condensed output, she can compare and verify the system state with the real world network component or the other way around. More powerful OCL queries are possible, which include configuration data from different network components, e.g., showing physical interfaces not posessing a link, every LAG or MC-LAG defined in the network, or virtual links not meeting the required resilience parameters (indicating broken physical links).

In practice, an administrator would have to compare the distributed configurations manually. Additionally she needs specific knowledge about the infrastructure itself combined with a sophisticated understanding of the influence of different configurations on the overall functionality of the network.

4.2 Incidents Found

The built USE system state does not satisfy every defined invariant. Nevertheless, the UML multiplicities together with the structure checking invariants are fulfilled, which states that the structure is valid and that the Cisco parser successfully parsed the configuration files. The three failing, configuration checking invariants can be examined further using the `check` command on the USE shell.

One of these invariants handles the uniqueness property of MAC addresses, whose transformation is not yet implemented in the parser. The second invariant detected orphaned "layer-2 subinterfaces", which do not have a corresponding interface. This could lead to miscommunication between network components, because the sent frames will not be processed accordingly. The third invariant will be the focus of further analysis, exemplifying the analysis process. A network administrator can get detailed information about the evaluation using the command `check -d`. The additional parameter leads to a detailed output, in which one can see which objects are violating the invariant:

```
use> check -d
     checking invariant 8: FAILED.
     Instances of Layer2Interface violating the invariant:
       Set{TrunkInterface23, TrunkInterface26}
     checked 48 invariants in 3.059s, 3 failures.
```

The focus is now on the failed `invariant 8`. This invariant checks if two corresponding interfaces are able to receive frames of certain VLANs the respective other interface is member of. The output of the command shows that the two objects `TrunkInterface23`, `TrunkInterface26` are violating the invariant. Object `TrunkInterface23` will be further used to demonstrate the options of the UML model in combination with USE.

Using a query, first the corresponding opposite interface of `TrunkInterface23` is found. For this specific case the query operation `getOpposited()` is implemented, which returns all corresponding interfaces. In this case, there is only one corresponding interface `TrunkInterface4`:

```
use> ?TrunkInterface23.getOpposites() -> Set{TrunkInterface4}
```

Next, we compare the VLAN memberships of the determined interfaces, as this was the cause of the invariant violation. To retrieve the VLAN memberships of an interface, the query operation `getVLANMemberships()` is used. Querying the difference of the VLAN memberships of the two interfaces shows the range of VLANs that are defined on one

interface and not on the other. Here, the object `TrunkInterface23` is member of ten more VLANs (1200–1209) than the object `TrunkInterface4`, which causes the invariant to fail.

```
use> ?(TrunkInterface23.getVLANMemberships() -
        TrunkInterface4.getVLANMemberships())->asSet()
     Set{1200,1201,1202,1203,1204,1205,1206,1207,1208,1209}
```

Now that the source of the violation is known, the network context can be explored further in order to identify the components with faulty configurations. Using a simple OCL query returning a tuple, the interface names and names of the associated network components are detected:

```
use> ?Set{TrunkInterface23,TrunkInterface4}->collect(i |
        Tuple{NM=i.name, NC=i.networkComponents.id().name})
     Bag{Tuple{NM='port-channel10',
               NC=Bag{'nexus-5500-a-1','nexus-5500-a-2'}},
         Tuple{NM='port-channel10',
               NC=Bag{'nexus-7000-a','nexus-7000-b'}}}
```

The output shows that both examined interfaces are named *port-channel10*. It also states that both interfaces are part of an MC-LAG, as they both are associated to two network components. Using the hide-and-show feature of USE, it is possible to visualize only the small examined context of the network as shown in Fig. 7. This confirms that the two interfaces are part of an MC-LAG. Overall, the analysis shows that the invariant violation was caused by an interface that interconnects the distribution layer with the access layer over an MC-LAG.

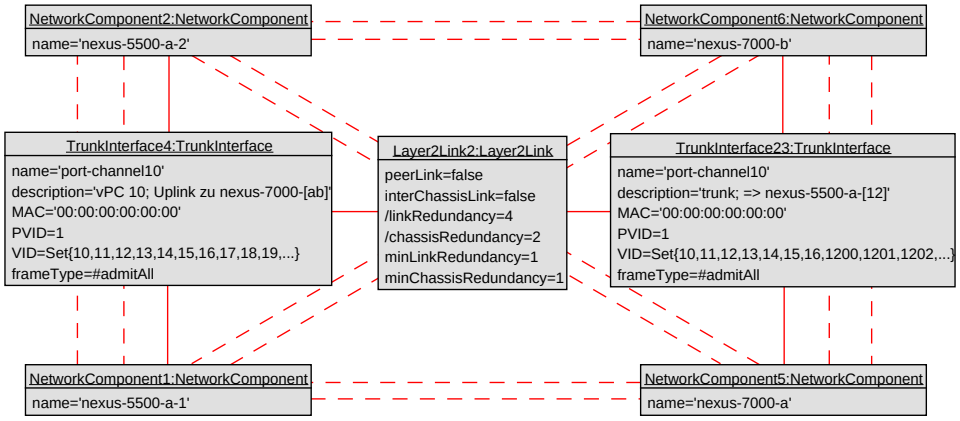

Fig. 7: Context of the found configuration error in the network components.

