A Graphical Modeling Tool Supporting Automated Schedule Synthesis for Time-Sensitive Networking

Morteza Hashemi Farzaneh, Stefan Kugele, and Alois Knoll
Department of Informatics
Technical University of Munich
Boltzmannstr. 3, 85748 Garching bei München
Email:{hashemif,kugele,knoll}@in.tum.de

***Abstract*—Time-Sensitive Networking (TSN) is a set of new standards which is being developed by the Institute of Electrical and Electronics Engineers to support mixed-criticality requirements based on Ethernet technology. These standards have recently raised the attention of real-time domains such as automation and automotive. To support tight timing guarantees, Time-Aware Shaper (*IEEE 802.1Qbv*) is introduced based on the theory of time-triggered communication. However, the configuration of Time-Aware Shaper requires expertise and is a time consuming procedure.**

We aim to automate this procedure reducing configuration overhead. A novel graphical modeling is introduced which combines the strengths of model-based and logic programming modeling paradigms. Using the graphical editor, network topology, dataflow based on a publisher and subscriber concept, and the quality of service requirements are specified. Facts for a network knowledge base are automatically derived from the model.

This knowledge base is used to generate constraints for schedule synthesis. These constraints are solved using a Satisfiability Modulo Theories solver to find a correct schedule. Moreover, we exploit the solver's capability of producing unsatisfiable cores and use it for network model correction. We annotated all generated schedule constraints and mapped them to the stream names to track the unsatisfiable streams from the solver's output.

We gained insightful results showing that this information significantly helps to correct an unsatisfiable network model to find a feasible schedule.

I. Introduction

Time-Sensitive Networking (TSN) [1] is a task group of the Institute of Electrical and Electronics Engineers (IEEE) which is developing new standards to support mixed-criticality communication with tight timing requirements based on Ethernet technology. These standards have recently raised the attention of real-time domains such as automation and automotive. To guarantee hard real-time requirements, Time-Aware Shaper (TAS) standard (*IEEE 802.1Qbv*) is introduced based on the theory of time-triggered communication. Using TAS and based on the frames' priority, frame queuing delays in the egress ports are minimized. Gate Control List (GCL) is a main component of TAS and contains a list of gate opening times for all 8 priorities queues based on *IEEE 802.1Q*. Each network port contains a GCL and it requires to be correctly configured in order to give latency and jitter guarantees and avoid deadline exceeding.

978-1-5090-6505-9/17/$31.00 ©2017 IEEE

Finding a feasible time-triggered schedule is a known NP-complete problem [2]. To deal with this problem, timing requirements are transformed to schedule constraints and prepared for constraint solvers such as Satisfiability Modulo Theories (SMT) [2]–[4] or Integer Linear Programming (ILP) [5] to find a feasible schedule for GCL configuration. To achieve this transformation, information about e. g. frames and affected egress ports in relation to their timing requirements have to be extracted.

However, without an adequate network model, this transformation is time-consuming especially at network design time when repeated changes are expected. It also requires expertise in the solver platform in order to obtain a correct transformation.

Contributions: In this paper, we aim to automate the GCL schedule synthesis to reduce the network configuration overhead. (i) We introduce a model-driven approach including a graphical editor to specify network topology, dataflow based on a publisher and subscriber concept, and the quality of service (QoS) requirements (following data-centric middleware approaches (cf. [6], [7]). This editor can be used by real-time application developers and network managers. (ii) We apply logic programming as a modeling paradigm to transform the designed graphical network model to a network knowledge base. The network knowledge base is the basis of our network query tool called TSN Declarative Network Manager (DNM). Using adequate queries, schedule constraints are generated. Moreover, developers and network managers can use DNM to verify reliability requirements in the network. (iii) We demonstrate our tool using an exemplary use case and show how a generated unsatisfiable core (a feature of SMT solvers) of a given network model helps to correct it and find a feasible schedule.

Outline: The remainder of this paper is structured as follows. First, we present the state-of-the-art in Section II, then we propose the main contributions in Section III. Section IV presents our case study and finally we conclude in Section V.

II. State-of-the-Art

Gomaa [8] presents a methodology to design real-time and distributed applications, which integrates object-oriented, and concurrency concepts by using the Unified Modeling Language (UML) [9]. In [10], a graphical notation is introduced

to characterize the automotive software architecture and to distribute processes and tasks on processors. In [11], a graphical notation is developed to model Multicore Programmable Logic Controllers (MPLC) in networked automation systems to increase analyzability of these systems. MATLAB Simulink and Stateflow [12] are graphical modeling and simulation tools which are widely used in different research and industrial fields such as in developing real-time and embedded systems. SysML [13] is a graphical modeling language based on an extended subset of UML. It is used for system engineering and supports mechanisms for system design and verification. Ptolemy [14] is an open-source platform supporting experimentation with an actor-oriented design which offers a graphical user interface to modify the models. MARTE [15] is a UML profile and is applied for modeling real-time systems. UPPAAL [16] is a tool box based on timed automata to support modeling, simulation and verification of real-time systems. It offers graphical and textual modeling tools.

In contrast to the mentioned modeling approaches, we define a metamodel with a formal semantics based on logic programming paradigm and use it in combination with the graphical modeling tool to create and maintain a knowledge base (in the sense of artificial intelligence) of network facts and use them to configure and verify the network.

Logic programming is used as a platform for modeling and verification of Cyber-Physical Systems (CPS) in [17]–[19]. The motivation is to bridge between logic programming and hybrid automata as the underlying model. Loo et al. [20] propose a declarative networking modeling approach based on logic programming language Datalog which is a subset of Prolog. This work is extended in [21] and used for declarative network verification of e. g. routing protocols. A Datalog-based declarative network management approach is proposed in [22]. Lopes et al. [23] show that Prolog is sufficiently expressive and suitable for implementation of distributed protocols. In contrast to our contribution, the mentioned approaches lack on graphical modeling features.

