



TECHNISCHE
UNIVERSITÄT
DARMSTADT

Correct-by-Construction Development of Dynamic Topology Control Algorithms

DEM FACHBEREICH ELEKTROTECHNIK UND INFORMATIONSTECHNIK
DER TECHNISCHEN UNIVERSITÄT DARMSTADT
ZUR ERLANGUNG DES AKADEMISCHEN GRADES
EINES DOKTOR-INGENIEURS (DR.-ING.)
GENEHMIGTE DISSERTATION

VON

ROLAND SPEITH (GEB. KLUGE), M.SC.

GEBOREN AM

16. NOVEMBER 1989 IN GIESSEN, HESSEN

REFERENT: PROF. DR. RER. NAT. ANDREAS SCHÜRR
KORREFERENT: PROF. DR. RER. NAT. HOLGER GIESE

TAG DER EINREICHUNG: 2018-11-30

TAG DER DISPUTATION: 2019-03-14

D17

DARMSTADT 2019

The work of Roland Speith was supported by the Corporate Research Center (CRC) 1053 *Multi-Mechanism-Adaptation für das künftige Internet (MAKI)* of the *Deutsche Forschungsgemeinschaft (DFG)* (<https://www.maki.tu-darmstadt.de>).

Speith, Roland

Correct-by-Construction Development of Dynamic Topology Control Algorithms
Darmstadt, Technische Universität Darmstadt

Year of publication at TUprints: 2019

URN: urn:nbn:de:tuda-tuprints-85627

URL: <https://tuprints.ulb.tu-darmstadt.de/id/eprint/8562>

Disputation date: 2019-03-14

Published under CC BY-SA 4.0 International

<https://creativecommons.org/licenses/>

©2019, Roland Speith

ABSTRACT

Wireless devices are influencing our everyday lives today and will even more so in the future. A wireless sensor network (WSN) consists of dozens to hundreds of small, cheap, battery-powered, resource-constrained sensor devices (motes) that cooperate to serve a common purpose. These networks are applied in safety- and security-critical areas (e.g., e-health, intrusion detection). The topology of such a system is an attributed graph consisting of nodes representing the devices and edges representing the communication links between devices. Topology control (TC) improves the energy consumption behavior of a WSN by blocking costly links. This allows a mote to reduce its transmission power. A TC algorithm must fulfill important consistency properties (e.g., that the resulting topology is connected).

The traditional development process for TC algorithms only considers consistency properties during the initial specification phase. The actual implementation is carried out manually, which is error prone and time consuming. Thus, it is difficult to verify that the implementation fulfills the required consistency properties. The problem becomes even more severe if the development process is iterative. Additionally, many TC algorithms are batch algorithms, which process the entire topology, irrespective of the extent of the topology modifications since the last execution. Therefore, dynamic TC is desirable, which reacts to change events of the topology.

In this thesis, we propose a model-driven correct-by-construction methodology for developing dynamic TC algorithms. We model local consistency properties using graph constraints and global consistency properties using second-order logic. Graph transformation rules capture the different types of topology modifications. To specify the control flow of a TC algorithm, we employ the programmed graph transformation language story-driven modeling. We presume that local consistency properties jointly imply the global consistency properties. We ensure the fulfillment of the local consistency properties by synthesizing weakest preconditions for each rule. The synthesized preconditions prohibit the application of a rule if and only if the application would lead to a violation of a consistency property. Still, this restriction is infeasible for topology modifications that need to be executed in any case. Therefore, as a major contribution of this thesis, we propose the anticipation loop synthesis algorithm, which transforms the synthesized preconditions into routines that anticipate all violations of these preconditions. This algorithm also enables the correct-by-construction runtime reconfiguration of adaptive WSNs. We provide tooling for both common evaluation steps. COBOLT allows to evaluate the specified TC algorithms rapidly using the network simulator SIMONSTRATOR. cMOFLON generates embedded C code for hardware testbeds that build on the sensor operating system CONTIKI.

ZUSAMMENFASSUNG

Drahtlos kommunizierende Geräte sind ein wesentlicher Bestandteil der zunehmenden Digitalisierung unserer Gesellschaft. Ein drahtloses Sensornetzwerk (engl. Wireless Sensor Network, WSN) umfasst eine Vielzahl günstiger, kleiner, batteriebetriebener Sensoren, die gemeinschaftlich einem Ziel dienen (z. B. in einer e-Health-Anwendung). Die Topologie eines WSNs ist ein attributierter Graph, dessen Knoten die Geräte und dessen Kanten deren Kommunikationsverbindungen darstellen. Topologiekontrolle (engl. Topology Control, TC) reduziert den Energieverbrauch eines WSN, indem bestimmte physische Nachbarn eines Geräts ausgeblendet werden, wodurch sich dessen Sendeleistung reduzieren lässt. Der Topologiekontrollalgorithmus muss dabei wesentliche Konsistenzeigenschaften erhalten (z. B. den Zusammenhang des Topologie).

Ungeachtet ihrer großen Bedeutung werden Konsistenzeigenschaften in der traditionellen Entwicklung von TC-Algorithmen nur in der anfänglichen Spezifikationsphase betrachtet. Die anschließende Implementierung des TC-Algorithmus wird von Hand durchgeführt und ist damit fehleranfällig und zeitaufwändig. Es ist schwierig nachzuweisen, dass die Implementierung die geforderten Konsistenzeigenschaften erfüllt, insbesondere da TC-Algorithmen typischerweise iterativ entwickelt werden. Die bestehenden Batch-TC-Algorithmen verarbeiten jeweils die gesamte Topologie ohne Wissen aus vorherigen Ausführungen zu nutzen. Daher ist es wünschenswert dynamische TC-Algorithmen zu entwickeln, die inkrementell auf Ereignisse reagieren.

In dieser Arbeit stellen wir eine modellbasierte Entwicklungsmethodik für garantiert korrekte dynamische TC-Algorithmen vor. Wir modellieren lokale Konsistenzeigenschaften mittels Graph-Constraints, globale Konsistenzeigenschaften mittels Prädikatenlogik zweiter Stufe sowie die möglichen Änderungen der Topologie und deren Kontrollfluss mittels programmierter Graphtransformationsregeln. Unter der Annahme, dass die lokalen Konsistenzbedingungen gemeinsam die globalen Konsistenzbedingungen implizieren, können wir mithilfe bestehender statischer Analysetechniken Vorbedingungen für die Graphtransformationsregeln synthetisieren, die sicherstellen, dass die modifizierten Regeln die Konsistenzbedingungen erhalten. Die resultierende Einschränkung ist für diejenigen Regeln, die das Umweltverhalten beschreiben, jedoch nicht zulässig. Wir stellen daher einen Algorithmus vor, der jede synthetisierte Vorbedingung systematisch in eine Antizipationsschleife überführt, welche alle unvermeidlichen Verletzungen der entsprechenden Vorbedingungen auflöst. Wir zeigen, dass sich dieser neuartige Algorithmus auch für die Rekonfiguration von adaptiven WSNs eignet. Um dem Nutzer unseres Ansatzes eine schnelle Erprobung eines entwickelten TC-Algorithmus zu ermöglichen, bieten wir Werkzeuge zur Evaluation im Netzwerksimulator (COBOLT) und Hardware-Testbed (cMOFLON) an.

TABLE OF CONTENTS

1	THE NEED FOR CORRECT-BY-CONSTRUCTION TOPOLOGY CONTROL	1
1.1	Challenges in wireless communication systems engineering	3
1.2	Goals	10
1.3	Contributions and thesis structure	11
1.4	Publications and supervised theses	14
1.5	Technical remarks	16
2	FUNDAMENTALS OF TOPOLOGY CONTROL	17
2.1	Graph theory	18
2.2	Network topologies	21
2.3	Performance of topologies	26
2.4	Topology control	27
2.5	Dynamic topology control	34
3	SELECTION OF SPECIFICATION LANGUAGES	45
3.1	Metamodeling	48
3.2	Local consistency properties	56
3.3	Global consistency properties	67
3.4	Elementary topology modifications	70
3.5	Execution order of topology modifications	76
3.6	Configuration space specification	81
3.7	Related work	84
4	SYNTHESIS OF CORRECT TOPOLOGY CONTROL MECHANISMS	93
4.1	Consistency preservation and violation	95
4.2	Constructive approach	102
4.3	Application of constructive approach to running example	106
4.4	Synthesis of anticipation loops	130
4.5	Applicability of anticipation loop synthesis algorithm	147
4.6	Related work	160
5	TOOL SUPPORT FOR RAPID EVALUATION	169
5.1	Selection of modeling tool	172
5.2	Model-based simulative evaluation with Cobolt	178
5.3	Model-based testbed evaluation with cMoflon	204
5.4	Related work	235

6	DEVELOPMENT OF FAMILIES OF TOPOLOGY CONTROL ALGORITHMS	239
6.1	Landscape of topology control algorithms	241
6.2	Specification of topology control algorithms	248
6.3	Specification of families of topology control algorithms	260
6.4	Proving global consistency properties	264
6.5	Discussion of applicability	267
6.6	Related work	273
7	CONCLUSION	275
7.1	Summary	276
7.2	Outlook	278
	BIBLIOGRAPHY	283
Appendix A	LIST OF FIGURES	316
Appendix B	LIST OF TABLES	320
Appendix C	LIST OF ALGORITHMS	321
Appendix D	LIST OF EXAMPLES	322
Appendix E	LIST OF THEOREMS	324
Appendix F	TABLE OF CONTENTS (DETAILED)	325

LIST OF ABBREVIATIONS

AC	Application condition
CMOF	Complete Meta-Object Facility
DSPL	Dynamic software product line
EMF	Eclipse Modeling Framework
EMOF	Essential Meta-Object Facility
IoT	Internet of Things
IQR	Inter-quartile range
LMST	Local minimum spanning tree
MDA	Model-Driven Architecture
MDE	Model-driven engineering
MOF	Meta-Object Facility
NFP	Nonfunctional property
OCL	Object Constraint Language
OMG	Object Management Group
NAC	Negative application condition
PAC	Positive application condition
SDM	Story-driven modeling
SPL	Software product line
TGG	Triple graph grammar
TC	Topology control
UDG	Unit disk graph
WSN	Wireless sensor network

THE NEED FOR CORRECT-BY-CONSTRUCTION TOPOLOGY CONTROL ALGORITHMS

Our society is currently undergoing a rapid digitalization with more and more electronic devices permeating our everyday life. The following examples illustrate that wireless communication systems are one cornerstone of this process.

- The Internet of Things (IoT) consists of numerous everyday objects that are connected to the Internet and among each other [273]. In 2016, the IoT consisted of 6 to 17 billion devices, depending on which device types are considered (e.g., smartphones, tablets) [170]. Wiring IoT devices is infeasible because these devices are usually mobile (e.g., smartphones, smartwatches, body area networks, drones). Often, multiple IoT devices cooperate locally to produce meaningful data.
- To create smart factories [265, 266], production companies equip factories with dense sensor networks to monitor each production step closely. This trend is also called Industry 4.0 [134]. Formerly, industrial sensor networks were wired, leading to high deployment and operational costs, error-prone installation, and low flexibility. For the prospected degree of digitalization in smart factories, only wireless technology is affordable [126]. The extent of certain types of industrial plants (e.g., solar power plants) requires that wireless sensor devices cooperatively collect and forward data toward a central data sink device. Modern agriculture also benefits from monitoring environmental parameters using wireless sensor networks (e.g., humidity, temperature) [203].
- An intelligent transportation system adapts itself to cope with traffic loads that change periodically throughout the day and due to accidents and large-scale public events [169]. Traditional traffic management systems are limited to counting vehicles (e.g., using induction loops). Wireless communication systems offer additional benefits [155]. Warning messages about traffic jams, accidents, or construction sites are broadcast regionally already today to avoid rear-end collisions [169]. In platooning, multiple vehicles (often trucks) form a convoy and drive autonomously at close distance (ca. 10 m to 15 m) to reduce drag [261]. The resulting safety demands require that the vehicles of a platoon communicate reliably and with a small guaranteed latency.

- Wireless technology saves lives. Wireless sensors are employed for (flash-)flood [32, 97], wildfire [148], or landslide warning systems [202], pollution detection [226], volcano monitoring [178], structural health monitoring of buildings and bridges [115, 116], or in e-health applications (e.g., to detect whether a person has fallen) [5]. We may expect that wireless devices will be used more extensively in these scenarios in the future.
- Wireless devices open new leisure opportunities. Multi-player augmented-reality games are becoming increasingly popular (e.g., Ingress¹, Pokémon Go²). In these games, players interact primarily via the Internet, which entails a conceivable latency if the mobile infrastructure is congested. Novel approaches enhance the infrastructure-based communication with additional ad-hoc communication channels, which allow users to communicate with fellow players that are in the geographic vicinity [207].
- Wireless sensors are used for wildlife and habitat monitoring (e.g., to protect natural reserves against intruders) [50, 154]. Here, wireless technology simplifies tasks for humans (e.g., counting or tracking animals) and avoids that animals are disturbed unnecessarily (e.g., during breeding season). The monitored area may be too large to cover it with radio towers, requiring collaboration among the sensor nodes to collect and forward data to a base station.

In all described examples, wireless mobile devices cooperate to achieve a common goal. Wireless sensors (also called motes) form a large class of such devices. A wireless sensor network (WSN) consists of a large number (i.e., dozens to thousands) of small, cheap, battery-powered, resource-constrained wireless sensor devices that collaboratively provide services [218]. Examples are periodic data collection (e.g., seismic vibrations for volcano monitoring) or event detection (e.g., a fallen person in an e-health application) [219, 267]. Kahn and Katz coined the term “Smart Dust” for motes because of their small size, low price, and flexible interconnection via standardized protocols [106]. WSNs serve as running example of this thesis due to their importance in research and society.

The underlay topology of a WSN is a graph that reflects the physical interconnections among the motes. A node of the underlay topology represents a mote, and an edge between two nodes states that a message can be transferred directly from the source to the target mote. The overlay topology reflects the application-specific interconnections of motes. For instance, a mote that transmits data to a dedicated central base station establishes a single link in the overlay topology, which usually corresponds to a path of underlay links. In WSNs, multi-hop routing is common because the transmission range of a single mote is often too small to reach the base station directly.

¹ Ingress page: <https://www.ingress.com/> (visited: 2018-09-17)

² Pokémon Go page: <https://www.pokemongo.com/> (visited: 2018-09-17)

1.1 CHALLENGES IN WIRELESS COMMUNICATION SYSTEMS ENGINEERING

Wireless networks (and especially WSNs) face several challenges due to the expected density of wireless devices in the Future Internet. In the following, we outline a subset of these challenges that is relevant for this thesis.

Performance goals

The first challenge is that scenario-specific performance goals need to be considered. For instance, if multiple wireless devices start transmitting concurrently, the individual messages may interfere and retransmissions of the same message become necessary, which leads to increased end-to-end latency and energy consumption. This problem even occurs if the motes wait for a free sending slot prior to transmitting (also called the hidden-terminal problem [200]). In fact, reducing the energy consumption is one of the most important performance goals for WSNs because, depending on the application scenario, recharging a mote is difficult, dangerous, costly, or impossible.

Topology control (TC) is an established approach to improve both the interference in and the energy consumption of a WSN [218, 219]. The idea of TC is to reduce the size of the neighborhood of each mote in the underlay topology. This sparser topology can then be used to lower the transmission power of the mote. Technically, a TC algorithm builds a virtual topology on top of the input topology. The virtual topology is a view of the input topology that contains only well-selected links. If TC is active, all network mechanisms that usually use the (unfiltered) input topology now operate on the virtual topology. A downside of this approach is that a message is often routed across more hops to reach its destination in the virtual topology compared to the input topology. The extended routing path length results in an increased end-to-end packet drop rate, latency, and risk of overloading relay motes. Dozens of TC algorithms have been proposed during the last decades (summarized, e.g., in [219, 264, 267]), each focusing on certain performance goals (e.g., minimizing energy consumption, minimizing latency, bounding path latency, ensuring fairness of resource consumption across all motes). With the advent of mm-wave communication, which requires directional instead of omni-directional antennas, additional performance goals arose (e.g., limited mote degree to reduce the computation complexity [234]). An ongoing debate is whether underlay and overlay topologies should be considered separately or jointly when engineering TC algorithms. A separate, application-agnostic approach fosters reuse, whereas a joint approach could leverage application-specific knowledge (e.g., routing information).

Example 1.1 (Underlay, virtual, and overlay topology). Figure 1.1 summarizes the concepts of underlay, virtual, and overlay topologies using a small WSN that consists of four motes and one base station. The motes are TELOSB devices^a [181]. This mote type is frequently used for testbed experiments in WSN research. The under-

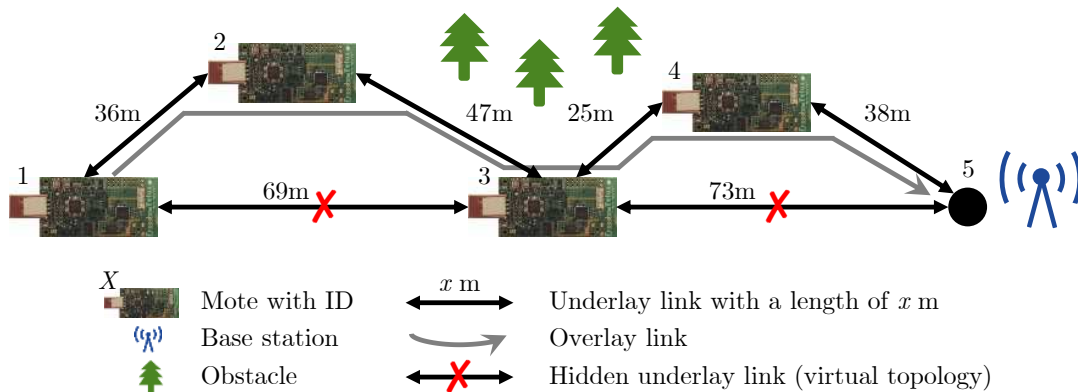


Figure 1.1: Example: WSN with underlay, virtual, and overlay topology

lay consists of two triangles of links and is shown as undirected black arrows, which are annotated with the lengths of the links. The tree-shaped icons in the center of the figure symbolize an obstacle that prevents motes 2 and 4 from communicating directly. Mote 1 is currently transmitting data to the base station, as indicated by the gray overlay link.

This example also shows the result of executing the TC algorithm kTC [227]. In general, kTC hides a link from the virtual topology if this link has the largest weight in a triangle of links and if its weight is at least k times larger than the minimal weight in this triangle. The parameter k is configurable and set to 1.3 in this example. Let's assume that the weight of a link equals its length. Then, kTC hides the link between motes 1 and 3 and the link between mote 3 and the base station from the virtual topology. Therefore, the path of underlay links that corresponds to the overlay link uses the relay motes 2, 3, and 4.

^a Image CC BY-SA 3.0 <https://commons.wikimedia.org/wiki/File:TelosB.jpg> (visited: 2018-11-07)

Dynamic topologies

The second research challenge is that traditional approaches assume that WSNs are static structures, which was known to be a false assumption already more than ten years ago [264, 277]. In fact, the topology of most WSNs changes over time due to environmental influences. For example, even if each mote is located at a fixed position, an obstacle may move between two motes leading to a disruption or a degradation of the quality of their connection. Furthermore, battery depletion and mote failures lead to disappearing motes, and the replacement of failed devices leads to additional motes in the topology. A TC algorithm should react dynamically to such context events. Ideally, this reaction is incremental and reprocesses only the relevant parts of the topology [232].

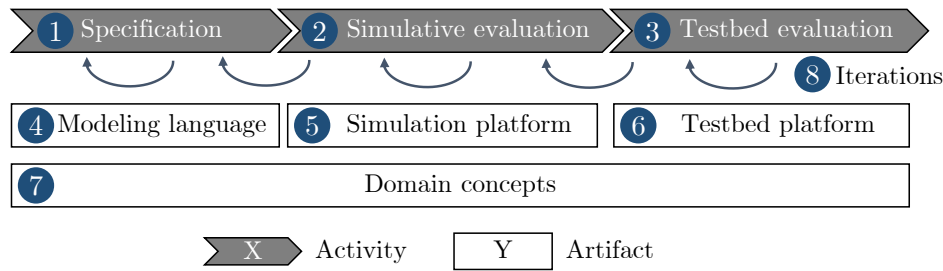


Figure 1.2: Overview of development phases of a TC algorithm

Consistency properties

The third research challenge is that, due to the safety-critical application scenarios of WSNs, it is important that a TC algorithm fulfills certain consistency properties. Two established consistency properties are the preservation of connectivity and coverage. Any TC algorithm must preserve connectivity, which means that the virtual topology is connected as long as the input topology is connected. Coverage applies to WSNs whose nodes can be disabled temporarily to save energy. In such WSNs, each node observes a small geographic region, and the observed regions of all devices shall jointly fulfill a coverage criterion (e.g., cover the boundary of a certain area).

A TC algorithm that violates consistency properties may cause dangerous situations. In an e-health system, a loss of connectivity may prevent the system to detect that a person has fallen because measurements are lost. Similarly, the inhabitants of a valley may be warned too late (or not at all) if information about an approaching avalanche is lost due to a disconnected topology. In a wildlife reserve that shall be protected from poachers by a WSN, a loss of coverage may endanger animals of a threatened species.

We shall discuss the necessary extensions to model coverage later. For now, we concentrate on the preservation of connectivity due to its fundamental role in almost all application scenarios of WSNs.

Tedious, error-prone development process

As a fourth major research challenge, the traditional development of TC algorithms is a tedious and error-prone manual process. As in many disciplines in communication systems engineering, the development of a new TC algorithm consists of two major tasks: specification and evaluation. In the following, we describe the individual steps of these tasks and highlight current shortcomings (Figure 1.2).

During the specification phase (see ① in Figure 1.2), relevant domain concepts (e.g., input, output, virtual topology and consistency properties, see ⑦) and the TC algorithm are specified using a formal modeling language (e.g., graph theory or game theory, see ④). Modeling languages are always selected to serve a particular purpose (e.g.,

to prove the correctness w.r.t. consistency properties). We are interested in modeling structural modifications of the WSN topology.

TC algorithms form two major groups: centralized and localized TC algorithms [267, Sec. 1.2]. A centralized TC algorithm is executed as a single process based on a global view of the topology. A localized TC algorithm is executed concurrently on multiple motes based on a limited per-mote local view of the topology. A typical local view in WSNs consists of all motes and links that are at most two hops away. A hop is the traversal of a single link [235].

Centralized algorithms have access to a larger scope of knowledge, which may lead to better decisions w.r.t. the desired performance goals. In comparison to centralized algorithms, localized algorithms tend to be more robust and scalable, at the expense of possibly less informed results due to their limited local knowledge. Also, the first version of a localized TC algorithm may be developed based on global knowledge for conducting initial simulation experiments. Our observation is that localized TC algorithms are more prominent in the literature than centralized TC algorithms [219, 264, 267]. One reason is that today's motes are still resource constrained w.r.t. processing power and working memory. This limits the computation complexity of the TC algorithm and the size of the local view that can be stored on a single mote. A second reason is that collecting, storing, and updating a global view of the topology on one or multiple motes entails a potentially large amount of protocol messages. Additional protocol messages are required to communicate the decisions of the centralized algorithm to all affected motes. These protocol messages reduce the available bandwidth for the actual applications running in the WSN.

The centralized and the localized perspective can be combined if each mote can be configured to have dedicated responsibilities for monitoring and updating the topology. A mote may only monitor its outgoing links because a link in the underlay topology is defined as a possible physical connection from a source mote to a target mote. Therefore, motes need to cooperate if a larger local view needs to be maintained. Each mote emits neighborhood messages (e.g., periodically), which contain the current local view of the mote. In general, if a K -hop local view is required on each mote, the neighborhood messages must contain a $(K-1)$ -hop local view.

In a localized TC algorithm, the neighborhood messages must contain enough information to build a local view that is sufficient to find the desired structural patterns of the active TC algorithm. For example, in case of k TC, each mote sends the set of its outgoing links (i.e., its 1-hop neighborhood) to all its neighbors because this information allows each mote to find out whether one of its outgoing links is part of a triangle that fulfills the k TC condition, which requires a 2-hop local view. In WSNs, each mote is responsible for updating its outgoing links because hiding (or unhiding, resp.) a link is often realized as adding (removing, resp.) the target mote of the link to (from, resp.) an internal blacklist. This operation is only possible on the source mote of the link.

In a centralized TC algorithm, all motes transmit their observed outgoing links to a selected central mote that builds a global view of the topology and centrally selects the links that shall be hidden from the virtual topology. The selection decisions are then broadcast to all motes in the network. Therefore, a centralized TC algorithm is capable of constructing, for instance, a global minimal spanning tree, which is not possible using K -hop local knowledge only (for a fixed K and arbitrary network sizes).

Example 1.2 (Local monitoring and updating responsibilities). Figure 1.3 illustrates the 2-hop local views of the motes with IDs 1 and 3 in the sample topology that is shown in Figure 1.1. Each local view is framed with a rectangle and decorated with the mote that owns the local view. In addition to the global view in Figure 1.1, the links in the local views in Figure 1.3 are decorated with a unit-less metric to indicate that, usually, the distance between motes is not available, but only some correlated metric (e.g., the received signal strength indicator (RSSI)).

Regarding the monitoring responsibility, the links that a mote can detect locally are labeled with «O» (e.g., the links from mote 1 to motes 2 and 3 in the local view of mote 1). All links that originate from neighborhood messages are shown without stereotype. For example, in the local view of mote 1, the links from mote 2 to motes 1 and 3 originate from a neighborhood message from mote 2.

Regarding the updating responsibility, each mote decides which of its outgoing links to hide from the virtual topology (e.g., the link from mote 1 to mote 3 in the local view of mote 1).

In this thesis, we characterize the correctness of a TC algorithm in terms of information that is present in the local view only. This characterization allows us to treat centralized and localized TC algorithms uniformly. In case of a centralized TC algorithm, we only consider the local view of the decision-making mote, which builds and maintains a global view of the topology. Regarding the connections between the local views in a WSN, we make the following assumptions. (i) The topology is jointly covered by all local views (complete coverage). (ii) Each link can be updated by exactly one mote (single responsibility). (iii) Each protocol messages eventually arrives at its destination (reliable communication). The specification is refined and adjusted iteratively until all required consistency properties can be shown (see 8).

Two subsequent evaluation phases follow the specification phase: the simulative evaluation phase and the testbed evaluation phase. During the simulative evaluation (see 2), the specification of the TC algorithm is implemented manually in a network simulator (see 5) and evaluated w.r.t. the desired performance goals. Typical programming languages of network simulators are JAVA, C, C++, and MATLAB. Examples of state-of-the-art network simulators are OMNET++³ [259] (C++-based) and NS-3⁴ [253]

³ OMNET++ page: <https://omnetpp.org> (visited: 2018-09-17)

⁴ NS-3 page: <https://www.nsnam.org/> (visited: 2018-09-17)

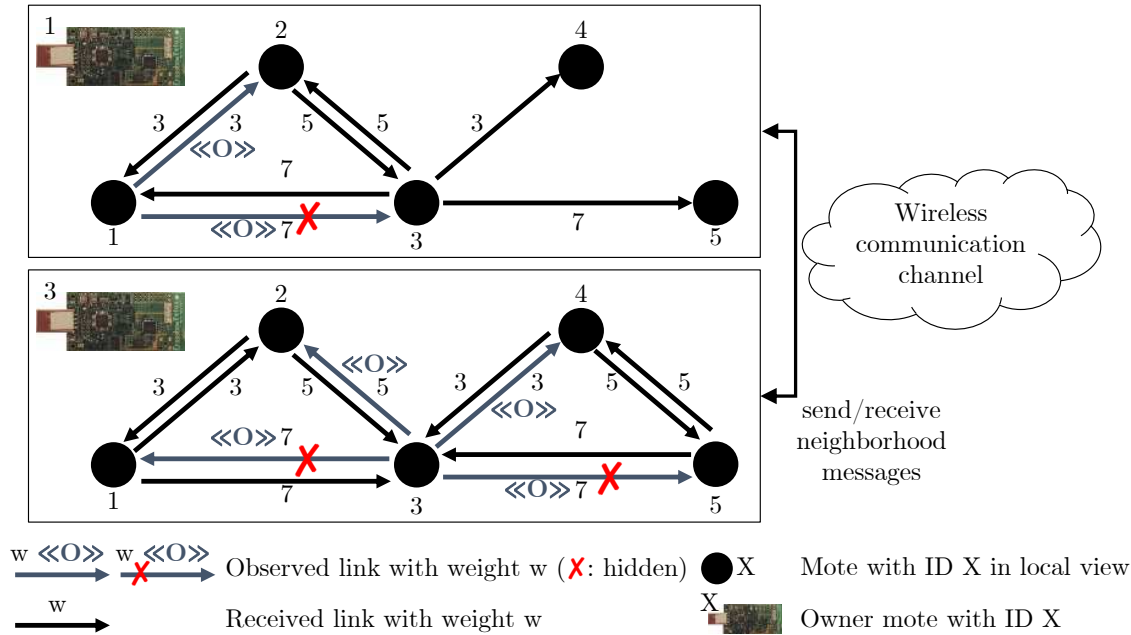


Figure 1.3: Example: Local knowledge

(C++-based), and SIMONSTRATOR⁵ [208] (JAVA-based). A network simulator allows us to evaluate a TC algorithm in a large number of scenarios. This evaluation underpins the general applicability of the TC algorithm. If the TC algorithm fails to show the desired effects, the specification is refined and the simulation implementation is adjusted to the modified specification (see 8). The major challenge of the simulative evaluation is to ensure that the implemented TC algorithm conforms to the specification, for which the consistency properties were proved. In the literature, high-level pseudocode usually serves to sketch the protocol that implements the specified TC algorithm. Implementing a localized TC algorithm for simulative evaluation is more difficult compared to a centralized TC algorithm because (at least for initial experiments) a centralized TC algorithm may access global-knowledge information, which is often provided by the simulator (e.g., consistent view of the physical topology, global-knowledge routing topology). As soon as the simulations show satisfactory results, the TC algorithm is evaluated in the final testbed evaluation.

During the testbed evaluation (see 3), the TC algorithm is evaluated in a hardware testbed (see 6). This step entails additional manual effort for porting or adjusting the simulation code to the target platform. Typical target languages for testbed evaluation are C, C++, and assembler dialects. It is crucial and difficult to ensure that the testbed implementation conforms to the specification. The testbed implementation is arguably

⁵ SIMONSTRATOR page: <http://simonstrator.com> (visited: 2018-10-04)

harder to get right compared to the simulation implementation because the programming languages tend to be on a lower level of abstraction (e.g., due to direct memory access and missing garbage collection), and validation support (e.g., unit testing) is barely available [239]. A prominent hardware platform is TELOS B [181]. An example of a hardware testbed is the FLOCKLAB⁶ [149] at ETH Zurich. If the TC algorithm fails to meet the performance goals, the developer needs to refine the specification and adjust the implementations (see 8).

The described process is iterative and, if carried out manually, error prone and time consuming. All phases require deep expertise of the WSN domain, the theoretical specification framework, the network simulator, and the hardware testbed. This may be the reason that testbed results of many TC algorithms are missing [66, Sec. 3]. Verifying that the manually created simulation and testbed implementations conform to the specification is difficult. Indeed, several lines of research used verification techniques to identify critical corner cases that were not considered during the specification and implementation of communication protocols. Seminal examples are the works by Zave on verifying the Session Initiation Protocol (SIP) [280] and Chord [281, 282]. In general, the lack of formal methods is still an open challenge in communication systems research [49, 156, 162, 193, 280, 281].

The major challenges of communication systems research that are relevant for this thesis are summarized in the following. (i) A novel TC algorithm must fulfill crucial consistency properties to allow for applying it in typical safety-critical application scenarios. (ii) Modern TC algorithms should cope with the inherently dynamic WSN topology to fulfill the consistency properties permanently. (iii) The traditional manual development of TC algorithms is tedious, time consuming, and error prone, and hinders the traceability between specification and implementation artifacts.

Example 1.3 (Challenges). We illustrate challenge (iii) using the Cooperative Topology Control Algorithm (CTCA) [35]. In [35], the authors first present a graph-based informal description of the goal of developing CTCA. The goal of CTCA is to improve the distribution of the motes' lifetimes in a WSN [35, Sec. III]. This goal is then formalized as an ordinary potential game [161]. The authors prove that CTCA creates a stable and optimal topology. The implementation in the paper consists of 83 lines of pseudocode in four listings. The pseudocode contains network-specific aspects such as handlers for sending and receiving the involved types of communication messages. It is highly nontrivial to understand the correspondences to the game-theoretic formalization [35, Sec. V]. In a simulation study, the authors compare CTCA with other state-of-the-art TC algorithms, without describing the simulation setup in detail [35, Sec. VI]. To the best of our knowledge, no testbed evaluation of CTCA has been published.

⁶ FLOCKLAB page: <https://flocklab.ethz.ch/> (visited: 2018-09-17)

1.2 GOALS

The described challenges show that the current way of developing TC algorithms must be improved. Therefore, our overarching goal is to devise a practical methodology for developing correct dynamic TC algorithms. More precisely, we pursue the following major goals.

- **Goal 1—Correctness by construction:** The methodology should produce TC algorithms that are guaranteed to be correct w.r.t. the specified consistency properties. To avoid the described gap between specification and implementation, we must translate the specification (semi-)automatically into implementations for simulative and testbed evaluation. We must ensure that the specification-to-implementation transformation preserves correctness. Tracing specification artifacts to the source code of the resulting implementation artifacts should be possible.
- **Goal 2—Dynamic TC:** To capture and cope with the inherent dynamics in modern WSNs, the proposed methodology should result in dynamic TC algorithms. This means that the developed TC algorithms should process context events instead of entire topology snapshots. Switching between incremental and batch processing mode should be possible to assess whether dynamic or batch TC is more suitable in a certain domain.
- **Goal 3—Practicality:** The methodology shall cover all important development phases of a TC algorithm: specification, simulative evaluation, and testbed evaluation. The specification artifacts should reflect the widely adopted graph-based perspective of TC. The methodology must be accompanied with appropriate tool support for simulative and testbed evaluation. The generated implementation artifacts should be competitive compared to manually implemented TC algorithms w.r.t. relevant metrics (e.g., code size for hardware devices). Finally, the proposed methodology should embrace the iterative development of TC algorithms.
- **Goal 4—Conceptual and technical reusability:** Numerous possible use cases for a correct-by-construction development methodology exist in the research areas of WSN and TC. The methodology should be flexible enough to serve as a blueprint for developing TC algorithms for further domains. Especially adaptive WSNs, which reconfigure or exchange algorithms at runtime, are a promising new application domain, which should be considered. Moreover, topologies are a cross-cutting area in the communication systems research domain. The proposed approach should expose clear formal and technical interfaces to enable its reuse in other domains.

1.3 CONTRIBUTIONS AND THESIS STRUCTURE

To achieve these goals, we employ techniques from model-driven engineering [22, 233]. A model is an abstraction of a system under construction and serves a certain purpose [129]. Model-driven engineering is a software development paradigm that treats models as central, prescriptive artifacts, in contrast to the role of models as mere descriptive artifacts (e.g., for documentation purposes) [233]. The system model is typically accompanied by a set of model transformations, which describe the behavior of the system in a platform-independent way. In the past, model-driven engineering proved to be suitable for real-life application scenarios [94], including communication systems engineering [4, 20]. Major challenges in model-driven engineering are (i) how to make models understandable by domain experts, (ii) how models can be validated, (iii) how the transformation of a model can be specified in a platform-independent way, (iv) how code for a target platform can be generated from models and model transformations such that the model elements can be traced to the generated code, and (v) how modeling tools can be integrated with existing software [233, p. 85]. All of these aspects are relevant in the context of this thesis. In the following, we summarize each chapter and state its contributions w.r.t. the goals stated in Section 1.2.

- In Chapter 1, we motivate why a systematic, correct-by-construction development methodology for TC algorithms is crucially needed due to the safety-critical role of WSNs in the ongoing digitalization of our society and in industry.
- In Chapter 2, we provide definitions for the domain concepts in the context of WSNs and TC (see 7 in Figure 1.2). These definitions are fundamental to understand the contributions of this thesis in detail. We also introduce our running example, the TC algorithm kTC [227].

As a *major contribution* of Chapter 2, we extend the standard definitions to specify dynamic TC algorithms and adaptive WSNs. To this end, we define the concepts of TC mechanisms, TC multi-mechanisms, and TC transitions.

Regarding the *goals* of this thesis, these definitions prepare the ground for developing dynamic TC (Goal 1) and discussing the application of our approach to adaptive WSNs (Goal 4).

- In Chapter 3, we introduce the modeling concepts for specifying topologies, consistency properties, and TC (see 4 in Figure 1.2). We specify valid topologies using metamodeling, local consistency properties using declarative graph constraints, and global consistency properties using second-order logic. To support the development of dynamic TC algorithms, we derive relaxed graph constraints that constitute an inductive invariant of the TC algorithms. The relaxed graph constraints must hold permanently (weak consistency), whereas the original graph constraints must hold eventually for a finite number of context events (strong consistency). To specify

atomic topology modifications during the execution of the TC algorithms or due to environmental influences, we use graph transformation rules. As language for specifying the control flow of a TC algorithm, we use the programmed graph transformation dialect story-driven modeling [60]. Story-driven modeling is a variant of UML activity diagrams [78] with graph transformation rule applications as actions. As a *major contribution* of Chapter 3, we show how to characterize local consistency constraints of TC algorithms in a systematic way using graph constraints.

Regarding the *goals* of this thesis, this chapter presents our model of dynamic TC (Goal 2). Furthermore, with metamodeling, graph constraints, graph transformation rules, and story-driven modeling, we use modeling techniques that represent TC algorithms on a graph-based abstraction level, which is common in the TC research area (Goal 3).

- In Chapter 4, we propose a two-step approach to obtain a correct TC algorithm specification (see ① in Figure 1.2).

First, we employ the constructive approach [44, 91] to establish the inductive variant that each rule application preserves weak consistency. For a given pair of rule and constraint, the constructive approach produces additional weakest application conditions for the rule. These additional application conditions prevent rule applications that would otherwise lead to consistency violations. The synthesized application conditions also restrict the applicability of graph transformation rules that specify context events (e.g., mote failures).

Therefore, in a second step, we show how to synthesize context event handlers. A context event handler anticipates imminent inevitable consistency violations. It retracts exactly those previous decisions of the TC algorithm that would lead to a consistency violation after the context event took effect. We present the anticipation loop synthesis algorithm, which transforms each synthesized application condition into a control-flow fragment that anticipates violations of the given synthesized application condition. We show how to generalize the anticipation loop synthesis algorithm based on further use cases.

The *major contributions* of Chapter 4 are the proposed anticipation loop synthesis algorithm and its application to further use cases.

Regarding the *goals* of this thesis, this chapter describes how we ensure correctness by construction (Goal 1). By proposing the anticipation loop synthesis algorithm, we ensure that the developed TC algorithms cope with all types of context events correctly (Goal 2). For each refinement step, we propose algorithms, which make the construction practically feasible (Goal 3).

- In Chapter 5, we present tool support for simulative and testbed evaluation of TC algorithms. First, COBOLT supports the rapid simulative evaluation (see ② in Figure 1.2) based on an integration of the graph transformation tool EMOFLO [135]

and the network simulator SIMONSTRATOR [208]. Its name alludes to the “Correct-by-construction development of topology control algorithms.” This tool integration allows the user to evaluate a TC specified with EMOFLON immediately inside simulations implemented in SIMONSTRATOR. Second, cMOFLON supports the rapid testbed evaluation of TC algorithms (see ③ in Figure 1.2). cMOFLON is a variant of EMOFLON that generates embedded C code for the CONTIKI IoT operating system [51]. The prefix “c” of cMOFLON alludes to CONTIKI.

The *major contributions* of this chapter are the model-driven tools, which allow the developer to generate TC algorithm implementations for the different evaluation phases based on a common specification.

Regarding the *goals* of this thesis, this chapter focuses on how we built both tools to be practical (Goal 3) and how we ensured the correctness of the involved code generators (Goal 1).

- In Chapter 6, we lift the results of the preceding chapters to families of TC algorithms (see ①, ②, and ③ in Figure 1.2). We show how to specify families of TC algorithms by employing techniques from dynamic software product line engineering [86]. The TC algorithms inside a family share structural constraints. A particular TC algorithm corresponds to a refinement of the graph constraint of its family. This observation allows us to conduct a proof of consistency preservation for an entire family of TC algorithms. We discuss the applicability of our approach using three families of TC algorithms: triangle-based, cone-based, and tree-based TC algorithms. We propose e-kTC, an energy-aware variant of the running example kTC [227] that was inspired by the CTCA algorithm [35].

The *major contribution* of this chapter is the application of our approach to families of TC algorithms, including the characterization of commonalities and differences as well as the required information that a TC algorithm requires for its operation.

Regarding the *goals* of this thesis, we focus on investigating the conceptual and technical reusability in this chapter (Goal 4). We investigate in how far our methodology is flexible enough to cover a set of representative TC algorithms.

- In Chapter 7, we conclude this thesis by discussing future research directions (e.g., regarding further application scenarios and extensions to probabilistic TC algorithms).

1.4 PUBLICATIONS AND SUPERVISED THESES

The results presented in this thesis originate from more than four years of research. The presented work was inspired and funded by the Corporate Research Center (CRC) 1053 *Multi-Mechanism-Adaptation für das künftige Internet (MAKI)*⁷ of the *Deutsche Forschungsgemeinschaft (DFG)*⁸.

PUBLICATIONS In the following, we summarize the most important publications in the context of this thesis. Roland Speith (né Kluge) also was a co-author or first author of the following additional publications [6, 8, 65, 118, 176, 177, 209, 210, 221, 236, 237, 240, 271, 274].

- Roland Kluge, Gergely Varró, and Andy Schürr: “A Methodology for Designing Dynamic Topology Control Algorithms via Graph Transformation,” in Proceedings of the International Conference on Model Transformation (ICMT), pp. 199–213, 2015, Ref. [122].

In this conference paper, we derived a correct-by-construction batch variant of the TC algorithm kTC using the constructive approach [44, 91].

- Roland Kluge and Michael Stein and Gergely Varró and Andy Schürr and Matthias Hollick and Max Mühlhäuser: “A Systematic Approach to Constructing Incremental Topology Control Algorithms Using Graph Transformation,” in Journal of Visual Languages and Computing (JVLC), pp. 47–83, 2016, Ref. [118].

In this article, which appeared in the special issue of JVLC in honor of Prof. SK Chang, we presented the full kTC case study together with an evaluation for correctness, incrementality, performance, and general applicability.

- Roland Kluge and Michael Stein and Gergely Varró and Andy Schürr and Matthias Hollick and Max Mühlhäuser: “A Systematic Approach to Constructing Families of Incremental Topology Control Algorithms Using Graph Transformation,” in International Journal on Software Systems and Modeling (SoSyM), pp. 1–41, 2017, Ref. [119].

In this article, which appeared in the STAF 2015 special issue of SoSyM, we (i) lifted the derivation of correct-by-construction TC algorithm development methodology to families of TC algorithms and (ii) introduced the rapid evaluation environment consisting of the graph transformation EMOFLON and the network simulator SIMONSTRATOR. This article was also invited for presentation at SE 2018 [120] and ICGT 2018 [121].

⁷ MAKI page: <https://www.maki.tu-darmstadt.de> (visited: 2018-09-17), English title: Multi-Mechanism Adaptations for the Future Internet

⁸ DFG page: <http://www.dfg.de/> (visited: 2018-09-17), English: German Research Foundation

- Roland Kluge, Michael Stein, David Giessing, Andy Schürr, and Max Mühlhäuser: “cMoflon: Model-Driven Generation of Embedded C code for Wireless Sensor Networks,” in Proceedings of the European Conference on Modeling Foundations and Applications, pp. 109–125, 2017, Ref. [117].