The impact of the discovered configuration error is a limited reachability of the network components. The `TrunkInterface23` is able to send frames belonging to VLAN 1200 to its connected interface `TrunkInterface4`. As the receiving interface of the frames is not a member of VLAN 1200, it will refuse and drop frames, resulting in a one-directional

miscommunication for VLANs 1200..1209. It is recommended to match the VLAN memberships of linked interfaces.

In practice this kind of error is hard to detect. The error will only become noticed when a certain reachability from one network component to another is not available. The task of error tracing can be very complex here due to the special conditions of the miscommunication, as the reachability for one VLAN might be present but fails for another. The search for misconfiguration can be simplified using our network configuration model. The defined invariants allow one to check the basic configuration and show the objects violating these constraints. With the use of a few query commands an administrator can extract and visualize the errors. An administrator can also create own queries for a quick inspection of the resilience attributes of certain links or any other data of interest.

4.3 Results and Discussion

When reporting the detected incidents to the administrator, the feedback was very positive. Ignoring the MAC addresses, in case of the first incident, it turned out that during the time of the acquisition of the configuration, an update has been pushed to some network components while still being scheduled for others. This led to a known inconsistency in the configuration that was later corrected. Nonetheless, this incident represents a configuration error in the extracted data and was identified as such by the model. As for the second incident, the misconfigured VLAN ranges have indeed not been recognized before and can be corrected with the detailed information extracted during the analysis.

Overall, the results show the suitability of the model to find errors of various kinds in the configuration of networks that would otherwise be hard to find in the distributed files containing configurations per network component. Additionally, the options to query the network configuration allows to locate causes of incidents directly. Furthermore, the model has demonstrated its suitability for industrial networks: in terms of the technologies that are used in such environments, which include proprietary equipment, and also in terms of performance.

5 Related Work

The “Interconnected Asset Ontology” (IO) [Bi12, BS14b, BS14a] resembles our work in its underlying goals. IO follows an ontology-based approach to the representation and inference of computer-network layers. It uses SPARQL as a query language to obtain information from computer networks. In particular, IO can represent VLANs, LAGs, and MC-LAGs as well. The topological model of our work can, hence, be seen as an UML/OCL-based alternative to IO, however, with a focus on the lower OSI-layers. An advantage of using UML is the fact that we can visualize network system states and exploit the USE model validator’s capability to complete system states.

Other works deal with the static analysis of computer networks. Xie et al. present an approach to determining reachability in IP-based networks [Xi05]. Their model is based on graphs and determines reachability between two network components by tracing all possible "headers" from a source to a target. Kazemian et al. refine this approach by also providing a technique for the detection of cycles and slices in network topologies [KVM12]. For this purpose, they use static configurations of network components, which are represented in a geometric model. This model also focuses on IP-based networks, but headers do not refer to a specific OSI layer. A similar approach is proposed by Al-Shaer et al., who model computer networks as finite state machines and follow a model-checking approach to analyze the model w.r.t. reachability and consistency [Al09].

Our approach resembles the aforementioned works in several ways. All techniques deal with the representation and analysis of static data of a computer network, however, with a focus on the IP layer of a computer network (including firewalls). In contrast, our work allows an administrator to consider the lower OSI-layers, which are not topic of the other works. In particular, we now support concepts such as VLAN and link aggregation while at the same time adopting ideas from the analysis of IP-based networks, such as reachability, cycle and slice analysis.

Our work has also connections to approaches for network configuration and administration based on formal semantics or formal tools and on proving-like techniques. A general formal configuration language is discussed in [AH16], and the application to real scenarios is demonstrated. In [Ma15] a tool based on SAT solving is used for network and access control list configuration. The configurations are checked through various standard and complex service access queries. [FM10] introduce a formal approach allowing to formally and optimally configure a network so that a given security policy is respected and by taking into account the quality of services. Special emphasis is put on the consideration of firewalls. The approach in [Ro06] presents a logic-based model that is suitable for describing networks and intrusions. The approach is implemented in Prolog and allows to analyze important static properties of networks. The work in [Fo06] proposes an abstract model for network configuration that is intended to help to understand fundamental underlying issues in self-configuration. When individual network components that are securely configured are connected together in an apparently secure manner, configuration cascades resulting in a misconfigured network are detected. Within the area of mobile-ad-hoc-networks (MANETS) the proposal from [So05] describes a formal approach to modeling and reasoning about auto-configuration protocols to support the detection of malicious nodes. Global security requirement for a network are defined that characterize the good behavior of individual nodes to assure a global property.