Ontology-based approaches are applied for network management tasks [24]–[27]. Vergara et al. [28] review these proposals regarding their advantages and shortcomings. The paper claims, that ontology-based interoperability frameworks can help to ease several tasks in the network management. It helps network administrators towards automation of parts of management tasks. In our previous work [29], an ontology-based approach was proposed to improve the modeling and plug-and-play capabilities of TSN. Describing network management policies such as firewall rules are proposed in [30] and [31] to show the capabilities of ontology-based network management. Martinez et al. [32] propose an ontology-based information extraction system to fill the configuration gap in hybrid *Software-Defined Networks* (SDN). One use case is the formalization of switch and router configurations to reduce the configuration overhead for administrators. In this paper, we apply logic programming as reasoning engine and replace the ontological visualization (cf. [29]) with a graphical modeling approach (i) to improve the visual features of our

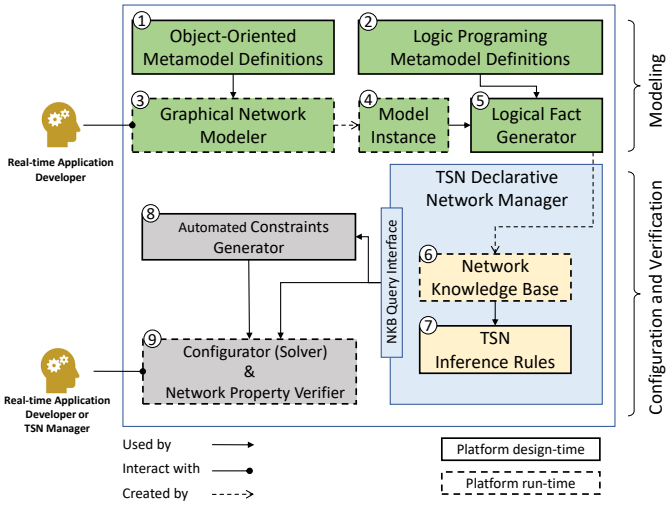

Fig. 1: Overview of the modeling approach

modeling tool including network topology and dataflow, and (ii) to embed the graphical modeling tool directly into a programming environment and use it for constraint generating and solving. The mentioned advantages are missing in the classical ontology tools such as Protégé [33].

Simulation-based approaches [34]–[38] are used to analyze the performance of Audio Video Bridging (AVB) and time-triggered Ethernet. They are very useful to gain an insight in these technologies. However, they cannot cover all corner cases and therefore are not suitable for formal verification of critical requirements. To overcome this problem, formal analysis methods such as in [39]–[41] are developed to verify the performance of the TSN shapers. Both simulation-based and formal methods do not deal with configuration challenges of TSN such as schedule synthesis. The schedule synthesis of time-triggered Ethernet is discussed in [2]–[5].

We use the theoretical contributions of these papers in our modeling tool and combine them with the logic programming approach towards automation of TSN configuration procedure and simplified network verification. Moreover, we exploit the solver's capability of producing unsatisfiable cores and analyze them in order to correct the network model.

III. Approach

As depicted in Figure 1, the modeling approach consists of a metamodel which is used as a basis for the graphical modeling tool to created graphical model instances. Application developers or TSN managers use the graphical editor to specify the dataflow and to annotate the QoS requirements of the developing real-time applications. For instance, it can be annotated that a specific application contains periodic data transmission with tight timing constraints or it has reliability requirements and therefore requires e. g. multiple disjoint paths (cf. *IEEE 802.1CB*). The modeling tool is also used to define the network topology and the physical properties of the components.

Using a formal metamodel based on logic programming and following our previous work [42], the graphical model

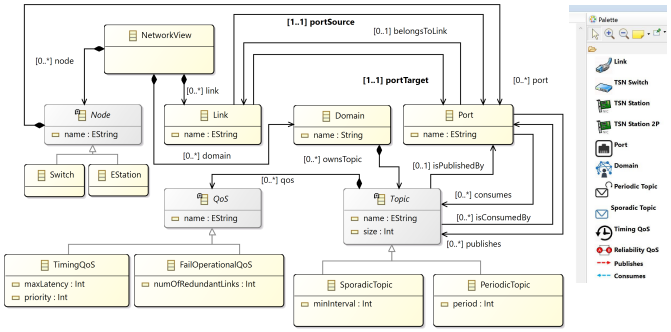

Fig. 2: Object-oriented metamodel of the network (left) and the graphical modeling palette (right)

instances are transformed into a knowledge base inside the DNM. We use Prolog as a concrete implementation of logic programming and transform the graphical network model instance into Prolog facts. Moreover, we implemented Prolog rules to use the advantages of logical inference to (i) query and verify network properties such as required redundant disjoint paths and (ii) query the required information from the network model to generate time-triggered schedule constraints. Using a Satisfiability Modulo Theories (SMT) solver, the constraints are solved and it will be decided whether there is a feasible schedule. The constraints can also be used for verification of manually created schedules. The solver checks the schedule and decides if all constraints (real-time requirements) are fulfilled.

A. Graphical Modeling Tool

To define the object-oriented model of the network, we use the Eclipse Modeling Framework (EMF)[1]. A network model consists of a *network view* which contains all graphical network components. A network *node* is either a *switch* or an *end-station*. Nodes have at least one Ethernet *port*. Switches and end-stations are connected through ports. A *link* connects two nodes and has exactly one *source* and one *target* port.

A real-time application is presented as a *domain*. Each domain contains one or more data *topics* which are *published* and *consumed* through ports within a domain. We distinguish between a *periodic topic* and a *sporadic topic*. The periodicity of the topics is highly relevant for schedule synthesis. In contrast to periodic topics, the sporadic topics have an attribute *minInterval* for minimum time between two consecutive frames. This parameter is important for worst case latency analysis of a sporadic frame.

Data topics with tight timing requirements have to be periodic in order to be considered in the schedule synthesis constraints in *IEEE 802.1Qbv*. Each topic may contain *QoS* requirements such as *timing* and *fail-operational* to specify e. g. the number of required redundant links. The metamodel and a palette of graphical modeling elements built on top of it are presented in Figure 2.

To demonstrate the modeling tool, we used it to model our in-vehicle automotive network example from [42]. The

[1]http://www.eclipse.org/modeling/emf/

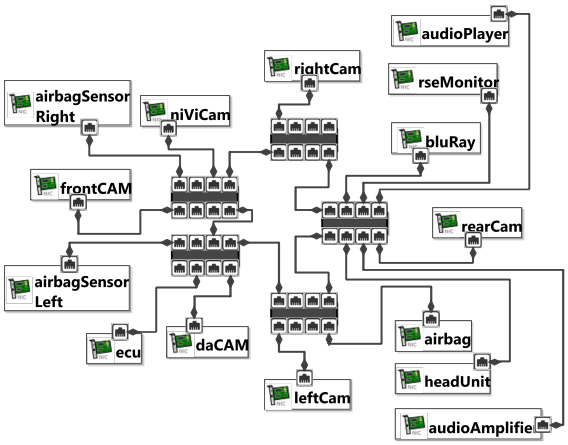

Fig. 3: Modeling network topology using the graphical editor

example network consists of 5 TSN switches with each 8 ports. End-stations are connected to these switches and build together the hardware topology as demonstrated in Figure 3. The end-stations belong to different domains (applications). The cameras are used in driver assistance system domain, airbag sensors and actuators are involved in the real-time airbag domain, and multimedia end-stations belong to the entertainment domain. Having a closer look on the dataflow of

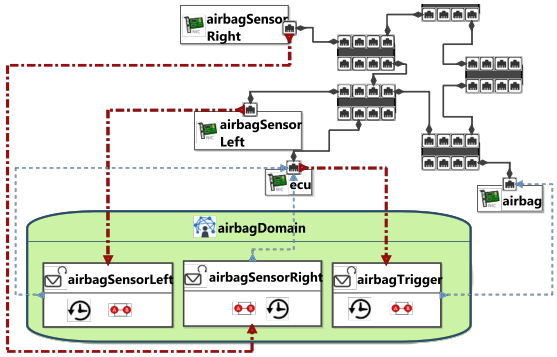

Fig. 4: Modeling dataflow and QoS requirements using the graphical editor

the airbag domain as depicted in Figure 4, *airbagSensorRight* and *airbagSensorLeft* data topics are published by the sensors and by *ecu* end-station. It controls the sensors' data and publishes *airbagTrigger* topic which is consumed by airbag actuator. The QoS requirements are inside the topics. For instance, *airbagTrigger* topic is a periodic topic and has both timing and reliability requirements. The appropriate parameter values such as period and size, etc. are entered textually in the modeling tool (not demonstrated here).

