Integrating UML/OCL Derived Properties into Validation and Verification Processes

Frank Hilken[1], Marcel Schuster[1], Karsten Sohr[1,2], and Martin Gogolla[1]

[1] University of Bremen, Computer Science Department, 28359 Bremen, Germany
[2] Center for Computing Technologies (TZI), 28359 Bremen, Germany
{fhilken,maschu,sohr,gogolla}@informatik.uni-bremen.de

Abstract. UML and OCL are rich languages offering a multitude of modeling elements. They provide modelers with simple and effective tools to express and visualize models, but their complex semantics are a great challenge for validation and verification processes, which are often limited by their underlying logic. On the basis of a network topology example model, describing lower physical and logical network levels, we demonstrate applications of derived attributes and associations that help checking network security aspects. This includes finding inconsistencies in an existing network, directing to potential configuration errors and property evaluation, e.g. reachability between components within the network. In addition, a transformation of derived properties into relational logic is presented to enable the use of a modern instance finder for further verification tasks and generation of valid networks configurations.

Keywords: Validation and Verification · Derived Property · Network Security · Tooling

1 Introduction

The management of modern network infrastructures has become a complex task in which an increasing number of requirements have to be considered. Besides availability concerns, also the important network security is a permanent requirement in increasingly growing corporate intranets. Models can help to keep track of the network and validate the consistency, but they mostly consider higher abstraction levels. In the OSI model (Open Systems Interconnection, [11]), a standard for communication in computer networks based on seven layers, the aforementioned models reside on the third layer as the lowest layer.

We present a UML/OCL network topology model based on OSI starting with especially the two lower layers. The model allows the representation of networks on the OSI layers 1 and 2. With the model, existing network configurations can be validated for consistency. Besides invariants ensuring consistency, the model is enhanced with derived properties, to abstract from complex network relations and further ensuring valid system states. We instantiate the model with the configuration of the data center of the University of Bremen and check it for consistency as well as for other properties. Furthermore, instances of the model are useful network documentations and can be used to interactively maintain the configuration with immediate feedback of the model's consistency.

Using a general UML/OCL validation and verification tool, the USE tool (UML-based Specification Environment, [8]) and its instance generator based on relational logic [13,12], additional verification use cases are demonstrated. In particular, the completion of partial system states, i.e. taking a partial (inconsistent) system state and complete it into a valid, consistent state, is used to allow network administrators to specify key data of their networks as well as security requirements and have the rest of the details generated to form a valid network that employs all requirements. The model also allows to check whether certain requirements, e.g. reachability between components, can be realized at all given a partial model as the basis.

In order to employ the derived properties with our instance generator, the USE model validator, a transformation of the semantics of derived properties into relational logic is presented. While derived properties can already be checked, using model transformations into simpler UML/OCL descriptions [10], the direct transformation in the tool causes less manual work and is more efficient, especially in regards to the search space. The transformation allows well-known verification use cases to be applied on models with derived properties.

The paper is structured as follows. Section 2 briefly introduces preliminary concepts used in the paper. Section 3 describes a network topology model able to analyze the lower connection layers of networks. Then, Section 4 demonstrates use cases how the derived properties help validating a real-life network and how the model can be further used to generate secure networks. Section 5 explains the transformation of derived properties into relational logic for usage in tools. Section 6 discusses further ideas regarding derived properties and tooling that were collected during the work. Finally, Section 7 concludes and outlines future work.

2 Preliminaries

2.1 Network Security

One task of network security administrators is to prevent the unhindered spreading of malware in corporate intranets, which use insecure connections to attack security-related network components like servers. This threat is much higher when an attacker comes from the inside or has already injected malware into the corporate intranet using known security vulnerabilities.

To obtain a better understanding of the computer network of the corporate intranet, it is important for network administrators to carry out an infrastructure analysis. Additionally, it should be possible to collect this information about the network infrastructure automatically. In particular, this is relevant for protection from inner threats. These attacks are often performed on the lower network layers of the OSI model (Open Systems Interconnection, [11]) (layer 1 and layer 2) as an attacker has physical access to the network in this case.

The importance of performing an infrastructure analysis in addition to a physical and logical network topology documentation is also stressed in the German “IT Baseline Protection”, a German standard for IT security management in organizations [4]. However, the task of generating a visual representation of the physical and logical topology is often difficult to fulfill. Depending on the current configuration of the network components, the logical topology might differ significantly from the physical topology.

Responsible for these difficulties are technologies from network standards such as IEEE 802.1AX (Link Aggregation) and IEEE 802.1Q (Virtual Bridged Local Area Networks). They allow network administrators to bundle physical links into one big logical link resulting in advantages regarding bandwidth, redundancy and partitioning the physical topology into multiple logical segments. Despite being widely used in common corporate intranets and corporate data centers, there is no known tool support for visualizing these kind of low-level functions in network topologies. Such a tool, however, would greatly support the process of security management in computer networks.

Our experience that we gained while carrying out our research project has shown that the network configuration and documentation are often generated and maintained manually. This leaves a lot of space for mistakes. A promising approach to improve on this situation is to create a UML/OCL network model enabling the analysis with validation and verification tools such as USE and the USE model validator.