Work on configuring networks has also been done in prover-like contexts. [Ha15] focuses on the behavior of individual switches, and demonstrates that even simple configuration rule updates can result in inconsistent packet switching. The paper demonstrates that consistent configuration updates require guarantees of strong switch-level atomicity from both hardware and software layers of switches. [DKC15] presents the tool topoS which

automatically synthesizes low-level network configurations from high-level security goals. The tool topoS is formally verified with Isabelle/HOL, which prevents implementation errors. None of the above approaches employs standardized modeling languages like UML and OCL to describe networks.

6 Lessons Learned and Conclusion

We have performed a full analysis of the static configurations on a real-life network and were able to show the effectiveness of our approach with the example of a professional network. In particular, with the model and the USE tool, two incidents in the extracted configuration snapshot were uncovered. The process was fully automatic in that the existing configuration files were processed by a parser to create the system state of the model on which USE evaluates the model invariants and gives detailed information about the results. In case of a failed invariant, the cause is reported and can be used as an entry point for further analysis steps. The analysis of errors works on the model level, which gives network administrators an abstract view and control over the state of the network, e.g. connections of network components can be interactively followed using navigation in OCL. For this purpose, the model contains several query operations that perform commonly used tasks, such as following virtual links.

Our approach models network configurations from the bottom, starting with network components and physical cables from OSI layer 1. Third-party tools offered by network companies only allow one to inspect network configurations from the third layer or above. Therefore, configuration errors in lower layers cannot be detected using these tools and less convenient methods have to be used to analyze the lower layers, e.g. the low-level options `ping` and `traceroute`.

Aside from static analysis, representing the network as a UML system state allows for modifications that directly affect the whole network. These modifications can be analyzed before they are implemented whether they are free of errors and have the desired effect on the model, e.g. regarding reachability.

OCL and its formal basis allow for verbally formulated requirements of a network to be precisely represented using the network configuration model. Thus custom requirements can be automatically checked on a system state or be implemented as invariants, to be used in the future. These requirements can also be used to create a new network. The generation of the links and configurations can then be generated by a model completion tool like USE. Additional parameters can be defined using higher layers of the model.

The documentation options of the network configuration model are useful at all development stages. Before installing a network, the network requirements can be checked and during network maintenance, changes can be simulated. With the parser for the Cisco configurations, the system state can be generated from existing networks at any time reducing initial costs.

More parsers can be implemented to support more technologies and brands. These parsers also allow for continuous integration during maintenance and for continuous documentation.

Being able to query the model, visualize the system state and test setups before they are physically installed are only some of the general features for UML/OCL models. Also, besides the USE tool, a multitude of tools exists to analyze yet more properties of general UML/OCL models, like the configuration model. Certainly, it also would have been possible to use a DBMS and define integrity rules, but employing USE allows us to automatically generate a series of test cases rather than defining only single cases.

In conclusion, we believe that the presented methodology helps network administrators. Specifically, administration of large-scale computer networks often becomes tedious and error-prone, but the abstractions and gained overview enable a better understanding of networks—existing and new ones—and validation detect faults in the configurations. Finally, we do not claim that the network configuration model presented in this paper is fully exhaustive. Further investigations can lead to more improvements, but the experience so far supports the claim that it is worthwhile to invest into model-based automatic validation techniques for networks and that these techniques pose a benefit for all networks.

Bibliography

[AH16] Anderson, P.; Herry, H.: A Formal Semantics for the SmartFrog Configuration Language. J. Network Syst. Manage., 24(2):309–345, 2016.

[Al09] Al-Shaer, E.; Marrero, W.; El-Atawy, A.; Elbadawi, K.: Network Configuration in A Box: Towards End-to-End Verification of Network Reachability and Security. In: ICNP. IEEE Computer Society, pp. 123–132, 2009.

[Bi12] Birkholz, H.; Sieverdingbeck, I.; Sohr, K.; Bormann, Carsten: IO: An Interconnected Asset Ontology in Support of Risk Management Processes. In: Seventh International Conference on Availability, Reliability and Security, ARES 2012. IEEE Computer Society, pp. 534–541, 2012.

[BS14a] Birkholz, H.; Sieverdingbeck, I.: Improving root cause failure analysis in virtual networks via the interconnected-asset ontology. In: Proceedings of the Conference on Principles, Systems and Applications of IP Telecommunications, IPTComm 2014, Chicago, Illinois, USA, October 1-2, 2014. ACM, pp. 2:1–2:8, 2014.