B. Logic-Programming Metamodel

We use Prolog (SWI implementation), a well-known logic programming language to define the formal metamodel of the network knowledge base. Prolog is based on Horn clauses and consists of *facts* and *inference rules*. Each rule has the form: $\alpha :- \beta_1, \beta_2, \ldots, \beta_n$ which is equivalent to

$$\beta_1 \wedge \beta_2 \wedge \cdots \wedge \beta_n \implies \alpha$$

In Prolog, α is called *head* and $\beta_1, \beta_2, \ldots, \beta_n$ is the *body* of the rule. The head is true if the body is true. Each β in the body is a call to a *predicate*. The predicates in the body are known as *goals*. They can be either a fact (a clause with empty body) or a rule.

The whole knowledge base of a graphically modeled network consists of facts. Rules are defined to be able to describe logical relations between facts and rules. We use rules for synthesis-related queries on the knowledge base in DNM. Considering the example from Figure 4,

```
publishes(airbagSensorRight, airbagSensorRight_p1,
          airbagDomain,airbagSensorRight).
```

is a fact which describes that the end-station *airbagSensorRight* uses its Ethernet port *airbagSensorRight_p1* to publish a data topic *airbagSensorRight* in the domain *airbagDomain* domain and the following

```
firstPort(Domain,Topic,Port) :- publishes(_,Port,Domain,
     Topic).
```

is a rule that declares that *Port* is the first egress port to transmit *Topic* in *Domain*, if there is a fact *publishes* in the knowledge base with the desired values. The query to find the first egress port when publishing e. g. *airbagTrigger* topic is:

```
firstPort(airbagDomain,airbagTrigger,Port).
```

and Prolog responds with *Port=airbagSensorRight_p1*.

In the following, the formal semantics of facts are described. Moreover, we present the inference rules which have been developed to query the knowledge base regarding verification and schedule synthesis.

1) Facts: The modeling clauses consist of the following facts: *topic*, *qos*, *publishes*, *consumes*, *end-station*, *switch*, and *isLinked*.

Let NODE denote the set of all network *nodes* consisting of *switches* and *end-stations* denoted by the sets ESTATION and SWITCH, respectively: NODE $=$ SWITCH$\cup$ESTATION. Moreover, let PORT be the set of all Ethernet ports, which can be connected using links, where LINK denotes the set of all links. We denote with DOMAIN the set of all *domains* and with TOPIC the set of all *topics*.

For a better presentation, we also assume things to be named (labeled) with strings by an injective function.

$$\ell\colon (\texttt{NODE} \cup \texttt{LINK} \cup \texttt{DOMAIN} \cup \texttt{TOPIC}) \rightarrowtail \texttt{STRING}$$

where STRING is the set of all possible finite words.

Definition 1 (Network Topology)**.** Let $\mathcal{N} = (N, L, D)$ be a *network topology*, where $N \subseteq$ NODE is the set of *nodes* of the network and $L \subseteq P \times P$ denotes the set of bi-directional *links* between node ports, where $P \subseteq$ PORT. Self-loops are prohibited, i. e., $\forall (p, p') \in L\colon p \neq p'$. Moreover, a port p can only be part of at most a single link, i. e., $|\{p \in P \mid (p, p') \in L \vee (p', p) \in L\}| \leq 1$. Each node has a set of attached ports: $\text{ports}\colon \texttt{NODE} \rightarrow \wp(\texttt{PORT})$. It holds that each port is *unique*, i. e., $\forall n, n' \in N, n \neq n'\colon \text{ports}(n) \cap \text{ports}(n') = \emptyset$. Switches and end-stations use Ethernet ports to transmit or forward data topics embedded in Ethernet frames. Moreover, let $D \subseteq$ DOMAIN be the set of all network *domains*. Each domain is assigned a set of *topics* $T \subseteq$ TOPIC: $\text{topics}\colon \texttt{DOMAIN} \times \wp(\texttt{TOPIC})$. Topics are *injectively* assigned to domains, i. e., $\forall d, d' \in D, d \neq d'\colon \text{topics}(d) \cap \text{topics}(d') = \emptyset$.

Commonly used topic types are *periodic* (per) and *sporadic* (spo); the set of all topic types is referred to as TTYPE $= \{\texttt{per}, \texttt{spo}\}$. Topic types are of importance for network configuration.

We derive *facts* from (i) the network topology (in particular nodes and links), (ii) topics and quality of service attributes, and (iii) published and consumed topics (w.r.t. data-centric middleware). The derived facts are given next.

Definition 2 (Facts)**.** (1) A *switch fact* f^{s} is a pair (ν, P) where $\nu \in$ STRING denotes its name and $P \subseteq$ PORT is the set of the switch's ports. Analogously, an *end-station fact* f^{e} is defined as a tuple (ν, P) where $\nu \in$ STRING denotes its name and $P \subseteq$ PORT is the set of the end-station's ports. (2) A *link fact* f^{l} is a triple (p, p', b) where $(p, p') \in L$ and $b \in \mathbb{N}_0^+$ is the link bandwidth. (3) A *domain fact* f^{d} is a pair (ν, T) where $\nu \in$ STRING denotes the name of domain d and $T = \text{topics}(d)$ is the set of owned topics. (4) A *topic fact* f^{t} is a tuple $(\nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \tau, \rho, \sigma)$ where $\nu^{\mathsf{d}}, \nu^{\mathsf{t}} \in$ STRING denote the domain and topic name, respectively. The topic type is denoted by $\tau \in$ TTYPE and its periodicity by $\rho \in \mathbb{N}_0^+$. Finally, $\sigma \in \mathbb{N}_0^+$ is the size of the topic. (5) A *publishes fact* f^{p} and a *consumes fact* f^{c} are both tuples $(\nu^{\mathsf{n}}, \nu^{\mathsf{p}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}}) \in \texttt{STRING}^4$ where $\nu^{\mathsf{n}}, \nu^{\mathsf{p}}, \nu^{\mathsf{d}}$, and ν^{t} denote the node, port, domain, and topic names. (6) A *Timing QoS fact* f^{qt} and *Reliability QoS fact* f^{qr} are defined as tuples $f^{\mathsf{qt}} = (\nu^{\mathsf{q}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \omega, \varrho)$ and $f^{\mathsf{qr}} = (\nu^{\mathsf{q}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \zeta)$ where $\nu^{\mathsf{q}}, \nu^{\mathsf{d}}, \nu^{\mathsf{t}} \in$ STRING, $\omega, \varrho, \zeta \in \mathbb{N}_0^+$ denote the QoS name, domain name, topic name, deadline, priority, and number of redundant links. For periodic topics, we assume $\omega \geq \rho$. Following *IEEE 802.1Qbv* with 8 priority classes, one has $0 \leq \varrho \leq 7$.