There are also other works that model computer networks in a formalism and allow an administrator to query information, e.g. reachability queries. For example, the Interconnected Asset Ontology (IO) models computer networks with the help of ontologies and uses SPARQL for querying the network description [3,2]. The UML/OCL network topology model presented in this paper is designed as an alternative to the ontology, to discover new possibilities with UML/OCL tools. IO’s feature pool is larger due to the longer research time among other reasons, but the new model presents advantages like the visually oriented tooling and the generation of networks using instance finders. In another work, a Prolog-based prototype for representing and reasoning about network descriptions is presented [5]. PRESTO allows configuration management for large-scale networks by providing generalized configlets by which concrete configurations for network devices can be created [7].

2.2 Derived Properties in UML/OCL

Derived properties in UML and OCL allow the modeler to represent conditions that are implicitly in the model and can be derived using existing information in system states [9]. These properties are defined on classes from the model, either as attributes or role ends, and do not need to be instantiated but rather are calculated using given OCL expressions. These so called derived expressions are used to describe attribute values and relations (associations) between classes in the form of links in a system state. Finally, the number of generated links in a system state must conform to the defined multiplicities of their respective derived association, allowing to put additional constraints on the model.

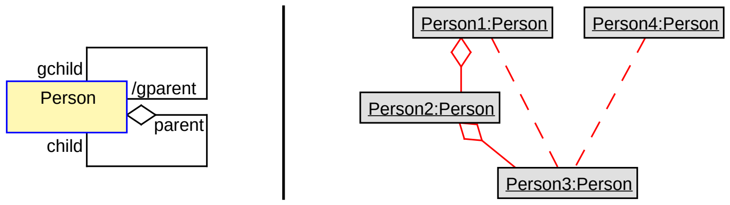


Fig. 1. Class diagram and partial object diagram of simple family tree model.

Definition 1 (Derived Expression). *Derived expressions are functions with the signature:*

$$derived : p_1 \times \ldots \times p_n \to T.$$

The inputs p_1 to p_n are scope parameters that can be used in the OCL expression. The number of parameters is determined by the model element. Derived attributes have exactly one parameter named `self` *that has the type of their respective class. Derived n-ary associations (including binary associations, $n = 2$) have exactly one derived role and $n - 1$ parameters, one for each non-derived role end to provide proper scope. The type of the output T is also determined by the model elements. For derived attributes it is the type of the attribute and for derived associations it is the type of the associated class of the derived role end, or a set thereof if the multiplicity is > 1.*

Example 1. A simple family tree model might just consist of the class `Person` and a `parent-child` association (see Fig. 1 on the left). These model elements are enough to specify a family tree, but they also contain a lot more information than is stated. For example, one could add a derived association, which displays the grandparents of each person. The definition of such association could look as follows:

```
association Grandparents between
  Person [*] role gparent  derived = self.parent.parent→asSet()
  Person [*] role gchild
end
```

The derived role `gparent` can be identified in the class diagram by the forward slash in front of the rolename. Figure 1 on the right shows a partial object diagram with four persons – parents at the top, children towards the bottom. The aggregations show `parent-child` links and the dashed lines represent derived grandparent relations. The persons 1, 2 and 3 are directly chained making `Person1` a grandparent of `Person3`. The object diagram also states that `Person4` is a grandparent of `Person3`, however there is no connection visible supporting this structure. This is due to the fact that a fifth person – not shown in the partial diagram – is a child of `Person4` and a parent of `Person3`. Even though the required objects for the derived link are hidden, the link itself is still visible. Herein lies a strength of the derived properties, being able to hide complex structures without losing information. For the visual representation, this greatly helps to comprehend models.

2.3 USE Tool and Model Validator Plugin

The USE tool (UML-based Specification Environment, [8]) is a modeling tool capable of validating and verifying UML models enhanced with OCL. USE supports several diagram types (class diagram, object diagram, sequence diagram, state machine, communication diagram) with a well-defined subset of UML and a near complete support of OCL. USE allows the interactive instantiation of system states while it is continuously checked for validity. Any violation of model invariants or multiplicities (including the ones from derived associations) is reported and can be analyzed in detail to find the inconsistent parts using several in-built (graphical) techniques allowing detection and correction of errors.

The USE model validator plugin is an instance finder for UML/OCL models that generates valid system states within given bounds. This bound configuration determines the number of objects and links as well as the domains of all types used by the model validator. These bounds, together with the UML structure and OCL constraints, are transformed into the relational logic of Kodkod [12,13] to be further transformed into a SAT problem and solved.[3] The solution instance is transformed back into a UML system state as an object diagram. In particular the capabilities of taking a partial, possibly invalid system state given by the modeler and completing it to a valid system state conforming the UML/OCL model will be important later.

3 Network Topology Model

To have a proper representation of network topologies, the following network topology model is oriented towards the OSI (Open Systems Interconnection) network model [11], which describes communication in computer networks in seven layers. To fill the gap of models for lower network layers, for now the network topology model only considers the OSI layers 1 and 2. The OSI layers 3 to 7 can be added later, but are not further discussed. The layers 1 and 2 of the OSI network model are defined as follows:

1. Layer 1 is the physical layer which specifies the transportation medium with its connectors and electrical resources. Furthermore it specifies how the bits are transferred to the receiver. For our example, this layer describes the physical cabling in between network components (e.g. routers and servers).
2. Layer 2 is the data link layer which separates the bit stream of layer 1 into several logical frames and adds data like checksums. Ethernet is the most popular example for a specification defining layer 2 of the OSI model.