[BS14b] Birkholz, H.; Sieverdingbeck, I.: Supporting Security Automation for Multi-chassis Link Aggregation Groups via the Interconnected-Asset Ontology. In: Ninth International Conference on Availability, Reliability and Security, ARES 2014. IEEE Computer Society, pp. 126–133, 2014.

[DKC15] Diekmann, C.; Korsten, A.; Carle, G.: Demonstrating topoS: Theorem-prover-based synthesis of secure network configurations. In (Tortonesi, M.; Schönwälder, J.; Madeira, E. R. Mauro; Schmitt, C.; Serrat, J., eds): 11th International Conference on Network and Service Management, CNSM 2015, Barcelona, Spain, November 9-13, 2015. IEEE Computer Society, pp. 366–371, 2015.

[FM10] Fall, M.N.; Mejri, M.: Network Security: Formal and Optimized Configuration. In (Fujita, H., ed.): New Trends in Software Methodologies, Tools and Techniques - Proceedings of the 9th SoMeT_10, September 29 - October 1, 2010, Yokohama City, Japan. volume 217 of Frontiers in Artificial Intelligence and Applications. IOS Press, pp. 229–246, 2010.

[Fo06] Foley, S.N.; Fitzgerald, W.M.; Bistarelli, S.; O'Sullivan, B.; Foghlú, M.: Principles of Secure Network Configuration: Towards a Formal Basis for Self-configuration. In: Autonomic Principles of IP Operations and Management, 6th IEEE International Workshop on IP Operations and Management, IPOM 2006, Dublin, Ireland, Proceedings. pp. 168–180, 2006.

[GBR07] Gogolla, M.; Büttner, F.; Richters, M.: USE: A UML-based specification environment for validating UML and OCL. Sci. Comput. Program., 69(1-3):27–34, 2007.

[GHD17] Gogolla, M.; Hilken, F.; Doan, K.-H.: Achieving Model Quality through Model Validation, Verification and Exploration. Journal on Computer Languages, Systems and Structures, Elsevier, NL, 2017. Online 2017-12-02.

[Ha15] Han, J.; Mundkur, P.; Rotsos, C.; Antichi, G.; Dave, N.H.; Moore, A. W.; Neumann, P. G.: Blueswitch: Enabling Provably Consistent Configuration of Network Switches. In (Brebner, G.J.; Bachmutsky, A.; Das, C., eds): Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for networking and communications systems, ANCS 2015, Oakland, CA, USA, May 7-8, 2015. IEEE Computer Society, pp. 17–27, 2015.

[IE14a] IEEE Computer Society: . IEEE Standard for Local and Metropolitan Area Networks: Bridges and Bridged Networks, 11 2014.

[IE14b] IEEE Computer Society: . IEEE Standard for Local and Metropolitan Area Networks: Link Aggregation, 12 2014.

[IT94] ITU-T – International Telecommunication Union: . X.200: Information technology – Open Systems Interconnection – Basic Reference Model: The basic model, 7 1994.

[KVM12] Kazemian, P.; Varghese, G.; McKeown, N.: Header Space Analysis: Static Checking for Networks. In: NSDI. USENIX Association, pp. 113–126, 2012.

[Ma15] Maity, S.; Bera, P.; Ghosh, S. K.; Al-Shaer, E.: Formal integrated network security analysis tool: formal query-based network security configuration analysis. IET Networks, 4(2):137–147, 2015.

[Ro06] Rolando, M.; Rossi, M.; Sanarico, N.; Mandrioli, D.: A formal approach to sensor placement and configuration in a network intrusion detection system. In (Bruschi, D.; Win, B. De; Monga, M., eds): Proceedings of the 2006 international workshop on Software engineering for secure systems, SESS 2006, Shanghai, China, May 20-21, 2006. ACM, pp. 65–71, 2006.

[So05] Song, T.; Ko, C.; Tseng, C. H.; Balasubramanyam, P.; Chaudhary, A.; Levitt, K. N.: Formal Reasoning About a Specification-Based Intrusion Detection for Dynamic Auto-configuration Protocols in Ad Hoc Networks. In (Dimitrakos, T.; Martinelli, F.; Ryan, P. Y. A.; Schneider, S. A., eds): Formal Aspects in Security and Trust, Third International Workshop, FAST 2005, Newcastle upon Tyne, UK, July 18-19, 2005, Revised Selected Papers. volume 3866 of LNCS. Springer, pp. 16–33, 2005.

[TW11] Tanenbaum, A.-S.; Wetherall, D.: Computer Networks. Pearson Prentice Hall, 5 edition, 2011.

[Xi05] Xie, G.G.; Zhan, J.; Maltz, D.A.; Zhang, H.; Greenberg, A.G.; Hjálmtýsson, G.; Rexford, J.: On static reachability analysis of IP networks. In: INFOCOM. IEEE, pp. 2170–2183, 2005.