Following, an excerpt of the facts from the automotive example is presented which is generated from the graphical model.

```
estation(airbagSensorRight, [es1_p1]).
isLinked(es1_p1, sw1_p1, 1000000000).
topic(airbagDomain, airbagTrigger, periodic, 250, 400).
qosReliability(q1, airbagDomain, airbagSensorRight, 2).
publishes(rightCam, es7_p1, adasDomain, rightCam).
consumes(daCAM, es6_p1, adasDomain, rightCam).
switch(sw5, [sw5_p1, sw5_p2, sw5_p3, sw5_p4, ...]).
```

2) Building the Network Knowledge Base: Using the before defined logic programming metamodel, we transform the graphical model instances of the network into the network knowledge base (Prolog facts). A part of the transformation algorithm is demonstrated in Algorithm 1. The graphical model is analyzed to extract all periodic topics in the network dataflow which are transformed into Prolog facts as a part of the whole Network Knowledge Base (NKB). The automated synthesis algorithms use NKB and inference rules to obtain information required for generating stream constraints.

The generated NKB is also a powerful tool for verification

Algorithm 1 Adding periodic topics to the knowledge base

Input: $\mathcal{N} = (N, L, D)$ ▷Network derived from the model
Output: NKB ▷Network knowledge base

1: **for all** $d \in D$ **do**
2: $\nu^{\mathsf{d}} \leftarrow d.getName()$
3: **for all** $t \in \text{topics}(d)$ **do**
4: **if** $periodic(t)$ **then**
5: $\nu^{\mathsf{t}} \leftarrow t.getName(), \tau \leftarrow \text{per}$
6: $\rho \leftarrow t.getPeriod(), \sigma \leftarrow t.getSize()$
7: $f^{\mathsf{t}} \leftarrow (\nu^{\mathsf{d}}, \nu^{\mathsf{t}}, \tau, \rho, \sigma).$
8: $NKB.assertz(f^{\mathsf{t}})$ ▷ Add f^{t} to NKB
return NKB ▷ Returns the updated NKB

of desired network properties. Using adequate queries, network managers and application developers can for example verify that there are at least two disjoint paths between two given network nodes. Such a property is important if there are fail-operational or redundancy requirements in the network. In [42], an example for such a use case is presented.

It can also be used as a simple network database which is used by the network managers or application developers to obtain any information about the network architecture. Such information is helpful when e. g. new applications are developed or the network architecture has to be modified.

3) Inference Rules: We developed a set of inference rules which relates the facts of the NKB and is used to extract information from NKB or to verify e. g. network reliability requirements. For instance, to make the `isLinked` fact symmetric, `link` is defined as:

```
link(P1,P2,BW):- isLinked(P1,P2,BW).
link(P1,P2,BW):- isLinked(P2,P1,BW).
```

To find the involved end-stations, switches, and ports involved in transmission of a topic, `path` is defined:

```
path(Start,End,Path) :- traveling(Start,End,[Start],Temp),
    reverse(Temp,Path).
```

where `traveling` and `reverse` are helper predicates. The rule `streamFinder` finds all streams transporting a given `topic` and extracts all involved ingress and egress ports on the streams' paths. Network streams carry topics through the network. Considering TAS, the egress ports are significant for schedule synthesis of periodic streams. It is important to find out, which periodicity ρ and data length σ a periodic stream has. These parameters are the basis for generating the stream schedule constraints. The definition of `streamFinder` is:

```
streamFinder(Name,Domain,Topic,CL2,EP,IP,Period,Length):-
    firstPort(Domain,Topic,FirstP),lastPort(Domain,Topic,
    LastP),path(FirstP,LastP,PL),removeSwitchDevice(PL,CL1)
    ,removeDevice(CL1,CL2),portClassifier(CL2,IP,EP),
    generateStreamName(Name,Topic,FirstP,LastP),topic(
    Domain,Topic,periodic,Period,Size),Length is Size+64.
```

where `removeSwitchDevice`, `removeDevice`, and `portClassifier` are helper predicates. It checks for a given topic, all communication paths from a publisher of the topic to its consumers including all ingress and egress ports on these paths. Each Ethernet frame has a minimum length of 64 Bytes. To each given pair of two topics, we use `overlappingStreams`:

```
overlappingStreams(N1,D1,T1,P1,L1,N2,D2,T2,P2,L2,SharedEP)
    :- streamFinder(N1,D1,T1,_,EP1,_,P1,L1), streamFinder(
    N2,D2,T2,_,EP2,_,P2,L2), N1\== N2, listIntersect(EP1,
    EP2,SharedEP).
```

This way, we identify non-overlapping streams carrying these topics where `listIntersect` is a developed helper rule to find the intersection of the egress port lists of each stream. Two streams have to be non-overlapping if they share at least one egress port. Using the rule `allOverlappingStreams`:

```
allOverlappingStreams(T1,T2,Out) :- findall({"domain1":
    DStr1,"stream1":N1,"period1":P1,"length1":L1,"domain2":
    DStr2,"stream2":N2,"period2":P2,"length2":L2},(
    overlappingStreams(N1,D1,T1,P1,L1,N2,D2,T2,P2,L2,_),
    atom_string(D1,DStr1),atom_string(D2,DStr2)),Out).
```

we collect all non-overlapping stream pairs in the network. These rules are used in Algorithm 2 to extracted all required input information to generate the constraints.

C. Constraint Generator for TSN Configuration

Using the NKB, schedule synthesis constraints are generated. The schedule of the periodic topics has to be non-overlapping [3]. In Algorithm 2, we describe the procedure of generating the non-overlapping constraints.