The network topology model (see Fig. 2) consists of three base classes: `NetworkComponent`, `Interface` and `Link`. The idea is that a computer network consists of network components having interfaces. The interfaces are in turn connected to other interfaces using links. These classes are part of the abstraction layer on which the OSI layers build upon. Modeling the association between the

[3] A list of all supported UML/OCL features in the tool is compiled in Appendix A.

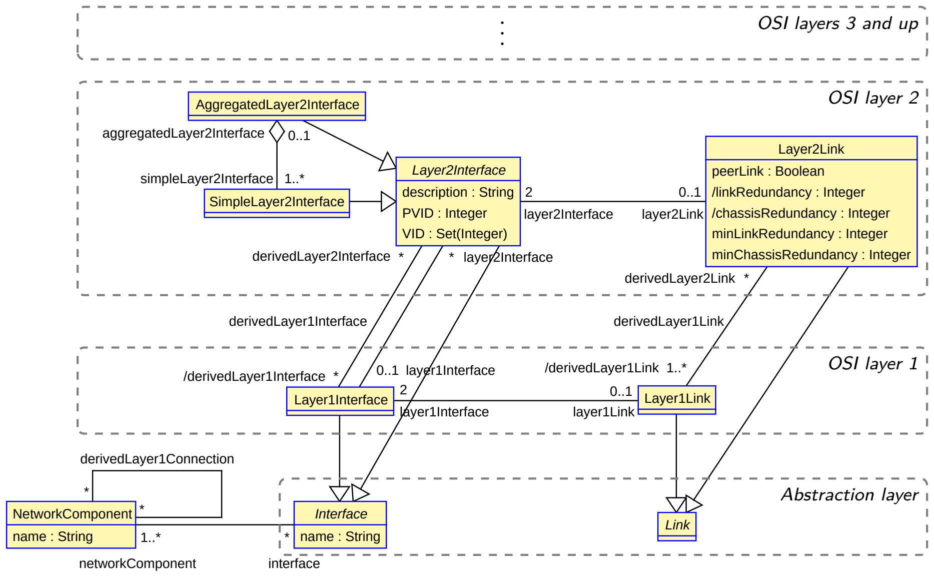


Fig. 2. UML class diagram representing a layered OSI-like network topology model.

classes `Interface` and `Link` with unions and subsets would be a nice approach in this model, but unfortunately this UML feature is not yet supported by the USE model validator (see Appendix A).

The specialization classes `Layer1Interface` and `Layer2Interface` represent the OSI-like interface layering. It is important to state here that the specialized interfaces can only be linked on the same layer. Thus a `Layer1Interface` can only have a `Layer1Link` while `Layer2Interface` can only have a `Layer2Link`. The multiplicity of "`0..1`" between Interfaces and Links results from the fact that interfaces don't require links to exist. The model contains further elements to handle the standards IEEE 802.1AX and IEEE 802.1Q for managing link aggregations and VLANs. Each layer has dependencies to the one below it and an extension of the model to support OSI layer 3 and above is indicated but not further pursued here.

The model contains two derived attributes in the class `Layer2Link`. The values of these attributes cannot be set manually but are calculated automatically using an OCL expression. The attributes represent the following information:

linkRedundancy This attribute computes on how many layer 1 links this layer 2 link depends. This is especially relevant when using link aggregation, where multiple physical connections are aggregated to one logical link. Any value greater than one represents redundancy where one physical link can fail while not affecting the availability of the layer 2 connection.

chassisRedundancy This attribute calculates the chassis redundancy of the given layer 2 link which may indicate single points of failure in the network topology. Due to the existence of multi-chassis systems, it is possible to create aggregated multi-chassis layer 2 interfaces as well which are distributed across the multi-chassis network components [2]. Any value greater than one represents redundancy where one network component can fail without affecting the availability of the layer 2 connection.

Using the attributes `minLinkRedundancy` and `minChassisRedundancy` of the class `Layer2Link` it is possible to define redundancy requirements which are useful for the completion of partial system states using the USE model validator later in Sect. 4.2. In addition to the aforementioned derived attributes, this model contains three derived associations:

derivedLayer1Connection This derived association links two network components, when they have at least one `Layer1Link` interconnecting them. This information is very useful when viewing layers in isolation, e.g. the information is preserved even when all layer 1 objects (`Layer1Interface`, `Layer1Link`) are hidden in an object diagram.

derivedLayer1Interface Layer 2 interfaces can either be directly or indirectly associated with layer 1 interfaces. An `AggregatedLayer2Interface` is associated to a `SimpleLayer2Interface` which is in turn directly associated with a `Layer1Interface`. As a result, the aggregated layer 2 interface is associated with the layer 1 interface using transitivity characteristics.

derivedLayer1Link Using this derived association, it is possible to show which logical layer 2 links are transmitted over an existing layer 1 link. This derived association results from the association of the layer 1 and layer 2 interfaces. The existence of a derived layer 1 link is mandatory and is ensured using an adequate multiplicity (`1..*`). This ensures valid layer 2 linking via a constraint enforced by a derived association and a violation of the multiplicity indicates invalid network configurations.

In this network topology model it would be possible to define far more derived associations, but this is not necessarily reasonable. The described derived attributes and associations are useful for the inspection of large system states, because of the calculation and visualization of implicit existing information that is otherwise not visible as easily. Additionally, the modeler can hide big parts of the data without losing important information (see Sect. 4.1).