In this conference paper, we introduced cMOFLON, which compiles batch programmed GT specifications into embedded C code for the CONTIKI IoT operating system. We conducted a case study with three TC algorithms: kTC, l*kTC (a routing-aware variant of kTC) , and LMST (a tree-based TC algorithm).

SUPERVISED THESES The following Bachelor’s and Master’s theses influenced the results of this Ph.D. thesis.

- Maximilian Herbst: “Graph Constraints meet Topology Control: Designing Correct graph-based Topology Control Algorithms,” Bachelor’s thesis supervised by Roland Kluge and Andy Schürr in 2016 [93].

This thesis builds on the SoSyM article [119] and investigates additional algorithms, leading to a feature-based modeling of the family of cone-based TC algorithms. Additionally, the thesis contains a discussion of negative results, e.g., why LMST cannot be encoded using simple graph constraints.

- David Giessing: “From Programmed Graph Transformation to Embedded C code: A Case Study,” Bachelor’s thesis supervised by Roland Kluge, Michael Stein, and Andy Schürr in 2016 [77].

In this thesis, an initial version of cMOFLON was implemented.

- Lukas Neumann: “Integration of a Graph Pattern Matcher into a Network Simulator,” Bachelor’s thesis supervised by Roland Kluge and Andy Schürr in 2016 [168].

In this thesis, the interpreted batch graph pattern matcher DEMOCLES [260] was integrated with the network simulator SIMONSTRATOR [208] in a prototype implementation. An advantage an interpreter is that patterns can be created or modified at runtime without the need to load compiled code dynamically. The tool integration that resulted from this thesis was used in [240].

- Dario Mirizzi: “Design and Implementation of a Demonstrator for Topology Adaptation Algorithms,” Bachelor’s thesis supervised by Roland Kluge, Michael Stein, and Andy Schürr in 2015 [160].

In this thesis, a demonstrator for exploring interactions of TC algorithms (on the underlay) with video streaming mechanisms (on the overlay) was designed and implemented, which we published as PerCom demonstration [237].

1.5 TECHNICAL REMARKS

This thesis is written in American English. Terms are shown in *italics* when they are defined (formally) for the first time. Metamodel elements (classes and operations) are printed in a sans-serif font. Proper nouns (e.g., tool or programming language names) are set in SMALL CAPS. Source code is shown using monospaced font (e.g., `int i = 3;`). Whenever possible, we use the symbols X, Y, Z, J, K as variables for counters (e.g., in loops or induction proofs) and the symbols N and M as variables for amounts (e.g., the length of a sequence).

As suggested in [54, p. 140], adjective-noun composites where the adjective relates to composite noun are typeset with a hyphen (-) to connect the composite nouns and an en-dash (–) to connect the composite noun to the adjective. For instance, in the construction “TC-algorithm–specific condition”, the attribute “specific” modifies “TC algorithm ” and “TC-algorithm–specific” jointly modifies “condition.”

Most of the contributions presented in this thesis have been published in conference papers and journal articles. Still, to avoid awkward formulations in past tense, we stick to present tense throughout the thesis and state unpublished contributions explicitly. The content of this thesis originates from joint work with numerous people—fellow researchers, students, and my supervisor. To honor the contributions of these people, I use the first person plural throughout the remainder of this thesis.

2

FUNDAMENTALS OF TOPOLOGY CONTROL

In this chapter, we introduce the required terminology for the subsequent chapters. We begin with summarizing standard definitions of graph theory, ad-hoc networks, topologies and static topology control [219, 264, 267]. Based on these definitions, we introduce compatible terminology for describing dynamic topology control. To the best of our knowledge, no single standard terminology for dynamic topology control exists. Therefore, we introduce novel terminology to characterize dynamic topology control and adaptive wireless sensor networks, which can reconfigure or exchange the enabled topology control algorithm at runtime. Figure 2.1 locates the role of this chapter in the entire TC algorithm development process (see also Figure 1.2).

Regarding notation, major definitions are presented in explicit environments (e.g., Definition 2.1) and auxiliary definitions are presented *emphasized* and in-text.

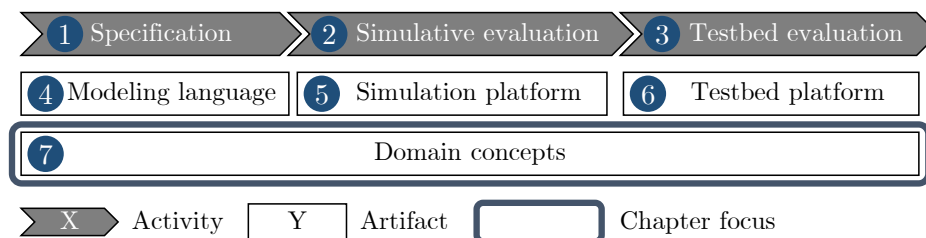


Figure 2.1: Location of Chapter 2 in TC algorithm development process

2.1 GRAPH THEORY

We begin with an introduction of standard graph-theoretic terminology to establish a common understanding of the relevant concepts.

Definition 2.1 (Graph). A (*directed*) graph $G = (V, E, \text{src}, \text{trg})$ consists of a set of nodes V , a set of edges E as well as source- and target-node mappings $\text{src}, \text{trg} : E \rightarrow V$. The term *graph element* subsumes the terms node and edge. In the remainder of this definition, let $e, e_1, e_2 \in E$ and $n, n_1, n_2 \in V$. The node n_1 is the *source node* of e if $n_1 = \text{src}(e)$, and the node n_2 is the *target node* of e if $n_2 = \text{trg}(e)$. In this case, we say that n_1 and n_2 are *adjacent*, n_1 and n_2 are *incident to* e , e is an *outgoing edge* of n_1 and an *incoming edge* of n_2 . If n_1 and n_2 are adjacent, we also say that n_1 is a *neighbor* of n_2 , and vice versa. The *in-degree* and *out-degree* of a node n are the numbers of incoming and outgoing edges of n , respectively. The *degree* of a node n is the sum of its in-degree and out-degree. The *density* of a graph G is the average degree of its nodes and can be calculated as $2 \cdot \frac{|E|}{|V|}$, where $|V|$ and $|E|$ denote the *node count* and *edges count* of the graph, respectively. The edge e_1 is called the *reverse edge* of an edge e_2 if $\text{src}(e_1) = \text{trg}(e_2)$ and $\text{trg}(e_1) = \text{src}(e_2)$. \square

We adopt the following notation: Edges are denoted with the symbol e . If an edge has two subscript indices (e.g., e_{12}), these indices represent the source and target node identifiers, respectively (e.g., $\text{src}(e_{12}) := n_1$ and $\text{trg}(e_{12}) := n_2$). If an edge has a single subscript index (e.g., e_1), we do not care about the source and target node identifiers.

Definition 2.2 (Element properties). A graph $G = (V, E, \text{src}, \text{trg})$ may be complemented by element properties. A *node property* $f : V \rightarrow \mathcal{D}$ is a function from V to a property domain \mathcal{D} . Similarly, an *edge property* $f : E \rightarrow \mathcal{D}$ is a function from E to a property domain \mathcal{D} . Examples of *property domains* \mathcal{D} that appear in this thesis are natural numbers \mathbb{N} , real numbers \mathbb{R} , positive real numbers (incl. zero) \mathbb{R}^+ , Boolean values $\mathbb{B} = \{\text{true}, \text{false}\}$, nodes V , and edges E . \square

In general, an edge may lack a reverse edge (i.e., the graph is *asymmetric*), the source and target of an edge may be identical (i.e., the graph contains *loops*), and multiple edges may have the same source and target nodes, respectively (i.e., the graph contains *multi-edges*). The following definitions describe special classes of graphs.

Definition 2.3 (Undirected graph). In an *undirected graph* $G = (V, E, \text{src}, \text{trg})$, each edge is *symmetric*, i.e., for each edge $e_{12} \in E$ if and only if $e_{21} \in E$, and the values of each edge property are identical for e_{12} and e_{21} . \square

We say that a graph is *structurally symmetric* if each edge has a reverse edge. In contrast to an undirected graph, pairs of reverse edges may differ in certain edge properties in graph that is only structurally symmetric.

Definition 2.4 (Simple graph). A *simple graph* $G = (V, E, \text{src}, \text{trg})$ contains neither *multi-edges* (also known as *parallel edges*) nor loops, i.e., $\forall e_1, e_2 \in E : \text{src}(e_1) = \text{src}(e_2) \wedge \text{trg}(e_1) = \text{trg}(e_2) \Rightarrow e_1 = e_2$, and $\forall e \in E : \text{src}(e) \neq \text{trg}(e)$. A graph that is not a simple graph is called a *multi-graph*. \square

The following definitions help to characterize whether one node is reachable from another node.

Definition 2.5 (Path). A *path* $P(e_1, e_2, \dots)$ in a graph $G = (V, E, \text{src}, \text{trg})$ is a sequence of edges $e_1, e_2, \dots \in E$ in which subsequent edges e_x and e_{x+1} share at least one node, i.e., $\text{trg}(e_x) = \text{src}(e_{x+1})$ or $\text{src}(e_x) = \text{trg}(e_{x+1})$. A path is *directed* if the target node of an edge e_x is the source node of its successor edge e_{x+1} (i.e., $\text{trg}(e_x) = \text{src}(e_{x+1})$). A path is *undirected* if it is not directed. A path may be empty. The *length of a path* P is equal to the number of edges on the path. A *hop* is the traversal of a single link on a path [66]. \square

Definition 2.6 (Connectivity). A graph is *strongly connected* if, for each pair of nodes n_1 and n_2 , a directed path from n_1 to n_2 exists. A graph is *weakly connected* if, for each pair of nodes n_1 and n_2 , an undirected path between n_1 and n_2 exists. \square

Each strongly connected graph is also weakly connected. The following definition introduces triangles. A triangle describes a structure within a graph where two nodes are connected by two alternative, disjoint paths of length one and two, respectively. Triangles are a recurring structure in research on topology control.

Definition 2.7 (Triangle). A *triangle* is a triple of edges (e_1, e_2, e_3) , where $\text{src}(e_1) = \text{src}(e_2) \wedge \text{trg}(e_1) = \text{trg}(e_3) \wedge \text{trg}(e_2) = \text{src}(e_3)$. \square

Example 2.8 (Graphs). Figure 2.2 shows a multi-graph $G = (V, E)$ with node set $V = \{n_1, n_2, n_3, n_4\}$ and edge set $E = \{e_{12}, e_{13}, e_{31}, e_{32}, e_{34}, e_{44}\}$. Without the loop e_{44} , the graph would be a simple graph. The graph contains one pair of reverse edges consisting of e_{13} and e_{31} . We denote pair of reverse edges with identical edge properties edges using a line with two arrow heads. The in- and out-degree of n_4 are 2 and 1, respectively, because the loop e_{44} contributes to the in- and out-degree of n_4 . Therefore, n_4 has a degree of 3. The density of G is $\frac{1}{4} (3 + 2 + 4 + 3) = 2 \cdot \frac{6}{4} = 3$. The graph G is weakly but not strongly connected. By adding e_{43} , the reverse edge of e_{34} , we can make G strongly connected. A directed path in the graph is (e_{12}, e_{21}) , and an undirected path that witnesses weak connectivity is (e_{12}, e_{32}, e_{34}) . The graph contains one triangle: (e_{12}, e_{13}, e_{32}) .

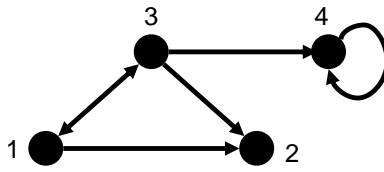


Figure 2.2: Example: Multi-graph

2.2 NETWORK TOPOLOGIES

In this section, we characterize ad-hoc networks, wireless sensor networks, and network topologies.

Definition 2.9 (Ad-hoc network [219, 264]). An *ad-hoc network* is a wireless communication system without predefined communication infrastructure that consists of battery-powered devices that use multi-hop communication to forward messages.

□

Multi-hop communication means that network devices are capable of forwarding (also called *relaying* or *routing*) messages on behalf of other devices.

Definition 2.10 (wireless sensor network [219, 264]). A *wireless sensor network (WSN)* is an ad-hoc network whose devices are low-power sensor devices (called *motes*) that cooperatively collect, aggregate, and forward data toward *data sinks* in the network.

□

Traditional WSNs follow a many-to-one communication pattern with a single sink node. WSNs are often employed in environments for which covering the entire area of interest using a fixed infrastructure (e.g., radio towers, wired connections) is cost-intensive (due to the extent of the observed area, e.g., for habitat observation [50, 247, 254], structural health monitoring [115, 116], plant monitoring [126], pollution detection [226]), or impossible (due to adverse environmental conditions, e.g., for volcano monitoring [178] and wildfire [148], (flash-)flood [32, 97] or landslide warning systems [202]). *Wireless Mesh Networks* are an alternative concept to WSNs, where any mote may be a (temporary) data sink. Wireless Mesh Networks are applied, for instance, in disaster recovery scenarios [228]. Further application scenarios of WSNs and Wireless Mesh Networks can be found in [264, Ch. 1], [138], and [219].

A mote can be *mobile* (e.g., because it is attached to an animal [50]). In contrast, the position of a *static mote* (also called *stationary mote*) does not change [219]. Traditionally, motes are equipped with *omnidirectional antennas*, which transmit with equal power into each direction [267, Sec. 1.1]. Motes that use mm-wave communication are equipped with directional antennas, which concentrate the transmission power in a particular direction [234].

Motes have different capabilities in terms of available mote metadata. Motes equipped with GPS may determine their position dynamically with high precision. In a static WSN, position information may be configured at deployment time. Additionally, a mote may determine relative angles of the motes in its vicinity. In the following, we assume that motes have unique identifiers (e.g., derived from its MAC address) [267, Sec. 1.1].

Definition 2.11 (Topology). The (*underlay*) topology of a WSN is a simple graph $G_T = (V_T, E_T)$ that models the motes and their interconnections. The nodes V_T represent the motes, and the edges E_T represent the actual or potential communication links between motes [267]. \square

Topologies are also called *communication graphs* [219, 264]. For conciseness reasons, we write topology for underlay topology. In the following, the terms *mote* and *link* refer to the nodes and edges of a topology to avoid confusion with general graph terminology. For the same reason, we use the variable ℓ to denote links in contrast to e for edges in general.

Depending on the purpose of modeling a topology, additional information is associated with the topology via element properties. Examples are the identifier and transmission range of a mote, and the weight of a link.

The *identifier* $\text{id}(n) \in \mathbb{N}$ of a mote n is unique within the network. From now on, the subscript of a mote denotes its identifier (i.e., $\text{id}(n_X) = X$). In practice, the mote identifier is (a function of) the MAC address of a mote. The *transmission range* $r(n) \in \mathbb{R}^+$ of a mote n is the maximum distance to another mote for which a connection is possible in the absence of obstacles. If an obstacle blocks the line of sight between two motes, communication may be impossible even if their distance is below their individual transmission ranges. Depending on the radio module and communication protocol, a mote may adjust its transmission power at runtime.

The *weight* $w(\ell_{12}) \in \mathbb{R}^+$ of a link ℓ_{12} describes the cost of transmitting a fixed amount of data from mote n_1 to mote n_2 . In the following, we denote the weight of a link ℓ_{12} with w_{12} (i.e., $w_{12} := w(\ell_{12})$). In practice, the weight of a link is often proportional to the Euclidean distance of its incident motes (if location information is available) or inverse proportional to the *received signal strength indicator (RSSI)*, which is provided by the radio module of a mote. Using the Euclidean distance as link weight is a common basis for estimating power consumption. In the absence of obstacles, the required power P to transmit a message successfully across a link grows with a power of the length d of the link: $P \propto d^\beta$, where the *attenuation* β is typically between 2 and 5 [267, Sec. 1.1].

A classic theoretical model that relates the transmission ranges of motes with the existence of links is the unit disk graph model. The unit disk graph model assumes that motes are embedded in the Euclidean plane and that all motes have a network-wide homogeneous *maximum transmission power* r_{\max} . In a *unit disk graph (UDG)* [264, Sec. 2.3], a link ℓ_{12} exists if the distance of n_1 and n_2 in the Euclidean plane is at most r_{\max} . The UDG model makes strong assumptions on the availability of communication links (e.g., absence of obstacles, reflections, and interference). Therefore, in recent years, the following extensions to the UDG model have been proposed to bridge the gap between theoretical model and real WSNs. In a *quasi-unit-disk graph (q-UDG)* [128] each mote has a lower and upper transmission range. If two motes are closer than the lower transmission range, successful communication is guaranteed and a link exists. Conversely, if two motes are farther apart than the upper transmission range, communication is impossible

and no link exists. If the distance between the motes lies between the lower and upper transmission range, a communication attempt may or may not be successful. A complementary refinement of the classic UDG model is the λ -UDG model, which imposes lower bounds on mote distances. In a λ *unit disk graph* (λ -UDG), the minimal distance between any two motes is at least λ . The λ -UDG model is also called *civilized UDG* or $\Omega(1)$ *model* [264]. UDG and its derivatives are models for the physical topology of a WSN and can be used to investigate theoretical geometric properties of topologies (e.g., planarity). From a technical viewpoint, network simulators use the UDG model to determine which motes may communicate (e.g., [208]). In the following, we do not presume a particular topology model and use the UDG model mainly for illustration purposes. As a weaker precondition, we assume that the maxpower topology, as defined in the following, serves as input topology of a topology control algorithm.

Definition 2.12 (Maxpower topology). The *maxpower topology* [219, p. 177] (also called *all-links graph* [66]) of a WSN is the topology that is determined during a *neighbor discovery phase*, during which each mote transmits with maximum power. \square

The fact that the maxpower topology is determined by measurement rather than calculation distinguishes it from the previously discussed UDG topology. For dynamic TC, the neighbor discovery happens periodically to detect modifications of the topology. Note that the definition of the maxpower topology assumes neither that the maximum transmission power is homogeneous across the network nor that the existence of a link is bound to the Euclidean distance of its incident motes.

Example 2.13 (Topology). Figure 2.3 shows a sample topology. We show mote identifiers next to each circle and omit link identifiers for better readability (e.g., •1 for mote 1). The labels next to each link represent the weight of the link, which is proportional to the Euclidean distance of its incident motes in this case. We use double-headed arrows to denote pairs of reverse links because their weights are identical in this example. The figure also depicts the minimal required transmission ranges to obtain the shown topology as gray circular arcs. The shown topology is not a UDG because the transmission ranges are not uniform ($r(n_1) \approx 39, r(n_4) \approx 23$).

The following definitions characterize the size of the part of the topology that is accessible from a particular mote.

Definition 2.14 (Global-view topology). For a given WSN, the *global-view topology* represents *all* devices and *all* communication links among the motes in the network. \square

The global-view topology is used by centralized algorithms, which require a complete view of the network. However, for many use cases, it is sensible to restrict the scope of the considered topology, which leads to the following definitions.

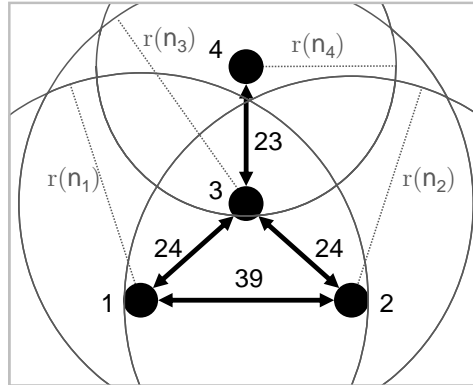


Figure 2.3: Example: Topology with transmission ranges

Definition 2.15 (Local-view topology). A *local-view topology* (*local view* for short) of a mote n represents the motes and their communication links in the vicinity of n . \square

The vicinity is often characterized in terms of the number of required hops to reach a particular mote, as captured by the following definition.

Definition 2.16 (K -hop local view [235]). A K -hop local view of a mote n_1 is a local-view topology that is defined by the maximum number of hops from n_1 to other motes in the topology. A mote n_2 is in the K -hop local view of n_1 if the shortest undirected path between n_1 and n_2 has a length of at most K . In this case, n_2 is called a K -hop neighbor of n_1 . A link is in the K -hop local view of n_1 if it is part of some undirected path that starts at n_1 and has a maximum length of K . \square

The preceding definition is intentionally general in that a K -hop local view only needs to be weakly connected (due to the required undirected paths) for certain domains. For the WSN domain, we assume that the input topology is structurally symmetric. This assumption can be enforced technically by not including undirected links in the local view of a mote. A typical value for K for WSNs is 1 or 2, depending on the density of the topology and the available memory of the mote. This limitation is sensible for the following reasons: The first reason for limiting K is the required storage (e.g., for the routing information and the local neighborhood). In a densely connected WSN, the size of a K -hop local view grows quadratically with K . The second reason for limiting K is freshness of the local view: Whenever the input topology changes, these changes need to be propagated to all $(K-1)$ -hop neighbors of a mote. Therefore, the probability that the local view is outdated grows with increasing local view size. The third reason for limiting the local view size is bandwidth overhead: A mote is, by definition, only able to observe its neighbors (i.e., its 1-hop neighborhood). To obtain a K -hop local view, motes need to receive and send *neighborhood messages*, which contain their current $(K-1)$ -hop

local view [267, p. 114] [264, Secs. 1.1, 5.3]. The size of the neighborhood messages may also grow quadratically with K in dense topologies.

Example 2.17 (Local view). In the topology shown in Figure 2.3, the one-hop neighborhood of n_1 consists of motes n_1, n_2, n_3 and links e_{12} and e_{13} . The two-hop neighborhood of n_1 additionally contains mote n_4 and links $e_{21}, e_{23}, e_{31}, e_{32}$, and e_{34} . This example also illustrates that a two-hop local view is required to check whether an outgoing link of a mote is symmetric (e.g., that w_{12} equals w_{21}).

Another technique for characterizing local views is the concept of virtual topologies, introduced in the following definition.

Definition 2.18 (Virtual topology [264]). A *virtual topology* is a subgraph of a given input topology that hides links of the input topology that should not be used by the application (or other network mechanisms in general) [264, Sec. 5.1]. \square

According to this definition, a virtual topology must contain all motes of the input topology, but may neglect certain links.

2.3 PERFORMANCE OF TOPOLOGIES

For a given set of nodes, many possible configurations (e.g., transmission range assignments) may fulfill the required consistency properties. However, not all of these configurations may perform equally well from the perspective of the application. The following definitions allow us to characterize the performance of a WSN precisely.

Definition 2.19 (Nonfunctional property [37]). A nonfunctional property (NFP) is a measurable value that can be derived from observing a network for a certain amount of time and that signifies the performance of the network in the context of the current application scenario. \square

To illustrate this rather abstract definition, we provide examples of typical NFPs: energy consumption per node or of the entire network [219], bandwidth per link or per end-to-end connection [219], node lifetime (i.e., the time until the depletion of the battery of a node) [264, Sec. 5.1], computational complexity of routing [264, Sec. 5.1], computational complexity of channel parameter negotiation [234]. Depending on the performance requirement of the application or end user, using the minimum, maximum, mean, or median as NFP value is reasonable.

The following definition captures whether increasing or decreasing the value of a particular NFP is favorable in the current application scenario.

Definition 2.20 (Performance goal). A *performance goal* assigns an *objective* (i.e., *minimization* or *maximization*) to an NFP. \square

For example, in the context of WSNs, maximizing throughput, maximizing network lifetime, minimizing energy consumption, and minimizing interference are examples of performance goals [219, 267].

A key problem in network research is that certain NFPs (e.g., network lifetime) can only be measured after a long time (or even after a network breakdown). Therefore, an established approach is to estimate NFPs using *metrics* (e.g., link weight, density, RSSI, node distance). These metrics should correlate with the desired NFP. For instance, the energy required to transmit a fixed amount of data across a wireless link grows at least quadratically with the length of the link [63, 230].

2.4 TOPOLOGY CONTROL

We are now ready to define one of the core concepts of this thesis: topology control.

Definition 2.21 (Topology control). *Topology control (TC)* is an approach to create and maintain a virtual topology to other network mechanisms based on an *input topology* with the goal to achieve a performance goal while preserving *consistency properties*. □

Definition 2.22 (Topology control algorithm). A *topology control algorithm* (TC algorithm) is an algorithm that instantiates the concept of TC by calculating a virtual topology from an input topology with the goal of achieving a particular performance goal. □

Definition 2.23 (Topology control mechanism). A *topology control mechanism* (TC mechanism) is a network component that contains a TC algorithm and additional control logic that, for instance, decides when to invoke the TC algorithm or when to update the input topology. □

These definitions are deliberately more general than in the related work on WSNs (e.g., [219, 264, 267]). They allow us to reuse the concept of TC in other application scenarios. For clarity and concreteness, we focus on TC for WSNs in the following. In 1984, Takagi and Kleinrock published one of the first papers mentioning the term “topology control” [251]. According to [219], TC refers to adjusting the transmission range of motes to achieve better energy consumption while maintaining connectivity of the network. More generally, in [267, p. 113], TC is defined as the task of selecting appropriate neighbors to improve network lifetime and throughput, again while maintaining connectivity. Another performance goal is to reduce the computational complexity for routing messages [264, Sec. 5.1] or negotiating channel parameters [234].

Preserving connectivity is probably the most prominent required consistency property of a TC algorithm [219, 264, 267]. In contexts with relaxed requirements, it may also suffice for TC to provide approximate connectivity [264, Sec. 5.2]. In contrast, in contexts with stricter safety requirements, *k-connectivity* may be necessary, which means that the topology remains connected even if k nodes fail [264, Sec. 5.2].

Another common, but more specialized required consistency property of a TC algorithm is the preservation of coverage. Preservation of coverage applies to WSNs where a mote observes a certain geographic region and may go into a *sleep state* for a certain amount of time if other motes cover the same region. During this period, the mote consumes considerably less energy, but also collects no data. A *coverage* requirement

for this type of WSN states that the set of awake motes must cover a predefined area (*blanket coverage*) or the border of a specified area [138].

Preserving connectivity and coverage are consistency properties that result from the application running in the WSN. Intermediate network layers may require that additional consistency properties hold. As a first example, a *geographic routing algorithm* requires that the virtual topology is planar (e.g., [109, 136]). *Planarity* means that no links intersect. Geographic routing algorithms derive forwarding decisions from the target location of the message (instead of the address of the target mote). This makes geographic routing algorithms generally faster than generic routing algorithms because no routing table needs to be maintained [267, Secs. 2.1,5.2]. However, if the input topology of the geographic routing algorithm is not planar, packets cannot be routed properly. Therefore, building a planar virtual topology is a further required consistency property if geographic routing is used.

Related consistency properties are that the virtual topology must be θ -separated and have bounded mote degree. A topology is θ -separated [143] if all outgoing links of a mote are separated by an angle of at least θ . A topology has *bounded mote degree* if the maximum number of neighbors of a mote is independent of the input topology size and density. As a second example, link layer mechanisms that employ acknowledgment messages require that the virtual topology is undirected [264, Sec. 5.2]. If the topology is asymmetric, acknowledgment messages get lost.

In the following, we discuss different dimensions for characterizing a TC algorithm according to its role in the network protocol stack, the minimal required information, and the locality of the decisions.

2.4.1 Topology control in the network stack

The first dimension is the role of TC in the network protocol stack. The Open Systems Interconnection (OSI) Reference Model [40] (left part of Figure 2.4) is an architecture for organizing the tasks of protocols in distributed systems into 7 layers. This reference model has been standardized by the International Standards Organization (ISO) in standard ISO 7498.

We briefly summarize the tasks of each layer and highlight those layers that are important for TC in WSNs

The *application layer* provides the functionality that is realized by the network. Often, applications are characterized by their communication pattern. The *communication pattern* of a network describes the possible ways in which motes may exchange data. If the application uses *many-to-one communication*, many or all motes in the network transmit toward a single sink device (*base station*), which is the usual use case in WSNs. If the application uses *many-to-many communication*, motes may exchange data in a pairwise manner. This communication pattern occurs for example in disaster recovery scenarios [228]. The *presentation layer* (layer 6) decouples the application from the concrete

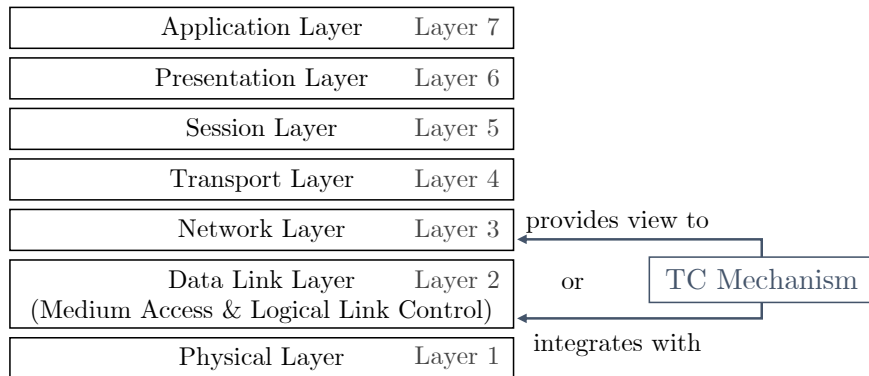


Figure 2.4: OSI Reference Model with possible roles of a TC mechanism

representation of algorithm-specific data (e.g., encoding). The *session layer* (layer 5) manages the state of application sessions. The *transport layer* (layer 4) provides an end-to-end perspective of communication, even if packets have to be routed across multiple devices or even networks in the background. The *network layer* (layer 3) is responsible for routing packets toward their destination using multi-hop communication if two network devices are not neighbors. The *data link layer* (layer 2) manages the access to the communication medium by negotiating channel parameters and listening for free transmission slots and provides hop-wise reliable communication. The *physical layer* (layer 1) is responsible for encoding the data in a way that they may be transmitted using the physical communication medium.

Traditionally, the layers of the OSI reference model are categorized into the *overlay* (layers 4 to 7), which is typically implemented and provided by end devices, and the *underlay* (layers 1 to 3), which is provided by the communication infrastructure [40].

Important layers for the TC are the application (layer 7), network (layer 3), data link (layer 2), and physical layer (layer 1). Presentation (layer 6), session (layer 6), and transport layer (layer 4) are less frequently considered [264, Ch. 2].

In the related work [264, Ch. 2], TC is usually integrated with the data link layer [103, 173, 228] or located between data link and network layer (right part of Figure 2.4). In the second case, TC provides a view of the topology maintained by the data link layer to the network layer (layer 3) [66]. Figure 2.5 sketches the interaction of a TC mechanism with other components in the network stack. A TC mechanism creates and maintains its input topology via a *topology monitoring component*, which observes an *input topology provider mechanism* (e.g., the raw MAC layer neighborhood or the output topology of the data link layer). The control logic of a TC mechanism decides when to update the virtual topology and the TC algorithm of the TC mechanism performs the actual updates. One or more *virtual topology consumer mechanisms* observe the virtual topology (e.g., the routing for adjusting the routing table or the radio module for adjusting the transmission power).

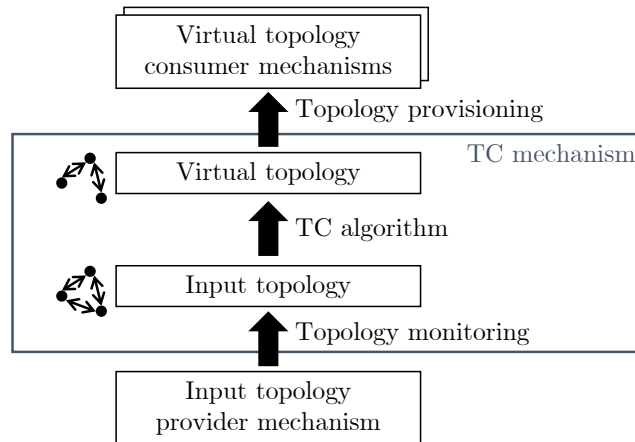


Figure 2.5: Interaction of TC mechanism with topology providers and consumers

2.4.2 Required information for topology control

The second dimension is the classification of TC algorithms according to the required information on a mote, which is often the major means for categorizing TC algorithms in surveys (e.g., [219, 267]). In *neighbor-based TC* [219], a mote needs to be able to identify the motes in its vicinity and to assign a link weight to the corresponding link. In *direction-based TC*, a mote needs to be able to determine the relative directions of its neighbor motes. In *location-based TC*, a mote needs to know its geographic position. Among these three categories, neighbor-based TC requires the least knowledge, and location-based TC requires the most knowledge. Besides *application-agnostic TC*, which is unaware of the current overlay, *application-aware TC* [239, 267] may also access information provided by higher-layer mechanisms (e.g., the hop count to the base station provided by the network layer).

2.4.3 Locality

The third and last dimension that we discuss is the locality of decision making. A *centralized algorithm* is executed on one mote and its decisions are communicated to all nodes in the network. The advantage of centralized algorithms is that they may derive better decisions due to their larger knowledge base. However, the major disadvantages of centralized algorithms are identical to the disadvantages of increasing the local-view size, as discussed earlier. A *distributed algorithm* is executed on multiple nodes in the network. During the execution of a distributed algorithm, nodes may communicate with other nodes to publish their decisions or to acquire additional knowledge. Even executing a distributed algorithm, a mote may collect information about the entire network. A *localized algorithm* is a distributed algorithm with limited local knowledge. Localized

algorithms tend to be more robust and scalable compared to centralized algorithms. However, due to the limited knowledge, decisions of a localized algorithms are usually more conservative compared to a centralized algorithm and not optimal.

2.4.4 Running example: *kTC*

The TC algorithm *kTC* [227] is the running example of this thesis. *kTC* excludes hides a link from the virtual topology if it fulfills the following *kTC condition*: A link ℓ_{12} is *not* part of the virtual topology of *kTC* if (i) ℓ_{12} is the weight-maximal link in some triangle $(\ell_{12}, \ell_{13}, \ell_{32})$ (i.e., $w_{12} \geq \max(w_{13}, w_{32})$) and (ii) w_{12} is at least k times larger than the minimal weight in the triangle (i.e., $w_{12} \geq k \cdot \min(w_{13}, w_{32})$). It is a common practice in the TC literature to characterize a TC algorithm in terms of links to be omitted from (rather than to be include in) its virtual topology. The first clause of the *kTC* condition ensures that, in each triangle, the link with the highest power consumption should be omitted. This may cause messages to be relayed via the intermediate mote n_2 . The second clause uses the configuration parameter k of *kTC* to control the aggressiveness of the link removal. Controlling the aggressiveness is sensible because removing the immediate communication link ℓ_{12} in favor of a multi-hop path (ℓ_{13}, ℓ_{32}) may increase the end-to-end packet drop rate and latency (e.g., if n_2 is already overloaded). *kTC* is a neighbor-based, localized algorithm that requires a 2-hop local view.

We think that *kTC* is a suitable running example for this thesis because its theoretical and practical properties has been investigated in multiple papers. Besides the original paper, which presents the specification and the simulation results [227], a testbed implementation of *kTC* in the CONTIKI operating system exists [228]. Several variants of *kTC* have been developed over the years (e.g., *g-kTC* [130, 238], *l-kTC* [130, 238], *l*kTC* [239], *e-kTC* [119]). Finally, *kTC* shares the triangular pattern with other *kTC* algorithms in the literature (e.g., *XTC* [270], *RNG* [109, 257], *GG* [70, 211]).

A recurring consideration during the development of TC algorithms is tie breaking. A *tie* represents a situation, where (i) two links are equally suitable for omission from the virtual topology, and (ii) omitting both links from the virtual topology violates consistency [264, Sec. 5.3].

A tie can be *broken* using an additional predicate that enforces a strict order on the links involved in a tie. The *kTC* condition reveals that *kTC* requires tie breaking because, otherwise, two links could be removed from the same triangle. Removing two links from a triangle may obviously break the connectivity of the topology. The following definition reflects a generic tie-breaking strategy for weight-based TC algorithms that relies on the fact that a mote has a unique identifier within a topology and that a topology is a simple graph.

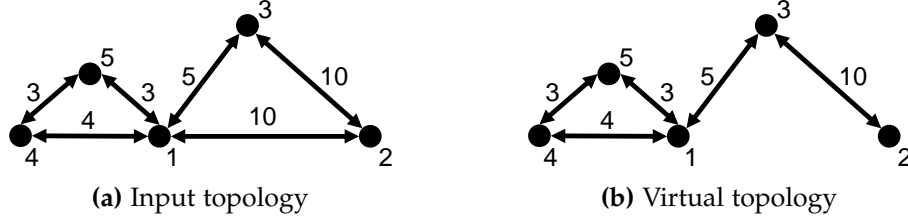


Figure 2.6: Example: Topology control with kTC ($k = 1.5$)

Definition 2.24 (Unique link weight). Let ℓ be a link with associated weight $w(\ell)$. The *unique link weight* $w'(\ell)$ of ℓ is defined as

$$w'(\ell) = (w(\ell), \text{id}(\text{src}(\ell)), \text{id}(\text{trg}(\ell))) \in \mathbb{R} \times \mathbb{N} \times \mathbb{N}.$$

□

Definition 2.25 (Canonical order of tuples). Let $\mathcal{D}_1, \dots, \mathcal{D}_N$ be domains for which a strict ordering $<$ is defined individually (e.g., \mathbb{R}, \mathbb{N}). Given two tuples $X = (x_1, x_2, \dots, x_N), Y = (y_1, y_2, \dots, y_N) \in \mathcal{D}_1 \times \mathcal{D}_2 \times \dots$. The *canonical (descending) order* \prec of X and Y is defined as

$$\begin{aligned} X \prec Y &\Leftrightarrow x_1 < y_1 \vee \\ &(x_1 = y_1 \wedge x_2 < y_2) \vee \\ &(x_1 = y_1 \wedge x_2 = y_2 \wedge x_3 < y_3) \vee \\ &\dots \\ &(x_1 = y_1 \wedge x_2 = y_2 \wedge \dots \wedge x_N < y_N) \end{aligned}$$

The *canonical ascending order* is the inverse of the canonical descending order:

$$X \succ Y \Leftrightarrow Y \prec X$$

The *canonical maximum* $\max(X, Y)$ of X and Y is X if $X \succ Y$, else Y . Similarly, the *canonical minimum* $\min(X, Y)$ of X and Y is X if $X \prec Y$, else Y . □

The idea behind Definition 2.24 is to use the canonical lexicographic order on the unique weight of links as tie breaker. When comparing two links w.r.t. w' , the identifiers of the source and target nodes of the links only take effect if both links have equal weights. Unique properties for other element properties may be derived in a similar way.

Example 2.26 (kTC). Figure 2.6 shows an input topology (Figure 2.6a) and a corresponding virtual topology (Figure 2.6b) for kTC with $k = 1.5$. Contrary to the previous example, link weights do not correlate with the length of the corresponding arrow in this and subsequent examples. According to the kTC condition, four links are eligible for inactivation ($l_{12}, l_{21}, l_{23}, l_{32}$). However, due to the tie breaking in the triangles consisting of motes n_1, n_2, n_3 , only l_{12} and l_{21} are hidden from and l_{23} and l_{32} are still visible in the virtual topology.

In [227], the authors show that kTC preserves connectivity and maintains a planar, θ -separated virtual topology under a UDG assumption and if identifier-based tie breaking is used.

2.5 DYNAMIC TOPOLOGY CONTROL

Even if the motes of a WSN do not move, the input topology usually changes over time [277]. For example, an obstacle between two motes may lead to an increased link weight or a link removal. The following definition summarizes the possible uncontrollable modifications of a topology by the environment.

Definition 2.27 (Context event). A *context event* is a modification of a topology that cannot be influenced by the TC algorithm. We distinguish between three categories of context events: (i) A *mote event* describes that a mote is added to (*mote addition*) or removed from (*mote removal*) the topology. (ii) A *link event* describes that a link is added to (*link addition*) or removed from (*link removal*) the topology. (iii) An (*element*) *property event* describes that a node property (*mote property event*) or link property (*link property event*) has changed. A *complex context event* consists of a sequence of context events. \square

In the following, we discuss the possible reasons of each type of context event. A mote removal occurs if a mote fails or if its battery has depleted. Mote additions occur, for instance, if a failed or empty mote is replaced manually or recharged (e.g., using solar power). A mote may be added to the topology, e.g., to replace a depleted mote. The second case, however, is rare since WSNs are often used in hostile environments, which make exchanging or recharging motes difficult. In the communication systems domain, the term *churn* summarizes a temporal sequence of mote events.

A link event is either induced by mote events or by one mote entering or leaving the transmission range of another mote. A link removal event occurs if (i) an obstacle moves between its incident motes, or (ii) the distance of its incident motes grows larger than the transmission range of its source mote. A link addition event occurs if (i) an obstacle between the incident motes of the link disappears, or (ii) its incident motes have converged so that their distance falls below the transmission range of the source mote of the link.

An element property event may be either immediate or derived. An *immediate property event* originates from the modification of the element that the property belongs to. A *derived property event* is triggered by another context event. For instance, if the link weight is calculated from the RSSI value of the underlying communication channel, a link-weight modification event is an immediate event. If the weight of a link is equal to the Euclidean distance of its incident motes (e.g., in case of location-based TC), a link-weight modification event is derived from mote-position modification events.

In traditional WSNs, context events were perceived as exceptional conditions. In the presence of today's mobile devices and the requirement that consistency properties should hold permanently, the handling of context events should be integrated constructively into the development process of TC algorithms. The following definition captures this requirement.

Definition 2.28 (Dynamic TC mechanism). A TC mechanism is *dynamic* if it accepts context events (in the sense of Definition 2.27) as input. \square

This definition is general and only mandates that context events must be observable. Dynamic TC is related to *adaptability*, which describes that a TC algorithm is capable of processing context events [264, Sec. 5.2]. Our notion of a TC mechanism separates clearly between a TC algorithm, which is a one-time transformation and (by definition of an algorithm) always terminates, and a TC mechanism, whose control logic may run continuously (similar to a process). The following definitions specify in how far the TC algorithm of a TC mechanism reacts to context events.

Definition 2.29 (Batch TC algorithm). A *batch TC algorithm* transforms an input snapshot of a topology into a virtual topology. \square

Definition 2.30 (Incremental TC algorithm). An *incremental TC algorithm* is a TC algorithm that takes a sequence of previous context events, previous TC algorithm decisions, and the effect of the current context event as input and returns a sequence of modifications of the virtual topology. \square

The sequence of previous context events that is provided as input to an incremental TC algorithm determines its input topology. Even though batch TC requires less effort for analyzing the context event, the required effort of reclassifying the entire topology may be inadequate if the effect of the context event is small compared to the size of the topology. The decision whether a batch or incremental TC algorithm is more suitable for a given scenario depends only on the dynamics of the topology. An incremental algorithm can always be used as batch algorithm by neglecting the provided information about previous decisions of the TC algorithm.

From the TC literature, we learn that most TC algorithms are designed as batch algorithms. This may originate from the complexity of defining appropriate routines for handling each possible type of a context event.

2.5.1 Connecting input and virtual topology

The subgraph relation between input and virtual topology can be represented in at least two ways (Table 2.1). The first alternative is to maintain *separate* input and virtual topologies. In this case, context events only affect the input topology and the TC mechanism synchronizes these modifications to the virtual topology by adding and removing topology elements to and from the virtual topology, and by updating element properties. During the synchronization, the TC algorithm has to ensure that the consistency properties are preserved (e.g., by adding a link that it previously removed from the virtual topology).

Table 2.1: Alternative representations of input and virtual topology

Property	Separate representation	Integrated representation
Involved graphs	2	3
TC decisions	difference of input and virtual topology	link property
Representation of pending context event	modification of input topology	markers at affected elements
Handling of context events	synchronization from input to output topology	modification of link states
Advantages	separation of input and virtual topology only two involved graphs	explicit bookkeeping of previous decisions more concise representation
Disadvantages	difficult to achieve incrementality due to implicitly encoded previous TC decisions	additional storage for TC mechanism topology need to maintain virtual topology as view

The second alternative is to *integrate* information of the input and output topology into a third, internal topology. We call this internal topology the *TC mechanism topology* because it serves as central source of knowledge for all components of the TC mechanism. The virtual topology is realized using an additional link property that marks whether or not a particular link is part of the virtual topology. When a context event is observed, a marker is attached to the affected topology element(s). The TC mechanism control logic then decides when to process the context event markers.