The required network facts are obtained using adequate queries on KNB. Each topic f^{t} is always carried by a stream S^{t} from publisher $S^{\mathsf{t}}.pub$ to consumer $S^{\mathsf{t}}.con$ end-stations. Topics can be carried by multiple streams. The stream variable for gate opening time $S^{\mathsf{t}}.tas$ (Line 10) is added to the solver which is desired for the configuration and has to be found by the solver. To calculate the transmission delay, the stream length is divided by the available bandwidth (Line 13). In each network node, processing times are required to analyze the packet headers, etc. The processing delay of each stream depends on the number of hops (Line 16). The interpacket gap delay of a specific stream is calculated using the number of MTU-sized frames of a stream multiplied with the defined interpacket gap α divided by the available bandwidth b (Line 18).

In [3]–[5], the gate opening times of a periodic stream are different for each egress port on the path. In contrast, we sum all relevant delays (Line 19) and keep the gate open for the worst-case delay $S^{\mathsf{t}}.\Phi_{all}$. This way, the $S^{\mathsf{t}}.tas$ is the same for all egress ports on the path. It reduces the number of constraints but also reduces the bandwidth utilization. The gate opening interval for a stream has to fit inside its period (Line 20).

We extract all non-overlapping stream pairs using `allOverlappingStreams` rule (Line 21). The non-overlapping constraints are generated and asserted (Line 30). When all constraints are generated and asserted to the solver context, the solving process starts (Line 31) to find a correct configuration (Line 32) or a proof that the constraints are not satisfiable (Line 34).

We can also use the automated synthesis procedure for timing verification of existing schedules. In this case, network managers or application evaluate use their manually (or using other tools) prepared schedule containing gate opening times for all streams. They use the modeling tool to generate schedule constrains and input the available gate opening times.

Algorithm 2 Periodic topics and non-overlapping constraints

SMT : Solver
$\text{lcm}(x, x')$ ▷ Least Common Multiple of x, x'
α ▷ Interpacket gap
mtu ▷ Maximum transmission unit
Φ_{proc} ▷ Switch processing delay
Φ_{prop} ▷ Propagation delay
S^{t} ▷ Stream carrying topic f^{t}
for all $S^{\mathsf{t}} \in \mathcal{N} = (N, L, D)$ **do**
  $S^{\mathsf{t}}.tas$ ▷ Desired gate opening time variable of S^{t}
  $SMT.assert(S^{\mathsf{t}}.tas)$ ▷ Add variable to the solver
  b ▷ Available bandwidth
  Φ_{trans} ▷ Transmission delay
  $S^{\mathsf{t}}.\Phi_{trans} \leftarrow (\frac{S^{\mathsf{t}}.\sigma}{b})$
  $S^{\mathsf{t}}.pub, S^{\mathsf{t}}.con$ ▷ Publisher and consumer of S^{t}
  $S^{\mathsf{t}}.hops$ ▷ # of hops from $S^{\mathsf{t}}.pub$ to $S^{\mathsf{t}}.con$
  $S^{\mathsf{t}}.\Phi_{proc} \leftarrow \Phi_{proc} S^{\mathsf{t}}.hops$
  $S^{\mathsf{t}}.\Phi_{ipg}$ ▷ Interpacket gap delay
  $S^{\mathsf{t}}.\Phi_{ipg} \leftarrow \left(\left\lceil \frac{S^{\mathsf{t}}.\sigma}{mtu} \right\rceil + 1\right) \cdot \frac{\alpha}{b}$
  $S^{\mathsf{t}}.\Phi_{all} \leftarrow S^{\mathsf{t}}.\Phi_{trans} S^{\mathsf{t}}.hops + S^{\mathsf{t}}.\Phi_{proc} + S^{\mathsf{t}}.\Phi_{ipg} + \Phi_{prop}$
  $SMT.assert\left(S^{\mathsf{t}}.tas \geq 0 \wedge S^{\mathsf{t}}.tas \leq S^{\mathsf{t}}.\rho - S^{\mathsf{t}}.\Phi_{all}\right)$
NOS ▷ List of all non-overlapping stream pairs
for all $pair = \{S_i^{\mathsf{t}}, S_j^{\mathsf{t}}\}, pair \in NOS \wedge S_i^{\mathsf{t}} \neq S_j^{\mathsf{t}}$ **do**
  $\Psi_i \leftarrow \left[0, \frac{\text{lcm}(S_i^{\mathsf{t}}.\rho, S_j^{\mathsf{t}}.\rho))}{S_i^{\mathsf{t}}.\rho} - 1\right]$
  $\Psi_j \leftarrow \left[0, \frac{\text{lcm}(S_i^{\mathsf{t}}.\rho, S_j^{\mathsf{t}}.\rho))}{S_j^{\mathsf{t}}.\rho} - 1\right]$
  for all $\psi_i \in \Psi_i, \psi_j \in \Psi_j$ **do**
    $\gamma \equiv ((\psi_i S_i^{\mathsf{t}}.\rho + S_i^{\mathsf{t}}.tas + S_i^{\mathsf{t}}.\Phi_{all} <$
      $\psi_j S_j^{\mathsf{t}}.\rho + S_j^{\mathsf{t}}.tas) \vee$
      $(\psi_j S_j^{\mathsf{t}}.\rho + S_j^{\mathsf{t}}.tas + S_j^{\mathsf{t}}.\Phi_{all} <$
      $\psi_i S_i^{\mathsf{t}}.\rho + S_i^{\mathsf{t}}.tas))$
    $SMT.assert\,(\gamma)$
if $SMT.check() == SAT$ **then** ▷ If constraints satisfied
  $\mathcal{C} = SMT.model()$ ▷ TAS Configuration
else
  $\mathcal{U} = SMT.unsatCore()$ ▷ Conflicting constraints

The SMT solver tries to satisfy the constraints using the input data. If all constraints are satisfied, the manually generated gate opening times are verified. If the solver is not able to satisfy all constraint, it returns an unsatisfiable core indicating which stream constraints are unsatisfiable. We exploited this feature and demonstrate it in Section IV.

IV. Case Study

The objectives of the case study are to demonstrate: (i) the capability of the graphical modeling tool to synthesize a correct schedule for TAS and (ii) to show how it gives feedback to developers and network managers to modify their model to correct its design problems and achieve a correct synthesized schedule.

We designed an exemplary network model which is presented in Figure 5. Two switches which 8 ports each connect end-stations that are physically separated by the switches.