To further ensure valid system states, the network topology model contains 20 class invariants which check advanced structural dependencies, e.g. associated layer 1 and 2 interfaces linking to the same network component. Furthermore, there exist class invariants that verify the integrity of the model which includes, for example, the check for valid VLAN configurations of linked layer 2 interfaces. Another example is stated in the following state invariant:

```
inv AssociatedLayer1InterfacesAreProperlyConnected:
  self.getOpposite() <> null implies
  self.getOpposite().getLayer1Interfaces()
      →includesAll(self.getLayer1Interfaces().getOpposite())
```

The operation `getOpposite()` is defined for every interface and returns the corresponding interface connected across an instance of the class `Link`. The presented state invariant is defined in the context of `AggregatedLayer2Interface` and ensures that the depending layer 1 interfaces of the linked aggregated layer 2 interfaces are properly connected. This prevents, for example, inconsistent cabling and therefore not fully working and redundant link aggregation.

4 Example Use Cases for Derived Properties

The following two sections present two possible and realized use cases for the described network topology model. The model can either be used to validate and visualize given configurations of network components or create and complete given partial system states towards an automated generation of configurations.

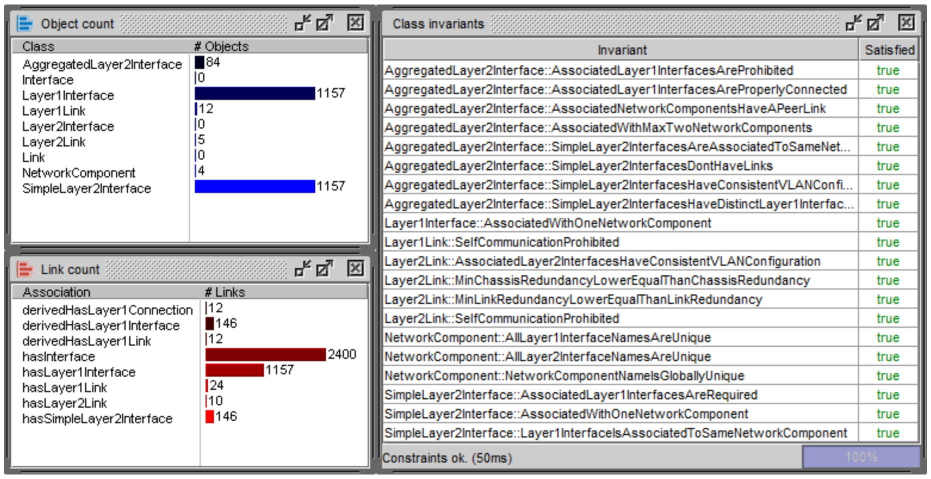
4.1 Validation of Existing Network Configurations

The network topology model described in Section 3 can be used to validate and visualize the configuration of network components. As a real-life example we chose four Cisco network switches which are placed in the data center of the University of Bremen. These switches belong to the core network meaning they are among the most central components of the network topology [6].

These core network components have high requirements regarding bandwidth and redundancy, because the failure of one component or link must not affect the availability of the whole data center [6]. In addition to the redundancy configuration the components contain a lot of configurations regarding the logical network design using VLANs. This is the reason why these core components have a lot of physical and logical interfaces which are hard to maintain manually. The configuration of each component consists of more than 3.000 Cisco-specific configuration commands provided in plain text files.

Using parts of the provided configuration files of the data center, it is possible to extract relevant data out of the static configuration information and build a system state from it. The emerging object diagram consists of 4 network components, 2.398 interfaces, 17 link instances with a total of 3.737 links and 170 derived links (see Fig. 3). Figure 3 also shows the evaluation of the invariants for the generated object diagram satisfying all class invariants. The low amount of layer 1 and layer 2 links comes from the fact that the configuration of neighboring network components was not processed. Without this information it is not possible to certainly create links, but VLAN properties can be checked on interface level. The present links model the interconnection of the four Cisco switches.

Beyond the validation using the defined multiplicities and class invariants, it is possible to do more security related analyses with the system state. It is possible to check whether two network components are in the same "broadcast domain", i.e. can reach each other by sending broadcast frames on layer 2. This logical segregation can be checked by querying through the linked layer 2 interfaces respecting the VLAN configuration of each interface. Extending the model



Object count

Class	# Objects
AggregatedLayer2Interface	84
Interface	0
Layer1Interface	1157
Layer1Link	12
Layer2Interface	0
Layer2Link	5
Link	0
NetworkComponent	4
SimpleLayer2Interface	1157

Link count

Association	# Links
derivedHasLayer1Connection	12
derivedHasLayer1Interface	146
derivedHasLayer1Link	12
hasInterface	2400
hasLayer1Interface	1157
hasLayer1Link	24
hasLayer2Link	10
hasSimpleLayer2Interface	146