A crucial drawback of the separate representation of input and virtual topology compared to the integrated representation is that information of previous TC algorithm decisions is only stored implicitly as difference of both topologies. A dynamic TC algorithm will probably need to recalculate these decisions frequently, which is an additional computation step that is unnecessary using the integrated representation. Therefore, we decided to use the integrated representation.

The *state* $s(\ell_{12})$ of a link ℓ_{12} stores which links are part of the virtual topology. For conciseness reasons, we introduce the following shorthand notation $s_{12} := s(\ell_{12})$. A link ℓ_{12} may be in one of three states, as detailed in the following.

- ℓ_{12} is *active* (i.e., $s_{12} = A$) if it has been selected by the TC algorithm to be *visible* of the virtual topology. This means that the link can be used by virtual topology consumer

mechanisms (e.g., routing, application). An active link ℓ_{12} is denoted by a solid arrow-headed line: $1 \bullet \rightarrow \bullet 2$.

- ℓ_{12} is *inactive* (i.e., $s_{12} = \text{I}$) it is *hidden* from the virtual topology. An inactive link ℓ_{12} is denoted by a dotted arrow-headed line: $1 \bullet \cdots \rightarrow \bullet 2$.
- ℓ_{12} is *unmarked* (i.e., $s_{12} = \text{U}$) if it needs to be (re-)processed by the TC algorithm. Unmarked links are *temporarily visible* in the virtual topology to ensure that the virtual topology fulfills all consistency properties (e.g., connectivity) and are denoted by a dashed line: $1 \bullet \dashrightarrow \bullet 2$.

Apart from the three concrete states of a link, we introduce the following terminology and notation to refer to *ranges of link states*.

- A *don't care* link ℓ_{12} may have an arbitrary state (i.e., $s_{12} \in \{\text{A}, \text{I}, \text{U}\}$) and is denoted by a gray solid arrow-headed line: $1 \bullet \xrightarrow{\text{gray}} \bullet 2$.

The definition of a three-valued link state property is similar to three-valued logic approaches to model uncertainty (e.g., in SQL [275] or distributed graph querying [29]). The *A-view* of a topology G_{\top} is the result of removing all inactive and unmarked links from G_{\top} . The *AU-view* of a topology G_{\top} is the result of removing all inactive links from G_{\top} .

We can now characterize the role of a TC algorithm within a TC mechanism as follows. The virtual topology of the TC mechanism is an AU-view of the TC mechanism topology. A TC algorithm is an algorithm that accepts a TC mechanism topology with zero or more unmarked links as input and updates the link state in such a way that (i) all links are marked upon termination, and (ii) the virtual topology fulfills all consistency properties. Note that the first condition implies that, upon termination of the TC algorithm, the A-view and AU-view of the TC mechanism topology are identical.

Example 2.31 (TC mechanism topology). We continue with the sample topologies discussed in Example 2.26. Figure 2.6 shows the separate representation of input and virtual topology at one point in time (for $k = 1.5$). Figure 2.7b shows the corresponding TC mechanism topology. Here, ℓ_{12} and ℓ_{21} are inactive (i.e., $s_{12} = s_{21} = \text{I}$), and all other links are active (e.g., $s_{13} = \text{A}$).

Figure 2.7a shows a completely unmarked topology, where every link is unmarked (e.g., $s_{12} = \text{U}$). This is the initial state of the TC mechanism topology prior to the first execution of the TC algorithm.

2.5.2 Context event handling

A TC mechanism must be able to react to context events. Therefore, a TC mechanism is not only equipped with a dynamic TC algorithm implementation, but also with corresponding context event handlers. A *context event handler* is an algorithm that processes

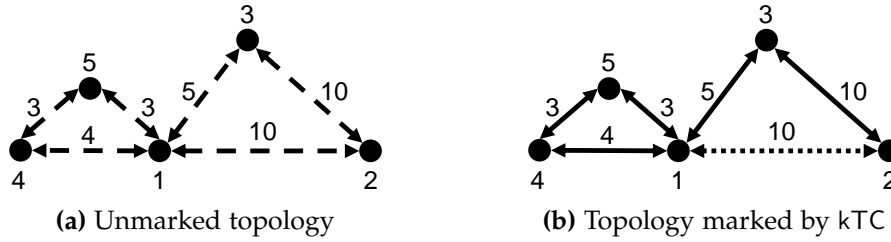


Figure 2.7: Example: Integrated representation of input and virtual topology

a certain type of context event marker such that (i) the context event marker is removed during the processing, (ii) the TC mechanism topology reflects the context event indicated by the context event marker, and (iii) fulfills all consistency properties. Therefore, the context event handlers of a TC mechanism must act in accordance with the TC algorithm of the TC mechanism. We say that a context event is *pending* if it is known to the TC mechanism (in the form of a context event marker) but not processed by a context event handler yet. In examples, we use a triangle with accompanying context event type indicator to denote a context event marker. The mnemonic \blacktriangle_{-e} next to a link ℓ denotes that the removal of the link is pending (denoted $\blacktriangle_{-e(\ell)}$ in text), the mnemonic $\blacktriangle_{+e(X,Y,W)}$ close to nodes n_X and n_Y indicates that the insertion of a link ℓ_{XY} with weight W is pending, and the mnemonic $\blacktriangle_{w=W}$ next to a link ℓ denotes that a weight-modification event with new weight W is pending for this link (denoted $\blacktriangle_{\text{mod-}w(\ell,W)}$ in text).

The TC mechanism control logic should invoke the TC algorithm and the context event handlers in a suitable order to process context events in a timely manner. This requirement is called *agility* [232, Ch. 4]. For instance, if the input topology is rather stable (i.e., the arrival rate of context events is relatively small), the TC algorithm may run to completion before checking for pending context events. In contrast, if the input topology is rather unstable (i.e., context events arrive frequently), the TC mechanism may interrupt the TC algorithm after processing one or a few unmarked links to process pending context events. Therefore, a TC algorithm should support interruptions after processing individual unmarked links.

A TC mechanism should adjust the reaction time depending on the type of context event to ensure stability of the virtual topology [232, Ch. 5]. For instance, the TC mechanism should react quickly to link removal events to ensure that the topology remains connected. In contrast, the TC mechanism may delay the handling of link addition events or define thresholds for link-weight-modification events to ensure that the new links are established reliably or to avoid fluctuating link state modifications. In the following, we assume that a separate *topology monitoring component* decides on the point in time when a context event should become visible in the TC mechanism topology.

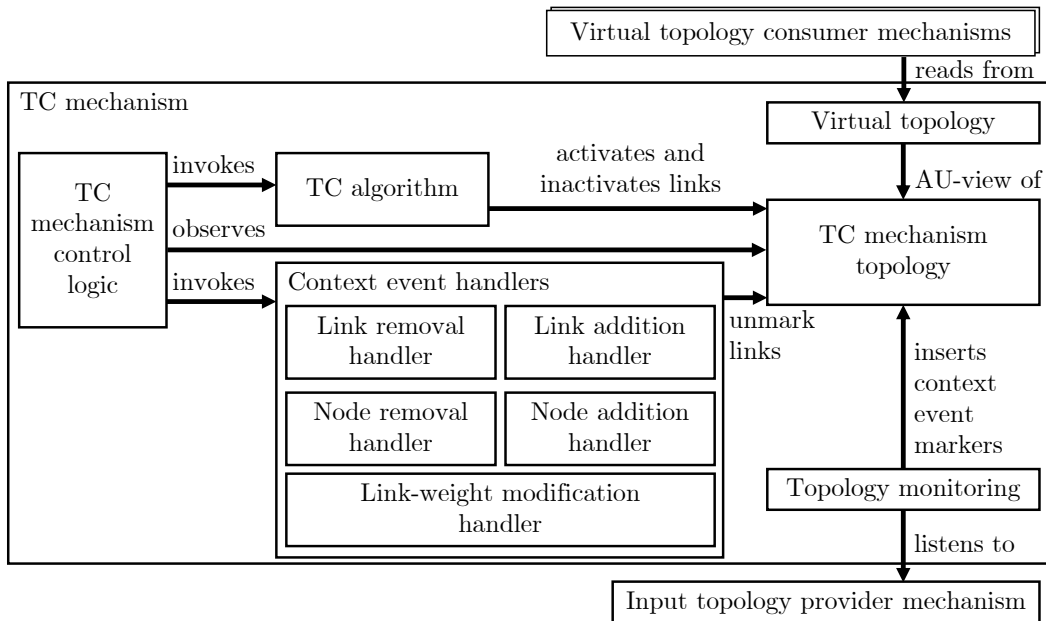


Figure 2.8: Architecture of a TC mechanism

2.5.3 Architecture of a topology control mechanism

Figure 2.8 shows the architecture of a TC mechanism that we build upon in the following. This view enables a more fine-grained perspective of TC mechanisms compared to Figure 2.5. The depicted TC mechanism is capable of handling all mote, link, and link-weight-modification events. The TC mechanism control logic is responsible for invoking the TC algorithm and context event handlers based on observed modifications of the TC mechanism topology, which is the integrated representation of input and virtual topology. The TC algorithm activates or inactivates links in the TC mechanism topology according to the underlying algorithm-specific condition. The context event handlers process context events and, thereby, revoke previous decisions of the TC algorithm to avoid violations of the consistency properties in the virtual topology. The virtual topology of the TC mechanism is an AU-view of the TC mechanism topology and is visible to virtual topology consumer mechanisms. The topology monitoring updates the TC mechanism topology based on the input topology provider mechanism that it listens to. Separating the tasks of (re-)marking (TC algorithm) and unmarking links (context event handlers) is reasonable because this separation allows the TC mechanism control logic to process multiple context events in sequence before invoking the TC algorithm. Invoking the TC algorithm inside a context event handler may waste computation time because the next invoked context event handler could unmark the just marked links again due to another pending context event.

Example 2.32 (Dynamic TC). This example illustrates the interplay of the components of a TC mechanism that is configured with kTC and $k = 1.5$. Figure 2.9a shows an unmarked initial topology. After invoking the TC algorithm, l_{12} and l_{21} are inactive, and all other links are active (Figure 2.9b). Now the topology monitoring indicates a pending removal of l_{23} and l_{32} (Figure 2.9c). The TC mechanism reacts to the new pending event by invoking the context event handler for link removals. This context event handler unmarks l_{12} and l_{21} because leaving these links inactive would lead to a disconnected virtual topology as soon as l_{23} and l_{32} have been removed (Figure 2.9d). The TC mechanism reacts to the freshly unmarked link by invoking the TC algorithm again (Figure 2.9e). Then, the topology monitoring detects a pending modification of w_{14} and w_{41} from 4 to 5 (Figure 2.9f). The context event handler for link-weight modifications is invoked by the TC mechanism and unmarks l_{14} and l_{41} because the modified link weights make these links (currently) eligible for inactivation (Figure 2.9g). Finally, the TC algorithm is invoked and inactivates l_{14} and l_{41} (Figure 2.9h).

2.5.4 Adaptive wireless sensor networks

In a traditional WSN, which is designed for a more or less fixed environment and application scenario, a mote is configured at design time to use a single TC mechanism. In a modern WSN, however, the set of active applications may vary over time. For example, in the context of the IoT, a wireless local-area network inside an apartment may only run smart home applications initially (e.g., for controlling the room temperature and ambient light). This setting is not safety critical and, at this point, the performance goal should be to reduce the energy consumption of the involved battery-powered devices, possibly at the cost of reduced robustness. Let's assume now that a medical incident requires that a resident of the apartment needs to transmit medical data frequently to an e-health service provider. This application is more safety critical than the smart-home application, and, probably, a more robust TC mechanism should be used to avoid data loss.

In an *adaptive WSN* [7, 61, 182, 183, 184, 263], a mote can exchange certain components at runtime. The following definition describes a component that allows a WSN or a mote to exchange the current TC mechanism transparently for input topology provider and virtual topology consumer mechanisms.

Definition 2.33 (Topology control multi-mechanism). A *topology control multi-mechanism* (TC multi-mechanism) is a component that encapsulates one or more compatible TC mechanisms and provides the same interface to input topology provider and virtual topology consumer mechanisms as a TC mechanism (Figure 2.10). Two

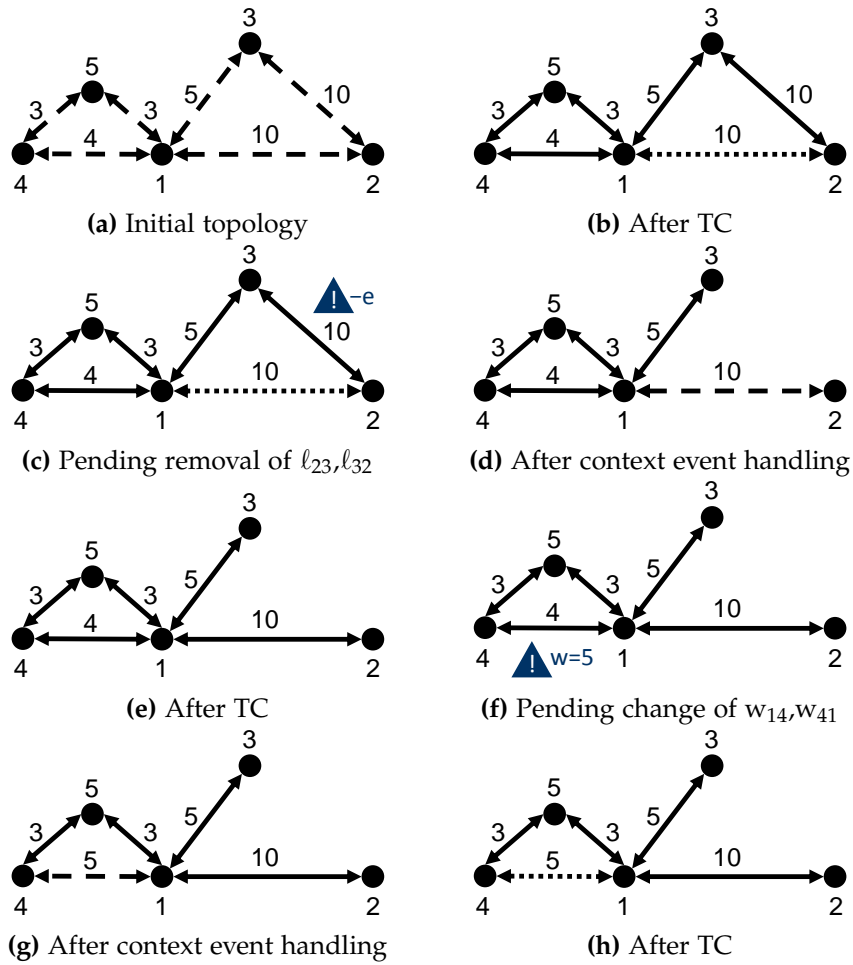


Figure 2.9: Example: Dynamic topology control ($k = 1.5$)

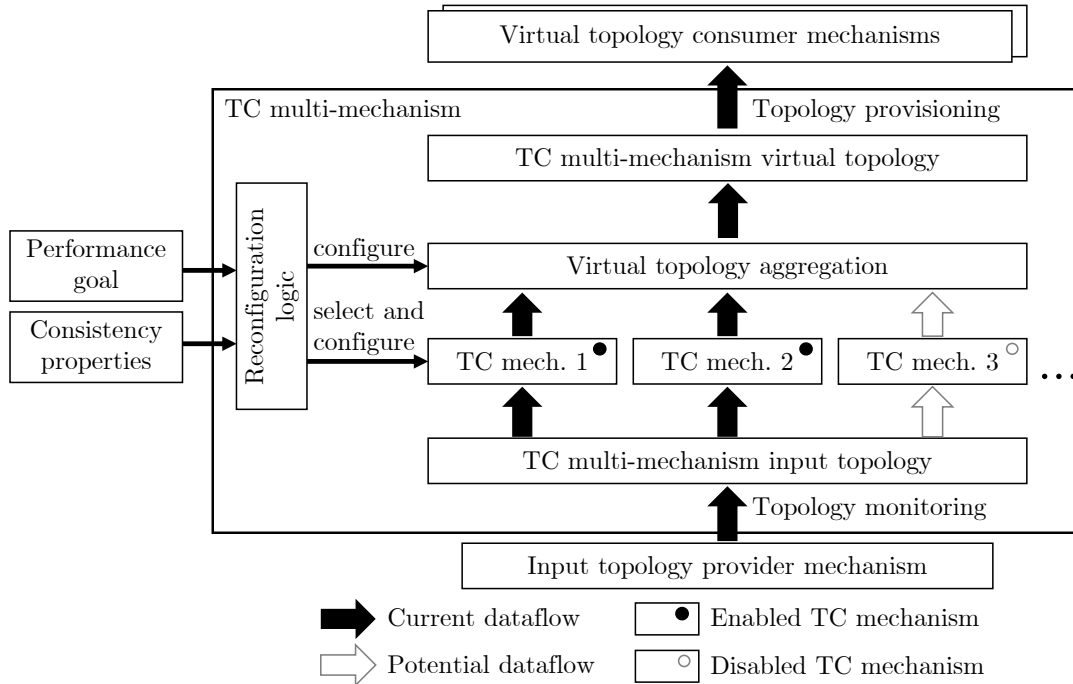


Figure 2.10: Architecture of a TC multi-mechanism

TC mechanisms are *compatible* if they are able to process the same types of context events. The *reconfiguration logic* enables, configures, and disables TC mechanisms and configures the *aggregation strategy* of the TC multi-mechanism virtual topology based on the current performance goals and consistency properties. Only the enabled TC mechanisms are provided with context events of the TC multi-mechanism input topology and maintain individual virtual topologies. The virtual topology of the TC multi-mechanism is aggregated from the virtual topologies of the enabled TC mechanisms. □

The selected aggregation strategy must ensure that the virtual topology of the TC multi-mechanism fulfills the consistency properties. One possible aggregation strategy is to activate a link in the virtual topology of the TC multi-mechanism if it is active in at least one virtual topology of an enabled TC mechanism. Using this aggregation strategy, the aggregated virtual topology is connected if at least one of the virtual topologies of the enabled TC mechanisms is connected. The following definition subsumes the possible actions of a TC multi-mechanism reconfiguration logic.

Definition 2.34 (Topology control transition). A *topology control transition* (TC transition) is the execution of one or more of the following actions: (i) reconfiguration of

the virtual topology aggregation strategy, (ii) enabling of a TC mechanism, (iii) disabling of a TC mechanism, or (iv) reconfiguration of a TC mechanism. A TC transition is carried out by the TC multi-mechanism reconfiguration logic. \square

This proposed TC multi-mechanism architecture implements the proxy pattern [71]. The interfaces to provider and consumer mechanisms are identical, which allows to exchange the current TC mechanism or even combine the decisions of multiple TC mechanisms transparently. Given that multiple TC mechanisms are enabled within a TC multi-mechanism, a possible virtual topology aggregation strategy is to create the union of the individual virtual topologies. This strategy may also be employed if one TC mechanism has been enabled just recently with the goal to replace another TC mechanism. Simply disabling the latter TC mechanism and enabling the new TC mechanism may result in a disruption of higher-layer mechanisms (e.g., the routing).

We introduced the concepts of TC multi-mechanisms and TC transitions to sketch how to integrate the developed TC mechanisms into an architecture of a modern WSN. Our focus during the subsequent chapters lies on developing individual correct TC mechanisms.

3

SELECTION OF SPECIFICATION LANGUAGES

In Chapter 2, we introduced terminology from the TC domain. In this chapter, we introduce the modeling languages that allow us to set up a model-driven methodology for developing correct TC mechanisms. We concentrate on the development of a single TC mechanism and, for now, neglect the details of the TC (multi-)mechanism control logic. We refer to the specification of a TC algorithm and its corresponding context event handlers collectively as *TC mechanism specification*, and to the topology of a TC mechanism simply as topology. Figure 3.1 locates the role of this chapter in the entire TC algorithm development process (see also Figure 1.2).

SELECTION CRITERIA AND CHAPTER STRUCTURE The specification languages that we introduce in this chapter cover the following four aspects of the target domain.

First, as a common basis for the entire specification, we want to characterize the set (or more technically, the language) of structurally valid topologies. A topology is structurally valid if it represents a topology in the sense of Definition 2.11 (i.e., an attributed simple graph). Metamodeling, introduced in Section 3.1, is a graph-based technique to describe the set of structurally valid topologies in terms of a single model, called the topology metamodel.

Second, we want to specify the TC-algorithm-specific conditions and further consistency properties (e.g., connectivity) declaratively. We refer to the TC-algorithm-specific conditions and consistency properties collectively as the *consistency specification* in the following. The set of topologies that fulfill the consistency specification is a subset of

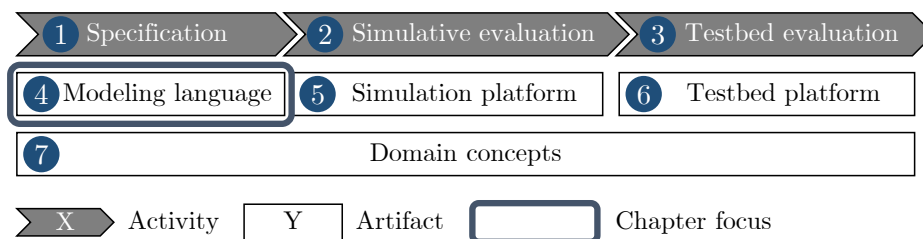


Figure 3.1: Location of Chapter 3 in TC algorithm development process

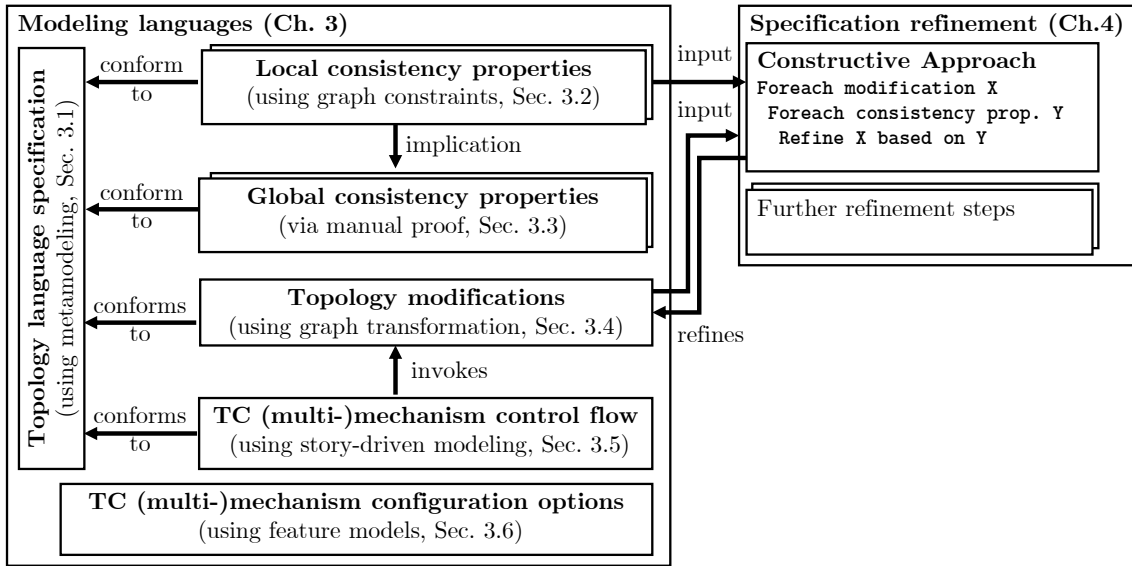


Figure 3.2: Connection of modeling languages and specification refinement (Chapter 4)

structurally valid topologies. Graph constraints, introduced in Section 3.2, capture local consistency properties, which can be expressed using first-order logic in terms of required or forbidden subtopologies (e.g., a triangle for which the kTC-specific condition holds). In Section 3.3, we illustrate how to handle global consistency properties, which cannot be expressed using graph constraints. The required preservation of connectivity is an important example of such a property [59].

Third, we want to specify possible types of modifications that the TC mechanism applies to the topology. Graph transformation rules, introduced in Section 3.4, capture possible types of modifications as pairs of pre- and postconditions (e.g., with an unmarked link as precondition and an active link as postcondition). An application of a graph transformation rule replaces an occurrence of its precondition with an occurrence of its postcondition.

Fourth, we want to specify the control flow of topology modifications in an imperative way. In Section 3.5, we propose to use story-driven modeling [60] for this purpose, a dialect of UML activity diagrams [78] that supports applying graph transformation rules as one type of action. A declarative specification, in our context, describes whether or not a given state of the topology is correct and what actions are available to modify the topology. An imperative specification describes the steps that are necessary to arrive at a particular topology.

To be able to derive a TC mechanism specification that fulfills the consistency specification, we must be able to check whether a given modification may or may not violate the consistency specification. This analysis shall be applicable not only to a concrete situation (i.e., a particular modification of a particular topology), but also on the level

of topology modifications and consistency properties. If a particular modification type violates the consistency specification, we need to be able to adjust this modification type systematically to ensure that the consistency specification is no longer violated. One state-of-the-art static analysis technique that identifies consistency-violating modifications and returns appropriate refinement instructions is the *constructive approach* by Heckel and Wagner [91]. Generally speaking, the constructive approach is an iterative algorithm that generates additional preconditions [47] for each possible type of modification. The constructive approach partly determines our choice of modeling techniques: graph constraints for specifying local consistency properties and graph transformation rules for specifying topology modifications. Our choice of story-driven modeling for composing topology modifications is not affected by the constructive approach, but rather motivated by its availability in the model-driven engineering tool `EMOFLON`.

In Section 3.6, we illustrate how the configuration options of individual TC mechanisms and entire TC multi-mechanisms can be specified using feature diagrams. Feature diagrams also allow for describing interdependencies of configuration options (e.g., to capture which TC mechanisms are suitable in which application scenarios).

Figure 3.2 illustrates the role of the modeling techniques presented in this chapter in the context of the constructive approach, which forms the basis of Chapter 4. As indicated in the figure, we propose further refinement steps that build on the constructive approach in Chapter 4. In Section 3.7, we conclude this chapter with a survey of related work on modeling approaches in communication systems engineering.

3.1 METAMODELING

Metamodeling is a technique for specifying possible states of a system from a syntactic point of view. In general, a *model* is a representation of another entity or set of entities that fulfills a particular purpose [129]. An entity may be a real-world system or a model that contains additional information compared to the currently considered model. To categorize models according to abstraction levels, we use the Model-Driven Architecture of the Object Management Group [233, Ch. 6]. The *Object Management Group (OMG)*¹ is a consortium, founded in 1989, that aims to foster the application of MDE in industry by defining conceptual and technical standards for MDE. The *Model-Driven Architecture (MDA)* [233, Ch. 12]², first presented by the OMG in 2001, is such a standard and defines different abstraction levels of models and the relations between abstraction levels in a modeling-language-independent way.

According to the MDA, models are categorized into *metalevels* (named M_0 , M_1 , etc.) Figure 3.3 illustrates these metalevels based on a small example. Models on higher metalevels are more abstract than models on lower metalevels. A model on metalevel N (e.g., M_1) *describes* a set of valid models on the next lower metalevel $N - 1$ (e.g., M_0). Conversely, a model on metalevel N is an *instance of* (or *conforms to*) a model on metalevel $N + 1$. An M_0 model is a special case because it is a *representation of* the state of a real-life system (e.g., the state of a WSN). An M_1 model describes a set of valid M_0 models and captures the concepts of a particular target domain (e.g., WSN topologies). In the same way as an M_1 model describes the set of valid M_0 models, we also need to declare what constitutes a valid M_1 model. This is the purpose of an M_2 model, which is often domain independent and contains abstract object-oriented concepts (e.g., classes, attributes, inheritance, associations). Prominent examples of M_2 models are the Unified Modeling Language (UML) [78, 80] and Ecore [241].

Conceptually, the metalevel hierarchy has no upper bound. For practical reasons, however, a model on a sufficiently high metalevel (e.g., Ecore on level M_2 in Figure 3.3) is usually self-descriptive. A *self-descriptive model* is abstract enough to express its own syntax. The proposed metamodeling hierarchy of the MDA is limited to four levels (M_0 to M_3). The universal M_3 model proposed by the MDA is the *Meta-Object Facility (MOF)* [233, Sec 12.2]. However, the high expressiveness of MOF hindered the implementation of MDE tools that should fully conform to the MOF. Therefore, the OMG defined *Essential MOF (EMOF)*, a subset of MOF that the standardization body deemed sufficient for most modeling tools. In contrast to EMOF, Complete MOF (CMOF) comprises all modeling concepts of MOF. The M_2 model of UML is formulated in terms of CMOF, while Ecore is specified in EMOF. Throughout this thesis, we use the Ecore meta-metamodel (M_2), which is the metamodel of the Eclipse Modeling Framework.

¹ OMG page: <https://www.omg.org/> (visited: 2018-09-17)

² MDA page: <https://www.omg.org/mda/> (visited: 2018-09-17)

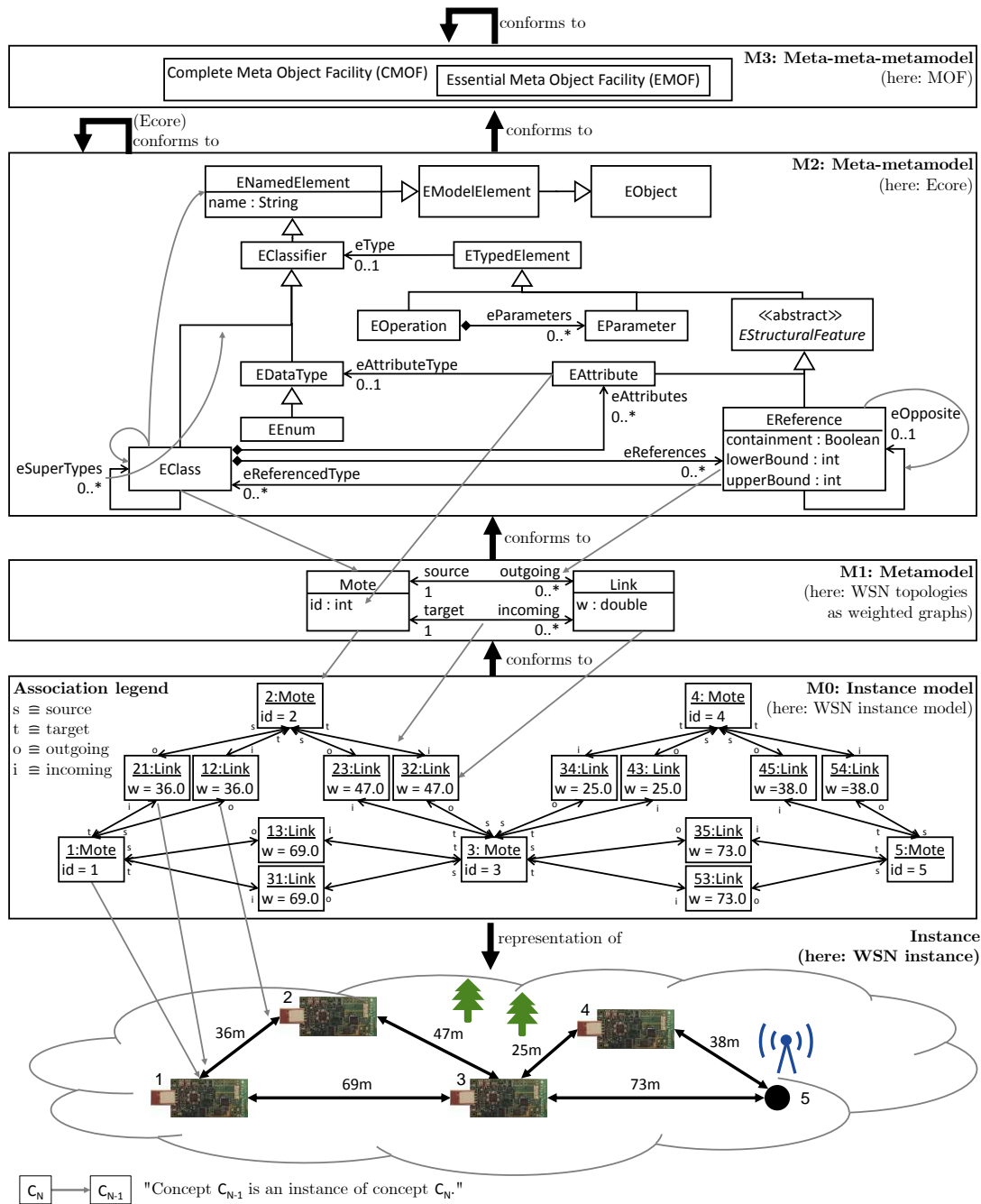


Figure 3.3: Four-layer MDA metamodeling hierarchy with excerpts of Ecore (M2) and the topology metamodel shown in Figure 3.4 (M1). For better readability, association role names (Mo) are abbreviated. Each gray arrow points from an abstract model element (e.g., on M1) to a (more) concrete model element (e.g., Mo).

The *Eclipse Modeling Framework*³ (EMF) [241] is an open-source Eclipse framework for modeling and code generation that constitutes the basis of numerous modeling tools in industry and academia. According to Ed Merks, the lead developer of EMF, Ecore was designed to be the de-facto reference implementation of EMOF [101]. For this reason, Ecore not only conforms to EMOF but is also self-describing.

The MDA terminology deviates from the practice in the MDE research community. In the MDA, an Mo model is called an “instance model”, an M1 model is called a “model”, and an M2 model is called a “metamodel” [233, Fig. 6.2]. In contrast, state-of-the-art MDE papers use the terms “model” for Mo models and “metamodel” for M1 models (e.g., in the following proceedings [98, 215]). For consistency with the related work, we apply the latter naming convention in the following, as captured by the following definition.

Definition 3.1 (Metamodeling terminology). An (*instance*) *model* is a representation of the state of a system at one point in time and resides on metalevel Mo. A *meta-model* is a model that describes the set of all possible instance models and resides on metalevel M1. A *meta-metamodel* is the specification of valid metamodels and resides on metalevel M2. □

The following example illustrates the metamodeling terminology and notation.

Example 3.2 (Metamodeling terminology and notation). Figure 3.3 illustrates the *metamodeling hierarchy* using the terminology introduced in Definition 3.1. A gray arrow indicates a mapping from an abstract to a concrete concept (e.g., from classes of the metamodel (M1) to objects in the instance model (Mo)).

The state of a WSN and its environment at a particular point in time, comprising the state of all communication components, batteries, the physical environment, etc., is indicated by the cloud at the bottom of the figure.

The instance model of the WSN is a directed weighted graph. For instance models, we use the *object diagram notation*, which depicts objects as rectangular boxes and associations between objects as directed arrows. The box is decorated with an *object identifier* (e.g., 1, 12) and the *object type* (e.g., Mote, Link), both underlined and separated by a column. The box of an object lists the properties of the object (e.g., the identifier of a mote or the weight of a link). An arrow in an Mo model represents an *association* and is labeled with a descriptive role name. A double-headed arrow represents a pair of opposite associations (e.g., source and outgoing).

The topology metamodel in this example declares that two types of objects exist: Mote and Link. For metamodels, we use the *class diagram notation*, which depicts each class as box (decorated with the class name) and each association type as arrow. The arrow of an association type is labeled with the descriptive role name and number

³ EMF page: <https://www.eclipse.org/modeling/emf/> (visited: 2018-09-19)

of allowed instances of an association per instance of a class. In this example, a mote (class Mote) has zero or more outgoing and incoming links (specified by the outgoing and incoming association types with multiplicity constraints 0..*). Conversely, a link (class Link) has exactly one source and one target mote (specified by the source and target association types with multiplicity constraints 1 = 1..1). The meta-metamodel is an excerpt of ECORE, which conforms to the EMOF meta-meta-metamodel. EMOF and CMOF are not shown here for conciseness reasons.

The following definition introduces all metamodeling concepts that occur in this thesis. Even though we use ECORE as meta-metamodel in this thesis, the terminology is independent of ECORE. Metamodel elements from the topology metamodel shown in Figure 3.4 serve as examples. This metamodel enriches the metamodel shown in Figure 3.3 by additional concepts (e.g., TC algorithm and context event handlers).

Definition 3.3 (Metamodeling concepts in detail). A *metamodel* is a graph whose nodes are classes and whose directed edges are relations between classes.

A *class* is named after the type of entity that it models (e.g., Mote in Figure 3.4), and contains attributes and operations. An *attribute* represents a property of an entity and possesses a name and a data type. For instance, a Mote instance has an integer-valued id attribute. We restrict attributes to have only primitive types. An *operation* of a class specifies its behavior. For instance, we can invoke the operation handle on any object of type ContextEventHandler. The *signature* of an operation consists of the operation name (handle), return type (e.g., Boolean), and parameter list. Each parameter has a name (e.g., e) and a type (e.g., ContextEvent). An *enumeration* is a special data type with a finite, predefined domain (e.g., State). An enumeration is decorated with the stereotype «Enum» and its box lists its possible values (e.g., {Unmarked, Active, Inactive}).

A *relation* can be an association or an inheritance relation. An *association* is a directed edge that allows to navigate from instances of the source class to instances of the target class (e.g., a mote to its outgoing links). An association is depicted as regular arrow and labeled with a descriptive *role name* (e.g., outgoing) and a *multiplicity constraint* (e.g., 0..*), which imposes a *lower bound* (e.g., 0) and an *upper bound* on the number of instances of the association in a valid model. If no upper bound exists, we use the asterisk symbol (i.e., *). Pairs of associations may be marked as each other's *opposite association*. In a valid model, the instances of opposite association types must exist together. For instance, if an outgoing association from a Mote to a Link instance exists, then a source association must exist from the Link to the Mote instance.

A *containment association* is a special type of associations that represents a *part-of relationship* and runs from a *containee class* (representing the “part”) to a *container class* (representing the “whole”). When an instance of a container class is removed

from a model, all instances of containee classes are removed as well. A containment association is shown as a black diamond at the end of the container class. For instance, Topology is the container class for the containee classes Mote and Link.

An *inheritance relation* points from a *subclass* to its *superclass* (e.g., from MoteEvent to ContextEvent). A subclass possesses all attributes and operations of all its (transitive) superclasses. A class may be a subclass of zero or more superclasses. The inheritance relation of a metamodel must be acyclic. An inheritance relation is depicted as arrow with an empty triangle as head.

A *class is abstract* if it cannot be the declared type of an object in a model (e.g., ContextEvent). Only concrete (i.e., non-abstract) subclasses of an abstract class may occur as object type (e.g., MoteAddition). An *operation is abstract* if it possesses no implementation. A class that contains abstract operations is also abstract. Abstract classes and operations are decorated with the stereotype «abstract». □

The following example illustrates the metamodeling concepts that were introduced in Definition 3.3.

Example 3.4 (Metamodel of running example). Figure 3.4 shows the topology metamodel of our running example. We do not explain each metamodel element in detail, but highlight the intent of the individual parts of the metamodel. We use the primitive types `int`, `double`, `Boolean` of `ECORE` for representing natural numbers, real numbers, and Boolean values.

The classes in the top-left part of the diagram represent the fact that a topology is a weighted graph (with node type Mote and edge type Link). `Mote::id` state that a mote has an identifier. `Link::state` and `Link::weight` represent the weight and state of a link. The *scoping operator* (`::`) indicates to which class an attribute or operation belongs.

The top-right part of the diagram shows the abstract superclass `TopologyControlAlgorithm` for TC algorithms, which specifies that each TC algorithm must provide a parameterless `run` operation. The diagram also shows two subclasses that represent the `kTC` and the `Maxpower` algorithm (`KtcAlgorithm`, `MaxpowerAlgorithm`).

The center part of the diagram shows the considered types of context events. The abstract superclass `ContextEvent` models that each context event is attached to exactly one topology (`ContextEvent::topology`) and that a topology may have zero or more attached pending context events (`Topology::pendingContextEvents`). We represent the affected motes and links in terms of primitive attributes (e.g., `motedId`) that relate to `Mote::id`. We use primitive attributes instead of associations to Mote and Link to keep the context event class hierarchy decoupled from concrete Mote and Link objects. Otherwise, for instance, a `LinkAddition` would require that the source and target mote of the link that shall be created exist, which may not be the case if the corresponding mote addition events are also still pending.

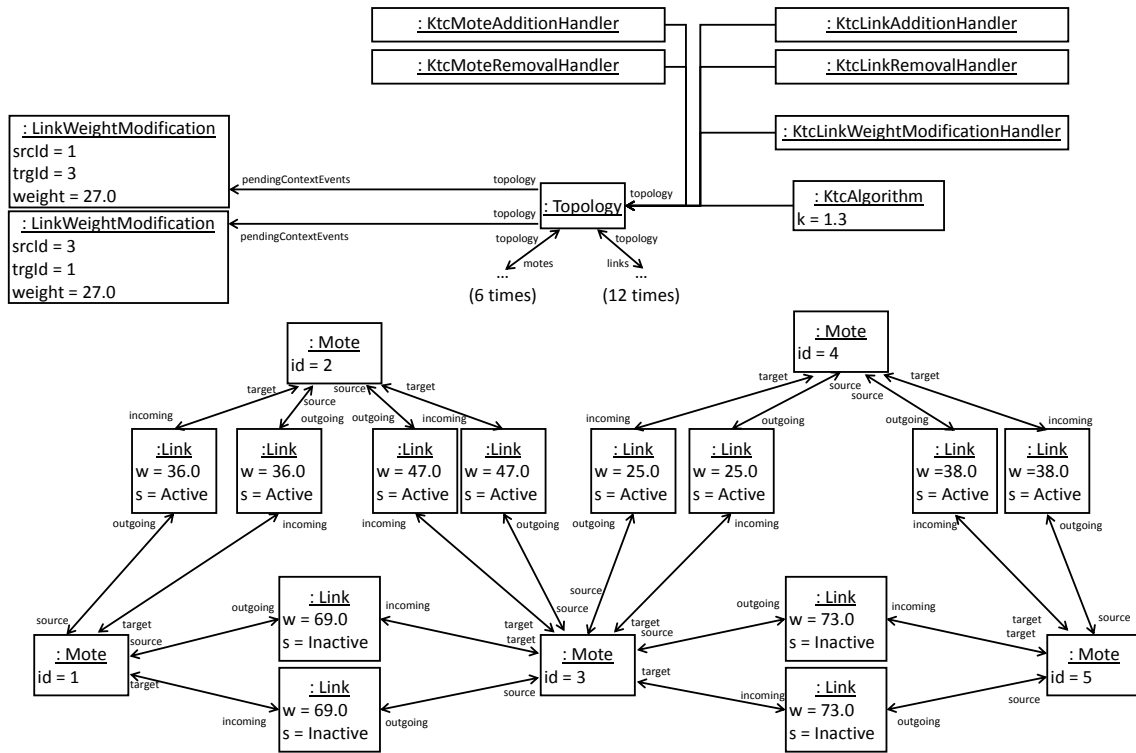
The bottom part of the diagram shows the context event handlers that correspond to the context events. The abstract superclass `ContextEventHandler` reflect the fact that each concrete context event handler has access to the topology (`ContextEventHandler::topology`) and provides a `handle` operation, which accepts a context event as parameter (`e: ContextEvent`) and returns `true` if this context event handler successfully handled the context event. This structure instantiates the visitor pattern [71] (i.e., each subclass of `ContextEventHandler` can visit any type of `ContextEvent`). The subclasses of `ContextEvent` provide refined operations that accept the respective subclass instances of `ContextEvent` (e.g., `MoteAddition::handleMoteAddition`).