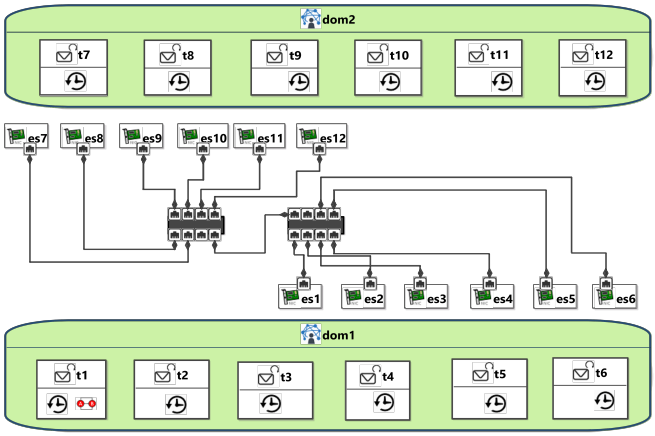

Fig. 5: Topology of the case-study

TABLE I: Stream properties and schedule synthesis results

Stream	Non-overlapping with	Domain	Topic	Publisher	Consumer	Period (ns)	Length (Byte)	First Gate Opening Time (ns)
s_1	s_2	$dom1$	t_1	$es_1\_p_1$	$es_2\_p_1$	500.000	400	38.072
s_2	s_1	$dom1$	t_2	$es_1\_p_1$	$es_2\_p_1$	2.000.000	1000	0
s_3	s_4, s_5, s_6	$dom1$	t_3	$es_2\_p_1$	$es_1\_p_1$	8.000.000	3000	0
s_4	s_3	$dom1$	t_4	$es_2\_p_1$	$es_3\_p_1$	16.000.000	6000	8.103.096
s_5	s_3, s_7, s_8, s_6	$dom2$	t_7	$es_7\_p_1$	$es_1\_p_1$	500.000	400	186.480
s_6	s_{13}, s_3, s_5, s_{12}	$dom2$	t_{12}	$es_{11}\_p_1$	$es_1\_p_1$	100.000.000	5000	7.705.352
s_7	s_8, s_5	$dom2$	t_7	$es_7\_p_1$	$es_8\_p_1$	500.000	400	0
s_8	$s_5, s_7, s_9, s_{10}, s_{12}, s_{13}$	$dom2$	t_7	$es_7\_p_1$	$es_{12}\_p_1$	500.000	400	167.608
s_9	$s_8, s_{14}, s_{10}, s_{11}, s_{12}, s_{13}$	$dom2$	t_8	$es_8\_p_1$	$es_{12}\_p_1$	500.000	400	186.480
s_{10}	$s_8, s_9, s_{11}, s_{12}, s_{13}$	$dom2$	t_9	$es_9\_p_1$	$es_{12}\_p_1$	5.000.000	5000	0
s_{11}	$s_8, s_9, s_{10}, s_{12}, s_{13}$	$dom2$	t_{10}	$es_9\_p_1$	$es_{12}\_p_1$	100.000.000	10000	99.205.352
s_{12}	$s_{11}, s_{10}, s_9, s_8, s_{13}, s_6$	$dom2$	t_{11}	$es_{11}\_p_1$	$es_{12}\_p_1$	100.000.000	5000	205.352
s_{13}	$s_6, s_8, s_9, s_{10}, s_{11}, s_{12}$	$dom2$	t_{12}	$es_{11}\_p_1$	$es_{12}\_p_1$	100.000.000	5000	481.128
s_{14}	s_9	$dom2$	t_8	$es_8\_p_1$	$es_7\_p_1$	500.000	400	0

Two domains are available which describe the communication topics. Using the NKB, we extracted the 14 streams carrying topics through the network. The details about streams, topics, non-overlapping relations of the streams, and the synthesized schedule are available in Table I. The stream's gate opening values are all in a correct range between 0 and the streams' periods. Streams $\{s_2, s_3, s_7, s_{10}, s_{14}\}$ do not have non-overlapping requirements and thus, have the same gate opening time $t = 0$.

In our demonstration setup, we used a desktop PC with an Intel Core i7 CPU and 4GB RAM. For the synthesis, we used the SMT solver Z3 Version 4.5.0[2]. We use Gigabit Ethernet in our models. We exploit the capability of SMT to produce unsatisfiable core *unsatCore*, if no feasible schedule is found. The *unsatCore* in our modeling use-case is a subset of constraints whose conjunction is unsatisfiable. To facilitate backtracking, we assigned an additional boolean variable to each constraint. The name of these Boolean variables are directly related to network stream names. This way, we are able to check which constraints related to which streams are unsatisfiable. To evaluate this approach, we designed three scenarios. In scenario one and two, we add two new streams in the model which we know that they are unsatisfiable together and analyze the output of solver's *unsatCore*. In scenario three, we add five unsatisfiable streams and demonstrate how to correct the model step-by-step using the *unsatCore* results.

[2]https://github.com/Z3Prover/z3

TABLE II: Step-by-step model correction using SMT's unsatisfiable core

New streams	Domain	Topic	Publisher	Consumer	Period (ns)	Length (Byte)
ns_1	$dom1$	t_6	$es_{12}\_p_1$	$es_1\_p_1$	100.000.000	40000
ns_2	$dom1$	t_5	$es_5\_p_1$	$es_8\_p_1$	100.000.000	40000
ns_3	$dom1$	t_1	$es_1\_p_1$	$es_7\_p_1$	500.000	400
ns_4	$dom2$	t_8	$es_8\_p_1$	$es_1\_p_1$	500.000	400
ns_5	$dom2$	t_9	$es_9\_p_1$	$es_1\_p_1$	5.000.000	5000
All constraints		**UNSAT-CORE**		**UNSAT_STREAMS**		
10252		402		ns_2, s_7 from (Table I)		

Model Correction Round 1			
Action	**Resulting UNSAT_STREAMS**	**All constraints**	**UNSAT-CORE**
period of t_5=20.000 ns	s_5 from (Table I), ns_1	5706	102
Model Correction Round 2			
Action	**Resulting UNSAT_STREAMS**	**All constraints**	**UNSAT-CORE**
modify ns_1: $es_{12}\_p_1$ to $es_6\_p_1$ (topic t_6)	s_5, s_7 from (Table I), ns_5	4628	4
Model Correction Round 3			
Action	**Resulting UNSAT_STREAMS**	**All constraints**	**UNSAT-CORE**
route t_5 to $es_4\_p_1$ and connect $es_{12}\_p_1$ to the right switch	No	0	0

A. Scenario 1

As depicted in Table III, two new streams ns_1 and ns_2 are added into the network model (cf. Table I). The length of ns_2 is 30000 Bytes and the period of ns_1 is $250\mu s$. The time which is required to transmit ns_2 is near to the period of ns_1. This combination leads to unsatisfiability. Considering the network model, these streams are isolated from the rest of the network. For this scenario, 4032 constraints were generated and the solver returned 4 streams in the *unsatCore* which are all related to the new streams. This information is helpful to modify the model e. g. reducing the length of ns_2 or increasing the period of ns_1.

B. Scenario 2

In this scenario and similar to scenario 1, we added two unsatisfiable new streams which are not isolated anymore and go through the two switches in the network (to es_7). The results, show that *unsatCore* contains an increased number of 136 unsatisfiable constraints. Interestingly, all of these constraints are directly related to the new added streams, while stream s_{14} also has non-overlapping constraints with the newly added streams. This information is as also very helpful to correct the model. The details of this scenario are presented in Table IV.