Class invariants

Invariant	Satisfied
AggregatedLayer2Interface::AssociatedLayer1InterfacesAreProhibited	true
AggregatedLayer2Interface::AssociatedLayer1InterfacesAreProperlyConnected	true
AggregatedLayer2Interface::AssociatedNetworkComponentsHaveAPeerLink	true
AggregatedLayer2Interface::AssociatedWithMaxTwoNetworkComponents	true
AggregatedLayer2Interface::SimpleLayer2InterfacesAreAssociatedToSameNet...	true
AggregatedLayer2Interface::SimpleLayer2InterfacesDontHaveLinks	true
AggregatedLayer2Interface::SimpleLayer2InterfacesHaveConsistentVLANConfi...	true
AggregatedLayer2Interface::SimpleLayer2InterfacesHaveDistinctLayer1Interfac...	true
Layer1Interface::AssociatedWithOneNetworkComponent	true
Layer1Link::SelfCommunicationProhibited	true
Layer2Link::AssociatedLayer2InterfacesHaveConsistentVLANConfiguration	true
Layer2Link::MinChassisRedundancyLowerEqualThanChassisRedundancy	true
Layer2Link::MinLinkRedundancyLowerEqualThanLinkRedundancy	true
Layer2Link::SelfCommunicationProhibited	true
NetworkComponent::AllLayer1InterfaceNamesAreUnique	true
NetworkComponent::AllLayer2InterfaceNamesAreUnique	true
NetworkComponent::NetworkComponentNameIsGloballyUnique	true
SimpleLayer2Interface::AssociatedLayer1InterfacesAreRequired	true
SimpleLayer2Interface::AssociatedWithOneNetworkComponent	true
SimpleLayer2Interface::Layer1InterfaceIsAssociatedToSameNetworkComponent	true

Constraints ok. (50ms) 100%

Fig. 3. Object count, link count and class invariants of an example object diagram of four interconnected network switches in the University of Bremen data center.

with OSI layer 3 would allow further security property analyses of IP routing and firewalls.

Sometimes it is not necessary for the modeler to have all information visible. With the help of the described derived associations, it is possible to hide every object of network layer 1. The resulting object diagram in Figure 4 shows only the logical layer 2 topology, which interconnects the four switches. Despite having 12 interconnecting layer 1 links, the logical layer 2 topology looks far smaller and clearer, because of the two aggregated multi-chassis layer 2 interfaces which are distributed across two network components each. Having this complex configuration visually present is not only useful for documentation purposes, but helps network administrators with the comprehension and further maintenance of the network topology.

The derived association `derivedLayer1Connection` is represented by the dashed lines in Figure 4 which states that the four switches are creating a meshed network topology (the diagonal links continue behind the layer 2 link, connecting the network components in the opposite corners). The responsible derived association for these links is defined as follows:

```
association derivedLayer1Connection between
  NetworkComponent[*] role sourceNetworkComponent
  NetworkComponent[*] role destinationNetworkComponent derived =
      self.getLayer1Interfaces().getOpposite().networkComponent→asSet()
end
```

The neighbored (or destination) network components are queried through the associated layer 1 interfaces and layer 1 links. Due to the symmetry of the layer 1 links, every destination network component acts as a source network component as well, which causes two links per network component pair.

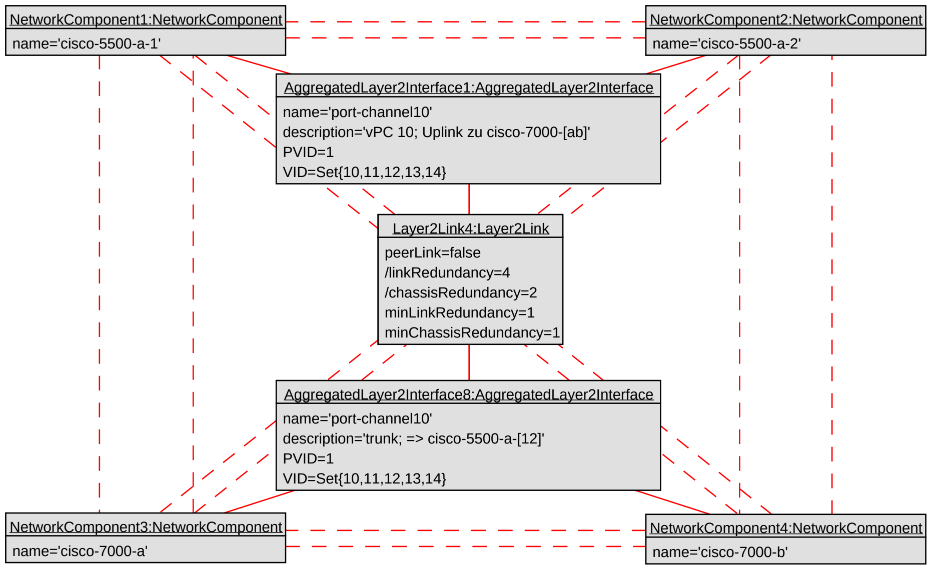

NetworkComponent1:NetworkComponent
name='cisco-5500-a-1'
NetworkComponent2:NetworkComponent
name='cisco-5500-a-2'
AggregatedLayer2Interface1:AggregatedLayer2Interface
name='port-channel10'
description='vPC 10; Uplink zu cisco-7000-[ab]'
PVID=1
VID=Set{10,11,12,13,14}
Layer2Link4:Layer2Link
peerLink=false
/linkRedundancy=4
/chassisRedundancy=2
minLinkRedundancy=1
minChassisRedundancy=1
AggregatedLayer2Interface8:AggregatedLayer2Interface
name='port-channel10'
description='trunk; => cisco-5500-a-[12]'
PVID=1
VID=Set{10,11,12,13,14}
NetworkComponent3:NetworkComponent
name='cisco-7000-a'
NetworkComponent4:NetworkComponent
name='cisco-7000-b'

Fig. 4. Object diagram of four interconnected network switches in the University of Bremen data center with hidden layer 1 objects. With the derived properties, all necessary information is visible.