The topology metamodel in Figure 3.4 is suitable for neighbor-based TC algorithms, which only rely on the graph structure and the link weights. To support, for instance, location-based TC algorithms, we need to add metamodel elements (e.g., `Mote::longitude` and `Mote::latitude` to represent the geographic position of a mote).

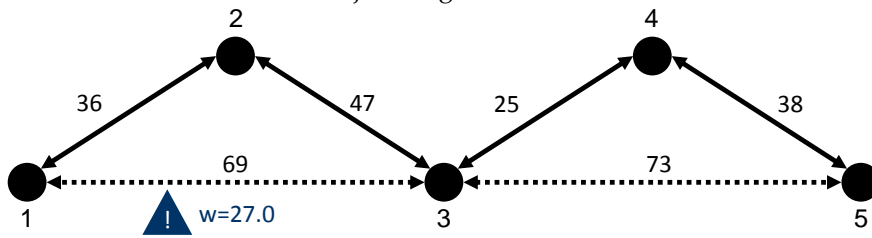
In the following example, we compare the object diagram notation with the compact notation, which we introduced for our running example in Chapter 2.

Example 3.5 (Topology model and notation). Figure 3.5 shows a sample topology model in object diagram and compact notation. The compact notation corresponds to the notation that we introduced for topologies in Chapter 2. Two link-weight modification events are attached to the topology, specifying that the weights of the links ℓ_{13} and ℓ_{31} will soon decrease from 69 to 27. The enabled TC algorithm is kTC with $k = 1.3$. Five context event handlers are connected to the topology to handle the five context event types.

The compact notation only contains information about the partial model consisting of Mote and Link objects (with their properties and associations) and the pending context events. In contrast, the object diagram notation represents all (mote and link) properties in a uniform and easily extensible way (e.g., the required additional attributes for location-based TC algorithms). However, a crucial drawback of the object diagram notation is its verbosity, that is, its extensive space consumption and the arguably low readability already for the small example discussed in Example 3.5. If we had drawn the six topology-nodes and the twelve topology-links associations of the sample model, the diagram would have become hardly understandable. Therefore, we use the compact notation whenever possible to represent topology models in the following. We assume that exactly one topology exists and provide information about the enabled TC algorithm separately.



(a) Object diagram notation



(b) Compact notation

Figure 3.5: Example: Topology with pending link-weight modification events at ℓ_{13} and ℓ_{31} , TC algorithm kTC with $k = 1.3$, and corresponding context event handlers (object diagram notation only)

3.2 LOCAL CONSISTENCY PROPERTIES

In this section, we describe how graph constraints [83, 91] can be used to specify local consistency properties. A *local consistency property* is a consistency property that can be expressed in terms of required and forbidden subtopologies together with constraints over the attributes of the involved motes and links. We distinguish between two types of local consistency properties. First, *structural consistency properties* complement the metamodel by expressing all those consistency properties that cannot be expressed using the metamodeling techniques introduced in Definition 3.3. For example, a topology should contain neither loops nor parallel links, which is not reflected by the topology metamodel in Figure 3.4. Second, *TC-algorithm-specific consistency properties* can also be local (e.g., the required triangle for kTC) that justifies the inactivation of a particular link). The following example provides precise definitions of the local consistency properties of the running example.

Example 3.6 (Local consistency properties). The following first-order logic expressions state sample structural and kTC-specific local consistency properties. In general, local consistency properties can refer to any elements of the model, not only to the topology-related parts. In our running example, G_T is a topology model that conforms to the metamodel in Figure 3.4. In the following, we interpret a topology model as directed graph $G_T = (V_T, E_T)$. The node set V_T of G_T is the set of Mote objects, and the edge set E_T is the set of Link objects. The src and trg functions are given by the source-outgoing and target-incoming associations in the topology model. For instance, for a given link l , $\text{src}(l)$ is the unique mote that is reachable from l via the source association, and $\text{trg}(l)$ is the unique mote that is reachable from l via the target association. Additionally, we interpret the id , weight , and state attributes of Mote and Link instances as properties of the motes and links in G_T .

We begin with the two structural local consistency properties that ensure that a topology is a simple graph. The following expression states that a valid topology contains no loops:

$$\forall G_T = (V_T, E_T) : \nexists l \in E_T : \text{src}(l) = \text{trg}(l). \quad (3.1)$$

The following expression states that a valid topology contains no parallel links:

$$\forall G_T = (V_T, E_T) : \nexists l_1, l_2 \in E_T : l_1 \neq l_2 \wedge \text{src}(l_1) = \text{src}(l_2) \wedge \text{trg}(l_1) = \text{trg}(l_2). \quad (3.2)$$

The local consistency properties of kTC describe under which circumstances a link may be active or inactive. In accordance with the practice in the TC literature,

we begin with the consistency property for inactive links. Typically, the local consistency specification of an inactive link requires that a “witness” exists in the topology for each inactive link. For kTC, the witness for a link l_1 is a triangle (l_1, l_2, l_3) (see $\varphi_{\text{triangle}}$) in which $l_1, l_2,$ and l_3 fulfill additional attribute constraints (see φ_{kTC}).

$$\forall G_T = (V_T, E_T) : \forall l_1 \in E_T : s(l_1) = I \Rightarrow \exists l_2, l_3 \in E_T : \\ \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{\text{kTC}}(w(l_1), w(l_2), w(l_3), \text{id}(\text{src}(l_1)), \text{id}(\text{trg}(l_1)), \text{id}(\text{trg}(l_2)))$$

with

$$\varphi_{\text{triangle}}(l_1, l_2, l_3) \Leftrightarrow \text{src}(l_1) = \text{src}(l_2) \wedge \text{src}(l_3) = \text{trg}(l_2) \wedge \text{trg}(l_3) = \text{trg}(l_1), \quad (3.3)$$

$$\varphi_{\text{kTC}}(w_{XY}, w_{XZ}, w_{ZY}, X, Y, Z) :\Leftrightarrow$$

$$(w_{XY}, X, Y) \succ \max((w_{XZ}, X, Z), (w_{ZY}, Z, Y)) \wedge$$

$$(w_{XY}, X, Y) \succ k \cdot \min((w_{XZ}, X, Z), (w_{ZY}, Z, Y))$$

$$\Leftrightarrow$$

$$w'_{XY} > \max(w'_{XZ}, w'_{ZY}) \wedge w'_{XY} > k \cdot \min(w'_{XZ}, w'_{ZY}). \quad (3.4)$$

The signature of the kTC-specific predicate φ_{kTC} contains the implicit assumption that the involved links form a triangle (see $\varphi_{\text{triangle}}$ predicate). The operator \prec refers to the descending canonical ordering on $\mathbb{R} \times \mathbb{N} \times \mathbb{N}$ (see Definition 2.25).

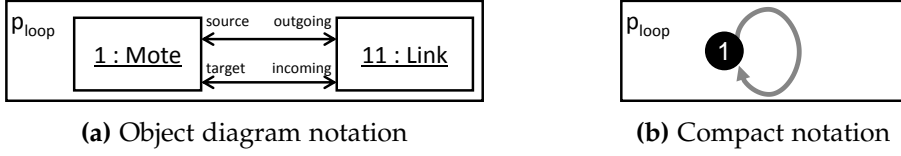
The kTC-specific condition for active links is complementary to Equation (3.4) in the sense that it forbids all those links to be active that are eligible for inactivation.

$$\forall G_T = (V_T, E_T) : \nexists l_1, l_2, l_3 \in E_T : s_1 = A \wedge \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{\text{kTC}}(l_1, l_2, l_3) \quad (3.5)$$

In Equations (3.4) and (3.5), it is not necessary to require that l_1, l_2, l_3 are pairwise different because φ_{kTC} can only be true if all three links are distinct.

A local consistency property is either positive or negative. A *positive local consistency property* consists of a premise part and a conclusion part. Each occurrence of the premise part in the local view (e.g., an inactive link l_1) must be extensible to an occurrence of the conclusion part (e.g., a witnessing triangle (l_1, l_2, l_3) for l_1). A negative condition consists of a premise part that may not occur within the topology (e.g., two parallel links). The subsequent definitions lead us to the introduction of graph constraints, a formalism that mirrors exactly this premise-conclusion structure. We begin with the definition of graph patterns, the building blocks of graph constraints.

Definition 3.7 (Graph pattern [56, 91, 214]). Let $G_{\text{MM}} = (V_{\text{MM}}, E_{\text{MM}})$ be a metamodel with a classes V_{MM} and association types E_{MM} . A (graph) pattern p relative to a metamodel G_{MM} consists of (i) a *pattern graph* $G_p = (V_p, E_p)$, whose nodes and edges are called *object variables* and *association variables* and serve as placeholders for objects and associations, respectively, (ii) a function type $:(V_p \cup E_p) \rightarrow (V_{\text{MM}} \cup E_{\text{MM}})$

Figure 3.6: Example: Loop pattern p_{loop}

that assigns to each variable a (*variable*) *type*, which is a class for object variables (i.e., $m|_{V_p} \subseteq V_{MM}^a$) and an association type for association variables (i.e., $m|_{E_p} \subseteq E_{MM}$), and (iii) a set of *attribute constraints* over the attributes of the object variable types. An attribute constraint consists of an operator and a number of parameters, where the parameter count must be consistent with the operator. All attribute constraints of a pattern are conjoined (i.e., implicitly connected by a logic conjunction). The typing function must preserve the source and target classes of association types:

$$\forall v_{12} \in E_p : \text{type}(\text{src}_p(v_{12})) = \text{src}_{MM}(\text{type}(v_{12})) \wedge \text{type}(\text{trg}_p(v_{12})) = \text{trg}_{MM}(\text{type}(v_{12}))$$

□

a For a function $f : X \rightarrow Y$ and a subset $W \subseteq X$, the function $f|_W$ is the *restriction of f on W* : $f|_W : W \rightarrow Y$

Typical operators are *relational operators* (e.g., $<$, $>$, \leq , \geq , $=$, \neq), but we also allow for user-specified operators, for which an interpretation must be given as first-order logic expression (similar to φ_{kTC} in Example 3.6).

Definition 3.8 (Topology pattern). A *topology pattern* is a special type of pattern whose metamodel is the topology metamodel. An object variables of a topology pattern has either Mote or Link as type and is called *mote variable* or *link variable*, respectively. □

The notion of topology patterns serves to introduce a concise terminology for our running example. In general, we use the symbol v to denote a pattern variable with mote variables having a one-digit index (e.g., v_1), and link variables having a two-digit index (e.g., v_{12}).

Example 3.9 (Structural consistency patterns). Figure 3.6 shows the pattern that describes a loop in the topology. Similar to Section 3.1, we introduce a generic object diagram notation (Figure 3.6a) and a compact notation (Figure 3.6b), which is tailored to our running example and simplifies understanding the pattern structure. The pattern consists of two object variables having types Mote and Link, respectively, and of four association variables having types Mote::outgoing, Mote::incoming, Link::source, and Link::target, respectively. The pattern has no attribute constraints

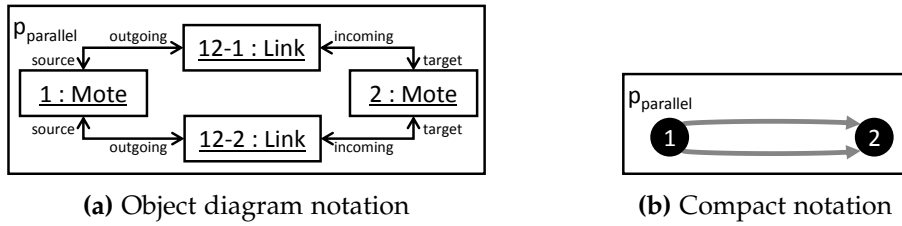


Figure 3.7: Example: Parallel-links pattern p_{parallel}

and is also a topology pattern with one mote variable and one link variable that has a identical source and target motes. This topology-specific perspective is also reflected by the compact notation shown in Figure 3.6b. The link variables are depicted in gray because the pattern does not impose any constraints on the link state. The compact notation usually shows the object identifiers of mote variables and omits the object identifiers of link variables.

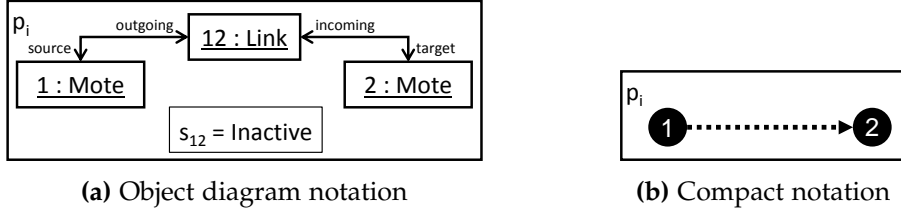
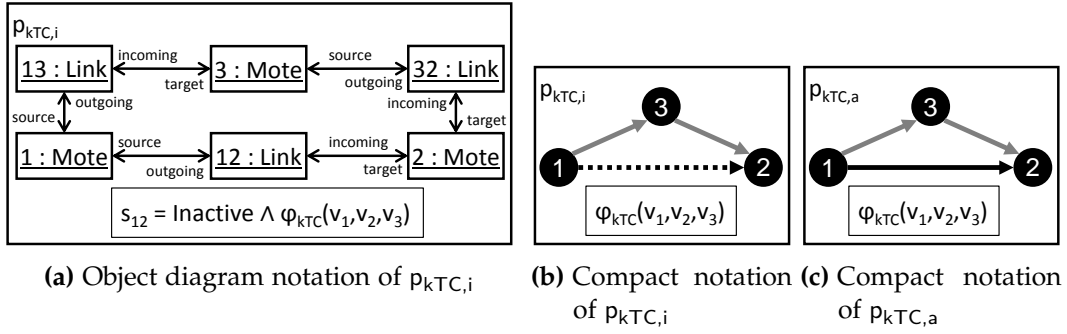
Figure 3.7 shows the second structural consistency pattern, which represents two parallel links. This topology pattern has four object variables, eight association variables, two mote variables and two link variables.

Example 3.10 (kTC-specific patterns). The inactive-link pattern p_i shown in Figure 3.8 represents a single inactive link. We collect the attribute constraints of a pattern in a separate box. The compact notation contains no such box because we encode attribute constraints that relate to the link state into the stroke and color of the corresponding arrows (e.g., dotted-black for inactive links). The pattern $p_{kTC,i}$ shown in Figure 3.9 represents a triangle in which the link represented by the link variable e_{12} should be inactive according to kTC. Finally, Figure 3.9c shows the complementary pattern $p_{kTC,a}$, which matches a forbidden triangle, that is, an active link in a triangle that should actually be inactive according to the kTC-specific conditions. The only difference between $p_{kTC,i}$ and $p_{kTC,a}$ lies in the link state constraints related to e_{12} .

The indices of mote and link variables in attribute constraint boxes relate to the object identifiers of the mote and link variables, and not to the mote identifiers. In the described patterns and in subsequent examples, we use the following abbreviated form of φ_{kTC} (introduced in Example 3.6) as shorthand notation for the kTC-specific attribute constraints:

$$\varphi_{kTC}(v_1, v_2, v_3) := \varphi_{kTC}(w(v_{12}), w(v_{13}), w(v_{32}), id(v_1), id(v_2), id(v_3)) \quad (3.6)$$

The following definition describes how the occurrence of a pattern in a model is represented.

Figure 3.8: Example: Inactive-link pattern p_i Figure 3.9: Example: kTC patterns $p_{kTC,i}$ and $p_{kTC,a}$

Definition 3.11 (Pattern match). Let p be a pattern with pattern graph $G_p = (V_p, E_p)$, $G_M = (V_M, E_M)$ be a model, where V_M is the set of objects and E_M is the set of associations, and p and G_M conform to the same metamodel MM . A *match* m of a pattern p in a model G_M is a mapping from the object variables V_p and association variables E_p of p to the objects and associations of G_M that preserves source and target objects of associations:

$$m : (V_p \cup E_p) \rightarrow (V_M \cup E_M)$$

$$m|_{V_p} \subseteq V_M$$

$$m|_{E_p} \subseteq E_M$$

$$\forall v_{12} \in E_p : m(\text{src}_p(v_{12})) = \text{src}_{G_M}(m(v_{12})) \wedge m(\text{trg}_p(v_{12})) = \text{trg}_{G_M}(m(v_{12}))$$

Additionally, (i) m must respect the variable type mappings, and (ii) the attribute constraints of p must be fulfilled when replacing the variables with the respective objects and association in the image of m . \square

A pattern match can be *injective* or *noninjective*. In the first case, distinct pattern variables are mapped to distinct objects. In the latter, general case, distinct pattern variables may be mapped to the same object. In this thesis, we apply injective matching. One reason for this selection is that the constructive approach requires injective matching. Still, we

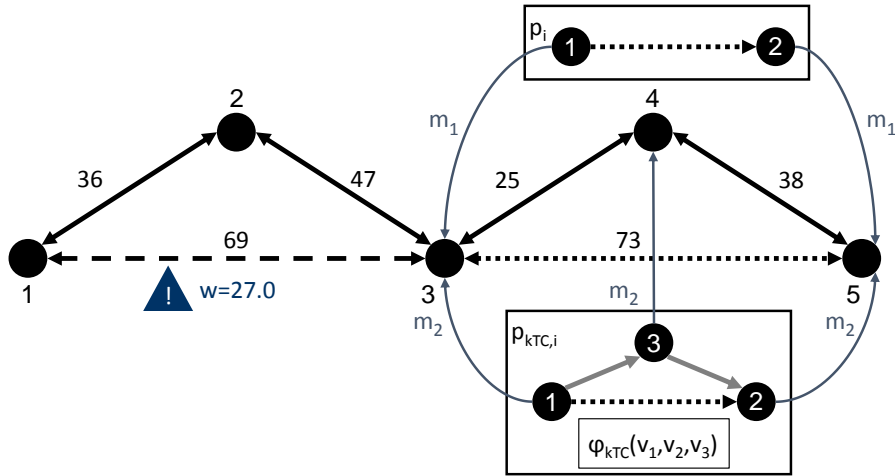


Figure 3.10: Example: Matches m_1 and m_2 of inactive-link and kTC pattern in sample topology

also think that an injective match is more intuitive to interpret for the user because no two variable mappings may coalesce to the same object in the model.

Example 3.12 (Pattern match). Figure 3.10 shows two matches m_1 and m_2 of the inactive-link pattern p_i and $p_{kTC,i}$ in the sample topology ($k = 1.3$). The mappings from mote variables to motes are depicted as (blue-gray) arrows that are labeled with the matches. The corresponding mappings from link variables to links are omitted for conciseness reasons. Match m_1 maps mote variables v_1 and v_2 of p_i to motes n_3 and n_4 , respectively, and link variable v_{12} to link l_{12} . Match m_2 additionally maps mote variable v_3 to the mote n_4 and link variables v_{13} and v_{32} to links l_{34} and l_{45} , respectively.

The following definitions capture when one pattern (or match) can be considered to be a part of another pattern (or match).

Definition 3.13 (Pattern extension). A pattern p_2 is an *extension of a pattern* p_1 if the pattern graph of p_1 is a subgraph of p_2 and if the restriction of any match of p_2 to the variables of p_1 fulfills the attribute constraints of p_1 . \square

Definition 3.14 (Match extension). Let G_M be a model, p_1 and p_2 be patterns and p_2 extends p_1 . A match m_2 of p_2 in G_M is an *extension of a match* m_1 of p_1 in G_M if the restriction of m_2 to the variables of p_1 is identical to m_1 : More formally,

$$m_2|_{(V_{p_1} \cup E_{p_1})} = m_1.$$

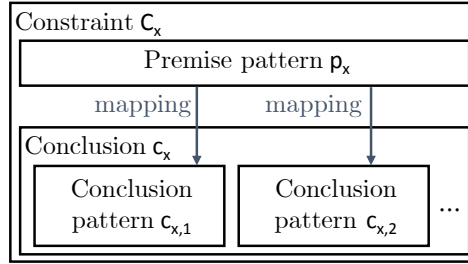


Figure 3.11: Structure of a graph constraint. The conclusion c_x may be empty. □

Example 3.15 (Pattern and match extension). The pattern $p_{kTC,i}$ (Figure 3.9) extends the inactive-link pattern p_i (Figure 3.8) by one additional mote variable, two link variables, and the two attribute constraints related to w' . Match m_2 in Figure 3.10 extends m_1 by a mapping from mote variable v_3 to mote n_4 .

The definition of graph patterns and pattern matches allow us to specify required and forbidden structures in the topology. The definitions of pattern and match extensions are the basis for specifying the premise-conclusion structure of local consistency properties using graph constraints, as described by the following definition.

Definition 3.16 (Graph constraint [91]). A *graph constraint* C_x consists of one *premise pattern* p_x (short: *premise*) and a (potentially empty) set of *conclusion patterns* $c_{x,1}, c_{x,2}, \dots$, which are collectively called the *conclusion* c_x of C_x (Figure 3.11). Each conclusion pattern is an extension of the premise. The constraint C_x is called a *positive constraint* if the conclusion contains at least one pattern. Otherwise, the constraint is called a *negative constraint*. □

A graph constraint characterizes whether a given model is consistent or not.

Definition 3.17 (Fulfillment of graph constraint). A *graph constraint* C_x is *fulfilled on a model* G_M if each match of the premise p_x in G_M can be extended to a match of at least one conclusion pattern. In this case, we also say that *the model* G_M *fulfills the graph constraint* C_x . □

This definition also sheds light on the dichotomy into positive and negative graph constraints (Definition 3.16). A negative graph constraint *forbids* the existence of certain submodels (described by its premise pattern), whereas a positive graph constraint *requires* the existence of a submodel (given each conclusion) if another, typically smaller submodel exists (i.e., the premise pattern). The following definition lifts the notion of fulfillment of individual graph constraints to sets of graph constraints.

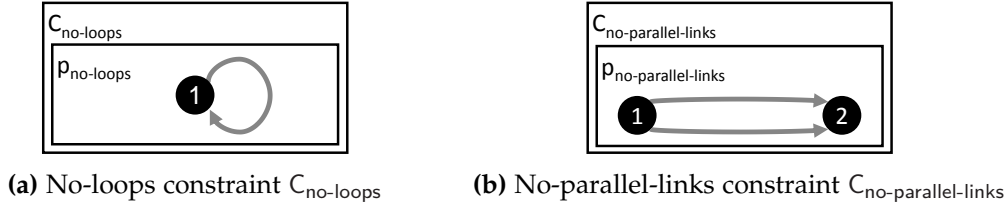


Figure 3.12: Example: Structural constraints $C_{\text{no-loops}}$ and $C_{\text{no-parallel-links}}$

Definition 3.18 (Consistency w.r.t. constraint sets). A model is *consistent w.r.t. a set of graph constraints* \mathcal{C} if it fulfills all constraints in $\mathcal{C} \in \mathcal{C}$. \square

The subsequent examples illustrate the concepts graph constraints, constraint fulfillment, and consistency. The shown graph constraints build on the already defined patterns. For consistency with the naming conventions of the premise and conclusion (patterns) inside a graph constraint, we renamed the pattern accordingly. We begin with an example that shows how to specify that topologies structurally valid.

Example 3.19 (Structural constraints). Figure 3.12 shows the two structural graph constraints $C_{\text{no-loops}}$ and $C_{\text{no-parallel-links}}$ that express the structural local consistency properties Equation (3.1) and Equation (3.2). The no-loops constraint $C_{\text{no-loops}}$ is a negative constraint that forbids loops in the topology. The no-parallel-links constraint $C_{\text{no-parallel-links}}$ is a negative constraint that forbids parallel links in the topology. The premises $p_{\text{no-loops}}$ and $p_{\text{no-parallel-links}}$ are equivalent to the patterns p_{loop} and p_{parallel} . A *topology is structurally consistent* if it is consistent w.r.t. the constraint set $\mathcal{C}_{\text{T}} = \{C_{\text{no-loops}}, C_{\text{no-parallel-links}}\}$.

In the next example, we provide the two graph constraints that specify the TC-algorithm-specific local consistency properties of kTC.

Example 3.20 (kTC-specific constraints). Figure 3.13 shows the two graph constraints that specify when a link may be inactive (Figure 3.13a) and may not be active (Figure 3.13b). According to the inactive-link constraint $C_{\text{kTC},i}$, each inactive link v_{12} (see $p_{\text{kTC},i}$) must be part of the triangle that fulfills the kTC condition (see $c_{\text{kTC},i,1}$). The premise $p_{\text{kTC},i}$ is identical to p_i in Figure 3.8 and the sole conclusion pattern $c_{\text{kTC},i,1}$ is equivalent to $p_{\text{kTC},i}$ in Figure 3.9. The required mapping from premise to conclusion patterns is given via the object variable identifiers. For example, v_1 of $p_{\text{kTC},i}$ is mapped to v_1 of $c_{\text{kTC},i,1}$. The active-link constraint $C_{\text{kTC},a}$ (Figure 3.13b) is a negative constraint that specifies that no active link may be part of a triangle that fulfills the kTC-specific condition.

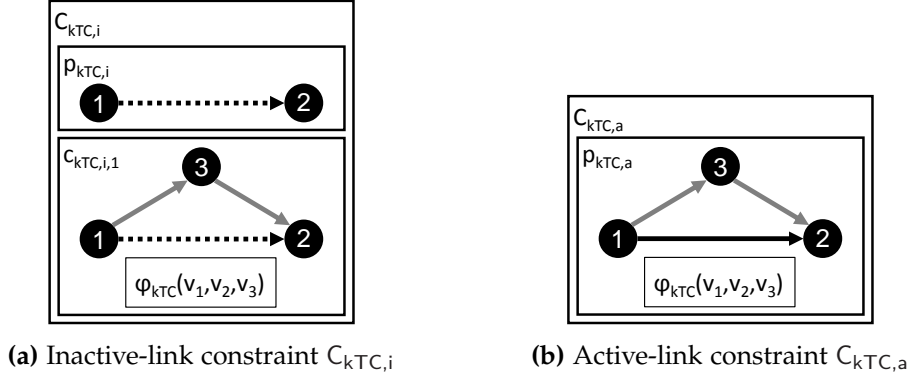


Figure 3.13: Example: kTC -specific graph constraints $C_{kTC,i}$ and $C_{kTC,a}$

A topology is *weakly consistent w.r.t. kTC* if it is consistent w.r.t. the constraint set $\mathcal{C}_W = \{C_{kTC,i}, C_{kTC,a}\} \cup \mathcal{C}_T$. This notion of consistency is “weak” because it only refers to the marked links in the topology. This means that a fully unmarked topology is weakly consistent but the virtual topology would still contain all links of the input topology. Weak consistency is an invariant of the TC algorithm and context event handler specification.

To obtain a stronger notion of consistency, we introduce the two equivalent *marking constraints* shown in Figure 3.14. The no-unmarked-links constraint C_u shown in Figure 3.14a requires that there are no unmarked links. The only-marked-links constraint C_u shown in Figure 3.14b requires that each link in the topology is either active or inactive. Both constraints are equivalent because State has finite domain. In subsequent examples, we use the no-unmarked-links constraint C_u .

A topology is *strongly consistent w.r.t. kTC* if it is consistent w.r.t. the constraint set $\mathcal{C}_S = \{C_{kTC,i}, C_{kTC,a}, C_u\} \cup \mathcal{C}_T = \{C_u\} \cup \mathcal{C}_W$. This means that a topology may only be strongly consistent if it is weakly consistent and all links are marked. We require strong consistency as a postcondition of the TC algorithm. If the topology is dynamic, which is the usual case, it is impossible to maintain strong consistency permanently because context events lead to new unmarked links in the topology.

The topology shown in Figure 3.10 is weakly consistent (for kTC with $k = 1,3$) because it fulfills all constraints in \mathcal{C}_W , but it is not strongly consistent because matches of the premise of the no-unmarked-links constraint C_u exist at l_{13} and l_{31} . If l_{13} and l_{31} were inactive, the topology would be strongly consistent.

We conclude the examples of graph constraints and consistency with the Maxpower-specific consistency constraints.

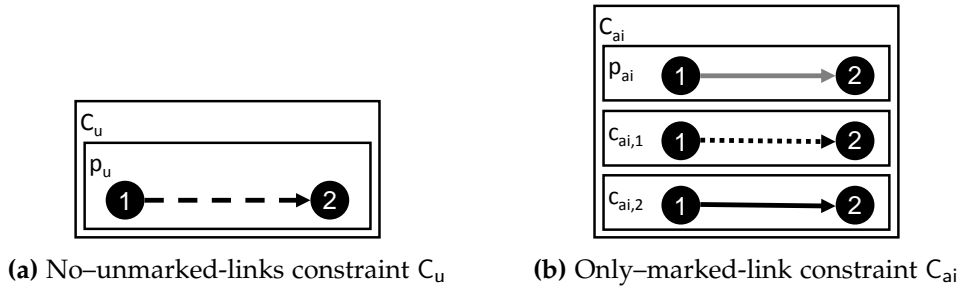


Figure 3.14: Example: Equivalent marking constraints C_u and C_{ai}

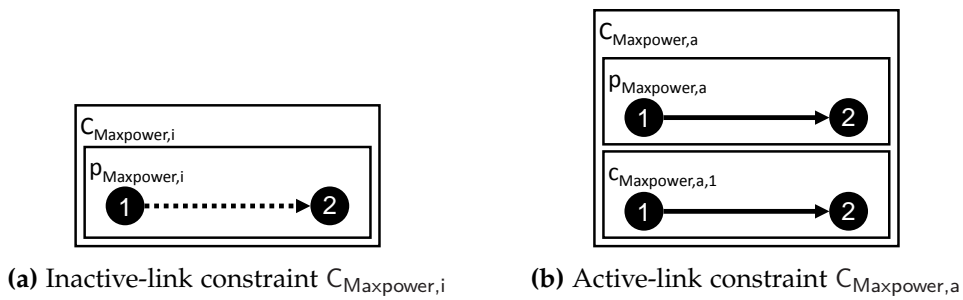


Figure 3.15: Example: Maxpower-specific graph constraints $C_{Maxpower,i}$ and $C_{Maxpower,a}$.

Example 3.21 (Maxpower-specific constraint). Figure 3.15 shows the consistency constraints of the Maxpower algorithm. Maxpower requires that each link in the topology is active. This requirement is specified by the inactive-link constraint $C_{Maxpower,i}$ (Figure 3.15a). The the active-link constraint $C_{Maxpower,a}$ (Figure 3.15b) is always fulfilled (due to its equal premise and conclusion patterns) and only shown here for consistency with the other examples.

The three levels of consistency, introduced in Example 3.19 and Example 3.20, are increasingly stricter. Structural consistency (Example 3.19) can be understood as a minimal technical requirement for structurally valid topologies. Weak and strong consistency (Example 3.20), in contrast, both imply (we could also say, rely on) structural consistency and are specific to the TC algorithm under construction. The following definition generalizes the notion of (structural, weak, and strong) consistency.

Definition 3.22 (Levels of consistency). We define *three levels of consistency* (Table 3.1): structural consistency, weak consistency, and strong consistency. Strong consistency implies weak consistency, and weak consistency implies structural consistency. A topology is *structurally consistent* if it conforms to the current topology metamodel and fulfills auxiliary consistency properties that only depend on the application do-

Table 3.1: Summary of consistency levels for TC mechanisms. The first column refers to the three levels of consistency and whether the particular consistency level is valid for the domain (e.g., TC algorithms in general) or for a particular algorithm (e.g., kTC). The second column specifies whether the respective constraints need to hold permanently (i.e., as invariant) or only at certain points in time (e.g., as postcondition). The third column lists the constraints in the constraint set of the given consistency level.

Level	Scope	Constraint set
structural (domain)	invariant of TC algorithm and context event handlers	$\mathcal{C}_T = \{C_{\text{no-loops}}, C_{\text{no-parallel-links}}\}$
weak (algorithm)	invariant of TC algorithm and context event handlers	$\mathcal{C}_W = \{C_{\text{no-loops}}, C_{\text{no-parallel-links}}, C_a, C_i\}$
strong (algorithm)	postcondition of TC algorithm	$\mathcal{C}_S = \{C_{\text{no-loops}}, C_{\text{no-parallel-links}}, C_a, C_i, C_{ai}/C_u\}$

main (i.e., not on the algorithm under development). A topology is *weakly consistent* if it fulfills structural consistency and all marked links fulfill the TC-algorithm-specific conditions. A topology is *strongly consistent* if it fulfills weak consistency and all links are marked. \square

In the following, we assume that weak consistency can be formulated in terms of two graph constraints: the active-link constraint C_a and the inactive-link constraint C_i . Still, our approach is not limited to this assumption.

3.3 GLOBAL CONSISTENCY PROPERTIES

Graph constraints are suitable for specifying consistency properties that can be expressed in first-order logic. Now, we illustrate how to handle *global consistency properties*, which cannot be expressed in first-order logic with attribute constraints.

An important global consistency property of TC algorithms is that the virtual topology must be connected as long as the input topology is connected. If the input topology is disconnected already, the virtual topology is disconnected as well because the virtual topology is a subgraph of the input topology. Unfortunately, connectivity is a graph property that cannot be expressed in first-order logic [59]. Graph constraints, as introduced in Definition 3.16, are a special type of nested graph constraints with one nesting level. We refer to the former type of graph constraints as *non-nested graph constraints* if necessary in the following. Nested graph constraints are as expressive as first-order logic [83]. Consequently, non-nested graph constraints are less expressive than first-order logic, which makes expressing connectivity using (either non-nested or nested) graph constraints impossible.

We require that the TC algorithm developer proves the following claim: If the topology fulfills weak consistency, all consistency properties are fulfilled. The following example illustrates how to prove of preserved connectivity for kTC.

Example 3.23 (Proof of connectivity preservation for kTC). For our running example, we have to show the following claim.

Claim: The virtual topology is connected if the input topology is connected and the TC mechanism topology is weakly consistent w.r.t. the kTC-specific graph constraints.

Proof sketch: The virtual topology of a TC mechanism is the AU-view of the TC mechanism topology and, therefore, is connected if each pair of motes is connected by an AU-path (i.e., a path of active or unmarked links) in the TC mechanism topology (see Definition 2.6). The mote and link sets of the input topology and TC mechanism topology are equal. This means that the TC mechanism topology is connected if the input topology is connected.

Therefore, it is sufficient to show the following claim. For a given path P between two motes n_{x_1} and n_{x_M} , an alternative AU-path P' between these motes can be constructed. In the following, we show for each link l_j on P (by induction) that its incident motes are connected by an AU-path P_j . The alternative path P' can be constructed by concatenating the per-link alternative AU-paths, as illustrated by Figure 3.16a.

Induction claim: If the topology is connected and weakly consistent, then the incident motes of each link l_j are connected by an AU-path P_j . We show that this claim holds by induction over the links of the topology $G_T = (V_T, E_T)$, sorted by w'

according to \prec . Let $L = (l_1, l_2, \dots)$ be a list that contains all links in E_T and let L be sorted by increasing w' (i.e., $w'(l_1) \prec w'(l_2) \prec \dots$).

Induction start ($j = 1$): Let's first assume that $l_j = l_1 = l_{XY}$ is inactive. In this case, the fulfillment of $C_{kTC,i}$ implies that there are links l_{XZ} and l_{ZY} that are both shorter than l_{XY} w.r.t. w' due to the kTC -specific condition (i.e., $w'_{XZ} \prec w'_{XY}$ and $w'_{ZY} \prec w'_{XY}$). This contradicts the assumption that l_1 is the shortest link w.r.t. w' . Therefore, l_1 is either active or unmarked and constitutes an AU-path.

Induction step ($j = N \rightarrow N + 1$): We assume that the induction claim holds for all links l_j with $j \leq N$. We have to show that the claim also holds for link l_{N+1} . We distinguish between the three possible states of l_j , as illustrated in Figure 3.16b. If l_j is active (Case 1) or unmarked (Case 2), the induction claim holds. If l_j is inactive, the fulfillment of $C_{kTC,i}$ implies that there are links l_{XZ} and l_{ZY} that are both shorter than l_{XY} w.r.t. w' due to the kTC -specific condition (i.e., $w'_{XZ} \prec w'_{XY}$ and $w'_{ZY} \prec w'_{XY}$). Therefore, $l_{XZ} = l_{j_1}$ and $l_{ZY} = l_{j_2}$ for some $j_1, j_2 \leq N$, and the induction claim holds for l_{XZ} and l_{ZY} . This implies that AU-paths P_1 and P_2 exist from n_X to n_Z and from n_Z to n_Y , respectively. The concatenation of P_1 and P_2 is an AU-path from n_X to n_Y , and the induction claim follows for $j = N + 1$. \square

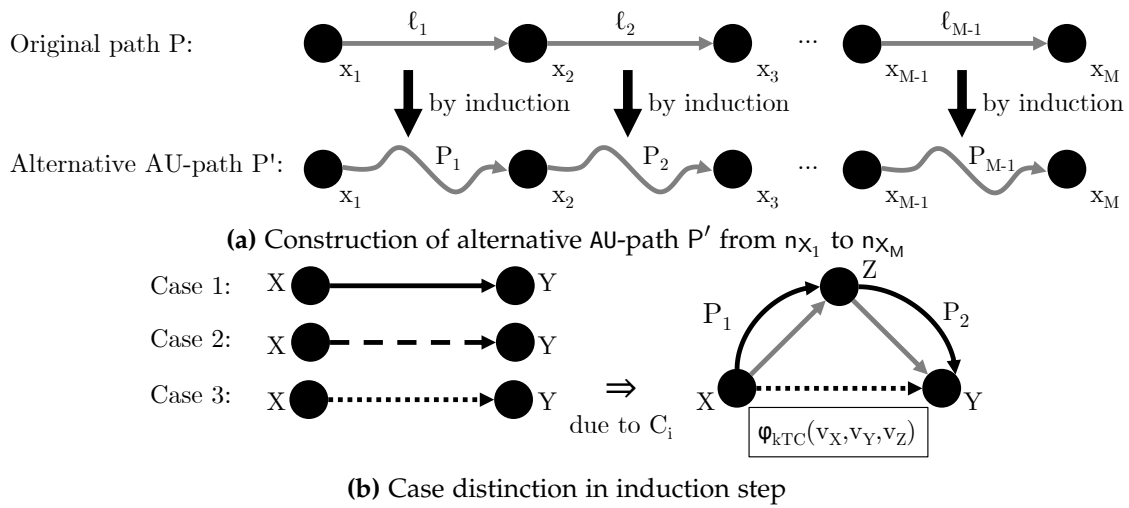


Figure 3.16: Sketches for proof of preserved connectivity for kTC

3.4 ELEMENTARY TOPOLOGY MODIFICATIONS

Graph constraints characterize valid and invalid states of a topology w.r.t. structural or TC-algorithm-specific consistency properties. The modeling technique introduced in this section formalizes elementary modifications of the topology. Examples of such modifications are the activation or inactivation of a link. As modeling technique, we selected graph transformation rules because they are suitable to capture structural modifications and constitute a (technical) prerequisite for applying the constructive approach.

Definition 3.24 (Graph transformation rule [56, 214]). A *graph transformation rule* (GT rule) R_X consists of a *left-hand side pattern* LHS_X , a *right-hand side pattern* RHS_X , and a potentially empty set of *application conditions* $AC_{x,1}, AC_{x,2}, \dots$. Each application condition is a (positive or negative) graph constraint [56, 82, 214]. Additionally, R_X contains a partial mapping^a from the variables of LHS_X to the variables of RHS_X and mappings from the variables of LHS_X to the premise of each application condition. A GT rule has zero or more *rule parameters* X_1, X_2, \dots . A rule parameter has a name and type. A rule parameter decorated with “out” is an *output rule parameter*. All other parameters are *input rule parameters*, which can be interpreted as partial match of LHS_X or used inside attribute constraints. Figure 3.17 depicts a GT rule schematically.

A GT rule must fulfill the following two well-formedness requirements. First, the attribute constraints of RHS_X represent attribute value modifications and may, therefore, only use the equality operator (i.e., =)^b. Second, the intersection of LHS_X and RHS_X must be a valid graph. This means that, if an association variable occurs in LHS_X and RHS_X , then the incident object variables of this association variable must also be part of both LHS_X and RHS_X . Without the second well-formedness

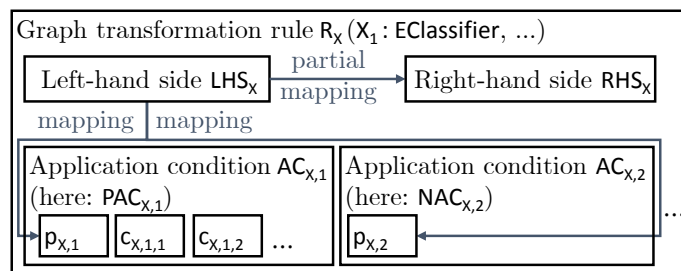


Figure 3.17: Structure of a GT rule with a positive application condition $AC_{X,1}/PAC_{X,1}$ and a negative application condition $AC_{X,2}/NAC_{X,2}$.

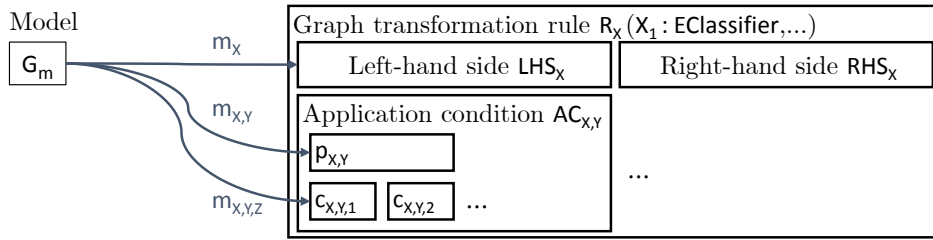


Figure 3.18: Illustration of GT rule applicability

requirement, the model resulting from an application of a GT rule may contain dangling associations. \square

- a* A partial mapping from a pattern p_1 to a pattern p_2 maps a subset of the variables of p_1 to variables in p_2 .
- b* In general, arithmetic expressions for calculating new attribute values are also allowed but not discussed here for conciseness reasons.

Regarding notation, the mappings inside a GT rule are implicitly given by equality of variable names across patterns (as usual). We use the abbreviations PAC and NAC for positive and negative application conditions respectively, and the abbreviation AC for application conditions in general. The rules that relate to topologies always have an implicit (i.e., hidden) variable for the unique topology. A GT rule is a declarative specification of a possible modification of a model. The following definitions capture how a concrete model can be modified by applying a GT rule to it.

Definition 3.25 (GT rule applicability). A GT rule R_x is *applicable at a match* m_x of LHS_x in a model G_M if the extension of m_x to a match $m_{x,y}$ of the premise $p_{x,y}$ of each application condition $AC_{x,y}$ can be further extended to a match $m_{x,y,z}$ of at least one conclusion pattern $c_{x,y,z}$ of $AC_{x,y}$ (Figure 3.18), and if the following dangling edge condition is fulfilled.

The *dangling edge condition* prescribes that a GT rule R_x is only applicable at a match if removing all model elements that are matched by a variable that occurs only in LHS_x and not in RHS_x results in valid graph (i.e., without dangling associations). \square

This definition implies that a GT rule is only applicable at a match if this match cannot be extended to any match of the premise of any negative application condition. For conciseness reasons, we refer to a match m of the LHS of a GT rule R in a model G_M as *the match m of the rule R in a model G_M* .

Definition 3.26 (GT rule application). Let R_x be a GT rule that is applicable at some match m in a model G_M . An *application of R_x at m* consists of the following three

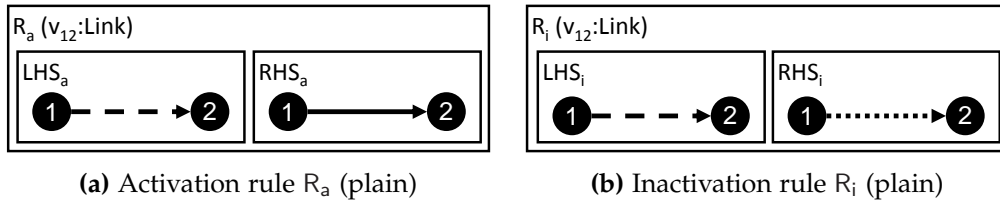


Figure 3.19: Example: TC rules

steps. (i) Each model element that is matched by a variable that occurs only in LHS_X and not in RHS_X is removed. (ii) For each variable that occurs in RHS_X but not in LHS_X, a fresh copy of this variable is inserted into the model G_M . (iii) Each equality constraint of RHS_X is interpreted as an attribute assignment.