C. Scenario 3

In this scenario, we add five new challenging streams to distribute unsatisfiable streams in the network. The details of these streams are available in Table II. We want to show how *unsatCore* helps to correct the model in order to obtain a correct configuration. Two streams ns_1 and ns_2 have a length of 40000 Bytes. High frequent streams ns_3 and ns_4 have a short period of $500\mu s$. The medium-sized stream ns_5 has a length of 5000 Bytes and a period of $5000\mu s$. For synthesis, 5706 constraints are generated. The result of the first execution of the solver returns 102 unsatisfiable constraints all pointing to ns_2 and s_7. In the first correction round, we decreased the length of ns_2 to 20000 Bytes. The next execution of the solver returned the same amount of unsatisfiable constraints but this time pointing to the streams s_5 and ns_1. Because of the big size of ns_1, we modified it such that es_{12} sends data to es_6 instead of to es_1. The results show a significant decrease of unsatisfiable constraints which are related to s_5, the modified ns_1, and s_7. In the final step, we modify the route of t_5 to $es_4\_p_1$ instead of to $es_8\_p_1$ and connect the physical port $es_{12}\_p_1$ to the right switch avoiding compatition of t_7 and t_6. The last execution of the solver returned a correct schedule. The details of the model correction steps are demonstrated in Table II.

TABLE III: Isolated non-schedulable streams

New streams	Domain	Topic	Publisher	Consumer	Period (ns)	Length (Byte)
ns_1	$dom1$	t_5	$es_4\_p_1$	$es_6\_p_1$	250.000	1500
ns_2	$dom1$	t_6	$es_5\_p_1$	$es_6\_p_1$	100.000.000	30000
All constraints		**UNSAT-CORE**		**UNSAT_STREAMS**		
4032		4		ns_1, ns_2		

TABLE IV: Cross-domain non-schedulable streams

New streams	Domain	Topic	Publisher	Consumer	Period (ns)	Length (Byte)
ns_1	$dom1$	t_5	es_4_p1	$es_7\_p_1$	250.000	1500
ns_2	$dom1$	t_6	$es_5\_p_1$	$es_7\_p_1$	100.000.000	30000
All constraints		**UNSAT-CORE**		**UNSAT_STREAMS**		
4444		136		ns_1, ns_2		

D. Discussion of scalability

Because the presented synthesis problem is NP-complete, it is necessary to mention the scalability limitations. We observed that synthesis time increases when (i) the number of streams increases, (ii) the difference of streams' periods increase (i.e., streams with long period length compete with streams having short period length) and leads to expansion of search space for a feasible schedule, and (iii) the stream's length increases and shrinks the solution space. The initial observations are compatible with the findings in [3], [4]. We observe that for up to 100 streams, the synthesis time remains short (few seconds to 4 minutes). Adding more streams, this time increases exponentially. We plan to extend our test cases to achieve an empirical evaluation. The problem of long synthesis times is discussed in [43].

V. Conclusion

To automate the procedure of time-triggered schedule synthesis for Time-Sensitive Networking, a novel graphical modeling tool is introduced which exploits object-oriented modeling, logic programming, and Satisfiability Modulo Theories. However, it is possible that the generated network models are non-schedulable. We used the feature of generating unsatisfiable cores in SMT solvers, to construct helpful feedback to correct non-schedulable network models step-by-step to find a feasible schedule. We only considered the non-overlapping requirements of time-triggered streams in this paper.

In a future work, we plan to extend the modeling tool in order to increase the precision of the models by covering more network details such as application level constraints. Moreover, we plan to formulate and generate constraints for peristaltic shaper of TSN (described in *IEEE 802.1Qch – Cyclic Queuing and Forwarding*) using our graphical modeling tool. In this case, only Algorithm 2 has to be modified and the rest of the framework can be used without modifications.

REFERENCES

[1] Time-sensitive networking. (Date last accessed 10-April-2017). [Online]. Available: http://www.ieee802.org/1/pages/tsn.html

[2] S. S. Craciunas and R. S. Oliver, "Combined task- and network-level scheduling for distributed time-triggered systems," *Real-Time Systems*, vol. 52, no. 2, pp. 161–200, 2016.

[3] W. Steiner, "An evaluation of smt-based schedule synthesis for time-triggered multi-hop networks," in *31st IEEE Real-Time Systems Symposium*, 2010, pp. 375–384.

[4] S. S. Craciunas, R. S. Oliver, M. Chmelík, and W. Steiner, "Scheduling real-time communication in ieee 802.1qbv time sensitive networks," in *Proc. of the 24th International Conference on Real-Time Networks and Systems*, ser. RTNS '16. ACM, 2016, pp. 183–192.

[5] L. Zhang, D. Goswami, R. Schneider, and S. Chakraborty, "Task- and network-level schedule co-synthesis of ethernet-based time-triggered systems," in *19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2014, pp. 119–124.

[6] Data distribution service. (Date last accessed 10-April-2017). [Online]. Available: http://www.omg.org/spec/DDS/

[7] C. Buckl, M. Geisinger, D. Gulati, F. J. Ruiz-Bertol, and A. Knoll, "Chromosome: a run-time environment for plug & play-capable embedded real-time systems," *ACM SIGBED Review*, vol. 11, no. 3, pp. 36–39, 2014.

[8] H. Gomaa, "Designing concurrent, distributed, and real-time applications with uml," in *IEEE Proc. of the 23rd International Conference on Software Engineering*, ser. ICSE '01, 2001, pp. 737–738.

[9] J. Rumbaugh, I. Jacobson, and G. Booch, *Unified Modeling Language Reference Manual, The (2Nd Edition)*. Pearson Higher Education, 2004.

[10] F. Huber, B. Schätz, A. Schmidt, and K. Spies, *AutoFocus — A tool for distributed systems specification*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1996, pp. 467–470.

[11] M. H. Farzaneh, S. Feldmann, C. Legat, J. Folmer, and B. Vogel-Heuser, "Modeling multicore programmable logic controllers in networked automation systems," in *IECON 2013 - 39th Annual Conference of the IEEE Industrial Electronics Society*, 2013, pp. 4398–4403.

[12] Stateflow. (Date last accessed 10-April-2017). [Online]. Available: https://de.mathworks.com/products/stateflow.html/

[13] T. Weilkiens, *Systems engineering with SysML/UML: modeling, analysis, design*. Morgan Kaufmann, 2011.

[14] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the ptolemy approach," *Proc. of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[15] F. Mallet and R. de Simone, "Marte: A profile for rt/e systems modeling, analysis–and simulation?" in *Proc. of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops*, ser. Simutools '08, 2008, pp. 43:1–43:8.