4.2 Generation of Network Configurations

The network topology model cannot only be used to validate and visualize the configuration of network components. It can also be used to create possible system states out of a given partial system state with the USE model validator.

The frame in Figure 5 marks the objects that were used as input for the model validator. The modeler requires two network components redundantly connected using a layer 2 link with aggregated layer 2 interfaces. With the attribute `minLinkRedundancy` it is enforced that at least two layer 1 links have to cover the layer 2 link. The constraints defined with the attributes affect the derived properties and these in turn affect the overall completion of the partial system state.

With the UML/OCL model and this partially given object diagram, the model validator is now capable of generating the missing layer 1 and 2 objects and links using given search boundaries in order to fulfill all model constraints including multiplicities. The overall translation time of the partial system state shown in the frame in Fig. 5 takes only 360 ms in addition to a solving time of 10 ms. Further security requirements could be employed before the completion to enforce further restrictions for the system state. In this example, the concrete component names were not important, hence the random strings like “string5” or “string7”. With a more sophisticated model validator configuration, more meaningful interface names like “Ethernet1/1” can be generated as well.

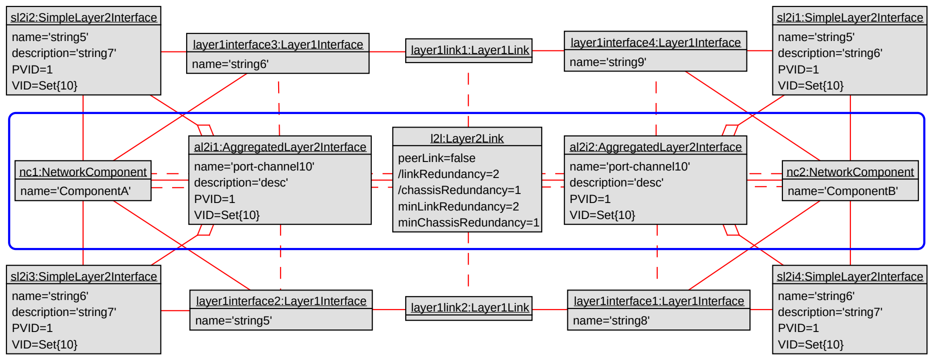


Fig. 5. Automatic generation of the missing layer 1 connections using the USE model validator with a given logical layer 2 topology (blue frame).

To think one step further, this system state can be used to generate the Cisco-specific configuration commands for the network components using a model-to-text transformation. Network administrators can model complete network structures or just the logical layer 2 topology as a partial system state, completed by the model validator, as shown in the example. A model-to-text transformation would then generate the Cisco-specific configuration commands. With this workflow, it would be possible to create consistent and valid configurations out of the network topology documentation which nowadays is usually done the other way around. Ultimately, this allows network administrators to focus on the higher, more abstract layers of the network infrastructure, knowing that the required lower layers will be generated automatically, fulfilling all security and availability requirements.

5 Transformation into Relational Logic

In order to use derived attributes and associations in the USE model validator, their semantics have to be translated into relational logic [12,13]. One of the challenges is that in the SAT-based instance finding there are no concrete objects on which the expressions can be evaluated, but rather the search space is manipulated to generate valid system states. This is also the reason for a limitation that applies to both features. The model validator does not support recursive query operations, because it is unknown how often the operation needs to be unrolled and for the same reason does not support recursive derived expressions. The extensive OCL support of the model validator is reused to transform the actual derived expression and the following sections describe the rest of the transformations.

5.1 Derived Attributes

Derived attributes are the simpler of the two properties to transform, since they do not employ further constraints on the model unless they are explicitly mentioned in model invariants. They are very similar to query operations but are represented as attributes in object diagrams. For this reason, the model validator does not transform the attribute itself, but rather substitutes all access to the attribute value with the derived expression on the fly during the transformation of OCL expressions. As a result, an expression `<expr>.attribute` is transformed into $\texttt{derived}_{\text{attribute}}\texttt{(<expr>)}$, where `<expr>` is an arbitrary OCL expression of the type of the derived attribute passed as the parameter to the derived expression function from Definition 1. The resulting expression is then transformed normally by the existing OCL transformation algorithm.

The advantage of this approach is that the derived attributes are not directly part of the model and therefore do not need to be specified in the problem bounds, meaning it does not extend the search space. The actual value is determined once the solution instance is transformed back into a USE system state, at which point USE evaluates the derived properties.

5.2 Derived Associations

The transformation of derived associations is more complex for two reasons:

1. Derived associations can be navigated in both directions, requiring that the derived expression can be applied backwards. However, derived expressions may be arbitrary OCL expressions that are not required to be bijective.
2. The semantics of derived associations put further constraints on the model that have to be fulfilled in order to have a valid system state. In particular, the number of calculated links must conform their multiplicities.

Navigating towards a derived role is similar to how derived attributes work. The derived expression function can be applied to the current object resulting in the linked objects, effectively simulating the navigation. However, there is no way of reversing an OCL expression to make it suitable for the navigation in the opposite direction. For the backwards navigation, all possibilities have to be considered and filtered for the ones that are applicable.