A model element that is neither removed nor created during the rule application is *preserved*. \square

Regarding the development of TC mechanisms, we distinguish between three types of GT rules. (i) A *TC rule* specifies an elementary modification that is performed by a TC algorithm (Example 3.27). (ii) A *context event rule* specifies an elementary modification of the topology that is caused by the environment (e.g., the removal of a link). As shown in Example 3.28, we associate with each context event class (e.g., *MoteAddition*) in the metamodel a corresponding context event rule whose application operationalizes the corresponding context event (e.g., R_{+n}). (iii) A *context event handler rule* is an additional rule that is used within the specification of a context event handler. We derive appropriate context event handler rules of kTC in the subsequent Chapter 4.

In our scenario, a GT rule is either restrictable or irrestrictable. A *restrictable GT rule* can be modified freely during the developed of the TC mechanism by adding application conditions (e.g., TC or context event handler rules). An *irrestrictable GT rule* represents a modification of the topology caused by the environment (i.e., context event rules). These modifications are not under the control of the sensor operating system and must be handled by adjusting the topology. In the following, we discuss all GT rules that are relevant for our running example.

Example 3.27 (TC rules). Figure 3.19 shows the activation rule R_a and the inactivation rule R_i , which specify the unconditional activation and inactivation of links. Both GT rules expect the link to be marked as input parameter, represented by the link variable v_{12} . Both LHS patterns impose no additional attribute constraints on v_{12} . The attribute constraint $s_{v_{12}} = A$ of RHS_a (indicated by the black solid line) states that given link is active after applying R_a . Conversely, v_{12} is inactive after applying R_i .

Example 3.28 (Context event rules). Figure 3.20 shows the five rules that represent the five different types of context events of the running example.

The node addition rule R_{+n} specifies the addition of a mote with a given mote identifier x . The LHS pattern is depicted as empty box because the new mote only requires the topology as context. The negative application condition $NAC_{+n,1}$ ensures that the mote v_1 is only created if the topology contains no other mote with the same identifier. The node removal rule R_{-n} specifies that a mote v_1 is removed from the topology. The two negative application conditions $NAC_{-n,1}$ and $NAC_{-n,2}$ ensure that v_1 may only be removed if it is isolated. This ensures that the removal of v_1 leaves no dangling links in the topology.

The link addition rule R_{+e} specifies the addition of a link from a mote v_1 to a mote v_2 having the given weight w . The negative application condition $NAC_{+e,1}$ ensures that no parallel links are added. The added link is unmarked after the rule application. The link removal rule R_{-e} specifies the removal of a given link v_{12} . Finally, the link-weight modification rule $R_{\text{mod-}w}$ specifies the change of the weight of the given link v_{12} to w . The modified link is unmarked after the rule application.

Table 3.2 maps the context event classes in Figure 3.4 to their corresponding context event rules.

Example 3.29 (Auxiliary rules). Figure 3.21 shows four auxiliary rules that we use to identify motes and links in a topology and to unmark a given link. The find-unmarked-link rule $R_{\text{find-u}}$ identifies some unmarked link and returns it as output parameter. The find-mote rule $R_{\text{find-n}}$ finds, for a given mote identifier x , a mote v_1 having this identifier. The find-link rule $R_{\text{find-e}}$ finds, for two given mote identifiers x and y , a link v_{12} with source mote n_x and target mote n_y . The application of $R_{\text{find-u}}$, $R_{\text{find-n}}$ and $R_{\text{find-e}}$ leave the topology unchanged because its LHS and RHS patterns are identical. The unmarking rule R_u unmarks a given link v_{12} regardless of the current state of v_{12} .

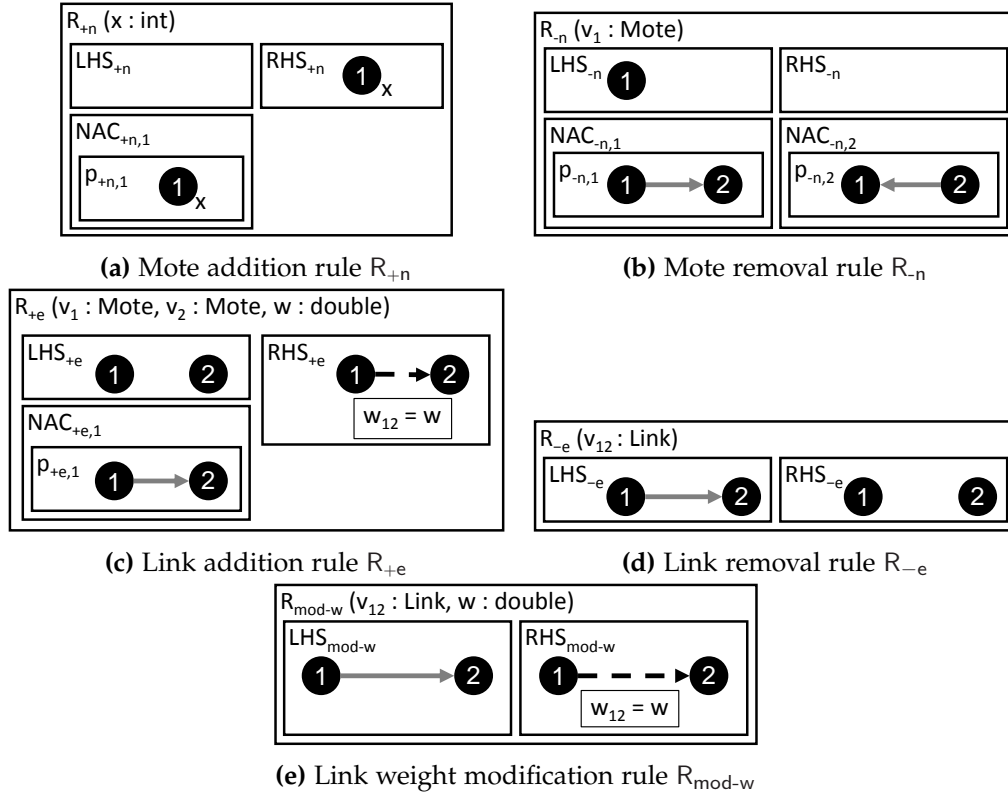


Figure 3.20: Example: Context event rules

Table 3.2: Operationalization of context events (First column: context event object, see also Figure 3.4; second column: corresponding context event rule application, see Figure 3.20)

Context event class	Operationalizing GT rule application
MoteAddition e	$R_{+n}(e.\text{moteld})$
MoteRemoval e	$R_{-n}(e.\text{moteld})$
LinkAddition e	$R_{+e}(e.\text{srclD}, e.\text{trglD}, e.\text{weight})$
LinkRemoval e	$R_{-e}(e.\text{srclD}, e.\text{trglD})$
LinkWeightModification e	$R_{\text{mod-}w}(e.\text{srclD}, e.\text{trglD}, \text{weight})$

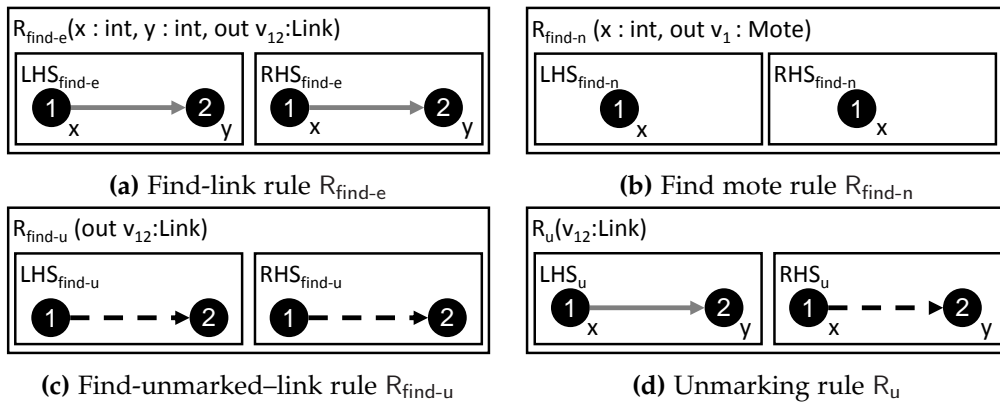


Figure 3.21: Example: Auxiliary rules $R_{\text{find-n}}$, $R_{\text{find-e}}$, $R_{\text{find-u}}$, and R_u

3.5 EXECUTION ORDER OF TOPOLOGY MODIFICATIONS

In the previous section, we introduced graph transformation as a modeling technique to specify elementary topology modifications. Given a model, a set of GT rules, and a set of matches of these rules in the model, it is generally undefined which rule to apply at which match first (*rule and match nondeterminism*).

In this section, we introduce a modeling technique that allows us to restrict the rule and match nondeterminism by (i) specifying the application order of GT rules and (ii) providing GT rules with input parameters to GT rule applications. The constructive approach does not restrict the choice of the control flow specification language. We propose to use story-driven modeling [60], a dialect of programmed GT that borrows from UML activity diagrams and allows for GT rule applications as actions [78].

Definition 3.30 (Story-driven modeling [60]). In story-driven modeling (SDM), each operation is specified by a story diagram [60]. A *story diagram* is a graph that specifies the control flow of an operation and consists of *activity nodes* and *activity edges*. An activity edge (\longrightarrow) may be labeled with a *success* ([S]) or *failure guard* ([F]). An activity node can be of one of four types: start node, stop nodes, story node, or operation node. The unique *start node* (\bullet) of a story diagram has exactly one outgoing, unguarded activity edge. Each of the zero or more *stop nodes* (\bullet) has at least one incoming activity edge and no outgoing activity edges. If the operation has a return type, each stop node is labeled with the return value.

A *story node* contains a GT rule application consisting of the rule name and parameter bindings for each input rule parameter. An *operation node* contains an invocation of another operation. A story or operation node has at least one incoming activity edge, and either one unguarded outgoing activity edge or two outgoing activity edges labeled with [S] and [F], respectively. An *operation variable* represent local variables of the story diagram. These variables are declared in parentheses next to the operation name (e.g., $\text{var } v_{12} : \text{Link}$) and are used to pass information to the operation or among rule and operation invocations. An operation variable can be *assigned* based on parameters of a successfully applied rule or invoked operation, which is denoted with a leading assignment operator (e.g., $v_{12} = R_{\text{find-u}}()$), and *passed* to rule applications or operation invocations, which is denoted with trailing parentheses (e.g., $R_a(v_{12})$). \square

Definition 3.31 (Execution of story diagram [60]). A story diagram is *executed* as follows: The execution begins at the (unique) start node and continues along activity edges until arriving at a stop node. The execution of story and operation nodes is similar: When the execution arrives at a story node, the contained graph transformation rule is applied (if possible). When the execution arrives at an operation node,

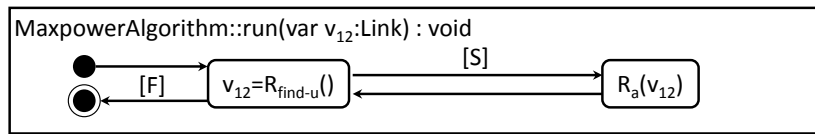


Figure 3.22: Story diagram of Maxpower algorithm

the contained operation is invoked. In case of two outgoing activity edges, if the rule application was successful or if the operation invocation returned a non-null result, the execution continues along the [S]-edge, else along the [F]-edge. In case of one one outgoing activity edge, the execution continues along this activity edge in any case after the rule application or operation invocation. \square

In contrast to UML activity diagrams [78], no single standardized version of SDM exists. The preceding definitions represent an SDM dialect that is provided (similarly) by the GT tool `EMOFLON` [135]. This SDM dialect is less expressive than the original implementation of SDM in, e.g., `FUJABA` [60]. By “less expressive”, we mean that `EMOFLON` rejects story diagrams for which `FUJABA` can generate code. The SDM dialect of `EMOFLON` simplifies the generation of source code for target programming languages such as `JAVA`, `C`, or `C++` by rejecting story diagrams that could only be realized using jumps (“goto”) in the generated source code. In contrast, the generated code of `FUJABA` used exceptions for jumping to arbitrary locations in the control flow.

Our first example for SDM provides a specification of the TC algorithm `Maxpower`. Then, we continue with story diagrams that specify the handle operations of the subclasses of `ContextEventHandler` in Figure 3.4. Finally, we provide story diagrams for the concrete context event handlers of `Maxpower`. The purpose of the next chapter is to derive a TC mechanism specification of `kTC` that is guaranteed to preserve weak consistency.

Example 3.32 (Story diagram of Maxpower algorithm). Figure 3.22 shows the story diagram that specifies the TC algorithm `Maxpower`. The specification consists of a loop whose condition is the invocation of `Rfind-u` (to identify some unmarked link v_{12}). The link variable v_{12} is an operation variable and does not count to the operation parameters (i.e., `MaxpowerAlgorithm::run` has no parameters). In the loop body, the activation rule `Ra` is applied to v_{12} . The execution of the story diagram terminates if the topology contains no more unmarked links.

It is straightforward to show that this TC algorithm terminates for any initial state of the topology. In each iteration, the number of unmarked links decreases by one. Given that a topology contains a finite number of links, the number of loop iterations is also finite.

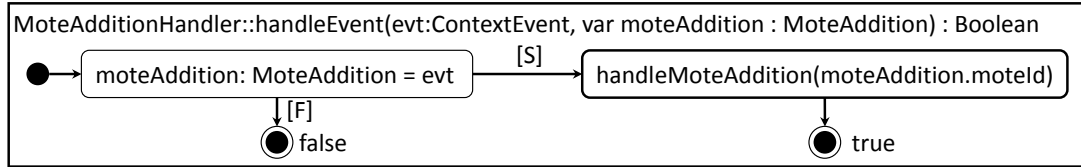
Example 3.33 (Story diagrams of generic context event handlers). Figure 3.23 shows the specifications of the generic handle operations of the abstract context event handler classes in Figure 3.4. The purpose of these specifications is to determine whether the context event handler is responsible for handling the event, and to invoke the specialized context event handling operation of the respective subclass (e.g., `MoteAddition::handleMoteAddition`).

Figure 3.23a specifies the handling of mote addition events. The leftmost story node in Figure 3.23a represents a type cast (attempt) of the parameter `evt` from the superclass `ContextEvent` to the subclass `MoteAddition`. If the type cast is successful (i.e., `evt` has the type `MoteAddition`), the execution continues along the [S]-edge, else the execution arrives at the leftmost stop node and returns *false*. We use this shortcut notation for type casts to keep the specification concise. A solution using GT rules is also possible but more verbose.

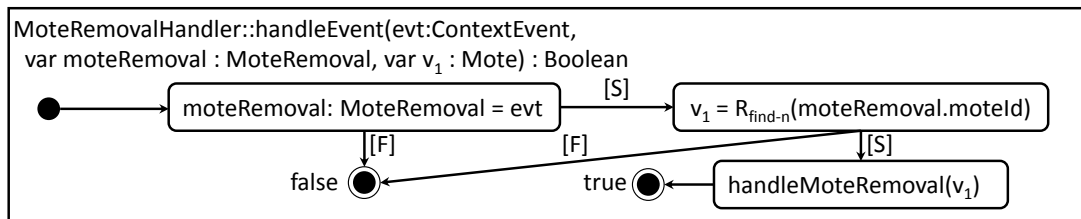
Figure 3.23b specifies the handling of mote removal events. It extracts the mote v_1 , which shall be removed and invokes the appropriate subclass method. For simplicity, we assume that v_1 is isolated already (i.e., that the incident links of v_1 have been removed by previously handled link removal events).

The specifications for link-related context event handlers in Figure 3.23 are completely analogous to the two discussed examples. Thus, we omit their detailed description here.

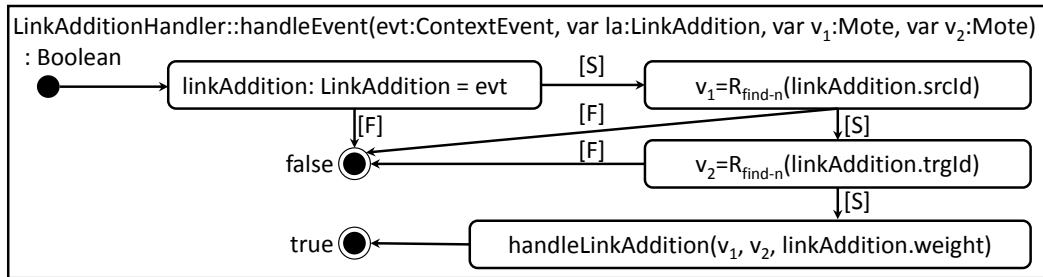
Example 3.34 (SDM specification of Maxpower-specific context event handlers). Figure 3.24 shows the specifications of all context event handlers that are specific to the Maxpower algorithm. The corresponding story diagrams forward the given parameters to the context event rules (Figure 3.20).



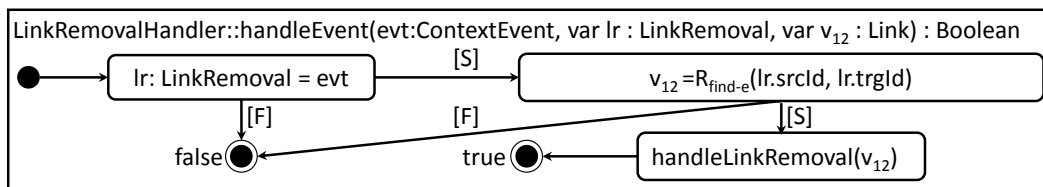
(a) MoteAdditionHandler::handleEvent



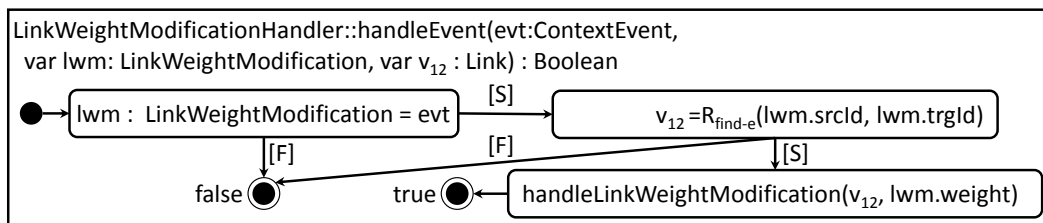
(b) MoteRemovalHandler::handleEvent



(c) LinkAdditionHandler::handleEvent



(d) LinkRemovalHandler::handleEvent



(e) LinkWeightModificationHandler::handleEvent

Figure 3.23: Story diagrams of context event handlers

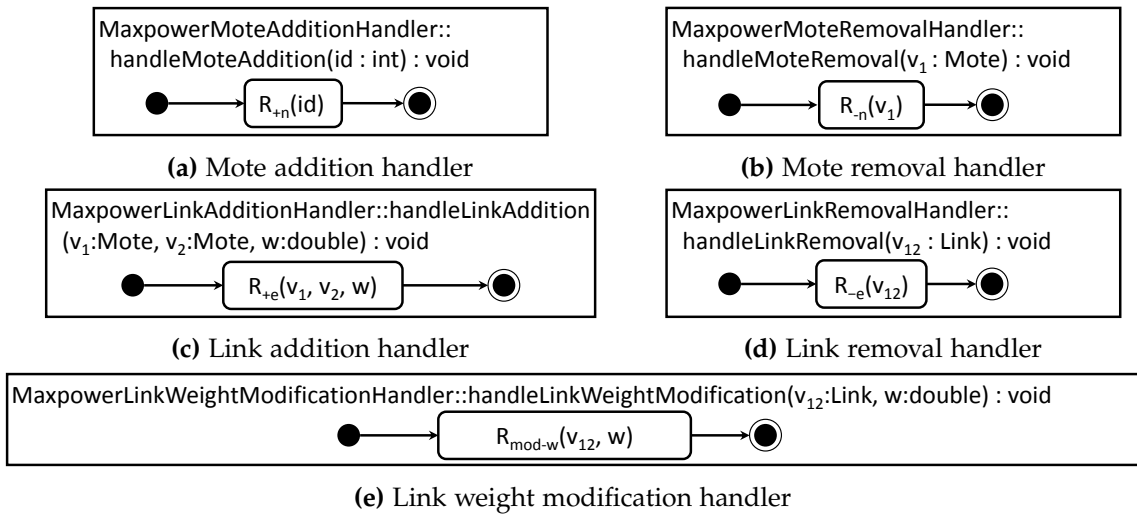


Figure 3.24: Story diagrams for Maxpower-specific context event handlers

3.6 CONFIGURATION SPACE SPECIFICATION

Modern WSN motes provide numerous configuration options, which can be either selected at compile time or, in case of adaptive WSNs, at runtime [183]. The space of configuration options can be captured using feature models from the domain of Software Product Line (SPL) engineering. A feature model usually provides a high-level, formal view of the system (also called the *problem space*). In contrast metamodeling (Section 3.1) is a technique for describing abstract representations of concrete systems that is closer to the actual implementation (also called the *solution space*) [180, 258]. While traditional SPLs typically describe the configuration space of a software system at or before its deployment, a *dynamic software product line* (DSPL) represents the possible reconfiguration at runtime [222].

A classic *feature model* [107, 108] represents only Boolean configuration options of a system as *features*. In [108], Karatas et al. introduced *extended feature models*, which also support to model configuration options with domains beyond Boolean values using *feature attributes* (e.g., integers, real numbers). In extended feature models, a feature attribute belongs to a parent feature. An established notation for feature models are feature diagrams in the style of the *Feature-Oriented Domain Analysis* (FODA) [107]. A feature diagram layouts a feature model in a tree-like structure. In the following, we introduce basic terminology of feature models and feature diagrams.

In FODA notation, a feature is denoted by a rectangular box and a feature attribute is denoted by a circle or ellipsis. Rectangles and circles are labeled with the name of the feature or feature attribute, respectively. A *feature model instance* assigns to each feature the value *true* (i.e., the feature is *selected*) or *false* (i.e., the feature is *deselected*). An attribute of a selected feature is configured with a value from its domain. A feature model specifies additional constraints among features and feature attribute. The unique *root feature* of a feature model is always selected in a valid feature model instance. The root feature is always shown on the top of a feature diagram. Except for the root feature, all features and feature attributes possess a *parent feature*. The parent-feature relation is acyclic. In a feature diagram, a child feature is connected to and placed below its parent feature. A child feature can only be selected if its parent feature is selected. A child feature can be either mandatory feature, an optional feature, or part of a feature group, which is labeled with either «XOR» or «OR». A *mandatory child feature* must be selected if its parent feature is selected (denoted with a solid black circle on top of the child feature's rectangle). An *optional child feature* may be selected if its parent feature is selected (denoted with a framed circle on top of the child feature's rectangle). Within an «XOR» *feature group*, exactly one child feature must be selected. Within an «OR» *feature group*, at least one child feature must be selected.

Cross-tree constraints can further restrict the set of possible feature model instances. Its name indicates that a *cross-tree constraint* involves two features that are not part of the parent-feature relation. A *require cross-tree constraint* from a source feature to a target

feature specifies that the target feature must be selected if the source feature is selected (denoted by a directed arrow labeled with «require»). An *exclude cross-tree constraint* among two features specifies that at least one of the features must be deselected (denoted by a two-headed arrow labeled with «exclude»).

A *context feature model* [88] is a feature model where each feature and feature attribute is assigned one of two categories. A *context feature* represents a configuration option that is determined by the system context and cannot be modified by the system (e.g., the density of notes). A *system feature* is configuration option that can be modified by the system (e.g., to react to modifications of the system context). A feature attribute is labeled according to its parent feature. A *context feature attribute* belongs to a context feature and a *system feature attribute* belongs to a system feature.

In recent years, the expressiveness of extended feature models has been enhanced by introducing multiplicities, which further reduces the gap between problem space and solution space [195, 221, 272]. These extensions could be used, e.g., to specify the maximum number of concurrently active TC mechanisms within a TC multi-mechanism.

Example 3.35 (Feature model and feature model instance). Figure 3.25 shows the feature diagram of a feature model with seven features and three feature attributes. System and context features are decorated with small rectangles containing either an S or C.

The root feature is Mote and has the optional system child feature TC and the mandatory context child feature PerformanceGoal. The child features of PerformanceGoal describe the possible performance goals of the mote: reducing the energy consumption (LowEnergy), increasing the robustness (HighRobustness), and reducing packet latency (LowLatency). Exactly one performance goal may be selected («XOR» group).

The system feature TC has two child features for the TC algorithms kTC [227] and l*kTC [239]. We omit details about l*kTC at this point. Exactly one TC algorithm may be selected («XOR» group). The TC algorithms have three real-valued three attributes: k belonging to kTC, and k and a belonging to l*kTC. The cross-tree constraints enforce that TC is disabled if high robustness is required («exclude») and that a TC algorithm must be selected if low energy consumption is desired («require»).

Features and feature attributes with a gray background describe a valid feature model instance. The value within the square brackets of the selected feature attribute k indicate that kTC has been configured with $k = 1.2$.

This example also illustrates that the problem-space perspective is usually more abstract compared to the solution-space perspective: In the feature model, kTC is represented by a feature with an attached feature attribute. In contrast, the meta-model in Figure 3.4 represents a TC algorithm using six classes for the TC algorithm and the different context event handlers.

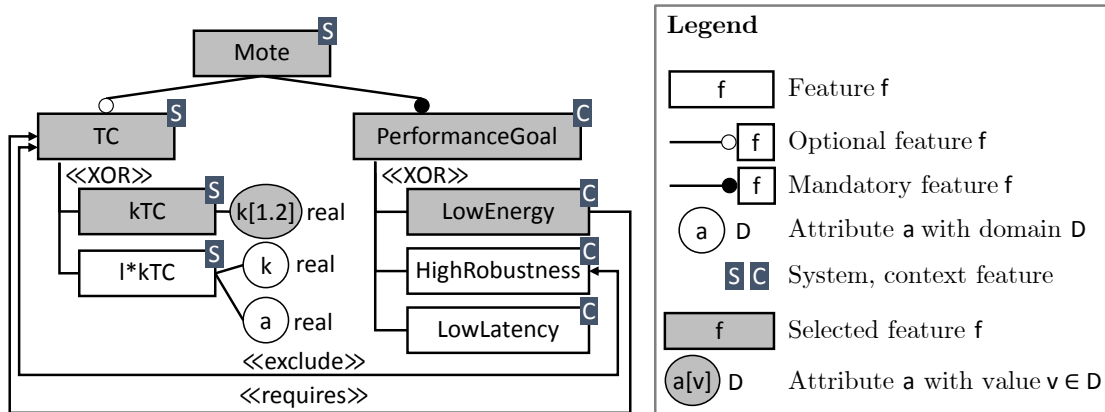


Figure 3.25: Example: Feature diagram with feature model instance (grey)

3.7 RELATED WORK

In this section, we survey related work on approaches for specifying TC algorithms and communication systems in general. Communication systems have been the running example or target domain of many works in the MDE and GT community (e.g., [90, 124, 157]). We also investigate additional modeling techniques w.r.t. their modeling purpose and potential for extending our approach.

3.7.1 *Snapshot-based approaches*

By a *snapshot-based approach*, we mean an approach that specifies a TC algorithm declaratively as relation over valid input and virtual topologies. This characterization is widely used in the WSN domain (see, e.g., the following surveys [219, 264, 267]).

Not all snapshot-based approaches can be specified using the modeling techniques presented in this chapter. This is the case if at least one of the following assumptions is violated: (i) The consistency specification is partly expressible in terms of graph constraints [91]; (ii) The joint fulfillment of all graph constraints in the consistency specification implies the fulfillment of the entire consistency specification.

For example, a centralized TC algorithm that requires the virtual topology of the entire WSN to be a global minimum spanning tree is not expressible in first-order logic. A *spanning tree* of an underlying weighted graph is an acyclic subgraph with identical node set and reduced edge set compared to the underlying graph. For a given underlying graph, a *minimum spanning tree* (MST) has the smallest sum over link weights among all spanning trees. We can also reformulate this consistency property as follows: The virtual topology forms a global minimum spanning tree if we inactivate all links that are the weight-maximal link on some cycle in the input topology (see cycle cancellation rule of Kruskal's algorithm [127]). Due to the required global coordination and the missing redundancy, the global minimum spanning tree algorithm is usually not used in practice. Instead, in case of many-to-one communication in a WSN, several routing protocols construct an approximately minimum spanning tree on top of the virtual topology of the TC algorithm (e.g., [99, 276]).

A related TC algorithm is the Local Minimum Spanning Tree (LMST) algorithm [140], which is a localized variant of the global minimum spanning tree algorithm. The LMST algorithm builds a minimum spanning tree within the local view of each mote and inactivates all of its outgoing links that are not part of the LMST. Non-nested graph constraints in the sense of [91] are not expressive enough to model LMST because, as for the global minimum spanning tree algorithm, the patterns of the graph constraints are not of a fixed size.

If the number of neighbors in the local view of a mote is limited by an upper bound (e.g., if a minimal geographic distance between motes is known), we can solve this problem using graph constraints as follows. Regarding the inactivation of links, we create

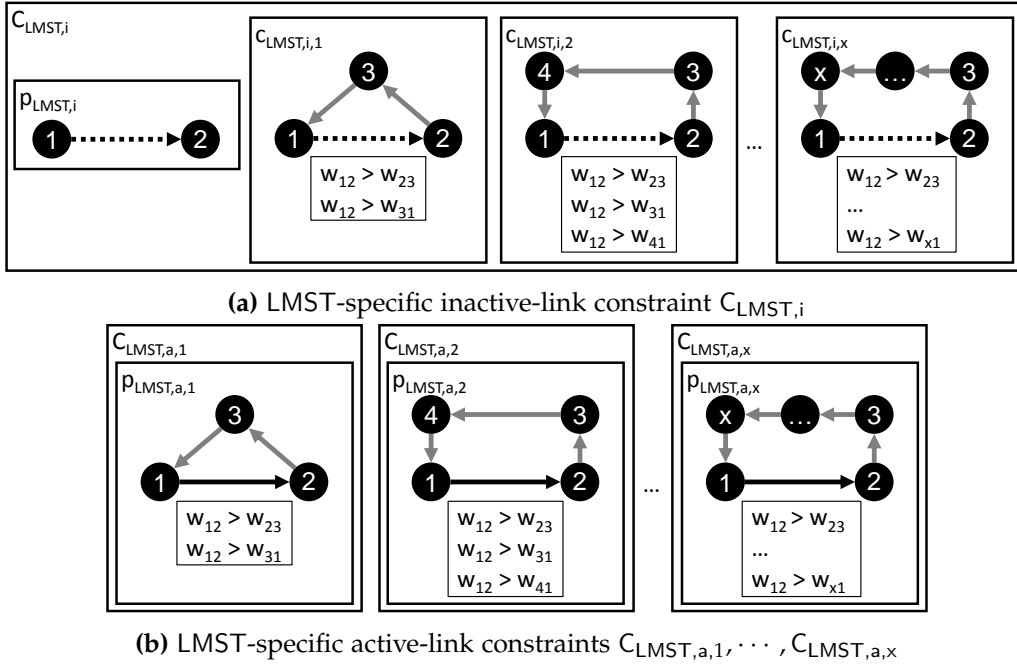


Figure 3.26: LMST-specific graph constraints for limited neighborhood size x

one positive graph constraint that contains a conclusion pattern for each possible length of a cycle that justifies the inactivation of a link if it is the weight-maximal link on the cycle. Regarding the activation of links, we create one negative graph constraint for each possible cycle length. From the end-user perspective, this approach is, of course, not desirable due to the excessive amount of resulting application conditions. Figure 3.26 sketches this solution for LMST, where the maximum neighborhood size is limited by some value x .

A less pragmatic and more foundational approach is the extension of graph constraints to nested graph constraints [55, 83], which allow to encode first-order logic expressions. Instead of simple conclusion patterns, the conclusion of a nested graph constraint can be another nested graph constraint. Therefore, arbitrary nesting is possible (conceptually). Still, first-order logic is not sufficient to express properties such as the acyclicity of a graph, which is required for the LMST algorithm.

To overcome this limitation, several formalisms have been proposed. In [167], Navarro et al. extend the logic of nested graph conditions [83] to represent paths of unbounded length. In contrast to nested graph constraints, Navarro et al. do not show how to ensure the correctness of a specification constructively w.r.t. their constraint formalism. Therefore, this constraint formalism is not compatible with our major goal of devising a correct-by-construction methodology.

In [84], Habel and Radke present HR^* constraints, a new type of graph constraints that allow to express path-related properties. They showed that the constructive approach is also applicable to HR^* constraints [196]. Therefore, using HR^* constraints instead of graph constraints in our scenario is possible, at least from a conceptual point of view. If we use HR^* constraints a formalization of consistency properties of tree-based TC algorithms (e.g., LMST [140]) should be possible.

3.7.2 Graph-grammar-based approaches

In this thesis, we model not only consistency properties of TC algorithms using graph constraints but also the possible topology modifications and their execution order using programmed GT.

In the following paragraphs, we survey works that solve challenges in communication systems engineering using programmed GT and graph grammars. We will also see how the proposed modeling techniques could be useful in our scenario to investigate additional (e.g., stochastic) properties of TC algorithms or to describe additional types of TC algorithms (e.g., probabilistic TC algorithms).

In [28], Bucchiarone et al. propose to specify self-adaptive systems using attributed graph grammars to identify dependencies and conflicts between GT rules. A *self-adaptive system* observes its internal state and environment and, based on these observations, adapts itself if necessary (e.g., to compensate for defect components (self-healing) or to improve its performance (self-optimization)) [42]. As running example, Bucchiarone et al. consider a case study of a car logistics scenario. Similar to this thesis, they distinguish between system rules, which represent actions of the self-adaptive system and context rules, which describe effects of the (uncontrollable) environment. The conducted dependency and conflict analysis (using the GT tool AGG [248]) helps to understand whether the specification always terminates for a given context event rule. In contrast to this work, they do not analyze whether the specification fulfills the required consistency properties.

In a survey paper on research challenges during the development of complex self-adaptive systems [19], Bennaceur et al. highlight that techniques such as the constructive approach are suitable to ensure that inter- and intra-model consistency requirements are preserved by a behavioral specification.

In [29], Bur et al. model a distributed cyber-physicalsystem as distributed runtime graph model that is dispersed over several (resource-constrained) devices. The purpose of their approach is to identify violations of consistency properties that are specified as graph queries. In contrast to the interpretation of graph constraints in this thesis, a successful evaluation of a graph query in [29] corresponds to a violation of the consistency specification. The distributed character of the runtime model is captured using 3-valued logic, where each local model element is either present, absent, or in an unknown state. In this thesis, we focus on the per-node perspective of a TC algorithm and assume

that the underlying communication mechanism ensures an eventual delivery of context events to neighbor nodes. The approach by Bur et al. is a promising extension of our approach, when message transfer shall be modeled explicitly during the specification phase (e.g., for detecting liveness issues such as deadlocks, livelocks, or starvation [61]).

In [124], Krause et al. propose *probabilistic GT* as a means to specify GT rules with multiple RHS patterns where each RHS pattern is annotated with a probability. The sum of probabilities over all RHS patterns of a GT rule must add up to 1.0. In comparison to multiple non-probabilistic GT rules with identical LHS patterns and distinct RHS patterns, a probabilistic GT rule can assign to each RHS pattern a different likelihood. The authors illustrate the usefulness of probabilistic GT based on a WSN case study with a gossiping application. One possible *gossiping* scenario is that a node forwards a message to one of its neighbor nodes based on a probability distribution [102]. For instance, a node may decide to forward a message with a probability of 90% and to not forward it with a probability of 10%. In [124], the transmission of a message is modeled as a probabilistic GT rule with two RHS patterns for modeling a positive forwarding decision with probability p and a negative forwarding decision with probability $1 - p$. The authors show how a probabilistic GT system can be translated into a Markov Decision Process based on a given start graph. To enable model checking, the resulting Markov Decision Process may only represent a finite state space. The analysis goals that are verified using the model checker PRISM [131] are formulated in Probabilistic Computational Tree Logic (PCTL) [87] (e.g., “How large is the probability that node X has received a message after Y transmission iterations in the given topology?”).

In [157], Maximova et al. propose *probabilistic timed GT* as a means to annotate GT rules with probabilities in the spirit of the already discussed probabilistic GT [124] and, additionally, clock constraints as known from timed automata (e.g., to specify that a given GT rule may only be applicable within a certain time period). As running example, the authors consider the RAILCAB scenario [92], which consists of small autonomous railroad shuttles. The RAILCAB shuttles need to communicate to avoid collisions, e.g., at railroad switches. Conceptually, the authors show how to translate a probabilistic timed GT system into a corresponding probabilistic timed automaton [132] based on a given start graph. As in probabilistic GT, the resulting probabilistic timed automaton represents a finite portion of the state space of the probabilistic timed GT system. Analysis goals for model checking a probabilistic timed GT system are specified in Probabilistic Timed Computational Tree Logic (PTCTL) [132] and verified using PRISM [131]. For example, one analysis goal is to estimate the probability of an emergency break given that the communication attempt between two shuttles fails with a certain probability within a given period of time.

In [90], Heckel et al. introduce *stochastic GT* as a means to annotate a GT rule with an application rate, which signifies the expected waiting time until the GT rule is applied at a match in a graph. As running example, they model a communication system with mobile devices, which connect to radio stations in their vicinity. The stochastic GT rules

specify the device movement behavior as well as failure and repair rates of radio towers. They propose to translate a stochastic GT system into a continuous-time Markov chain (CTMC) based on a given start graph via an intermediate representation as labeled transition system (LTS). The resulting continuous-time Markov chain can only represent a finite number of graphs. The authors propose to specify analysis goals in Continuous Stochastic Logic (CSL) [10]. To evaluate the analysis goals, the authors use the model transformation tool GROOVE [204] for creating a finite part of the state space of a given stochastic GT system as labeled transition system. This labeled transition system is annotated with the application rates and passed to the PRISM model checker to evaluate the analysis goals specified in Continuous Stochastic Logic. Complementary to [90], Torrini et al. present a simulation environment for evaluating specifications based on stochastic GT in [256].

In [17], Becker and Giese present a development approach for the architecture of self-adaptive systems [42] based on UML class and object diagrams as well as SDM [60]. First, they identify eight requirements that such a modeling approach should fulfill (e.g., multiple levels of abstraction, techniques for specifying dynamics, time constraints and correctness, representation of the system environment). Regarding the notion of correctness, they discuss the utility of simulation and verification using inductive invariants [53]. The former method provides anecdotal evidence based on selected execution traces of the system, and the latter method allows to verify statically the correctness of the specification. Similar to the already discussed probabilistic and stochastic approaches to GT [90, 124] the authors propose to analyze the timing behavior of the specification based on an analysis of a finite subset of the state space of the system in a model checker.

Probabilistic, timed probabilistic, and stochastic GT are complementary to the techniques used in this thesis because their purpose is to verify probabilistic and time-related properties of a system based on a finite state space. Our goal is to refine a system specification to ensure that consistency properties hold, regardless of the system size. Stochastic GT could be used in the context of the development of TC algorithms to specify arrival rates of context events (e.g., to determine whether it makes sense to invoke the TC algorithm specification at a particular point in time or for a given unmarked link). Probabilistic GT could be used to model gossip-based TC algorithms, which decide whether to activate and inactivate a link based on probability distribution. Under such circumstances, the graph constraints that we use for specifying local consistency properties can not longer be used as is. Instead, we would need to characterize consistency properties, for instance, as a threshold on the number of constraint violations or as an expectation values thereof. This extension of the concept of graph constraints is outside the scope of this thesis. Another alternative could be to adjust the consistency specification to be more restrictive w.r.t. link inactivation. For instance, we could require that a link may only be inactive if it is part of a triangle that fulfills the kTC condition and if, additionally, none (or at most one) of the other two links is inactive

already. This modification could limit the expected stretch factor, which signifies how much longer a path in the virtual topology is compared to the corresponding path in the input topology.

3.7.3 *Game-theoretic approaches*

In game-theoretic approaches, the individual motes are considered as player who try to improve their individual benefit according to a local reward function, which maps each combination of state and action of a player to a reward. If the reward function is properly designed [161], the pursuit of a mote's individual goal also maximizes some desired global property (e.g., minimizes the overall energy consumption or the fairness w.r.t. energy consumption). Surveys of game-theoretic approaches to developing TC algorithms can be found in [153, 231].

In Chapter 1, we shortly discussed the example of the CTCA algorithm [35, 36] for illustrating the difficulty of bridging the gap between the formal (game-theoretic) specification and the actual implementation (as pseudo- and simulation code). The idea behind CTCA is close to the idea of kTC: From a graph-theoretic point of view, CTCA also resolves triangles but the cost function is different from the distance-based cost function of kTC. Each mote knows about its battery level (i.e., its remaining energy) and the estimated energy that is required to transmit a fixed amount of data across each of its links in the input topology. Each mote shares this information with its one-hop neighbors. A mote estimates its expected remaining lifetime and the expected remaining lifetime of its neighbors based on the active links with largest transmission power. CTCA inactivates a link if it is part of a triangle, where the other two links have a larger expected remaining lifetime, and activates all other links. Using the expected remaining lifetime as link weight, a mote with high energy level may activate an energy-intensive link to assist a neighbor with low energy level.

Our observation is that, in contrast to the traditional style of formulating a TC algorithm as relation over input and virtual topologies, the structure of game-theoretic problem formulations is naturally incremental because each mote performs elementary actions (e.g., add a link to or remove a link from the virtual topology). These elementary actions can be modeled as GT rules. To apply our approach, we would need to enrich each such GT rule with a context in which taking the corresponding action is beneficial in the game-theoretic sense. To provide a concrete example for this idea, we show how to specify the game-theoretic CTCA algorithm as an energy-aware variant of kTC in Chapter 6.

3.7.4 *Role-centric approaches*

Role-centric approaches focus on separating the concerns of different stakeholders of the TC algorithm development process. Classic roles are the domain expert, who wants

to focus on a particular use case of a WSN (e.g., wildlife monitoring), and the network expert, who is an expert for a particular sensor platform and its operating system (e.g., Tiny OS [95]). Our observation is that this research space is huge. Therefore, we summarize only a few of the existing works in the following.

In [212, 213], Rodrigues et al. present an MDE approach to developing WSN algorithms in a platform-independent way. The development workflow consists of (i) a transformation from a platform-independent model (represented as textual DSL) to a platform-specific model and, (ii) a model-to-text transformation via platform-specific templates. The business logic of a WSN algorithm is specified as UML state machine [78]. The discussed metamodel contains only two types of topology (i.e., flat and hierarchical), but it appears that their approach could be extended to support additional TC algorithms. The proposed approach provides no means to verify correctness properties or to integrate such properties constructively.

In [252], Tei et al. present an MDE approach to develop data-processing WSN applications. The application programmer specifies the data flow from the sensor nodes to the sink node and the network expert provides the implementation of data-processing operations and the code templates for the sensor platform. As an intermediate representation, the data flow model is transformed into a group model, which describes the deployment plan for a concrete WSN. Their approach treats the choice of the network topology as design decision of the network expert. The topology is mapped to the selected routing algorithm. Tei et al. support tree, star, and flat topologies. Possible extensions of the specification to general TC algorithms are not discussed. Our approach to develop TC algorithms in a model-driven manner is complementary to the approach by Tei et al. because the active TC algorithm should not affect the functionality of a specified data-processing application.