[16] K. G. Larsen, P. Pettersson, and W. Yi, "Uppaal in a nutshell," *International Journal on Software Tools for Technology Transfer*, vol. 1, no. 1, pp. 134–152, 1997.

[17] N. Saeedloei and G. Gupta, "A logic-based modeling and verification of cps," *SIGBED Rev.*, vol. 8, no. 2, pp. 31–34, 2011.

[18] G. Gupta and E. Pontelli, "A constraint-based approach for specification and verification of real-time systems," in *Proc. Real-Time Systems Symposium*, 1997, pp. 230–239.

[19] N. Saeedloei and G. Gupta, "A methodology for modeling and verification of cyber-physical systems based on logic programming," *SIGBED Rev.*, vol. 13, no. 2, pp. 34–42, 2016.

[20] B. T. Loo, T. Condie, M. Garofalakis, D. E. Gay, J. M. Hellerstein, P. Maniatis, R. Ramakrishnan, T. Roscoe, and I. Stoica, "Declarative networking," *Commun. ACM*, vol. 52, no. 11, pp. 87–95, 2009.

[21] A. Wang, P. Basu, B. T. Loo, and O. Sokolsky, "Declarative network verification," in *International Symposium on Practical Aspects of Declarative Languages*. Springer, 2009, pp. 61–75.

[22] T. L. Hinrichs, N. S. Gude, M. Casado, J. C. Mitchell, and S. Shenker, "Practical declarative network management," in *Proc. of the 1st ACM Workshop on Research on Enterprise Networking*, ser. WREN '09. New York, NY, USA: ACM, 2009.

[23] N. P. Lopes, J. A. Navarro, A. Rybalchenko, and A. Singh, "Applying prolog to develop distributed systems," *Theory and Practice of Logic Programming*, vol. 10, no. 4-6, pp. 691–707, 2010.

[24] A. Guerrero, V. A. Villagrá, J. E. L. De Vergara, and J. Berrocal, "Ontology-based integration of management behaviour and information definitions using swrl and owl," in *Ambient Networks*. Springer, 2005, pp. 12–23.

[25] S. Van der Meer, B. Jennings, D. O'Sullivan, D. Lewis, and N. Agoulmine, "Ontology based policy mobility for pervasive computing," 2005.

[26] J. Keeney, D. Lewis, D. O'Sullivan, A. Roelens, V. Wade, A. Boran, and R. Richardson, "Runtime semantic interoperability for gathering ontology-based network context," in *10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006, pp. 56–65.

[27] P. Ray, N. Parameswaran, J. Strassner *et al.*, "Ontology mapping for the interoperability problem in network management," *IEEE Journal on Selected Areas in Communications*, vol. 23, no. 10, pp. 2058–2068, 2005.

[28] J. E. L. De Vergara, A. Guerrero, V. A. Villagrá, and J. Berrocal, "Ontology-based network management: study cases and lessons learned," *Journal of Network and Systems Management*, vol. 17, no. 3, pp. 234–254, 2009.

[29] M. H. Farzaneh and A. Knoll, "An ontology-based plug-and-play approach for in-vehicle time-sensitive networking (tsn)," in *IEEE 7th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)*, 2016.

[30] A. K. Bandara, A. Kakas, E. C. Lupu, and A. Russo, "Using argumentation logic for firewall policy specification and analysis," in *Large Scale Management of Distributed Systems*. Springer, 2006, pp. 185–196.

[31] T. Klie, F. Gebhard, and S. Fischer, "Towards automatic composition of network management web services," in *10th IFIP/IEEE International Symposium on Integrated Network Management*, 2007, pp. 769–772.

[32] A. Martinez, M. Yannuzzi, J. Lopez de Vergara, R. Serral-Gracia, and W. Ramirez, "An ontology-based information extraction system for bridging the configuration gap in hybrid sdn environments," in *IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2015, pp. 441–449.

[33] N. F. Noy, M. Sintek, S. Decker, M. Crubézy, R. W. Fergerson, and M. A. Musen, "Creating semantic web contents with protege-2000," *IEEE intelligent systems*, vol. 16, no. 2, pp. 60–71, 2001.

[34] H. T. Lim, D. Herrscher, and F. Chaari, "Performance comparison of ieee 802.1q and ieee 802.1 avb in an ethernet-based in-vehicle network," in *8th International Conference on Computing Technology and Information Management (ICCM)*, vol. 1, 2012.

[35] F. Reimann, S. Graf, F. Streit, M. Gla, and J. Teich, "Timing analysis of ethernet avb-based automotive e/e architectures," in *IEEE 18th Conference on Emerging Technologies Factory Automation (ETFA)*, 2013.

[36] G. Alderisi, A. Caltabiano, G. Vasta, G. Iannizzotto, T. Steinbach, and L. L. Bello, "Simulative assessments of ieee 802.1 ethernet avb and time-triggered ethernet for advanced driver assistance systems and in-car infotainment," in *IEEE Vehicular Networking Conference (VNC)*, 2012, pp. 187–194.

[37] P. Meyer, T. Steinbach, F. Korf, and T. C. Schmidt, "Extending ieee 802.1 avb with time-triggered scheduling: A simulation study of the coexistence of synchronous and asynchronous traffic," in *IEEE Vehicular Networking Conference*, 2013, pp. 47–54.

[38] T. Steinbach, H. T. Lim, F. Korf, T. C. Schmidt, D. Herrscher, and A. Wolisz, "Beware of the hidden! how cross-traffic affects quality assurances of competing real-time ethernet standards for in-car communication," in *IEEE 40th Conference on Local Computer Networks (LCN)*, 2015.

[39] S. Thangamuthu, N. Concer, P. J. L. Cuijpers, and J. J. Lukkien, "Analysis of ethernet-switch traffic shapers for in-vehicle networking applications," in *Design, Automation Test in Europe Conference Exhibition (DATE)*, 2015, pp. 55–60.

[40] D. Thiele, R. Ernst, and J. Diemer, "Formal worst-case timing analysis of ethernet tsn's time-aware and peristaltic shapers," in *IEEE Vehicular Networking Conference (VNC)*, 2015, pp. 251–258.

[41] D. Thiele and R. Ernst, "Formal worst-case performance analysis of time-sensitive ethernet with frame preemption," in *IEEE 21st International Conference on Emerging Technologies and Factory Automation (ETFA)*, 2016.

[42] M. H. Farzaneh, S. Shafaei, and A. Knoll, "Formally verifiable modeling of in-vehicle time-sensitive networks (tsn) based on logic programming," in *IEEE Vehicular Networking Conference (VNC)*, 2016.

[43] F. Pozo, G. Rodriguez-Navas, H. Hansson, and W. Steiner, "Smt-based synthesis of ttethernet schedules: A performance study," in *International Symposium on Industrial Embedded Systems (SIES)*. IEEE, 2015, pp. 1–4.