Example 2 (Role Navigation). Consider this simple derived association:

```
association AB between
  A [2]    role a
  B [1..4] role b  derived = <OCL expression>
end
```

The navigation from `A` to `B` is achieved by applying the derived expression. The navigation from `B` to `A`, e.g. by an expression `self.a` in an invariant of `B`, has to be calculated using all objects of type `A`. The expression `self` can be substituted with any arbitrary OCL expression of the correct type. In OCL the navigation expression would be:

```
A.allInstances()→select( a | derived(a)→includes( self ) )
```

In relational logic, this same semantics is achieved with a comprehension:

$$\{a : \texttt{one A} \mid \texttt{self} \in \texttt{derived}(a)\}.$$

Note that the relational logic encoding does not have single object values. These are represented as a set containing one element. Therefore, multiplicities with an upper bound of 1 do not need special treatment.

The second challenge, transforming the semantics of multiplicities, is solved with a constraint for every association end, which is added to the model. Roles with non-restricting multiplicities `0..*` can be ignored.

Example 3 (Multiplicity Constraint). For the previous association, the multiplicity constraints formulated in relational logic are as follows:

$$\begin{aligned}&(\texttt{all}\ a : \texttt{one A} \mid \#\texttt{derived}(a) \geq 1 \wedge \#\texttt{derived}(a) \leq 4) \wedge \quad && \text{// mult. role b}\\ &(\texttt{all}\ b : \texttt{one B} \mid \#\{a : \texttt{one A} \mid b \in \texttt{derived}(a)\} = 2) && \text{// mult. role a}\end{aligned}$$

For n-ary associations both transformations are similar, however more parameters need to be bound. Here the role navigation for n-ary associations is exemplified, but the changes to the multiplicity constraints are similar.

Example 4 (Ternary Navigation). Adding a third role `c` to the previous association changes the formula for the navigation from `B` to role `a` to the following:

$$\Big\{a : \texttt{one A} \,\Big|\, \big(\underbrace{\texttt{some}\ c : \texttt{one C}}_{\text{bind additional parameters}} \mid \texttt{self} \in \texttt{derived}(a, c)\big)\Big\}.$$

6 Future Ideas for Derived Properties

During the development and work with the derived properties, we came across further ideas regarding the tooling and features that are not yet supported by the UML and OCL standards.

6.1 Discussion of Tool Support

The system state completion use case from Sect. 4.2 uses partial system states to provide key data for a network infrastructure that is then completed by the model validator to a valid system state showing all components and wiring. This partial state fixes key data of the final result including objects, links and also attribute values. However, these attribute values include derived attributes as well, which are evaluated on the partial system state that might not reflect the intend of the modeler. For this reason, the derived attribute values are ignored when transforming the partial system state in the model validator. In the USE tool, derived properties are calculated model elements that automatically refresh their values when necessary. The implementation does not allow to manually assign values at all. Also the model validator plugin does not account for this situation. In the current implementation, where derived properties are not explicitly bounded, also the configuration is no help. Therefore, in the current state it is not possible to specify exact values for derived properties in partial system states as it is for regular properties. However, this feature is very useful and should be supported by tools that deal with derived properties, for derived attributes and associations alike.

6.2 Derived Classes and Association Classes

Already in 2003 a method was presented to define derived classes for database views using tuple-typed derived attributes [1]. There, derived classes are modeled as a derived attribute of type `Set(Tuple(...))`, where the contents of the tuple represent attributes on the derived class. These attributes of the derived class must be derived themselves. However, the representation of these objects within an attribute does not have the same features as real objects, e.g. the operation `allInstances()` cannot be invoked on this derived type. Thus a nicer integration is desired, but the concepts already exist and show the feasibility and applications.

Building upon the idea of derived classes, also derived association classes come to mind. The instances are defined by a derived role like regular derived associations, but allow derived attributes to be defined on them, similar to the derived classes. These derived classes can be used as the context for model invariants in order to employ constraints. These new features are a straight forwards combination of existing elements in the standards and reuse their components. Adding them would complete the support for derived properties in UML/OCL.

7 Conclusion and Future Work

We have presented a network topology model in UML/OCL to describe the lower OSI layer connections of network components. Derived attributes and associations are employed to further constraint the model and improve visual representations, providing a better understanding. With this model it is possible to validate given real-life networks for consistency as demonstrated with parts of the data center of the University of Bremen. Ideas for a model transformation from system states to network configurations were sketched.

Additionally, the support for derived properties was added to the USE model validator in order to verify further properties of the model and to increase the capabilities of the tool itself. In particular, a way of generating network structures from key data points was presented. A transformation of the semantics of derived properties into the relational logic of Kodkod was introduced.

Future work can concentrate on many aspects. The network topology model can be extended with higher OSI layers like layer 3 (IP) to take into account firewall rules as well. The USE model validator plugin can be improved by adding more features of the UML, e.g. union and subsets association end properties, two other derived features that are not based on derived expressions but rather the inheritance hierarchy. They improve the abstraction features in the network topology by automatically deriving connections in the abstract layer from lower hierarchy level links. Finally, the performance of the tool when generating larger networks needs to be investigated further.

Acknowledgement. We thank Henk Birkholz for the helpful discussions about the UML network topology model and Markus Germeier for providing the configuration files of the University of Bremen data center.