Berardinelli et al. invented the AGILLA Modeling Framework⁴ to simulate applications written in the agent-based AGILLA language [61]. AGILLA is an agent-based specification framework for specifying WSN applications. This paradigm allows a WSN application to move from one node to another one (roughly similar to a minimalist virtual machine). The AGILLA Modeling Framework builds on Foundational UML (fUML) [158], a subset of the UML with formalized semantics. In the AGILLA Modeling Framework, activity diagrams are used to represent the control flow of WSN applications. This is similar to the usage of SDM in this thesis to describe the control flow of TC algorithms. However, the abstraction level of a specification in the AGILLA Modeling Framework is lower in comparison to the topology patterns that we employ in this thesis. For instance, in AGILLA, typical instructions serve to manipulate actors or fetch sensor values.

In [4], Al Saad et al. present SCATTERCLIPSE, an MDE tool for developing WSN applications based on the SCATTERWEB [220] WSN platform. The authors model the control flow of test cases for a WSN using hierarchical activity diagrams, where an activity node can contain an activity subdiagram. Operations that are supported by the framework

⁴ AMF page: <http://sealabtools.di.univaq.it/tools.php> (visited: 2018-09-17)

include logging as well as sending and receiving packets. The instruction set appears to be smaller compared to AGILLA. The specification is translated into platform-specific code using a code generator that can be configured using custom code templates. The specification can be instrumented with assertions that are evaluated later on the sensor nodes. The reduced instruction set implies that a considerable part of the application logic needs to be implemented as platform-specific code. Even though the authors mention model checking as a means to verify the specification, their notion of model checking appears to be limited to syntactic and semantic checks of the visual application model (e.g., that naming conventions are obeyed to).

All of the role-centric MDE approaches have in common that their major goal is to provide a modeling environment for WSN applications that relieves the application developer from the low-level details of a concrete sensor platform. These approaches usually provide tool support for visual modeling or Domain-Specific Languages and allow to configure the code generation using code templates. The abstraction level of the provided modeling primitives is usually higher than the abstraction level of programming the target platform immediately (e.g., using C-dialects or Assembler). Several of the approaches that we surveyed support the reconfiguration of the topology. Still, this reconfiguration is often only possible at design time, and only a limited set of pre-configured topologies is available (e.g., flat vs. tree). In contrast, our approach focuses on the topology of the network and is orthogonal to the specification of the concrete WSN application, even though the selection of a particular TC algorithm may have a drastic influence on the performance of a particular WSN application. Finally, separating concerns of experts for specification frameworks and the network experts is also an important goal of this thesis.

4

SYNTHESIS OF CORRECT TOPOLOGY CONTROL MECHANISMS

In Chapter 3, we introduced techniques for specifying (i) when a topology is structurally valid (using metamodeling and structural graph constraints, see Sections 3.1 and 3.2), (ii) when the topology fulfills the consistency specification (using TC-algorithm-specific graph constraints, see Sections 3.2 and 3.3), (iii) what types of elementary topology modifications exist (using GT rules, see Section 3.4), and (iv) how GT rules can be composed into a TC mechanism specification (using SDM, see Section 3.5). Finally, we provided a specification of the Maxpower algorithm (Example 3.32). The example of the Maxpower algorithm is arguably simple and served to provide an overview of the introduced modeling techniques.

CHAPTER STRUCTURE In this chapter, we describe an iterative refinement approach to synthesize a correct TC mechanism specification. As before, kTC serves as running example. We begin with applying the constructive approach [44, 91] and, based on the resulting TC mechanism specification, we determine further required refinement steps to come up with a correct TC mechanism specification. Figure 4.1 locates the role of this chapter in the entire TC algorithm development process (see also Figure 1.2).

In Section 4.1, we characterize in how far applying a GT rule preserves or violates a graph constraint. This characterization leads us to the definition of the correctness of a TC mechanism specification (Definition 4.7).

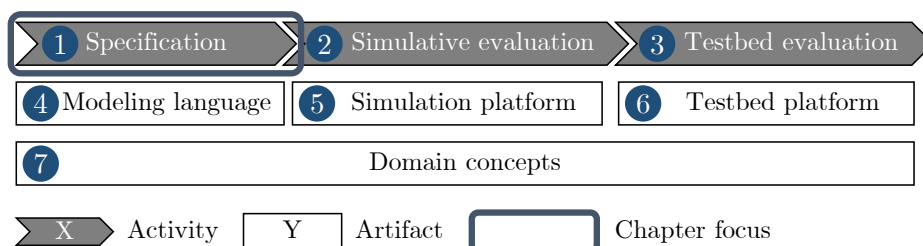


Figure 4.1: Location of Chapter 4 in TC algorithm development process

In Section 4.2, we explain how the constructive approach by Heckel and Wagner [91] allows us to refine the TC mechanism specification to ensure that each topology modification preserves weak consistency. Intuitively, the constructive approach restricts the applicability of a given GT rule based on a given graph constraint by transforming the graph constraint into a set of additional *synthesized application conditions* for the rule. After repeating this procedure for each combination of GT rule and graph constraint in the constraint set of weak consistency, all refined GT rules preserve weak consistency. Applying the constructive approach leaves the control flow specification unchanged; only individual GT rules are modified. As a result, weak consistency is an inductive invariant of the TC mechanism specification [53]. This means that each GT rule application preserves weak consistency and, therefore, the entire control flow specification preserves weak consistency.

The TC mechanism specification is consistency preserving by construction at this point. However, applying the constructive approach to context event rules, reduces the applicability of these GT rules. From the domain perspective, restricting the applicability of a GT rule means that we forbid certain environmental changes to occur (e.g., a link may not be removed if the resulting topology would violate weak consistency). This contradicts the requirement that context event rules are irrestrictable. In Section 4.4, we show how to resolve this problem by introducing an additional refinement step that transforms each synthesized application condition of a context event rule into an anticipation loop. An *anticipation loop* for a particular application condition unmarks exactly those links that would otherwise prevent the application of the context event rule due to a violation of this application condition. The algorithm for synthesizing anticipation loops is the major contribution of this section (see Algorithm 4.3).

In Section 4.5, we discuss the applicability of the proposed anticipation loop synthesis algorithm in general (Section 4.5.1) and based on two additional use cases: the handling of parameter modifications of the TC algorithm (Section 4.5.2) and the constructive refinement of the TC algorithm specification to ensure termination (Section 4.5.3).

RELATED PUBLICATIONS AND THESES In [122], we presented how to employ the constructive approach to TC algorithms for the first time. In [118], we presented the full kTC case study with all intermediate synthesis steps. In [119], we presented the anticipation loop synthesis algorithm and, additionally, used it to ensure constructively that the TC algorithm specification terminates (see also Section 4.5.3).

4.1 CONSISTENCY PRESERVATION AND VIOLATION

In this section, we investigate how applying a GT rule (see Definition 3.24) to a model influences the fulfillment of graph constraints (see Definition 3.16). We first concentrate on individual GT rule applications and, then, consider the influence of the control flow specification. The following definition focuses on consistency preservation in general.

Definition 4.1 (Consistency preservation in general). A modification of a model *preserves consistency w.r.t. a consistency level \mathcal{C}* if the model is consistent w.r.t. \mathcal{C} after the modification given that the model was consistent w.r.t. \mathcal{C} prior to the modification. \square

We now concretize the preceding general definition to consistency preservation for GT rules w.r.t. graph constraints. We begin with a situation where a particular GT rule application preserves or violates a particular constraint and lift this definition to constraint sets, GT rules, and, finally, story diagrams.

Definition 4.2 (Constraint and consistency preservation of GT rule application). The *application of a GT rule R at a match m preserves a constraint C* if the topology fulfills C prior to and after the application of R at m . Otherwise, the *application R at m violates C* .

The *application of a GT rule R at a match m preserves consistency w.r.t. a constraint set \mathcal{C}* if the rule application preserves all constraints in \mathcal{C} . Otherwise, the *application of R at m violates consistency w.r.t. \mathcal{C}* . \square

A closer look at the premise-conclusion structure of graph constraints allows us to characterize *constraint violation reasons*. The violation of a positive graph constraint due to a GT rule application can have one of two reasons: The first possible reason is that the rule application destroys a match of a conclusion pattern and this match is the only extension of a match of the premise without either (i) destroying the corresponding match of the premise or (ii) creating another extension of the premise match to some conclusion pattern match. The second possible reason is that the rule application creates a match of the premise that cannot be extended to a match of any conclusion pattern.

A violation of a negative graph constraint due to a GT rule application can only occur if a new match of the premise of the constraint is created by applying the GT rule. In any case, the violation of a graph constraint can be represented as a match of the premise of the graph constraint.

The preceding definition considers a concrete situation where a GT rule is applied at a particular match in a model. The following definition lifts the preceding definition to GT rules in general.

Definition 4.3 (Constraint and consistency preservation of GT rule). A GT rule R preserves a constraint C if, for all valid models G_M and all matches m of R in G_M , the application of R at m preserves C . Otherwise, R violates C .

A GT rule R preserves consistency w.r.t. a constraint set \mathcal{C} if, for all models G_M and all matches m of R in this model, the application of R at m preserves consistency w.r.t. \mathcal{C} . Otherwise, R violates consistency w.r.t. \mathcal{C} . \square

Finally, the following definitions lift the notion of constraint and consistency preservation to the execution of story diagrams.

Definition 4.4 (Constraint and consistency preservation of story diagram execution). The execution of a story diagram SD preserves a constraint C if each GT rule application that occurs until the execution reaches the stop node preserves C . Otherwise, the execution of SD violates C .

The execution of a story diagram preserves consistency w.r.t. a constraint set \mathcal{C} if each GT rule application that occurs until the execution reaches the stop node preserves consistency w.r.t. \mathcal{C} . Otherwise, the execution of SD violates \mathcal{C} . \square

The following definition lifts the notion of constraint and consistency preservation to a perspective that is independent of an individual execution of a story diagram.

Definition 4.5 (Constraint and consistency preservation of story diagram). A story diagram SD preserves a constraint C if, for each valid model G_M , the execution of SD preserves C . Otherwise, SD violates C .

A story diagram SD preserves consistency w.r.t. a constraint set \mathcal{C} if, for all valid models G_M , the execution of SD preserves consistency w.r.t. \mathcal{C} . Otherwise, SD violates consistency w.r.t. \mathcal{C} . \square

The preceding definitions (Definition 4.1 to Definition 4.5) disregard models that are inconsistent prior to the GT rule application or story diagram execution, respectively. This means that a consistency preserving GT rule or story diagram has undefined behavior for inconsistent input models. Therefore, achieving consistency preservation is usually only sensible if it can be established as invariant. For instance, weak consistency is an invariant of the TC algorithm and the context event handlers and should be preserved consequently.

However, not all consistency properties can be maintained as invariant. For instance, strong consistency is not an invariant but only the postcondition of the TC algorithm because context events necessarily produce unmarked links and thereby violate strong consistency (e.g., when a link is created by a link addition event). To characterize the interplay of such consistency properties with modifications of the model, we introduce the notion of consistency enforcement in the following.

Definition 4.6 (Consistency enforcement in general). A modification of a model G_M enforces consistency w.r.t. a precondition consistency level C_{pre} and a postcondition consistency level C_{post} if G_M is consistent w.r.t. C_{post} after the modification given that G_M was consistent w.r.t. C_{pre} before the modification. \square

The definition of consistency enforcement (Definition 4.6) is a generalization of the definition of consistency preservation (Definition 4.1). In the latter case, the precondition and postcondition consistency levels coincide: $C_{pre} = C_{post}$. Analogous specialized definitions for constraint and consistency enforcement can be given for GT rule applications, GT rules, story diagram executions, and story diagrams (Definition 4.1 to Definition 4.5). For the sake of brevity, we omit these analogous definitions here. In the literature, consistency-enforcing modifications are also called *consistency guaranteeing* (e.g., in [91]) or *consistency establishing* (e.g., in [89]).

The preceding general definitions of consistency preservation lead us to the following definition of correctness of TC mechanism specifications.

Definition 4.7 (Correctness of TC mechanism). A TC mechanism is *correct w.r.t. a given three-level consistency specification* (in the sense of Definition 3.22) if the following six correctness criteria CC1, ..., CC6 are fulfilled. (i) The TC algorithm specification terminates (CC1), preserves weak consistency (CC2), and enforces strong consistency (CC3). (ii) each TC-algorithm-specific context event handler terminates (CC4), preserves weak consistency (CC5), and processes the context event (CC6).

A context event handler processes a context event if it (i) removes the corresponding context event marker from the topology, and (ii) applies the corresponding operationalizing context event rule according to Table 3.2. \square

In the following example, we illustrate Definition 4.7 by proving the correctness of the Maxpower specification.

Example 4.8 (Correctness of Maxpower specification). We now sketch why the Maxpower specification is correct by showing that all six correctness criteria are fulfilled. The Maxpower specification consists of the story diagrams shown in Figures 3.22 and 3.24.

Criterion CC1: The Maxpower algorithm specification in Figure 3.22 terminates. *Proof idea:* The execution of the story diagram terminates if the topology contains no more unmarked links (see [F]-edge at the story node containing the application of the find-unmarked-link rule R_{find-u}). The number of unmarked links decreases by one with each iteration of the loop because the activation rule R_a is always applicable to an unmarked link v_{12} identified by applying the find-unmarked-link rule R_{find-u} . For a given topology, the set of links is finite and so is the number of loop iterations.

Criterion CC2: The Maxpower algorithm specification in Figure 3.22 preserves weak consistency. *Proof idea:* The constraint set of weak consistency for Maxpower consists of $C_{\text{no-loops}}$, $C_{\text{no-parallel-links}}$, the inactive-link constraint $C_{\text{Maxpower,i}}$ (Figure 3.15a), and the active-link constraint $C_{\text{Maxpower,a}}$ (Figure 3.15b). The constraint $C_{\text{Maxpower,a}}$ is always fulfilled due to its equal premise and conclusion patterns. The constraint $C_{\text{Maxpower,i}}$ is violated if an inactive link exists in the topology. We show the preservation of consistency by considering the effect of each possible successful application of a GT rule in Figure 3.22. Applying the find-unmarked-link rule $R_{\text{find-u}}$ preserves weak consistency because this rule does not modify the topology. Applying the activation rule R_a activates the given link v_{12} and, therefore, neither creates a loop ($C_{\text{no-loops}}$) nor a parallel link ($C_{\text{no-parallel-links}}$), and never inactivates a link ($C_{\text{Maxpower,i}}$). Therefore, the Maxpower algorithm specification preserves weak consistency.

Criterion CC3: The Maxpower algorithm specification in Figure 3.22 enforces strong consistency. *Proof idea:* An execution of the Maxpower algorithm specification may only terminate if $R_{\text{find-u}}$ is inapplicable. This is the case if the topology contains no unmarked links and, therefore, fulfills the no-unmarked-links constraint C_u . In conjunction with CC2, we conclude that the Maxpower specification enforces strong consistency.

Criterion CC4: The execution of each Maxpower-specific context event handler specification shown in Figure 3.24 terminates. *Proof idea:* No story diagram in Figure 3.24 contains loops. Therefore, the execution of each story diagram always terminates.

Criterion CC5: Each Maxpower-specific context event handler specification shown in Figure 3.24 preserves weak consistency. *Proof idea:* The argumentation in this case is similar to the one for CC2: Neither rule application in the context event handler specification creates a loop ($C_{\text{no-loops}}$) or a parallel link ($C_{\text{no-parallel-links}}$), or inactivates a link ($C_{\text{Maxpower,i}}$). Therefore, the context event handler specification preserves weak consistency.

Criterion CC6: Each Maxpower-specific context event handler specification shown in Figure 3.24 processes the corresponding context event. *Proof idea:* The single GT rule application in each context event handler diagram corresponds to the operationalizing GT rule application that is required according to Table 3.2.

In the next step, we prepare the construction of a correct TC mechanism specification for the kTC example. We begin with specifying initial control flow template in the following examples.

Example 4.9 (Control flow specification templates). We propose to use a modified version of the Maxpower specification as starting point for developing the kTC specification. We expect a TC mechanism developer to provide suitable control flow specification templates for the TC mechanism under construction.

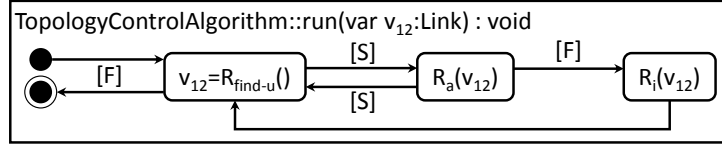


Figure 4.2: Story diagram of TC algorithm template

As generic *TC algorithm specification template*, we use the story diagram shown in Figure 4.2, which extends the specification of the Maxpower algorithm by an application of the inactivation rule R_i if the activation rule R_a is inapplicable. The order in which unmarked links are identified is undetermined, which allows the execution environment to choose freely which links to process first. The application of R_i is unreachable in the template because R_a is always applicable to a link v_{12} identified by applying $R_{\text{find-u}}$. This behavior is intended because we refine R_a and R_i in subsequent steps, which cause R_a to be inapplicable for certain unmarked links v_{12} . The order of applying first R_a and then R_i is chosen arbitrarily here.

As *context event handler specification templates*, we use the story diagrams of the Maxpower-specific context event handlers shown in Figure 3.24. Each of these story diagrams constitutes a minimal implementation that fulfills the correctness criterion CC6, which requires that a context event handler processes a pending context event according to Table 3.2.

Now, we investigate in how far the proposed TC algorithm template preserves weak consistency w.r.t. $k\text{TC}$. This is an optional step that serves to understand which constraints are preserved or violated by which GT rules.

Example 4.10 (Consistency violations by template specification). Table 4.1 summarizes which of the $k\text{TC}$ -specific weak consistency constraints $\{C_{k\text{TC},i}, C_{k\text{TC},a}\}$ are violated by which of the GT rules in the template specification (Figures 3.24 and 4.2). The parameter k of $k\text{TC}$ is set to 1.3.

The structural constraints $C_{\text{no-loops}}$ and $C_{\text{no-parallel-links}}$ are preserved by all GT rules thanks to (i) the negative application condition $\text{NAC}_{+e,1}$ of the link addition rule R_{+e} , which prevents the creation of parallel links, and (ii) the injectivity of matches, which prevents that the mote variables v_1 and v_2 in R_{+e} , which represent the source and target motes of the created link, are mapped to the same mote.

For each constraint violation, the constraint-violating GT rule applications refers to the sample topology shown in Figure 4.3. The GT rule applications $R_a(\ell_{911})$, $R_a(\ell_{119})$, $R_{+e}(n_5, n_7, 1)$, $R_{+e}(n_7, n_{10}, 1)$, $R_{\text{mod-w}}(\ell_{79}, 1)$, and $R_{\text{mod-w}}(\ell_{97}, 1)$ each create a match of the premise of the negative constraint $C_{k\text{TC},a}$. The GT rule applications $R_i(\ell_{12})$ and $R_i(\ell_{21})$ create a match of the premise of the positive constraint $C_{k\text{TC},i}$ that

Table 4.1: Example: Preservation (✓) and violation (✗) of weak consistency w.r.t. kTC ($k = 1.3$) by template specification (Figures 3.24 and 4.2). Consistency-violating GT rule applications refer to Figure 4.3. ($C_{nl} = C_{no-loops}$, $C_{npl} = C_{no-parallel-links}$)

C_x	R_a	R_i	R_{+n}	R_{-n}	R_{+e}	R_{-e}	R_{mod-w}
$C_{kTC,a}$	✗ $R_a(l_{9,11})$, $R_a(l_{11,9})$	✓	✓	✓	✗ $R_{+e}(n_4, n_5, 1)$, $R_{+e}(n_7, n_{10}, 1)$	✓	✗ $R_{mod-w}(l_{79}, 1)$, $R_{mod-w}(l_{97}, 1)$
$C_{kTC,i}$	✓	✗ $R_i(l_{12})$	✓	✓	✓	✗ $R_{-e}(l_{46})$, $R_{-e}(l_{64})$	✗ $R_{mod-w}(l_{23}, 8)$, $R_{mod-w}(l_{32}, 8)$
C_{nl}	✓	✓	✓	✓	✓	✓	✓
C_{npl}	✓	✓	✓	✓	✓	✓	✓

cannot be extended to a match of the conclusion pattern $c_{kTC,i,1}$. The GT rule applications $R_{-e}(l_{46})$, $R_{-e}(l_{64})$, $R_{mod-w}(l_{23}, 8)$, and $R_{mod-w}(l_{32}, 8)$ each destroy a match of the conclusion pattern $c_{kTC,i,1}$ of $C_{kTC,i}$ without destroying the corresponding match of the premise $p_{kTC,i}$ or creating a new match of $c_{kTC,i,1}$.

The premise matches that represent violations of $C_{kTC,a}$ and $C_{kTC,i}$ are shown as red cross marks (✗) in Figure 4.3a.

After introducing the concepts of constraint and consistency preservation and violation and defining when a TC mechanism specification is correct, we focus on how to ensure constructively that a (potentially incorrect) TC mechanism specification becomes correct in the subsequent section.

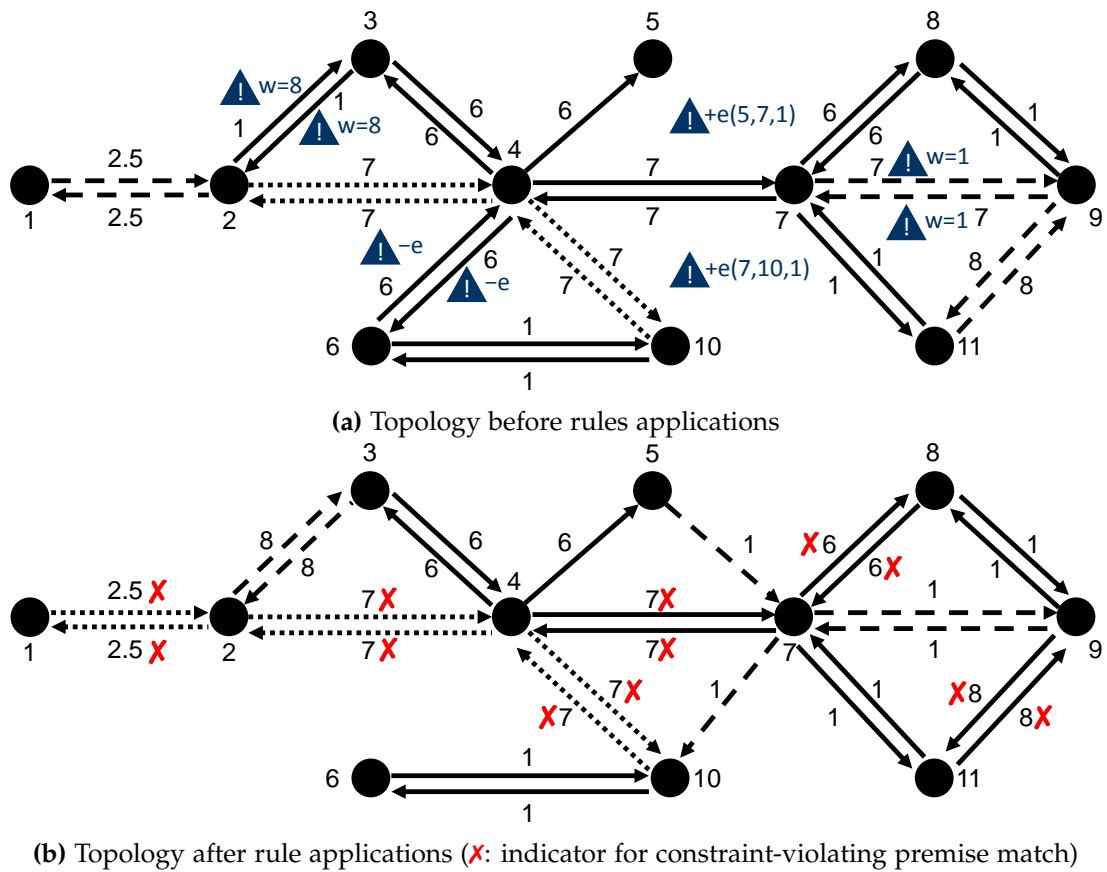


Figure 4.3: Example: Violation of consistency (for Example 4.10, kTC with $k = 1.3$)

4.2 CONSTRUCTIVE APPROACH

In this section, we introduce the goal of and ideas behind the constructive approach by Heckel and Wagner [91]. The goal of this approach is to refine a set of GT rules \mathcal{R} based on a set of graph constraints \mathcal{C} such that the refined GT rules \mathcal{R}' preserve consistency w.r.t. the constraint set \mathcal{C} . To achieve this goal, the approach iteratively refines each GT rule R_X in \mathcal{R} based on each constraint C_Y in \mathcal{C} (Algorithm 4.1). The synthesis algorithm `SYNTHESIZEAPPLICATIONCONDITIONS` is the core of the constructive approach and is responsible for synthesizing a set `acs` of additional application conditions (line 4) for the GT rule R_X based on the graph constraint C_Y (line 5). The original synthesis algorithm presented in [91] is only able to process purely structural GT rules and graph constraints, whose patterns contain no attribute constraints. Deckwerth and Varró [44] extended the synthesis algorithm with an additional step for handling attribute constraints. In [44], the authors discussed only negative application conditions, but their results carry over to positive application conditions in an analogous way.

Figure 4.4 illustrates how the synthesis algorithm generates the required application conditions for R_X by conducting two major steps: the specialization step ① and the anticipation step ②. At this point, we describe the general idea behind each step and describe the detailed (technical) steps in the context of the running example in Section 4.3. A formal description of the algorithm can be found in [91].

During the *specialization step* (①, [91, Prop. 3.1]), the graph constraint C_X is transformed into a set of postconditions $PC_{X,Y,J}$ of R_X . This step is called specialization step because its purpose is to transform a (global) graph constraint, which is independent of any GT rule, into a postcondition, which is associated with a particular GT rule. Each rule-constraint pair (R_X, C_Y) results in zero or more postconditions (as indicated by the index J). A *postcondition* of R_X is similar to an application condition, but it relates to RHS_X (instead of LHS_X). This means that a mapping from the variables of RHS_X to each postcondition premise $p'_{X,Y,J}$ exists. Each postcondition premise $p'_{X,Y,J}$ is a possible overlapping of RHS_X and p_Y and each postcondition conclusion pattern $c'_{X,Y,J,Z,K}$ is a possible extension of $p'_{X,Y,J}$ according to the conclusion patterns $c_{Y,Z}$ of C_Y . Multiple

Algorithm 4.1 Constructive approach algorithm

```

1: procedure CONSTRUCTIVEAPPROACH( $\mathcal{R}$ : Set of GT rules,  $\mathcal{C}$ : Set of graph constraints)
2:   for all  $R_X \in \mathcal{R}$  do
3:     for all  $C_Y \in \mathcal{C}$  do
4:       acs = SYNTHESIZEAPPLICATIONCONDITIONS( $R_X, C_Y$ )
5:        $R_X$ .applicationConditions =  $R_X$ .applicationConditions  $\cup$  acs
6:     end for
7:   end for
8: end procedure

```

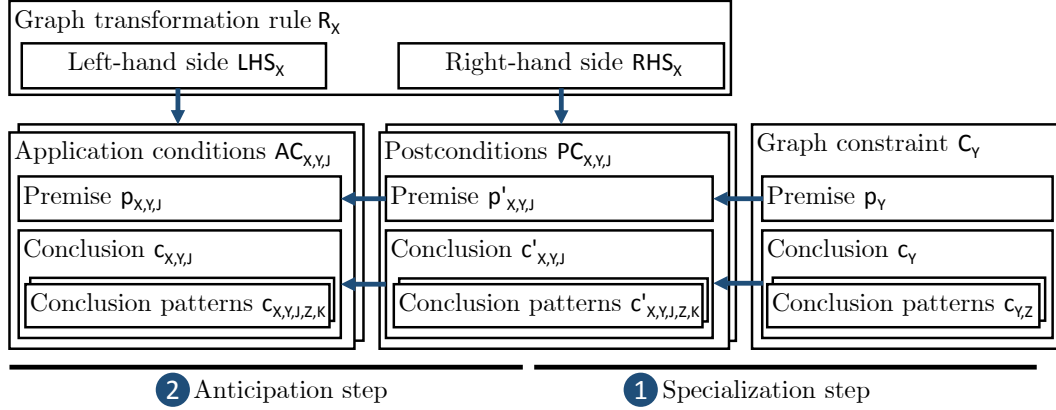


Figure 4.4: Schema of synthesis algorithm SYNTHESIZEAPPLICATIONCONDITIONS

postcondition conclusion patterns may result from each combination of postcondition premise $p'_{X,Y,J}$ and constraint conclusion pattern $c_{Y,Z}$ (as indicated by the index K). If multiple postcondition conclusion patterns result originate from one constraint conclusion pattern $c_{Y,Z}$, this situation reflects different situations in which the application of R_X may preserve C_Y via $c_{Y,Z}$.

For example, one postcondition conclusion pattern may reflect that the required match of the constraint conclusion pattern still exists after the application of R_X , and another postcondition conclusion pattern may reflect that an alternative extension of the match of p_Y shall be found. An example of this phenomenon is the synthesis algorithm iteration for the rule-constraint pair $(R_{\text{mod-w}}, C_{kTC,i})$.

After introducing postconditions of GT rules, we extend the set of criteria that determine when a GT rule is applicable as follows.

Definition 4.11 (GT rule applicability (with postconditions)). A GT rule is applicable if all criteria in Definition 3.25 are fulfilled and if any extension of RHS_X in the model resulting from applying R_X to a match of a postcondition premise $p'_{X,Y,J}$ can be further extended to a match of a postcondition conclusion pattern $c'_{X,Y,J,Z,K}$. \square

According to Definition 4.11, the GT rule R_X already preserves C_Y after the specialization step. Still, the disadvantage of a postcondition compared to an application condition is that a postcondition is checked *after* applying the GT rule, which implies that we have to roll back a GT rule application if the resulting model violates a postcondition.

To overcome this drawback, the second *anticipation step* (2, [91, Prop. 3.2]) transforms each synthesized postcondition $PC_{X,Y,J}$ into an application condition $AC_{X,Y,J}$. This step is called anticipation step because the created application conditions anticipate any violation of C_Y without applying R_X . The anticipation step works by applying R_X in reverse order to each pattern in $PC_{X,Y,J}$.

Table 4.2: Role of indices in synthesis algorithm description

Index	Relates to	Appears in steps
X	GT rule R_X	all steps
Y	constraint C_Y	all steps
Z	constraint conclusion pattern $c_{Y,Z}$ of C_Y	1.2, 1.3, 2
J	postcondition $PC_{X,Y,J}$	all steps
K	condition conclusion patterns $c_{X,Y,J,Z,K}/c'_{X,Y,J,Z,K}$	1.2, 1.3, 2

To sum up the roles of the involved indices in the synthesis algorithm, (i) the index X refers to the currently processed R_X (e.g., $X = i$ for R_i), (ii) the index Y refers to the currently processed graph constraint C_Y (e.g., $Y = i$ for $C_{kTC,i} = C_i$), (iii) the index Z refers to the currently processed constraint conclusion pattern $c_{Y,Z}$, (iv) the index J indicates that multiple postconditions may result for a given rule-constraint pair, and (v) the index K indicates that multiple application and postcondition conclusion patterns may result from a given pair of GT rule R_X and constraint conclusion $c_{Y,Z}$. Table 4.2 summarizes the described roles of the index variables.

Deckwerth and Varró's approach for handling attribute constraints [44] combines the attribute constraints of RHS_X and C_Y during the specialization step 1 in a similar way as described earlier. In contrast, the anticipation step 2 is more elaborate because we cannot simply drop attribute constraints that refer to variables that occur in RHS_X and not in LHS_X . We will discuss the attribute handling in detail for our running example.

Our goal is to establish weak consistency as an inductive invariant of the TC mechanism specification. We assume that weak consistency holds as a precondition prior to any GT rule application. This assumption allows us to filter the synthesis results using the filter rules summarized in Table 4.3. A synthesized pattern can be filtered out if it never matches in a weakly consistent topology. For instance, a pattern that contains a loop link variable or two parallel link variables never matches because $C_{no-loops}$ and $C_{no-parallel-links}$ are fulfilled for a weakly consistent topology (denoted by ∇_{nl} and ∇_{npl}). We also filter patterns that contain a contradiction in the attribute constraints (denoted by ∇_{cac}). A possible contradiction is a link variable that must have two different states (e.g., $s(v_{12}) = A \wedge s(v_{12}) = U$). If a postcondition premise $p'_{X,Y,J}$ is filtered out due to one of the filter rules, the entire corresponding post- and application conditions $PC_{X,Y,J}$ and $AC_{X,Y,J}$ is removed as well. Finally, we filter out a synthesized application condition after the anticipation step if it is always fulfilled for a weakly consistent topology (denoted by ∇_{wc}).

Table 4.3: Filter rules during synthesis algorithm

Symbol	Scope	Justification
∇_{cac}	pattern	no matches due to contradiction in attribute constraints
∇_{nl}	pattern	no matches due to loop link variable
∇_{npl}	pattern	no matches due to parallel link variables
∇_{wc}	application condition	application condition always fulfilled due to weak consistency precondition

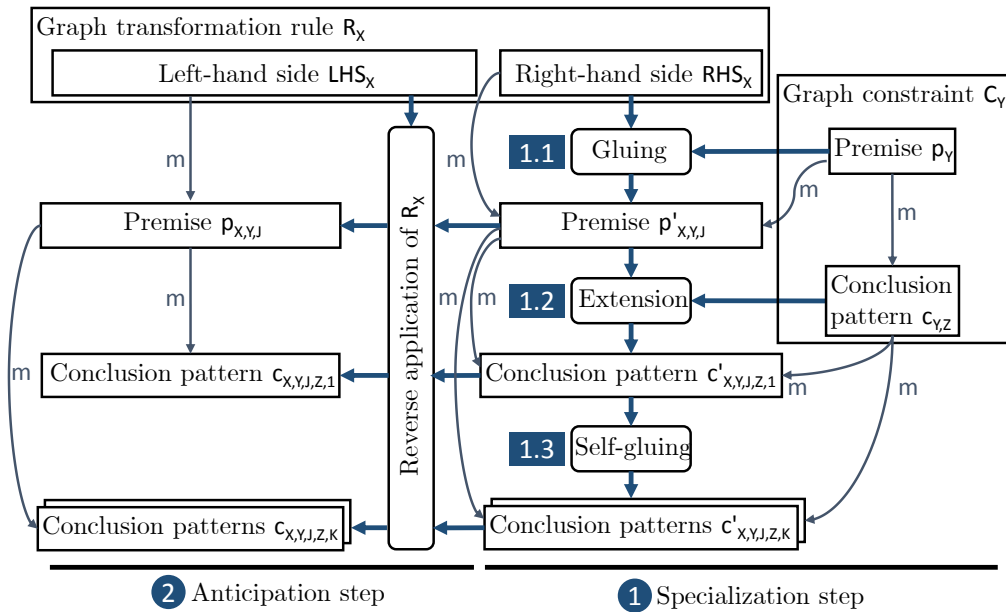


Figure 4.5: Detailed view of synthesis algorithm in Figure 4.4 (regular rectangles: artifacts, rounded rectangles: algorithm (sub-)steps, m : variable mapping)

4.3 APPLICATION OF CONSTRUCTIVE APPROACH TO RUNNING EXAMPLE

In this section, we apply the constructive approach to the TC and context event rules of the running example (Sections 4.3.1 and 4.3.2) and interpret the resulting synthesized application conditions (Section 4.3.3). In Section 4.2, we kept the discussion of the constructive approach deliberately declarative and refrained from presenting technical details of the application condition synthesis algorithm. In this section, we describe how to conduct each step precisely. Figure 4.5 is a refined view of Figure 4.4 that shows the substeps of SYNTHESIZEAPPLICATIONCONDITIONS as rounded rectangles with their respective input and output artifacts as regular rectangles.

4.3.1 Refinement of topology control rules

We now explain the synthesis algorithm step by step using the rule-constraint pair $(R_i, C_{kTC,i})$. For orientation purposes, Table 4.4 summarizes all results and the page counts for all six iterations of the synthesis algorithm for kTC.

When executing the synthesis algorithm, we have the choice among two perspectives of the involved patterns. The generic perspective based on graph patterns is to interpret object variables as nodes and association variables as edges (see Definition 3.7). The domain-specific perspective based on topology pattern is to interpret mote variables as nodes and link variables as edges (see Definition 3.8). As discussed earlier, the advantage of the latter perspective is the more concise and easier-to-understand representation

Table 4.4: Summary of application condition synthesis results for kTC per rule-constraint pair (R_X, C_Y) (num. gluings: number of postcondition premises in 1.1, num. ACs: number of application conditions after filtering 2, start, end: page range of discussion, result: figure showing refined rule)

Property	$(R_i, C_{kTC,i})$	$(R_a, C_{kTC,a})$	$(R_{+e}, C_{kTC,+a})$	$(R_{-e}, C_{kTC,-i})$	$(R_{\text{mod-w}}, C_{kTC,+a})$	$(R_{\text{mod-w}}, C_{kTC,-i})$
Num. gluings	6	12	12	6	12	6
Num. ACs	1	1	2	2	2	2
Start page	107	115	116	120	121	121
End page	111	118	118	121	121	125
Result (Fig.)	4.9g	4.11	4.12	4.14	4.18	4.18

of patterns. Also, no information is lost when using the latter perspective because the only difference is that we interpret the combination of a source-outgoing association variable, an incoming-target association variable, and an object variable of type Link as a single link variable (as apparent in Figures 3.6 to 3.9). Therefore, we use the domain-specific view of topology patterns in the following. Still, all steps can also be performed using generic graph patterns as well.

We abbreviate $C_{kTC,i}$ as C_i , $C_{kTC,a}$ as C_a , and φ_{kTC} as φ in all figures of this section to improve the readability of the synthesis results.

Synthesis iteration for (R_i, C_i) . During the gluing step [1.1](#), the premises $p'_{X,Y,J}$ of the synthesized postconditions $PC_{X,Y,J}$ are determined by calculating all possible overlaps of RHS_X and p_Y . The gluing step also determines the set of additional postconditions $PC_{X,Y,J}$. An overlap of two patterns is represented formally by a gluing.

Definition 4.12 (Gluing). A *gluing* $g_{A,B}$ of two patterns p_A and p_B is a pattern and two jointly surjective mappings m_A and m_B from p_A and p_B to the variables of $g_{A,B}$. The joint surjectivity of the mappings implies that each variable in $g_{A,B}$ has a preimage w.r.t. m_A or m_B , or both. The images of m_A and m_B must overlap in at least one mote variable. \square

The *preimage of an element $b \in B$ w.r.t. a function $f : A \rightarrow B$* , is an element $a \in A$ for which $f(a) = b$. A pair of functions $f_1 : A \rightarrow C, f_2 : B \rightarrow C$ is jointly surjective if each element in C has a preimage w.r.t. at least one of the functions f_1 or f_2 . In the context of the gluing step, we use the variables N_A and N_B to denote the number of mote variables in p_A and p_B , respectively. Due to the required overlap of the ranges of m_A and m_B , a gluing $g_{A,B}$ has at most $N_A + N_B - 1$ mote variables.

The link variables of a gluing are derived from the link variables of p_A and p_B as follows: If two mote variables in p_A or p_B are connected by a link variable, then their images in $g_{A,B}$ are also connected by a link variable. Each mote variable pair in $g_{A,B}$ is connected by at most one link variable. Technically, gluings with parallel link variables are possible as well. Still, such gluings can never be matched because we assume that the topology fulfills the no-parallel-links constraint $C_{no-parallel-links}$. Therefore, we only keep gluings with merged parallel link variables (∇_{npl}).

To encode the mappings m_A and m_B of a gluing, we label a gluing $g_{A,B}$ with a sequence of mote variables $g_{A,B}[v_{AB_1}, \dots, v_{AB_{N_A+N_B-1}}]$, where $v_{AB_1}, \dots, v_{AB_{N_A+N_B-1}} \in V_{p_B} \cup \{-\}$ (illustrated by Figure 4.6). All mote variable identifiers of p_B occur in the gluing sequence, and the remaining $N_A - 1$ entries are filled with the placeholder “-”.

In the following, we assume that mote variables can be compared by their identifier (i.e., a total ordering $<_{mv}$ exists on the set of mote variables). Furthermore, all mote variables are smaller than the placeholder “-” according to $<_{mv}$. The *lower (gluing) subsequence* consists of the first N_A entries of $g_{A,B}$ (denoted by $g_{A,B}[1, N_A]$) and represents the gluing mote variables that are in the image of m_A . The *upper (gluing) subsequence*

consists of the remaining $N_B - 1$ entries of $g_{A,B}$ (denoted by $g_{A,B}[N_A, N_A + N_B - 1]$) and represents the gluing mote variables that are only in the image of m_B .

The placeholder “_” has a different meaning depending on the part of the sequence in which it occurs. If a “_” occurs in the lower subsequence, it indicates that the corresponding gluing mote variable has no preimage in m_B . If a “_” occurs in the upper subsequence, it indicates that the corresponding gluing mote variable is absent from this gluing. To avoid creating isomorphic gluings, which only differ in the names of mote variables, we require that the subsequence $g_{A,B}[N_A + 1, N_A + N_B - 1]$ is sorted in ascending order according to $<_{mv}$.

Example 4.13 (Gluing). Figure 4.7 an exemplary gluing $p'_{i,i,2}$ of RHS_i , which is the RHS pattern of the inactivation rule R_i , and the premise $p_{kTC,i}$ of the inactive-link constraint $C_{kTC,i}$ (abbreviated as p_i and C_i in the figure, respectively). Instead of the symbol g for gluings in general, we use the naming convention for postcondition premise patterns directly. The figure shows the corresponding mappings m_{RHS-i} and m_{p-i} from the mote variables of RHS_i and $p_{kTC,i}$ to $p'_{i,i,2}$. The gluing sequence $g[1 _ | 2]$ represents that variables v_1 and v_2 of $p_{kTC,i}$ are mapped to the mote variables v_1 and v_3 of $p'_{i,i,2}$.

Algorithm 4.2 sketches an algorithm for generating all gluings according to the preceding description. The algorithm follows the generate-and-filter paradigm: First, an initial gluing sequence g_0 is populated where the first N_B entries correspond to the mote variable identifiers of p_B and the remaining entries (if exist) are filled with _ (lines 2–4). Then, all permutations $g_{A,B}$ of g_0 are generated (line 6). Finally, only those generated permutations are kept whose upper sequence is sorted ascendingly according to $<_{mv}$ (lines 8–13).