References

1. Balsters, H.: Modelling Database Views with Derived Classes in the UML/OCL-Framework. In: Stevens, P., Whittle, J., Booch, G. (eds.) The Unified Modeling Language, Modeling Languages and Applications, UML 2003. LNCS, vol. 2863, pp. 295–309. Springer (2003)
2. 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. pp. 126–133. IEEE Computer Society (2014)
3. Birkholz, H., Sieverdingbeck, I., Sohr, K., Bormann, C.: IO: An Interconnected Asset Ontology in Support of Risk Management Processes. In: Seventh International Conference on Availability, Reliability and Security, ARES 2012. pp. 534–541. IEEE Computer Society (2012)
4. Bundesamt für Sicherheit in der Informationstechnik: BSI Standard 100-2 IT-Grundschutz Methodology, Version 2.0 (2008), `https://www.bsi.bund.de/SharedDocs/Downloads/EN/BSI/Publications/BSIStandards/standard_100-2_e_pdf.pdf`
5. Burns, J., Cheng, A., Gurung, P., Rajagopalan, S., Rao, P., Rosenbluth, D., Surendran, A., Jr, D.M.: Automatic Management of Network Security Policy. DARPA Information Survivability Conference and Exposition, 2, 1012 (2001)
6. Cisco Systems, Inc.: Campus Network for High Availability Design Guide (May 2008), `https://www.cisco.com/c/en/us/td/docs/solutions/Enterprise/Campus/HA_campus_DG/hacampusdg.pdf`
7. Enck, W., Moyer, T., McDaniel, P., Sen, S., Sebos, P., Spoerel, S., Greenberg, A., Sung, Y.W.E., Rao, S., Aiello, W.: Configuration Management at Massive Scale: System Design and Experience. IEEE J.Sel. A. Commun. 27(3), 323–335 (Apr 2009)
8. Gogolla, M., Büttner, F., Richters, M.: USE: A UML-Based Specification Environment for Validating UML and OCL. Science of Computer Programming 69, 27–34 (2007)
9. Hamann, L., Gogolla, M.: Endogenous Metamodeling Semantics for Structural UML 2 Concepts. In: Moreira, A., Schätz, B., Gray, J., Vallecillo, A., Clarke, P.J. (eds.) Model-Driven Engineering Languages and Systems, MODELS 2013. LNCS, vol. 8107, pp. 488–504. Springer (2013)
10. Hilken, F., Niemann, P., Gogolla, M., Wille, R.: From UML/OCL to Base Models: Transformation Concepts for Generic Validation and Verification. In: Kolovos, D.S., Wimmer, M. (eds.) Theory and Practice of Model Transformations, ICMT 2015. LNCS, vol. 9152, pp. 149–165. Springer (2015)
11. ITU-T – International Telecommunication Union: X.200 : Information technology – Open Systems Interconnection – Basic Reference Model: The basic model (July 1994), `http://www.itu.int/rec/T-REC-X.200-199407-I`
12. Kuhlmann, M., Gogolla, M.: From UML and OCL to Relational Logic and Back. In: France, R.B., Kazmeier, J., Breu, R., Atkinson, C. (eds.) Model Driven Engineering Languages and Systems, MODELS 2012. LNCS, vol. 7590, pp. 415–431. Springer (2012)
13. Torlak, E., Jackson, D.: Kodkod: A Relational Model Finder. In: Grumberg, O., Huth, M. (eds.) Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2007. LNCS, vol. 4424, pp. 632–647. Springer (2007)

A USE Model Validator supported UML/OCL Features

Unified Modeling Language (UML)		
Class features		
✓ Class		
✓ Abstract Class		
✓ Inheritance		
✓ Multiple Inheritance		
✓ Attribute		
✓ Derived Value	new in this paper	
✗ Initial Value		
✓ Enumeration		
✓ Invariant		
Association features		
✓ Binary Association		
✓ N-ary Association		
○ Aggregation	limited support of cycle freeness (otherwise ✓)	
○ Composition	limited support of cycle freeness (otherwise ✓)	
✓ Multiplicity		
✓ Association Class		
✓ Derived Association End	new in this paper	
✗ Qualified Association		
✗ Redefines, Subsets, Union		
Operation features		
✓ Query Operation		
✓ Parameter		
✓ Return Value		
✗ Recursion		
✗ Operation Call (non query)	checking behavior possible via filmstripping	
✗ Parameter	∟ with filmstripping	
✗ Return Value	∟ with filmstripping	
✗ Pre-/Postcondition	∟ with filmstripping	
✗ Nested Operation Call		
Object Constraint Language (OCL)		
OCL types		
✓ Boolean	✓ Integer	✓ Class Type
○ String	○ Real	✗ UnlimitedNatural
✓ Set	✗ Bag	✗ Sequence
✗ OrderedSet	✗ Nested collections	
OCL operations		
✓ Comparison Operators	✓ Boolean Operations	✓ Integer Operations
○ String Operations	✗ substring	✗ concat
✓ <Class>.allInstances	✗ <Assoc>.allInstances	✓ size
✓ isEmpty/notEmpty	✓ includes/excludes	✓ including/excluding
✓ forAll/exists	✓ select/reject	✓ one
✓ isUnique	✓ union/intersection	○ any
✓ collect	✓ closure	✗ iterate
✓ toString	○ sum	✓ oclIsType/KindOf
✓ selectByType/Kind	✓ oclAsType	✗ oclType

✓ supported element – ✗ unsupported element – ○ partially supported element