Finally, the attribute handling of the gluing substep consists of the following two modifications of the synthesized postcondition premises. For each postcondition premise, (i) the attribute constraints of RHS_X and C_Y are combined and attached to the postcondition premise, and (ii) the attribute variables in the combined attribute constraints are renamed according to the variable names of the postcondition premises.

$$\begin{array}{l}
 \text{Preimage in mapping } \mathbf{m}_A: v_{A,1}, \dots, v_{A,N_A} \\
 \text{Gluing } \mathbf{g}_{A,B}: \underbrace{[v_{AB,1}, \dots, v_{AB,N_A}]}_{\text{lower subsequence}} \mid \underbrace{[v_{AB,N_A+1}, \dots, v_{AB,N_A+N_B-1}]}_{\text{upper subsequence}}, v_{AB,C} \in V_B \cup \{_ \} \\
 \mathbf{g}_{A,B}[1, N_A] \qquad \mathbf{g}_{A,B}[N_A+1, N_A+N_B-1], \\
 \qquad \qquad \qquad \text{sorted according to } <_{mv}
 \end{array}$$

Figure 4.6: Gluing sequence (schematic)

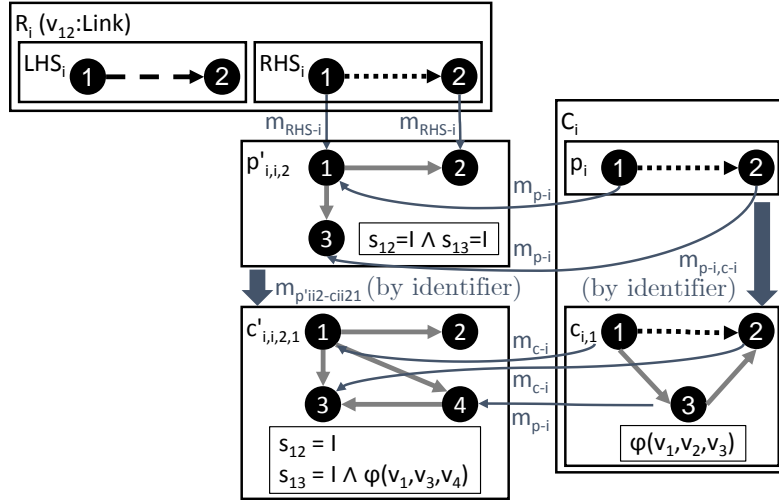


Figure 4.7: Example: Gluing $p'_{i,i,2}$ of RHS_i and $p_{kTC,i}$ with extension to $c'_{i,i,2,1}$

Example 4.14 (Gluing substep 1.1 for $(R_i, C_{kTC,i})$). Figures 4.8c to 4.8h show the six synthesized postcondition premises $p'_{i,i,1}, \dots, p'_{i,i,6}$ for the rule-constraint pair $(R_i, C_{kTC,i})$. To improve readability, we repeat the inactivation rule R_i and $C_{kTC,i}$ in Figures 4.8a and 4.8b, respectively. We list the attribute constraints copied from RHS_i and $C_{kTC,i}$ in the first and second row of the attribute constraints box, respectively. To illustrate how the synthesized attribute constraints are derived, we do not employ the compact notation to encode the link state information in all figures that relate to the specialization step 1.

Each postcondition premise $p'_{i,i,j}$ is labeled with the corresponding gluing sequence, which encodes the mappings to RHS_i and $C_{kTC,i}$. The lower and upper gluing subsequences are separated by a vertical bar (“|”) to improve readability. This separation is also represented by the mote variables of the lower and upper subsequences appearing in two different rows in the figure. The gluing mote variables v_1 and v_2 in this example always relate to the mote variables v_1 and v_2 of R_i , respectively, whereas the mote variables v_3 and v_4 only have preimages in $p_{kTC,i}$. For example, the gluing sequence $[1 _ | 2]$ that corresponds to $p'_{i,i,2}$ represents that the mote variables v_1 and v_3 of $p'_{i,i,2}$ map to v_1 and v_2 of $C_{kTC,i}$, respectively, and that v_2 of $p'_{i,i,2}$ has no preimage in $C_{kTC,i}$.

None of the postcondition premises $p'_{i,i,j}$ can be filtered out according to the rules in Table 4.3. Therefore, we create six postconditions for R_i in this iteration.

In the next two substeps, the extension substep 1.2 and the self-gluing substep 1.3, we construct the postcondition conclusion $c'_{X,Y,J}$ for each postcondition $PC_{X,Y,J}$ separately. During the extension substep 1.2, each constraint conclusion pattern $c_{Y,Z}$ is used

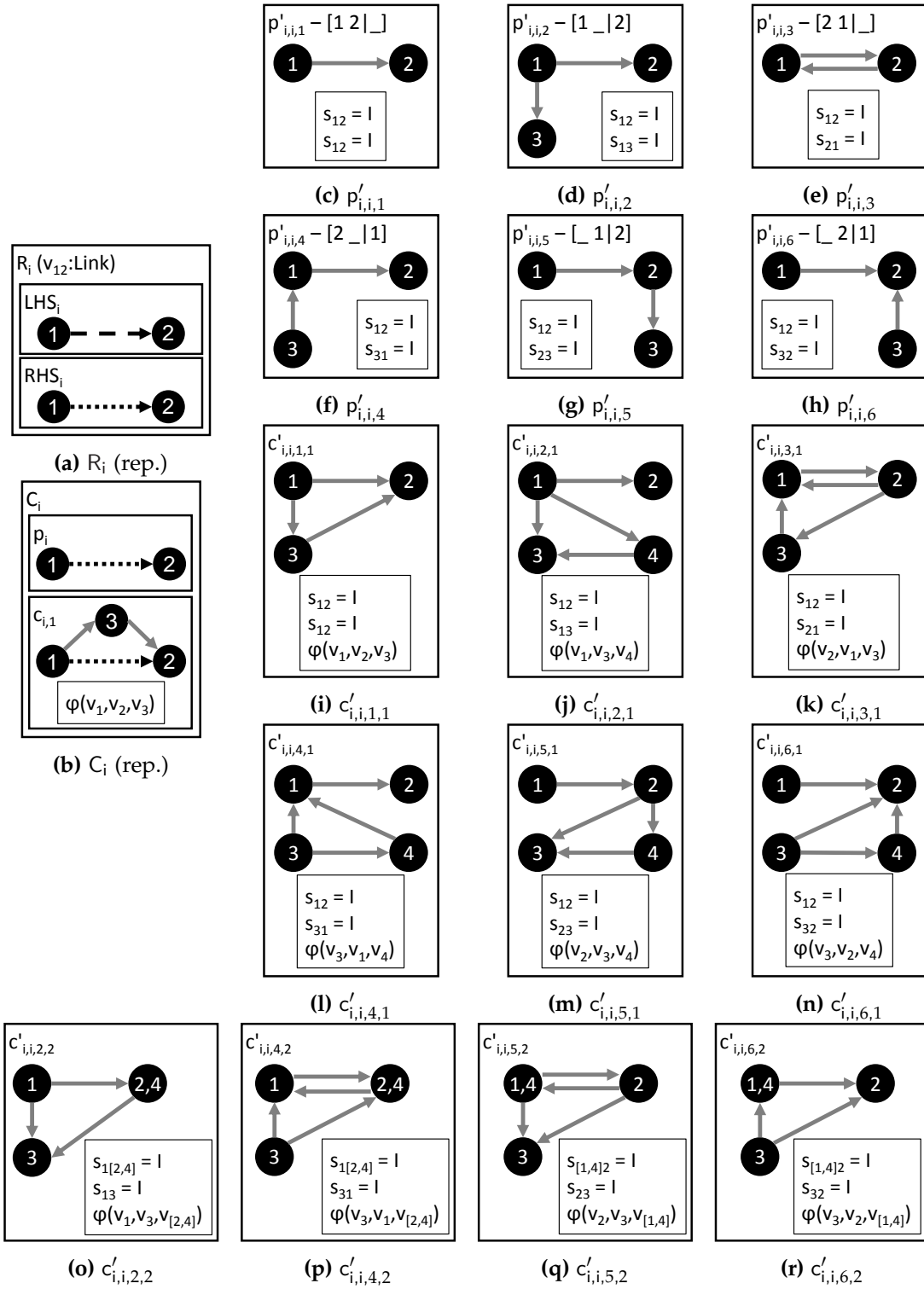


Figure 4.8: Results of specialization step 1 for $(R_i, C_{kTC,i})$

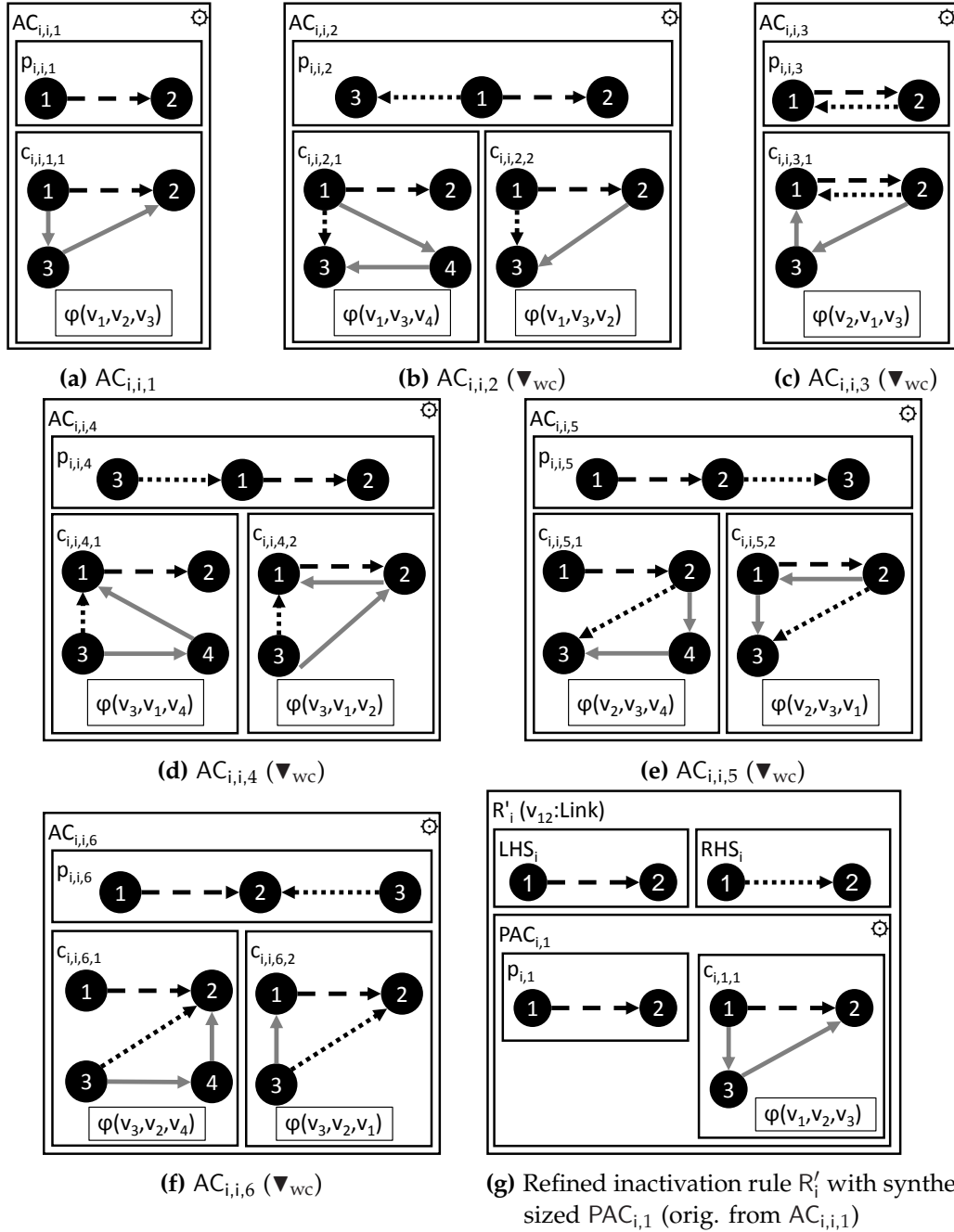


Figure 4.9: Results of anticipation step 2 for $(R_i, C_{kTC,i})$

Algorithm 4.2 Gluing generation algorithm

```

1: function ALLGLUINGS ( $p_A$ : Pattern,  $p_B$ : Pattern)
2:    $g_0$ : Gluing sequence of length  $N_A + N_B - 1$ 
3:    $g_0[1, N_B] = V_{p_B}$   $\triangleright$  Copy mote variable identifiers of  $p_B$  into first  $N_B$  entries
4:    $g_0[N_B + 1, N_A + N_B - 1] = -$   $\triangleright$  Fill remaining entries
5:
6:   allGluing : Set of gluings = ALLPERMUTATIONS( $g_0$ )
7:
8:   filteredGluings : Set of gluings =  $\emptyset$ 
9:   for all  $g_{A,B} \in$  allGluing do
10:    if  $g_{A,B}[N_A + 1, N_A + N_B - 1]$  is sorted ascendingly according to  $<_{mv}$  then
11:      filteredGluings = filteredGluings  $\cup$   $g_{A,B}$ 
12:    end if
13:  end for
14:
15:  return filteredGluings
16: end function

```

separately to extend the postcondition premise $p'_{X,Y,J}$ to a *basic postcondition conclusion pattern* $c'_{X,Y,J,Z,1}$. The resulting basic postcondition conclusion pattern is unique. The extension is carried out long the two mappings from (i) constraint premise p_Y to postcondition premise $p'_{X,Y,J}$ and (ii) constraint premise p_Y to constraint conclusion pattern $c_{Y,Z}$.

Example 4.15 (Extension substep 1.2 in detail). To illustrate the extension substep in detail, we use Figure 4.7 again. In the center part, the figure shows one possible gluing $p'_{i,i,2}$ of RHS_i and $p_{kTC,i}$ and its extension along the premise-gluing mapping m_{p-i} and the premise-conclusion mapping $m_{p-i,c-i}$.

For the given combination of gluing $p'_{i,i,2}$ and constraint conclusion pattern $c_{kTC,i,1}$, the resulting pattern $c'_{i,i,2,1}$ is unique and obtained by adding mote variable v_4 and link variables v_{14} and v_{43} as well as the attribute constraint $\varphi(v_1, v_2, v_3)$. In this example, the constraint and postcondition premise-conclusion mappings are shown as block arrows instead of individual mote mappings to improve readability. As usual, the exact mappings are implicitly given by equal mote variable identifiers in premise and conclusion patterns.

Example 4.16 (Extension substep 1.2 for $(R_i, C_{kTC,i})$). Figure 4.8i to Figure 4.8n show the six basic conclusion patterns $c'_{i,i,1,1}, \dots, c'_{i,i,6,1}$ that result from the extension substep for $(R_i, C_{kTC,i})$. All constraints in our example have only one conclusion pat-

tern. Therefore, to improve readability, we omit the counter Z , which is always 1. For instance, $c'_{i,i,3,1}$ in Figure 4.8k represents $c'_{i,i,3,1,1}$ according to the naming scheme shown in Figure 4.5.

The basic conclusion patterns use the shorthand notation $\varphi_{kTC}(v_1, v_2, v_3)$ to denote that the triangle consisting of the mote variables v_1 , v_2 , and v_3 fulfills the kTC -specific condition for inactivating a link (see also Equation (3.6)).

The subsequent self-gluing substep **1.3** is performed for each basic postcondition conclusion pattern $c'_{X,Y,J,Z,1}$ separately. During this step, additional postcondition conclusion patterns are synthesized by gluing the basic postcondition conclusion pattern $c'_{X,Y,J,Z,1}$ with itself. The set of conclusion patterns $c'_{X,Y,J,Z,K}$ corresponds to all different ways how a match of the constraint premise p_Y (represented by a match of the postcondition premise $p'_{X,Y,J}$) can be extended to a match of the constraint conclusion pattern $c_{Y,Z}$. To obtain a self-gluing, we merge mote variables of the basic postcondition conclusion pattern. Only those self-glued postcondition conclusion patterns $c'_{X,Y,J,Z,K}$ are kept for which an injective mapping from $p'_{X,Y,J}$ and a total injective mapping from $c_{Y,Z}$ exists. If the first mapping were not injective, a match of the postcondition premise $p'_{X,Y,J}$ could never be extended to a match of the self-glued conclusion pattern $c'_{X,Y,J,Z,K}$. If the second mapping were not injective, a match of the postcondition conclusion pattern $c'_{X,Y,J,Z,K}$ could not witness that the graph constraint C_Y is fulfilled.

The required injectivity from $p'_{X,Y,J}$ and $c_{Y,Z}$ to $c'_{X,Y,J,Z,K}$ implies that only those mote variables that were introduced freshly during the preceding extension substep **1.2** can be merged with the other mote variables. As during the gluing substep, we keep only those gluings without parallel link variables (∇_{npl}). After the self-gluing substep, all postconditions are complete.

The attribute handling for the extension substep and the self-gluing substep is performed analogously to the gluing substep. The attribute constraints of $c'_{X,Y,J,Z,K}$ consist of the attribute constraints of $p'_{X,Y,J}$ plus a copy of the attribute constraints of $c_{Y,Z}$ with appropriately renamed variables.

Example 4.17 (Self-gluing substep **1.3** for $(R_i, C_{kTC,i})$). Figures 4.8o to 4.8r shows the resulting self-glued postcondition conclusion patterns $c'_{i,i,J,2}$ for $(R_i, C_{kTC,i})$. Only the four basic conclusion patterns $c'_{i,i,2,1}$, $c'_{i,i,4,1}$, $c'_{i,i,5,1}$, and $c'_{i,i,6,1}$ are eligible for the self-gluing substep because each self-glued postcondition conclusion pattern must have at least three mote variables due to the required injective mappings from $p'_{i,i,J}$ and $C_{kTC,i,1}$.

If two mote variables are glued together, the resulting mote variable in the self-gluing postcondition conclusion pattern contains a comma-separated list of the original mote variable identifiers (e.g., **2,4** in $c'_{i,i,2,2}$, which results from gluing v_2 and v_4 in $c'_{i,i,2,1}$). In attribute constraints, the identifiers of glued mote variables are shown in square brackets (e.g., “[2,4]” in $c'_{i,i,2,2}$). Technically, additional self-gluings are pos-

sible (e.g., of mote variables v_1 and v_3). However, all of these gluings result in link variables that form loops and never match due to the weak consistency precondition (∇_{nl}). This observation can be generalized as follows: During the self-gluing sub-step, we only consider pairs of mote variables in the basic postcondition conclusion pattern that are not neighbors (e.g., v_2 and v_4 in $c'_{i,2,2}$).

The final anticipation step ② is carried out for each synthesized postcondition $PC_{X,Y,J}$ separately. We obtain the synthesized application condition $AC_{X,Y,J}$ from $PC_{X,Y,J}$ by applying the GT rule R_X in reverse order to the postcondition premise $p'_{X,Y,J}$ and all conclusion patterns $c'_{X,Y,J,Z,K}$. During the inverse application of R_X , we first consider modifications of the pattern graph (i.e., additions and removals of mote and link variables) and then handle the attribute constraints. We carry out the attribute handling for each premise and conclusion pattern separately: (i) The attribute constraints of LHS_X are added to the attribute constraints of the application condition pattern. (ii) If an attribute value is reassigned by the GT rule application (e.g., $s_{12} = I$ in the right-hand side of R_i) and the left-hand side LHS_X also refers to this attribute (e.g., $s_{12} = U$ in R_i), then we distinguish the LHS and RHS attribute values. We the suffix R to the variable that occurs in RHS_X (e.g., s_{12R}) and insert an existential quantification over the domain of the RHS variable (e.g., $\exists s_{12R} \in \{A, I, U\} : s_{12R} = I$). (iii) We apply the same strategy if an attribute variable relates to an element created by R_X (i.e., a variable that occurs in RHS_X and not in LHS_X). During the anticipation step, such elements are removed from the postcondition patterns because R_X is applied in reverse order. (iv) All attribute variables that are neither reassigned nor related to added elements are left unchanged. After the anticipation step, we may filter the synthesized application conditions. We drop a synthesized application condition if it is always fulfilled on a weakly consistent topology (∇_{wc}).

Example 4.18 (Anticipation step ② for $(R_i, C_{kTC,i})$). Figure 4.9 shows the unfiltered set of synthesized application conditions for $(R_i, C_{kTC,i})$. In contrast to the preceding examples, we employ the compact notation here to reduce the size of the figures. Using the compact notation is possible because the attribute constraints (esp. related to link states) are guaranteed to be free of contradictions due to the previous filtering steps (∇_{cac}). Additionally, we renamed all merged mote variables to conform to the mote variable identifiers of R_i . For instance, v_2 in $c_{i,i,2,2}$ corresponds to $v_{[2,4]}$ in $c'_{i,i,2,2}$. From now on, we mark synthesized application conditions with a gear symbol (⚙️) to distinguish them from the application conditions that existed prior to executing the application condition synthesis algorithm.

Regarding the attribute handling, the state of v_{12} is reassigned by R_i . This means that the attribute constraint $s_{12} = I$ is replaced with the following attribute constraint during the anticipation step:

$$\exists s_{12R} \in \{A, I, U\} : s_{12R} = I.$$

This attribute constraint is a tautology, which is always fulfilled. In this and the following examples, we omit an attribute constraint that constitutes a tautology.

The five application conditions $AC_{i,i,2}, \dots, AC_{i,i,6}$ can be filtered out because they are always fulfilled on weakly consistent topologies (∇_{wc}). This means that the only additional application condition synthesized for $(R_i, C_{kTC,i})$ is $AC_{i,i,1}$. The refined inactivation rule R'_i is shown in Figure 4.9g. In R'_i , we rename $AC_{i,i,1}$ to $PAC_{i,i,1}$ according to the naming conventions of GT rules.

The original constructive approach in [91] contains a third major step: the *minimization step*, which removes synthesized application conditions that block GT rule applications to inconsistent models. The purposes of this minimization step and our proposed filter rules in Table 4.3 are similar. However, the reduction criterion of the minimization step in [91, Constr. 3.3.] is not applicable to GT rules that contain attribute constraints because the reduction criterion relates only to created elements of R_X . If we applied the original minimization step in our scenario, too many candidates of premises would be dropped and the resulting refined rule R'_X would still violate the constraint C_Y . Therefore, we skip the minimization step from [91] in this work and apply the filter rules shown in Table 4.3 instead.

After considering the steps of the application condition synthesis algorithm for the rule-constraint pair $(R_i, C_{kTC,i})$ in detail, we present the corresponding results for the remaining rule-constraint pairs in this section and Section 4.3.2. Conceptually, the content of these subsections constitutes one large example. Still, to improve readability we refrain from presenting this content in example environments. We only discuss the synthesis results for combinations of GT rules and graph constraints that result in additional application conditions for the GT rules (X in Table 4.1). For all other combinations, the constructive approach produces either no additional application conditions or weaker application conditions, which are already implied by the existing application conditions. We continue with $(R_a, C_{kTC,a})$.

Synthesis iteration for $(R_a, C_{kTC,a})$. Figure 4.10c to Figure 4.10n show the twelve synthesized postcondition premises $p'_{a,a,1}, \dots, p'_{a,a,12}$ for $(R_a, C_{kTC,a})$ generated during the gluing substep 1.1. To improve readability, we repeat the activation rule R_a and the active-link constraint $C_{kTC,a}$ in Figures 4.10a and 4.10b here. In comparison to $(R_i, C_{kTC,i})$, twice as many postcondition premises are created because the premise of $C_{kTC,a}$ has three mote variables (instead of two mote variables for $C_{kTC,i}$). Therefore, the postcondition premise has at most $2 + 3 - 1 = 4$ mote variables. The lower and the upper gluing subsequence have both length 2.

This example also illustrates why only gluings with ascending upper gluing sequence are kept: The gluing sequence $[12 \mid 3 _]$ is kept and corresponds to $p'_{a,a,1}$, but the gluing sequence $[12 \mid _ 3]$ is dropped because the resulting gluings would be isomorphic to $p'_{a,a,1}$: We obtain the latter gluing by renaming mote variable v_3 of the former gluing to v_4 . None of the postcondition premises can be filtered according to Table 4.3. Thus, twelve postconditions $PC_{a,a,1}, \dots, PC_{a,a,12}$ are created from the postcondition premises during the gluing substep.

The extension substep 1.2 and the self-gluing substep 1.3 can be skipped for the rule-constraint pair $(R_a, C_{kTC,a})$ because the conclusion of $C_{kTC,a}$ is empty. Therefore, we continue with the anticipation step 2 by applying R_a in reverse order to the twelve postcondition premises $p'_{a,a,j}$. The only effect of applying R_a in reverse order to a postcondition premise is to introduce an attribute constraint $s_{12} = U$ in each premise and to replace the RHS attribute constraint $s_{12} = A$ with the attribute constraint $\exists s_{12R} \in \{A, I, U\} : s_{12R} = A$, which is a tautology and omitted therefore. Figures 4.100 to 4.10r show four selected application condition premises. We chose these application condition premises because $p_{a,a,1}$, $p_{a,a,2}$, and $p_{a,a,8}$ are the only postcondition premises that have three mote variables and contain merged link variables of R_a and $C_{kTC,a}$. Furthermore, $p_{a,a,3}$ is a representative of the remaining nine application condition premises, which all contain four mote variables and no merged link variables. Each application condition premise results in one potential negative application condition of R_a . For conciseness reasons, we show the premises instead of the full negative application conditions here.

Some of the application conditions can be filtered out because they are always fulfilled on weakly consistent topologies (∇_{wc}). For example, a weakly consistent topology never contains a match of $p_{a,a,3}$ (shown in Figure 4.10q). Otherwise, the motes and links matched by the link variable triangle consisting of v_1, v_3, v_4 would constitute a match of the premise—and, thereby, a violation—of $C_{kTC,a}$. In total, we can filter out eleven of the twelve synthesized postcondition premises: $p_{a,a,2}, \dots, p_{a,a,12}$. Thus, only the application condition $AC_{a,a,1}$ must be added to the activation rule R_a to ensure that the refined GT rule R'_a preserves $C_{kTC,a}$.

The refined activation rule R'_a , whose application condition $NAC_{a,1}$ corresponds to $p_{a,a,1}$, is shown in Figure 4.11.

The iteration of the synthesis algorithm for $(R_a, C_{kTC,a})$ is the last one that affects a TC rule.

4.3.2 Refinement of context event rules

The remaining four iterations of the synthesis algorithm for the rule-constraint pairs $(R_{+e}, C_{kTC,a})$, $(R_{-e}, C_{kTC,i})$, $(R_{\text{mod-w}}, C_{kTC,a})$, and $(R_{\text{mod-w}}, C_{kTC,i})$ affect context event rules and are discussed in the following.

Synthesis iteration for $(R_{+e}, C_{kTC,a})$. Figure 4.13 shows the twelve generated postcondition premises $p'_{+e,a,1}, \dots, p'_{+e,a,12}$ that result from the gluing substep 1.1 for $(R_{+e}, C_{kTC,a})$.

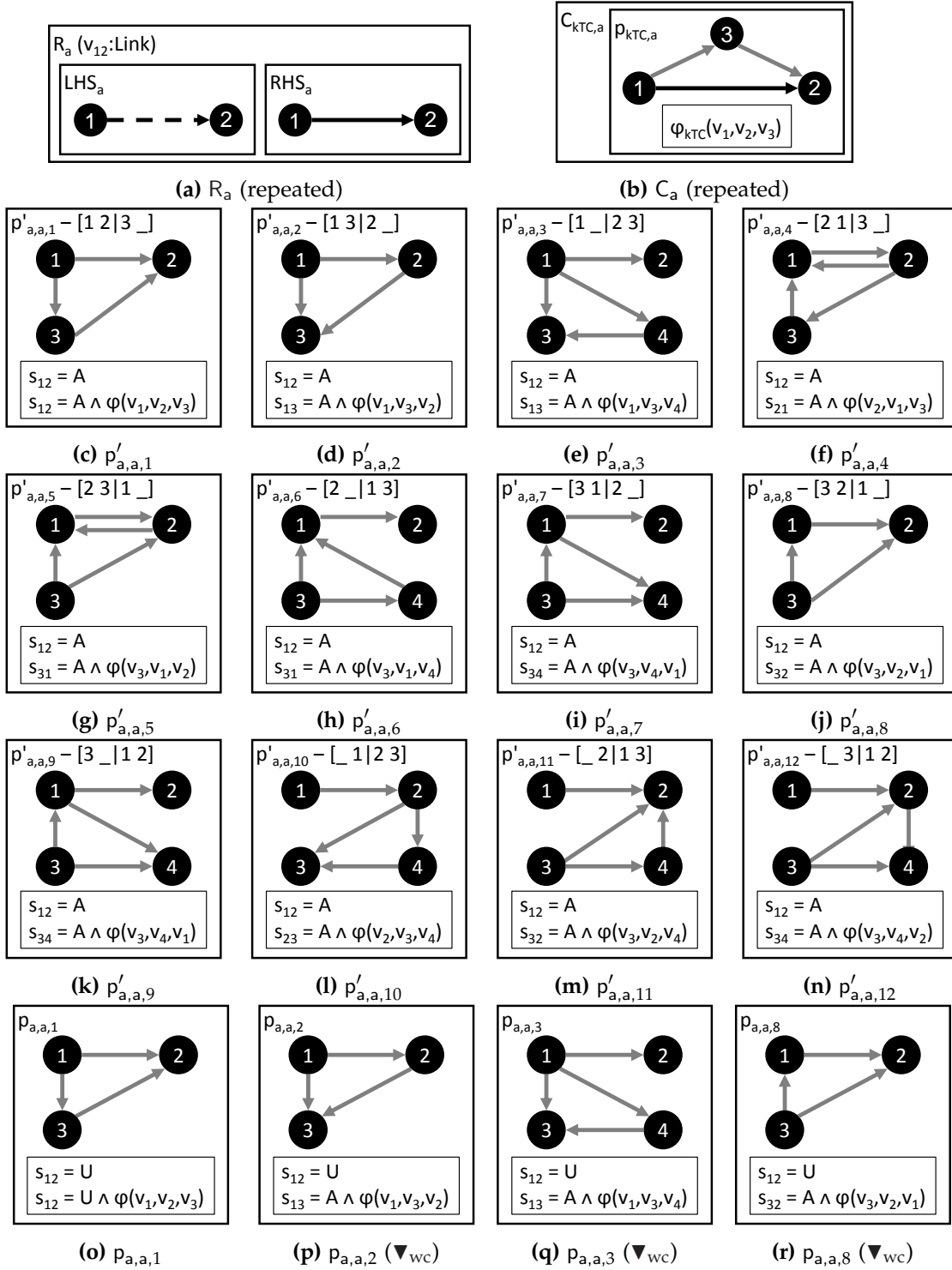


Figure 4.10: Postcondition and application condition premises $p'_{a,a,j}/p_{a,a,j}$ for $(R_a, C_{kTC,a})$

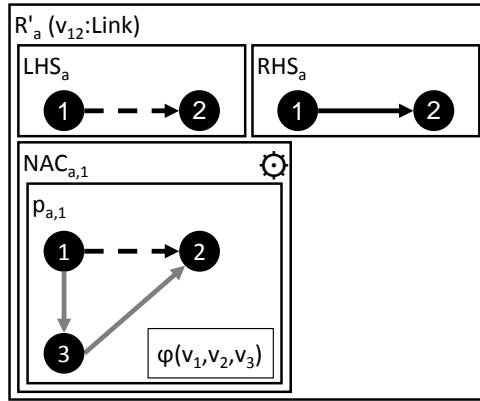


Figure 4.11: Refined activation rule R'_a with synthesized application condition $NAC_{a,1}$ (resulting from $p_{a,a,1}$ and ensuring preservation of $C_{kTC,a}$)

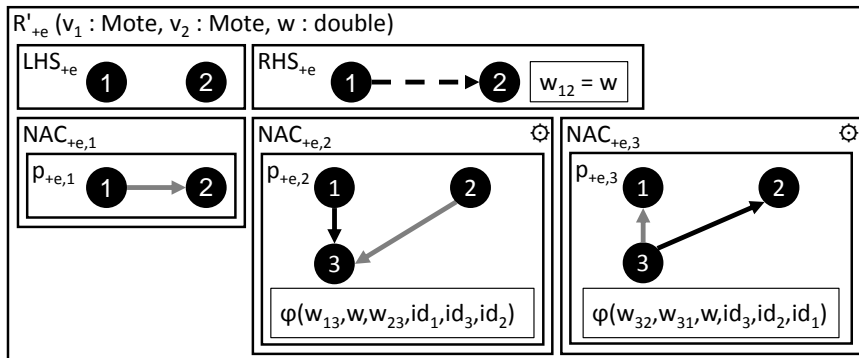
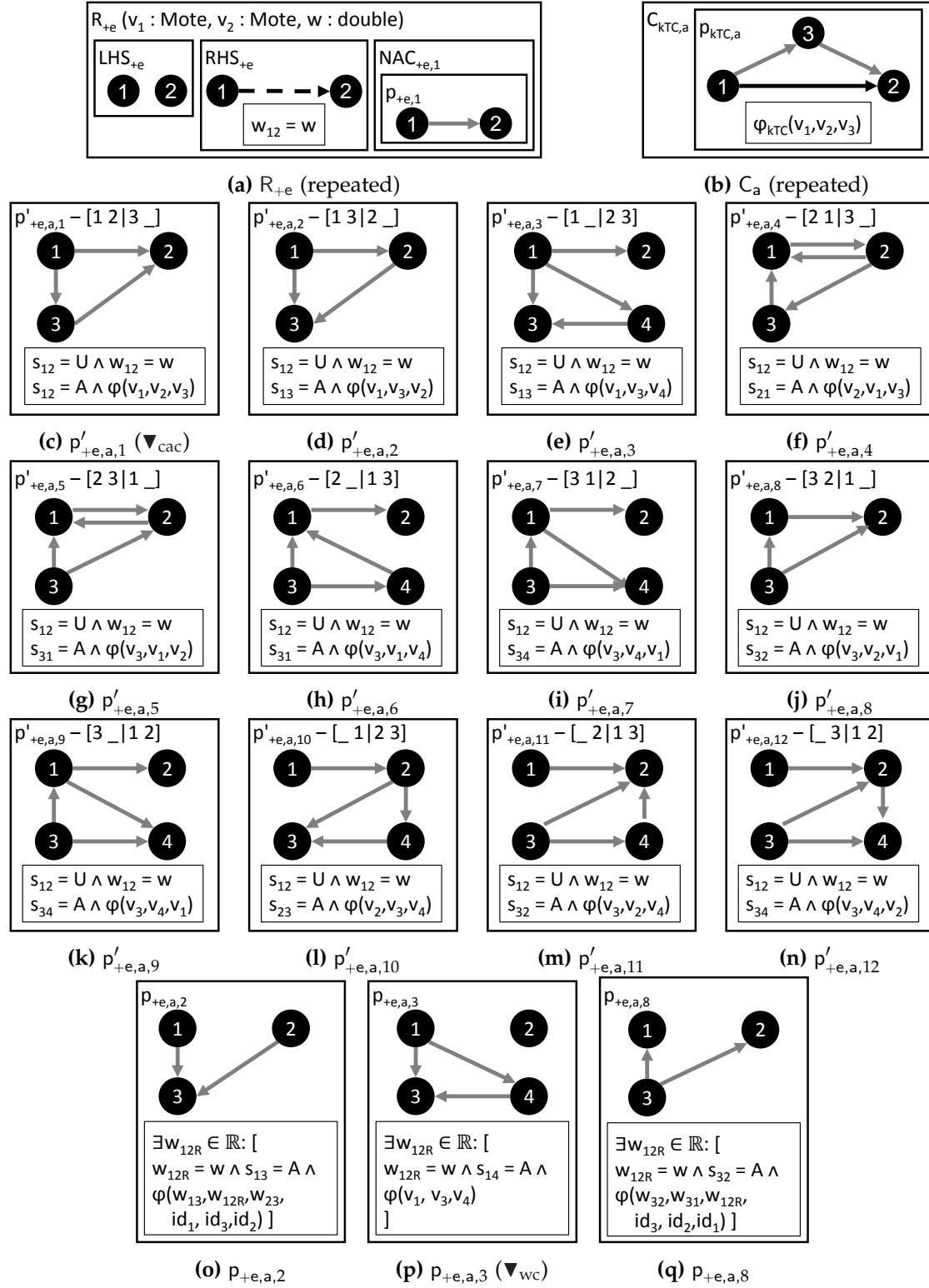


Figure 4.12: Refined link addition rule R'_{+e} with synthesized application conditions $NAC_{+e,1}$ and $NAC_{+e,2}$ (originating from $p_{+e,a,2}$ and $p_{+e,a,8}$ and ensuring the preservation of $C_{kTC,a}$)



The pattern graphs of the gluings are almost identical compared to $(R_a, C_{kTC,a})$. Only the first line of the combined attribute constraints, which originates from RHS_{mod-w} differs: $s_{12} = U \wedge w_{12} = w$, where w is an implicit parameter, instead of $s_{12} = A$. During this substep, we filter out $p'_{+e,a,1}$ because of the contradictory attribute constraints $s_{12} = U \wedge s_{12} = A$ (∇_{cac}). The remaining eleven of the twelve postcondition premises $p'_{+e,a,2}, \dots, p'_{+e,a,12}$ all result in new postconditions $PC_{+e,a,J}$ of R_{+e} . The extension substep 1.2 and the self-gluing substep 1.3 can be skipped due to the empty conclusion of $C_{kTC,a}$.

For the anticipation step 2, we chose the representative application condition premises $p'_{+e,a,1}$, $p'_{+e,a,2}$, $p'_{+e,a,3}$, and $p'_{+e,a,8}$. The argumentation for their representativeness is analogous compared to $(R_a, C_{kTC,a})$. In comparison to $(R_a, C_{kTC,a})$, the attribute handling during the anticipation step for $(R_{+e}, C_{kTC,a})$ is different because the reverse application of R_{+e} to the postcondition premises $p'_{+e,a,J}$ removes v_{12} . Therefore, we (i) introduce existential quantifications for all attributes of the removed link variable v_{12} that are referenced in the attribute constraints (i.e., w_{12}), (ii) append the suffix “R” to their indices to avoid confusion with attributes of LHS variables (i.e., $w_{12} \rightarrow w_{12R}$), and (iii) use the kTC -specific predicate with six parameters (see Equation (3.4)) in the application condition premises $p_{+e,a,J}$ for $p_{+e,a,2}$ and $p_{+e,a,8}$. For $p_{+e,a,2}$, the expression $\varphi_{kTC}(v_1, v_3, v_2)$, which is equal to $\varphi_{kTC}(w_{13}, w_{12}, w_{23}, id(v_1), id(v_3), id(v_2))$, becomes $\varphi_{kTC}(w_{13}, w_{12R}, w_{23}, id(v_1), id(v_3), id(v_2))$. For $p_{+e,a,3}$, we keep the ternary constraint $\varphi_{kTC}(v_1, v_2, v_4)$ because its parameters do not refer to the existentially quantified variable w_{12R} . As in the previous iterations, we omit the attribute constraint $s_{12R} \in \{A, I, U\} : s_{12R} = U$ because it is a tautology.

The refined link addition rule R'_{+e} is shown in Figure 4.12 with the negative application conditions $NAC_{+e,2}$ and $NAC_{+e,3}$, which correspond to the synthesized application conditions $p_{+e,a,2}$ and $p_{+e,a,8}$. To improve readability, we also substitute the existential quantification of w_{12R} and the assignment constraint $w_{12R} = w$ with an immediate reference to the parameter w .

Synthesis iteration for $(R_{-e}, C_{kTC,i})$. Figures 4.15c to 4.15h show the six postcondition premises $p'_{-e,i,1}, \dots, p'_{-e,i,6}$ that result from the gluing substep 1.1 for $(R_{-e}, C_{kTC,i})$. We omit the results of the extension substep 1.2 and self-gluing substep 1.3 for $(R_{-e}, C_{kTC,i})$ because they are analogous compared to the results for $(R_i, C_{kTC,i})$ shown in Figure 4.8. The sole difference between $c'_{-e,i,J,1}$ and $c'_{i,i,J,1}$ is that that the former patterns lack the link variable v_{12} . In the self-glued postcondition conclusion patterns $c'_{-e,i,J,2}$ the removed link variable v_{12} is partly reconstructed: $v_{[1,4]2}$ for $c'_{-e,i,5,2}$ and $c'_{-e,i,6,2}$ (see also Figures 4.15m and 4.15n), and $v_{1[2,4]}$ for $c'_{-e,i,2,2}$ (see also Figure 4.15j).

During the anticipation step 2, the reverse application of R_{-e} causes the addition of a link variable v_{12} (Figure 4.15). This link variable is also added if the postcondition premise and conclusion patterns already contains a link variable v_{12} , leading to parallel link variables. We remove all patterns that contain parallel link variables (∇_{npl}). This filter rule applies to the premise $p_{-e,i,1}$ (leading to the removal of $AC_{-e,i,1}$) as well as the

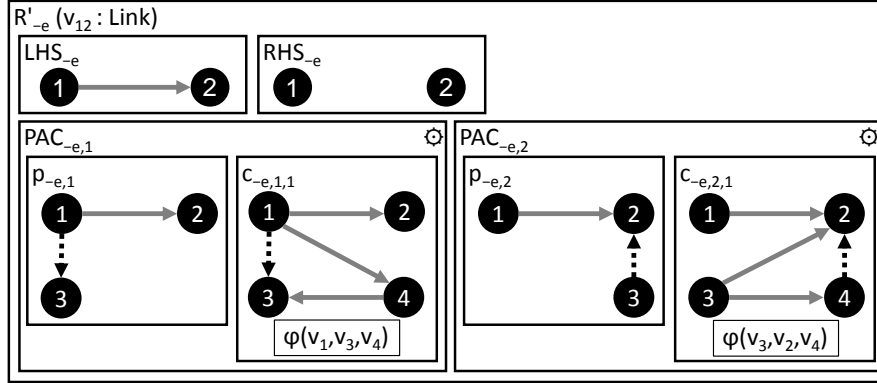


Figure 4.14: Refined link removal rule R'_e with synthesized application conditions $PAC_{-e,1}$ and $PAC_{-e,2}$ (corresponding to $AC_{-e,i,2}$ and $AC_{-e,i,6}$)

self-glued conclusion patterns $c_{-e,i,2,2}$ and $c_{-e,i,6,2}$. Furthermore, the following application conditions are always fulfilled for weakly consistent topologies and can be removed (∇_{wc}): $AC_{-e,i,3}$, $AC_{-e,i,4}$, and $AC_{-e,i,5}$. After applying all filtering rules, only two application conditions remain: $AC_{-e,i,2}$ (with a single conclusion pattern $c_{-e,i,2,1}$) and $AC_{-e,i,6}$ (also with a single conclusion pattern $c_{-e,i,6,1}$).

The refined link removal rule with the positive application conditions $PAC_{-e,1}$ and $PAC_{-e,2}$, which correspond to $AC_{-e,i,2}$ and $AC_{-e,i,6}$, is shown in Figure 4.14.

Synthesis iteration for $(R_{mod-w}, C_{kTC,a})$. The result of the gluing substep 1.1 for the rule-constraint pair $(R_{mod-w}, C_{kTC,a})$ is identical to the gluing of RHS_{+e} and p_a because the patterns involved in the gluing substep are identical in both constellations (see Figure 4.13). The anticipation step 2 for $(R_{mod-w}, C_{kTC,a})$, in contrast, is similar to the anticipation step for $(R_{+e}, C_{kTC,a})$. The only difference is that the link variable v_{12} is also present in the application condition premises $p_{mod-w,a,j}$. Still, as for $(R_{+e}, C_{kTC,a})$, the kTC -specific attribute constraint φ_{kTC} relates to the RHS link weight w_{12R} of v_{12} . All but two application condition premises can be omitted because they never match on a weakly consistent topology: $p_{mod-w,a,2}$ and $p_{mod-w,a,8}$ (Figure 4.16). Finally, we rename the mote variables in $p_{mod-w,a,2}$ and $p_{mod-w,a,8}$ to conform to the mote variable names in R_{mod-w} . Figure 4.18 shows the refined link-weight modification rule R'_{mod-w} with the corresponding negative application conditions $NAC_{mod-w,1}$ and $NAC_{mod-w,2}$. The figure also shows the results of the following, final iteration of the synthesis algorithm for $(R_{mod-w}, C_{kTC,i})$.

Synthesis iteration for $(R_{mod-w}, C_{kTC,i})$. Figures 4.17c to 4.17h show the six postcondition premises $p'_{mod-w,i,1}, \dots, p'_{mod-w,i,6}$ that result from the gluing substep 1.1 for the rule-constraint pair $(R_{mod-w}, C_{kTC,i})$. To improve readability, we repeat the link-weight modification rule R_{mod-w} and the inactive-link constraint $C_{kTC,i}$ in Figures 4.17a and 4.17b. The premise $p'_{mod-w,i,1}$ can be filtered out because it never matches due to the contra-

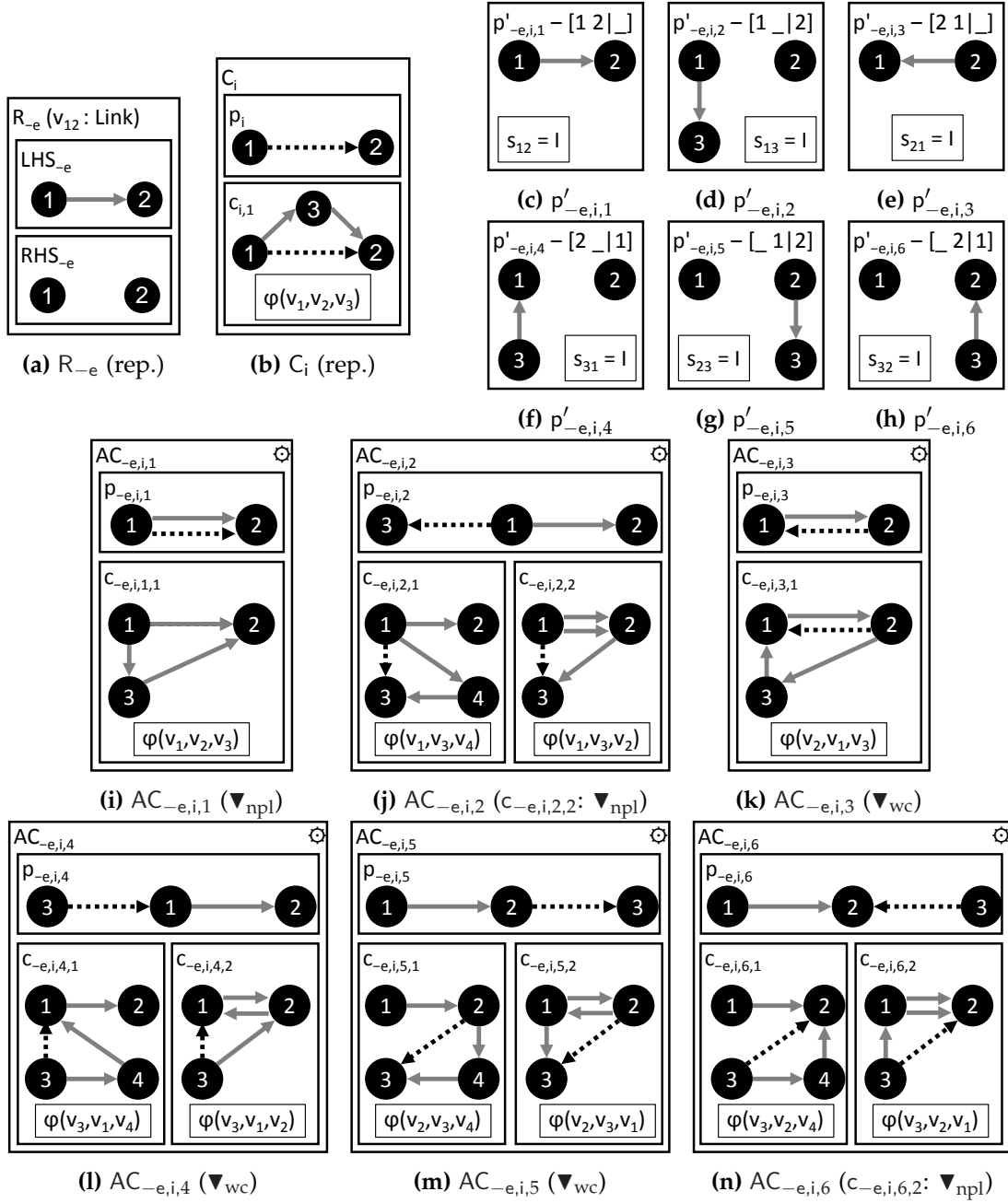


Figure 4.15: Synthesis results for $(R_{-e}, C_{kTC,i})$

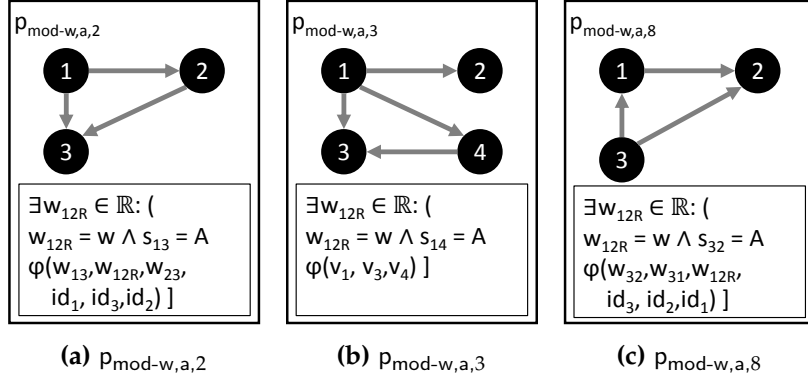


Figure 4.16: Selected application condition premises $p_{\text{mod-w,a,j}}$ for $(R_{\text{mod-w}}, C_{\text{kTC,a}})$ (after 2)

dictory attribute constraints (∇_{cac}). The results of the extension substep 1.2 and the self-gluing substep 1.3 are omitted here because they are similar compared to the post-condition conclusion patterns shown in Figure 4.8. The pattern graphs are identical, but the first row of the attribute constraints differs. Compared to $(R_i, C_{\text{kTC,i}})$, the attribute constraint $s_{12} = I$ is replaced with $s_{12} = U \wedge w_{12} = w$. In total, five postconditions $PC_{\text{mod-w,i,1}}, \dots, PC_{\text{mod-w,i,6}}$ result from the specialization step.

During the anticipation step 2, the attribute constraints are adjusted by replacing the RHS variable w_{12} with an existential quantification of w_{12R} and an assignment attribute constraint (as for $(R_{+e}, C_{\text{kTC,a}})$ and $(R_{\text{mod-w}}, C_{\text{kTC,a}})$). We omit the corresponding constraint for the state of v_{12} (i.e., $\exists s_{12R} \in \{A, I, U\} : s_{12R} = U$) as usual because it is a tautology.

Three application conditions $AC_{\text{mod-w,i,3}}, \dots, AC_{\text{mod-w,i,5}}$ can be filtered out because they are always fulfilled for weakly consistent topologies (∇_{wc}). Only the remaining application conditions are shown in Figures 4.17i and 4.17j.

Notably, in contrast to $(R_i, C_{\text{kTC,i}})$, we cannot filter out any gluings of these application conditions. This means that the remaining two application conditions $AC_{\text{mod-w,i,2}}$ and $AC_{\text{mod-w,i,6}}$ both contain two conclusion patterns. The reason for this difference to $(R_i, C_{\text{kTC,i}})$ is that the modification of w_{12} may or may not lead to the destruction of a match of $c_{\text{mod-w,i,6,2}}$. In the case of $(R_i, C_{\text{kTC,i}})$, the inactivation of v_{12} does not affect weak consistency. To sum up, the two conclusion patterns of the application condition $AC_{\text{mod-w,i,2}}$ reflect that the weight of a link v_{12} may only change to a new value w if for each outgoing inactive link v_{13} of v_1 either an alternative triangle (consisting of v_{13}, v_{14}, v_{43}) exists for which the kTC predicate φ_{kTC} holds ($c_{\text{mod-w,i,2,1}}$), or the triangle consisting of v_{13}, v_{12}, v_{23} fulfills φ_{kTC} after the link weight of v_{12} has changed to w ($c_{\text{mod-w,i,2,2}}$). The application condition $AC_{\text{mod-w,i,6}}$ represents the analogous situation for all incoming inactive links of v_2 .

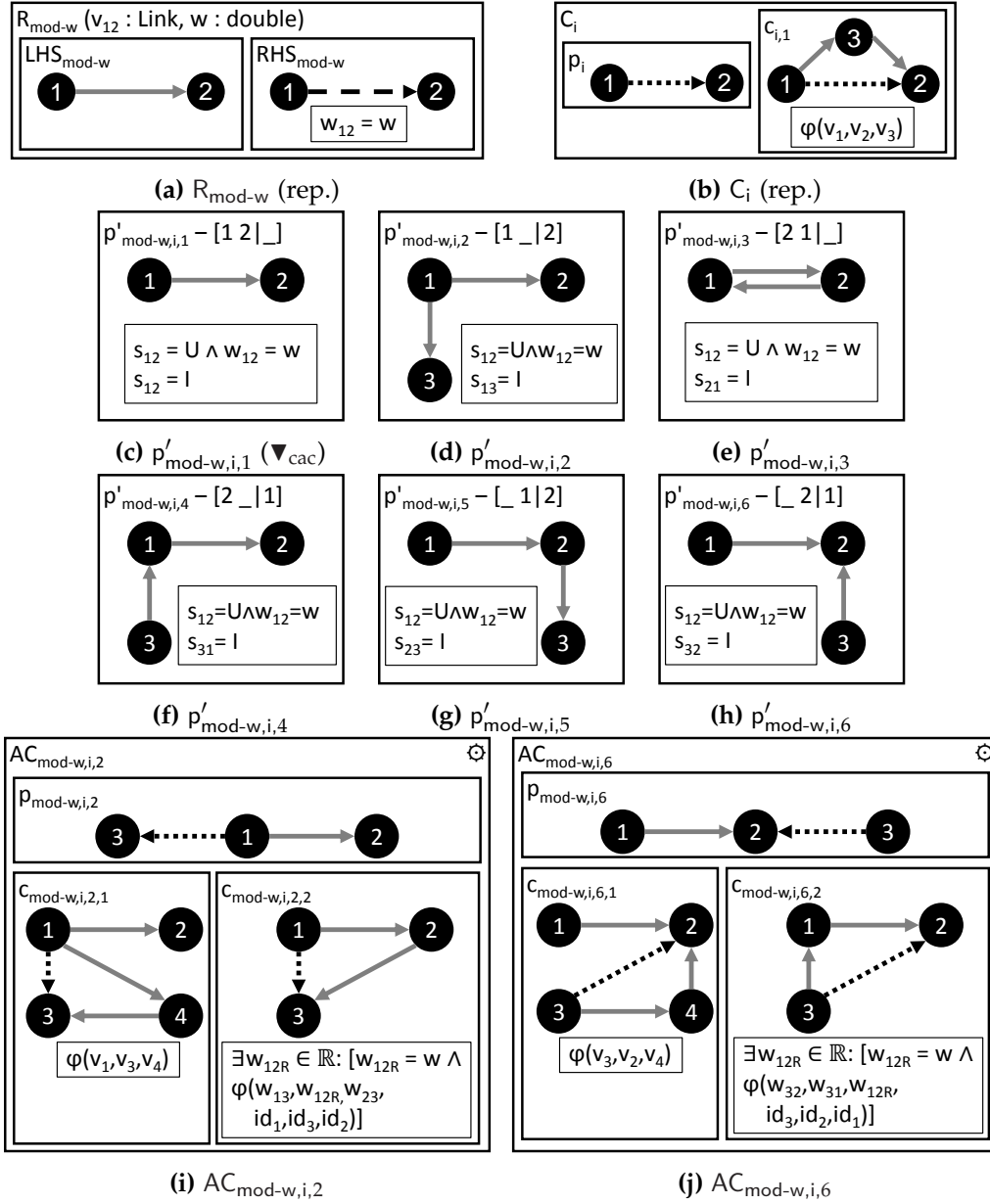


Figure 4.17: Selected synthesis results for $(R_{\text{mod-w}}, C_{kTC,i})$

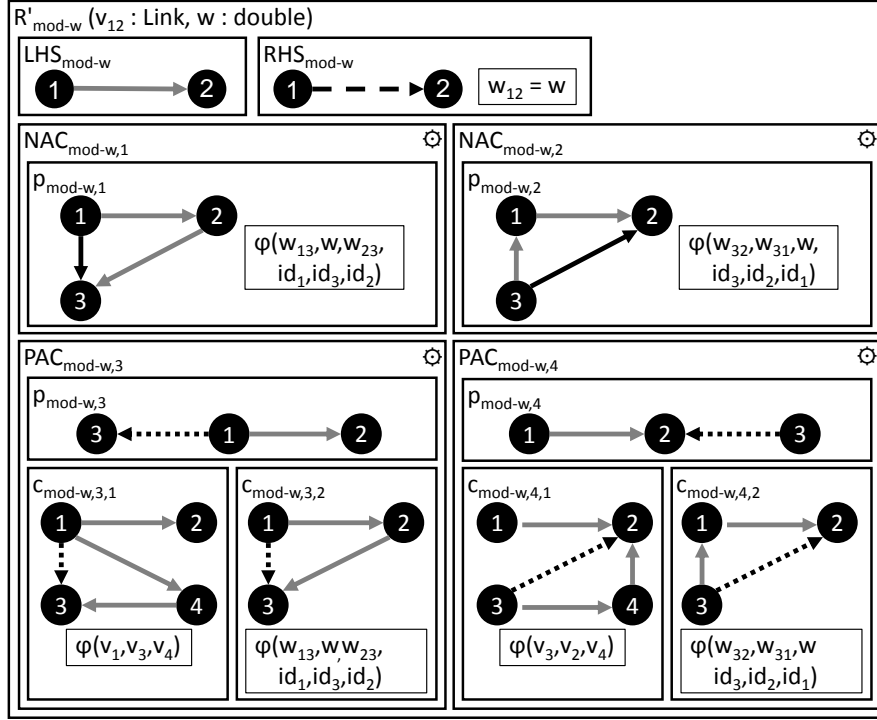


Figure 4.18: Refined link-weight modification rule $R'_{\text{mod-w}}$ with synthesized application conditions $\text{NAC}_{\text{mod-w},1}$ and $\text{NAC}_{\text{mod-w},2}$ (originating from $p_{\text{mod-w},a,2}$ and $p_{\text{mod-w},a,8}$ and ensuring the preservation of $C_{k\text{TC},a}$) as well as $\text{PAC}_{\text{mod-w},3}$ and $\text{PAC}_{\text{mod-w},4}$ (originating from $\text{AC}_{\text{mod-w},i,2}$ and $\text{AC}_{\text{mod-w},i,6}$ and ensuring the preservation of $C_{k\text{TC},i}$)

Finally, we attach $\text{AC}_{\text{mod-w},i,2}$ and $\text{AC}_{\text{mod-w},i,6}$ to the link-weight modification rule $R_{\text{mod-w}}$ and perform the usual renaming of variables and the replacement of w_{12R} with the parameter w . Figure 4.18 shows the resulting refined link-weight modification rule.

We discuss the effect of the two distinct application condition conclusion patterns of $\text{PAC}_{\text{mod-w},3}$ and $\text{PAC}_{\text{mod-w},4}$ using the sample topology Figure 4.19 ($k\text{TC}$ with $k = 1.3$). Here, six context events are pending, and four of the pending context events can be applied. The link-weight modification event that relates to l_{12} can be applied because, after the link-weight modification from 1 to 5.5, the match $\{v_{12} \mapsto l_{14}\}$ of $p_{k\text{TC},i}$ can be extended to a match of the conclusion pattern $c_{k\text{TC},i,1}$ that corresponds to the alternative triangle (n_1, n_4, n_3) . Application condition $\text{PAC}_{\text{mod-w},3}$ and, in particular, the application condition conclusion pattern $c_{\text{mod-w},3,1}$ capture this situation. Similarly, the link-weight modification event that relates to l_{21} can be applied because, after the link-weight modification from 1 to 5.5, the match $\{v_{12} \mapsto l_{41}\}$ of $p_{k\text{TC},i}$ can be extended to a match of the conclusion pattern $c_{k\text{TC},i,1}$ that corresponds to the alternative triangle (n_4, n_1, n_3) . Application condition $\text{PAC}_{\text{mod-w},4}$ and, in particular, the application condition conclusion pattern $c_{\text{mod-w},4,1}$ capture this situation. The situation is different for the next pair of

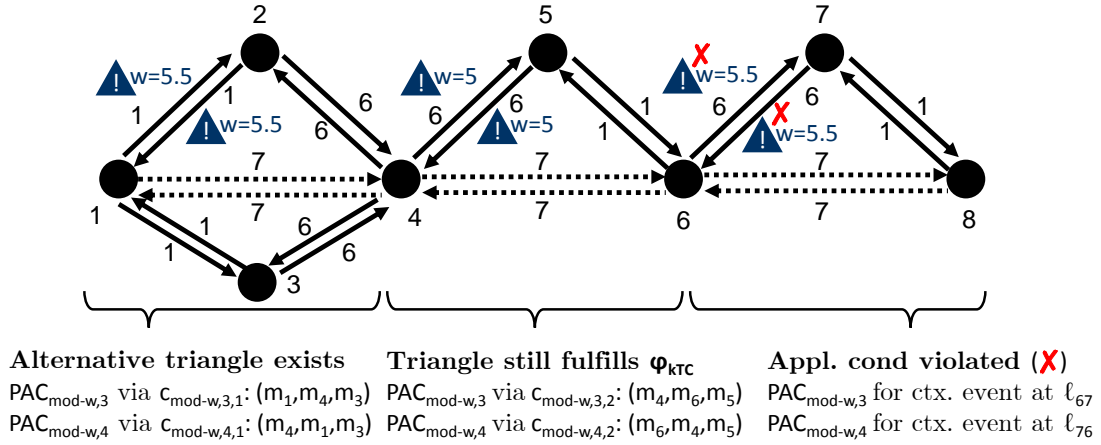


Figure 4.19: Effect of synthesized positive application conditions $PAC_{mod-w,3}$ and $PAC_{mod-w,4}$

pending context events, which determine that the weight of l_{45} and l_{54} are about to change from 1 to 5. In this case, no alternative triangle exists, but the existing extensions of the matches from the premise $p_{kTC,i}$ to the conclusion pattern $c_{kTC,i,1}$ are still valid after applying the link-weight modification events ($7 > 1.3 \cdot 5 = 6.5$). The situation for l_{45} is captured by $PAC_{mod-w,3}$ and $c_{mod-w,3,2}$, and the situation for l_{54} is captured by $PAC_{mod-w,4}$ and $c_{mod-w,4,2}$. Only the context events that relate to l_{67} and l_{76} cannot be applied because either $PAC_{mod-w,3}$ (for l_{67}) or $PAC_{mod-w,4}$ (for l_{76}) cannot be fulfilled. In both cases neither an alternative, φ_{kTC} -fulfilling triangle exists nor would the existing triangle fulfill φ_{kTC} after applying the context event.

4.3.3 Interpretation of synthesized application conditions

In this subsection, we investigate in how far the synthesized application conditions cause a different behavior of the TC mechanism specification compared to the template specification. Figure 4.20 shows the story diagram of the kTC algorithm, whose control flow is identical to the template TC algorithm specification and whose GT rule applications contain the refined rules R'_a (Figure 4.11) and R'_i (Figure 4.9g). We obtain the story diagrams of the context event handler specification from the template specification in Figure 3.24 in an analogous manner. For conciseness reasons, we do not show these story diagrams here. In Example 4.19, we now attempt to prove the correctness of the current state of the kTC specification.

Example 4.19 (Correctness of kTC specification after synthesis algorithm). We evaluate which correctness criteria the kTC specification fulfills after applying the constructive approach. Figure 4.20 shows the TC algorithm specification, which con-

tains the refined TC rules R'_a and R'_i . We obtain the refined context event handler diagrams from Figure 3.24 by using the refined context event rules R'_{+e} (Figure 4.12), R'_{-e} (Figure 4.14), and $R'_{\text{mod-w}}$ (Figure 4.18). For conciseness reasons, we do not show the refined context event handler diagrams here.

Criterion CC1: The kTC specification in Figure 4.20 terminates. *Proof idea:* The execution of the story diagram terminates if the topology contains no more unmarked links (see application of $R_{\text{find-u}}$). We now show that the number of unmarked links decreases by one with each iteration. This is the case if at least one of the two GT rules R'_a and R'_i is applicable to the unmarked link v_{12} . Thus, v_{12} is left unmarked if and only if neither R'_a nor R'_i is applicable. We now assume that neither GT rule is applicable and show that this implies a contradiction. The fact that R'_a is inapplicable allows us to conclude that v_{12} is part of a triangle together with links v_{13} and v_{32} for which φ_{kTC} is true. The fact that R'_i is inapplicable allows us to conclude that no such triangle exists because no match of $c_{i,1,1}$ exists. This is a contradiction. Therefore, for a given unmarked link v_{12} , at least one of the two rules R'_a and R'_i is applicable. For a given topology, the set of links is finite and so is the number of loop iterations.

Criterion CC2: The kTC specification in Figure 4.20 preserves weak consistency. *Proof idea:* Each GT rule application in the story diagram (i.e., $R_{\text{find-u}}$, R'_a , R'_i) preserves weak consistency; the find-unmarked-link rule $R_{\text{find-u}}$ preserves weak consistency because it does not modify the topology, and R'_a and R'_i preserve weak consistency by construction because, by applying the constructive approach to these GT rules, we added application conditions that prevent any application of R'_a and R'_i that would otherwise introduce a violation of $\mathcal{C}_W = \{C_{\text{kTC},i}, C_{\text{kTC},a}\}$. Therefore, the kTC diagram preserves weak consistency.

Criterion CC3: The kTC specification in Figure 4.20 enforces strong consistency. *Proof idea:* An execution of the story diagram shown in Figure 4.20 may only terminate if $R_{\text{find-u}}$ is inapplicable. This is the case if the topology contains no unmarked links and, therefore, fulfills the no-unmarked-links constraint C_u . Together with the previously shown criterion CC2 we conclude that the kTC specification enforces strong consistency.

Criterion CC4: The execution of each kTC-specific context event handler story diagram terminates. *Proof idea:* Each context event handler story diagram in Figure 3.24 contains a single GT rule application and no loops. Even though the applicability of the context event rules has decreased due to the additional application conditions, the termination behavior of the story diagrams has not changed because the outgoing activity edge of the respective story nodes has no guard. Therefore, the execution of each context event handler diagram always terminates.

Criterion CC5: Each kTC-specific context event handler specification preserves weak consistency. *Proof idea:* By applying the constructive approach to R_{+e} , R_{-e} ,

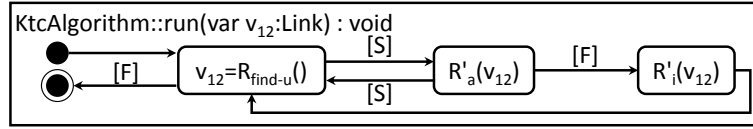


Figure 4.20: Story diagram of kTC algorithm after applying the constructive approach

and $R_{\text{mod-w}}$, we added application conditions to these GT rules that prohibit the application of each GT rule at a match that would otherwise lead to a violation of weak consistency. Therefore, each kTC-specific context event handler diagram preserves weak consistency.

Criterion CC6: Each kTC-specific context event handler specification processes the corresponding context event. *Observation:* We reuse the sample topology from the beginning of this chapter to illustrate why CC6 does not hold. Figure 4.21 shows the sample topology from the beginning of the chapter (Figure 4.3a) together with the refined link-weight modification rule $R'_{\text{mod-w}}$. For the moment, we only consider the pending link-weight modification events. The four synthesized application conditions of R'_{+e} are placed close to the part of the topology that contains a violation of the respective application condition. The violation of each application condition is marked as a match of the premise of the application condition that cannot be extended to a match of one of conclusion patterns: m_1 for $\text{NAC}_{\text{mod-w},1}$, m_2 for $\text{NAC}_{\text{mod-w},2}$, m_3 for $\text{PAC}_{\text{mod-w},3}$, m_4 for $\text{PAC}_{\text{mod-w},4}$. Similarly, we can identify matches of the premises of the synthesized application conditions of R'_{+e} and R'_{-e} in Figure 4.3a that cannot be extended to a match of a conclusion pattern.

Verdict: The current kTC specification is not correct because CC6 is violated.

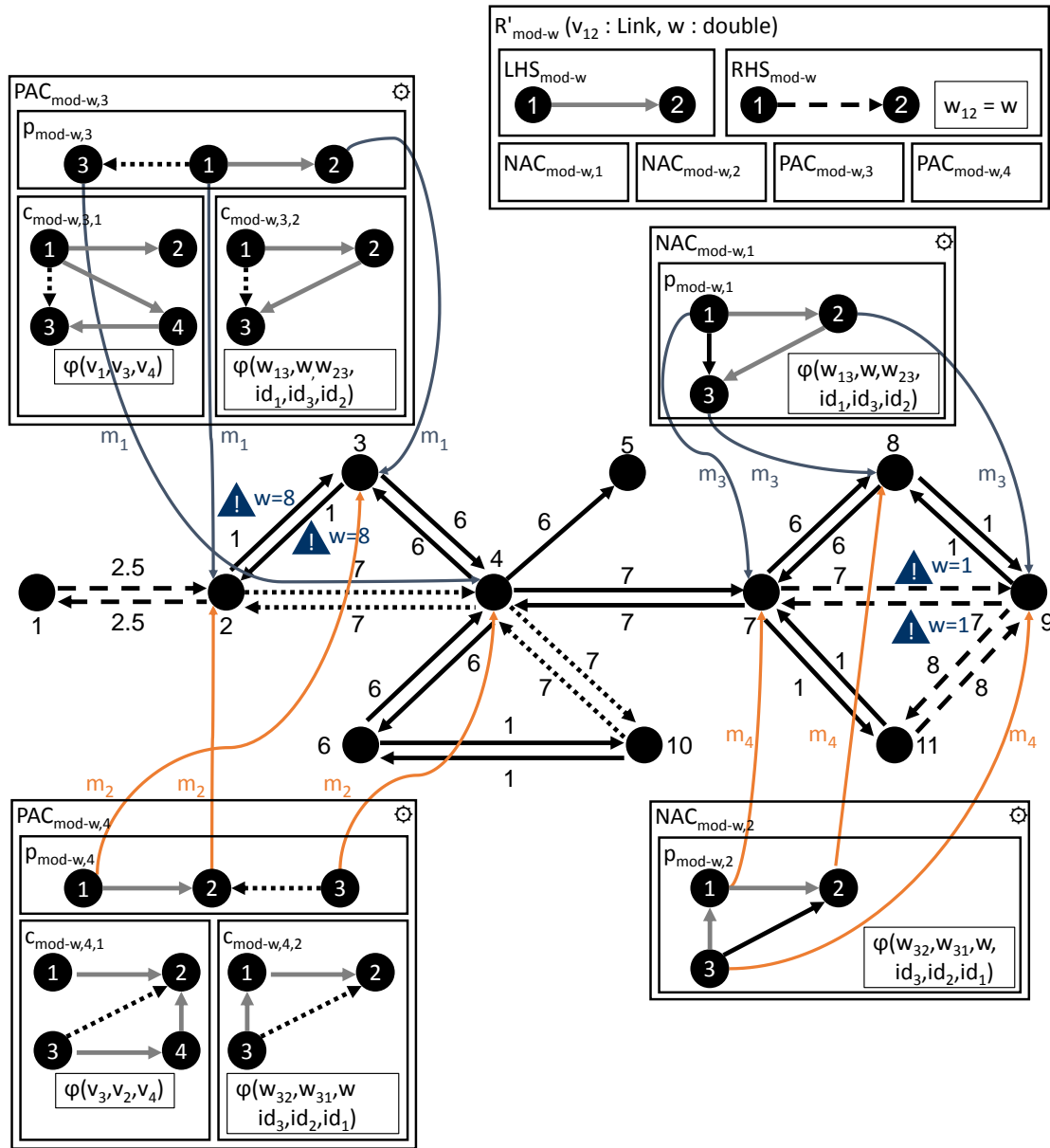


Figure 4.21: Violations of application conditions of $R'_{\text{mod-w}}$ in Figure 4.3a

4.4 SYNTHESIS OF ANTICIPATION LOOPS

The preceding attempt to prove the correctness of the kTC specification failed because the application of the constructive approach restricted the applicability of the context event handler rules (see Example 4.19). In this section, we show how to re-establish the applicability of the context event rules in a systematic, consistency-preserving way. We introduce the anticipation loop synthesis step ③, an additional refinement step that ensures that each context event handler processes a given context event without violating weak consistency. The anticipation loop synthesis algorithm is one major contribution of this thesis.

This step expects as input a TC mechanism specification that has been refined using the constructive approach and that fulfills the correctness criteria CC1 to CC5. This implies that the TC algorithm specification is already correct because the only related correctness criteria CC1 to CC3 are fulfilled. Therefore, we focus on the context event handler specification in the following.

In contrast to the constructive approach, the anticipation loop synthesis step modifies the control flow specification and synthesizes additional rules, the context event handler rules.

4.4.1 Violation of application conditions









The underlying idea of the envisioned anticipation loop synthesis step is to transform each synthesized application condition of the context event rules into control flow fragment that unmarks exactly those links that are part of a violation of the considered application condition. We use the term “anticipation” here because each anticipation loop foresees the violation of a particular application condition of the context event rule that follows. We begin with a definition that characterizes the violation of an application conditions.

Definition 4.20 (Violation of application condition). *A violation of an application condition $AC_{X,Y}$ of a GT rule R_X is an extension of the match of LHS $_X$ to the premise of $AC_{X,Y}$ that cannot be extended to a match of any conclusion pattern $c_{X,Y,Z}$.* \square

The following example illustrates this definition.

Example 4.21 (Violation of application condition and resolution). To illustrate the concept of violated application conditions, we reuse the initial example of this chapter, shown in Figure 4.3. In Figure 4.22a, each link that is labeled with a red cross mark (X) indicates a violation of a synthesized application condition of the refined context event rules R'_{+e} (Figure 4.12), R'_{-e} (Figure 4.14), or $R'_{\text{mod-w}}$ (Figure 4.18). In comparison to Figure 4.3, the violations of the synthesized application conditions of

Table 4.5: Example: Violations of application condition in Figure 4.3a

Context event	 $+e(5,7,1)$	 $+e(7,10,1)$	 $-e(\ell_{64})$	 $-e(\ell_{46})$
AC	$NAC_{+e,2}$	$NAC_{+e,3}$	$PAC_{-e,1}$	$PAC_{-e,2}$
Violation	$\{v_{13} \mapsto \ell_{47}\}$	$\{v_{32} \mapsto \ell_{74}\}$	$\{v_{13} \mapsto \ell_{410}\}$	$\{v_{32} \mapsto \ell_{104}\}$
Context event	 $\text{mod-w}(\ell_{79},1)$	 $\text{mod-w}(\ell_{97},1)$	 $\text{mod-w}(\ell_{23},8)$	 $\text{mod-w}(\ell_{32},8)$
AC	$NAC_{\text{mod-w},1}$	$NAC_{\text{mod-w},2}$	$PAC_{\text{mod-w},3}$	$PAC_{\text{mod-w},4}$
Violation	$\{v_{13} \mapsto \ell_{78}\}$	$\{v_{32} \mapsto \ell_{87}\}$	$\{v_{13} \mapsto \ell_{24}\}$	$\{v_{32} \mapsto \ell_{42}\}$

the TC rules (i.e., $PAC_{i,1}$ at ℓ_{12} and $NAC_{a,1}$ at ℓ_{911}) are resolved by activating and inactivating the links ℓ_{12} and ℓ_{21} as well as ℓ_{911} and ℓ_{119} , respectively. This resolution is ensured by the TC algorithm specification (CC3).

In contrast, all red cross marks shown in Figure 4.22a are problematic because they prevent the application of the pending context events (CC6). During the (partially) failed proof of correctness for the current version of the kTC specification, we discussed why the refined link-weight modification rule $R'_{\text{mod-w}}$ is inapplicable to each pending link-weight modification event in Figure 4.3a. The exact violations of each application condition are shown in Example 4.19.

Table 4.5 summarizes the violations of the application conditions of the refined link-weight modification rule $R'_{\text{mod-w}}$ as well as the violations of the refined link addition rule R'_{+e} and the refined link removal rule R'_{-e} .

which links constitute a violation of which application condition during the processing a given context event. We only provide the partial match that involves the link variable of the application condition premise that is associated with a marking constraint. For instance, the violation of $NAC_{+e,2}$ contains the mapping from the v_{12} with marking constraint $s(v_{12}) = A$ to ℓ_{47} .

Notably, all violation reasons shown in Figure 4.22a can be resolved by setting the respective link to unmarked. Unmarking a link always preserves weak consistency and, at the same time, allows the context event handlers to process the pending context events because the premise of each synthesized application condition contains a constraint of the form $s(v_{YZ}) = A$ or $s(v_{YZ}) = I$.

Figure 4.22b shows the topology after unmarking all links that cause the violation of a synthesized application condition of a context event rule. In this state of the topology, all pending context events can be processed. The result is shown in Figure 4.22c. After handling all context events, the next step is to execute the TC algorithm specification to ensure that all links are marked (not shown here).

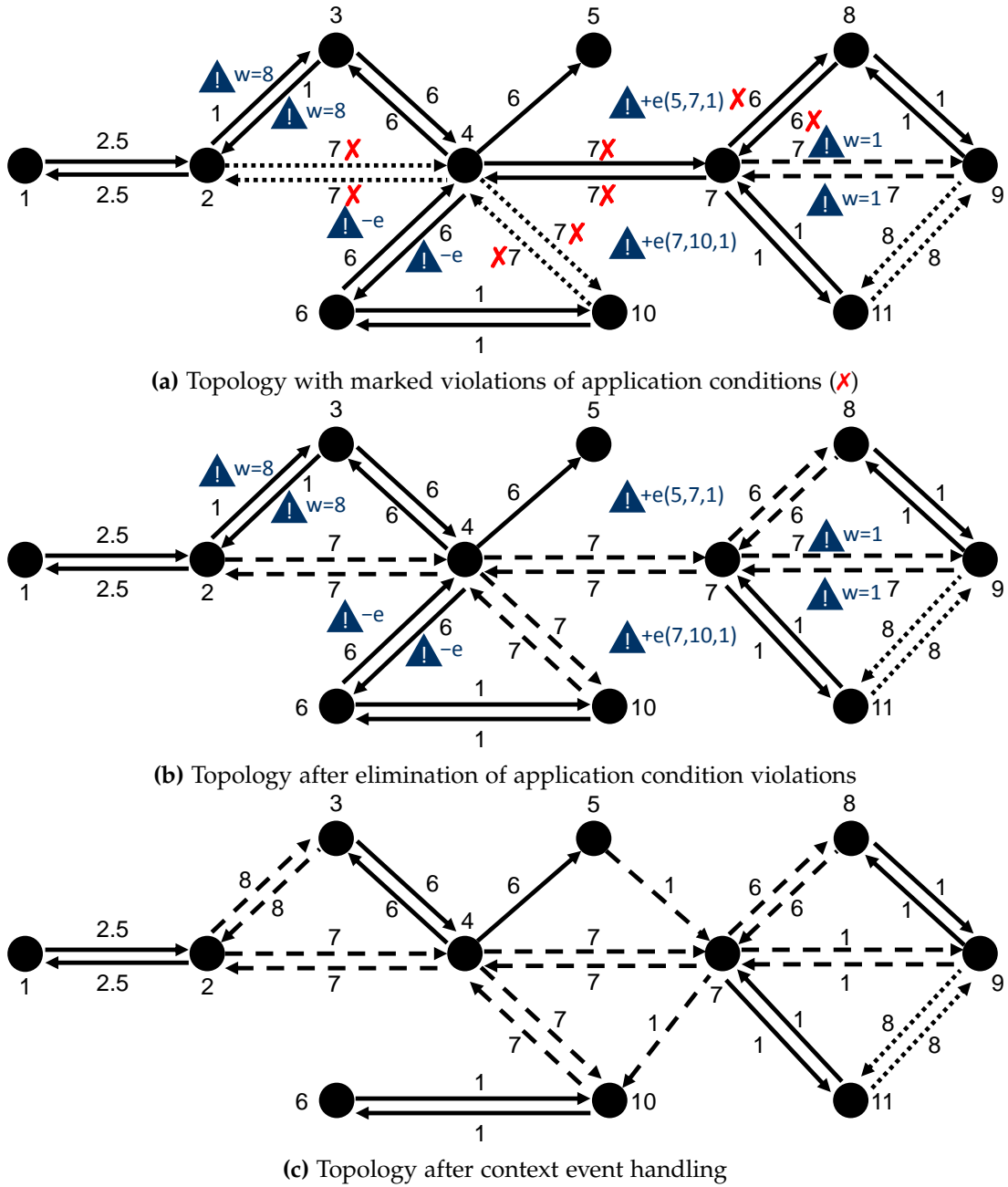


Figure 4.22: Example: Violation of application conditions and resolution via anticipation with kTC parameter k set to 1.3 (continued from Figure 4.3)

4.4.2 Anticipation loop synthesis algorithm

PRELIMINARIES We now generalize the insights gathered from Example 4.21. The following definition summarizes the necessary preconditions that allow us to resolve a violation of an application conditions by unmarking links.

Definition 4.22 (Anticipability preconditions). The strategy of unmarking links to ensure the applicability of context event rules relies on two preconditions that each synthesized application condition in the context event handler specification must fulfill. The *first anticipability precondition* is that each application condition premise contains at least one *marking constraint*, which is an attribute constraint that restricts the state of at least one link variable (i.e., $s(v_{YZ}) = A$ or $s(v_{YZ}) = I$ for some link variable v_{XY}). The *second anticipability precondition* is that unmarking a link (i.e., applying the unmarking rule R_u) preserves weak consistency. \square

Note that the first precondition is fulfilled if the premises of each weak consistency constraint contains a marking constraint. The reason is that the context event rules generally do not contain attribute constraints related to the state of the involved links in the LHS pattern. This implies that the reverse application of the context event rule to the postcondition patterns during the anticipation step ② does not introduce fresh marking constraints. Therefore, any marking constraint in the synthesized application condition originates from the graph constraint that caused the addition of the application condition during the synthesis algorithm.

Example 4.23 (Anticipability preconditions). For our running example, the first anticipability precondition is fulfilled because all eight synthesized application conditions contain marking constraints: $s_{13} = A$ in $NAC_{+e,2}$ and $NAC_{mod-w,1}$, $s_{32} = A$ in $NAC_{+e,3}$ and $NAC_{mod-w,2}$, $s_{13} = I$ in $PAC_{-e,1}$ and $PAC_{mod-w,3}$, $s_{32} = I$ in $PAC_{-e,2}$ and $PAC_{mod-w,4}$. All marking constraints result from the attribute constraints of the premises of $C_{kTC,a}$ and $C_{kTC,i}$, respectively.

The second anticipability precondition is also fulfilled because applying the unmarking rule R_u preserves \mathcal{C}_W .

Based on the anticipability preconditions and to enforce that CC6 is fulfilled, the procedure `SYNTHESIZEANTICIPATIONLOOPS` in Algorithm 4.3 refines each context event handler in the TC mechanism specification by synthesizing anticipation loops.

Definition 4.24 (Anticipation loop). An *anticipation loop w.r.t. a pending context event CE and a synthesized application condition $AC_{X,Y,J}$* is a loop in a story diagram that unmarks all links in the topology that are part of a violation of $AC_{X,Y,J}$ during the handling of CE. \square

The algorithm operates in-place by modifying the given context event handler specification handle_X . In the following, the context event rule that is invoked by handle_X is denoted by R'_X and the story node that contains the application of R'_X is denoted by SN_e . After the algorithm has terminated, the refined context event handler diagram fulfills CC6. The pending context event CE is encoded as the values of the parameter list (X_1, X_2, \dots) of handle_X .

Example 4.25 (Notation of anticipation loop synthesis algorithm). This example illustrates the usage of the variables handle_X , SN_e , and (X_1, X_2, \dots) . The pending addition of a link with weight 1 from mote n_5 to mote n_7 (i.e., $\blacktriangle_{+e(n_5, n_7, 1)}$) in Figure 4.22a is represented by an invocation of $\text{handle}_X = \text{KtcLinkAdditionHandler::KtcLinkAdditionHandler}$ with parameter values $(X_1, X_2, X_3) = (n_5, n_7, 1)$.

The pending removal of l_{46} (i.e., $\blacktriangle_{-e(l_{46})}$) in Figure 4.22a is represented by an invocation of $\text{handle}_X = \text{KtcLinkRemovalHandler::KtcLinkRemovalHandler}$ with parameter values $(X_1) = (l_{46})$.

The pending modification of the weight of l_{23} to 8 (i.e., $\blacktriangle_{\text{mod-w}(l_{23}, 8)}$) in Figure 4.22a is represented by the invocation of the context event handler operation $\text{handle}_X = \text{KtcLinkWeightModificationHandler::handleLinkWeightModification}$ (see Figure 3.23e) with the parameter values $(X_1, X_2) = (l_{23}, 8)$.

For each of discussed pending context events, the story node SN_e is equal to the single story node of the context event handler diagram.

Without loss of generality, we make the following technical assumptions: First, the story node containing the invocation of R'_X has one incoming activity edge. Second, the parameter list (X_1, X_2, \dots) of handle_X is equal to the parameter list of R'_X without modifications. Third, the parameter list unambiguously determines the context event, that is, the extension of the partial match that is represented by (X_1, X_2, \dots) to a match of LHS_X is always unique. If the first assumption were violated, all incoming activity edges of the story node containing the application of R'_X could be redirected to a freshly introduced story node in front of the context event story node. If the second assumption were violated, we could introduce an intermediate operation that reorganizes the parameter list and refine this intermediate operation as described in the following. If the third assumption were violated, the parameter list of handle_X and R'_X would not specify the context event precisely, which is undesirable.

EXPLANATION OF ALGORITHM We now describe the steps of the anticipation loops synthesis algorithm in detail. We assume that the anticipability preconditions (Definition 4.22) hold.

The synthesis algorithm for anticipation loops processes each synthesized application condition $\text{AC}_{X,J}$ separately by invoking `SYNTHESIZEANTICIPATIONLOOP` (lines 3–5). For a given synthesized application condition $\text{AC}_{X,J}$, the procedure `SYNTHESIZEANTICIPATION-`

Algorithm 4.3 Anticipation loop synthesis algorithm

Require: Anticipability preconditions (Definition 4.22).

Ensure: Correctness criterion CC6 fulfilled for handle_x (see Definition 4.7)

procedure SYNTHESIZEANTICIPATIONLOOPS(handle_x : context event handler diagram)

R'_x = Refined context event rule in handle_x

for all synthesized application condition $AC_{X,J}$ of R'_x **do** ▷ See \odot symbol

 SYNTHESIZEANTICIPATIONLOOP($\text{handle}_x, R'_x, AC_{X,J}$)

end for

end procedure

procedure SYNTHESIZEANTICIPATIONLOOP(handle_x : context event handler diagram,
 R'_x : refined context event rule, $AC_{X,J}$: synthesized application condition of R'_x)

SN_e = Story node in handle_x containing the invocation of R'_x

(X_1, X_2, \dots) = Parameter list of invocation of R'_x and handle_x

 ▷ Create anticipation rule $R_{\text{anticipate},X,J}$

 Create new GT rule $R_{\text{anticipate},X,J}$ with parameters (X_1, X_2, \dots)

$LHS_{\text{anticipate},X,J}$ = extension of LHS_x along $LHS_x-p_{X,J}$ mapping in R'_x

v_{YZ} = link variable in $LHS_{\text{anticipate},X,J}$ with $s(v_{YZ}) = A$ or $s(v_{YZ}) = I$

$RHS_{\text{anticipate},X,J}$ = pattern with graph of $LHS_{\text{anticipate},X,J}$ and constraint $s(v_{YZ}) = U$

for all conclusion pattern $c_{X,J,K}$ of $AC_{X,J}$ **do**

$NAC_{\text{anticipate},X,Y}$: NAC = Fresh negative application condition

$P_{\text{anticipate},X,J,K}$ = extension of $LHS_{\text{anticipate},X,J}$ along $p_{X,J}-c_{X,J,K}$ mapping in $AC_{X,J}$

end for

 ▷ Create application of $R_{\text{anticipate},X,J}$ into handle_x

$SN_{\text{anticipate},X,J}$ = Fresh story node containing $R_{\text{anticipate},X,J}(X_1, X_2, \dots)$

 Redirect incoming activity edge of SN_e to have target $SN_{\text{anticipate},X,J}$

 Insert [S]-edge with source and target $SN_{\text{anticipate},X,J}$

 Insert [F]-edge with source $SN_{\text{anticipate},X,J}$ and target SN_e

end procedure

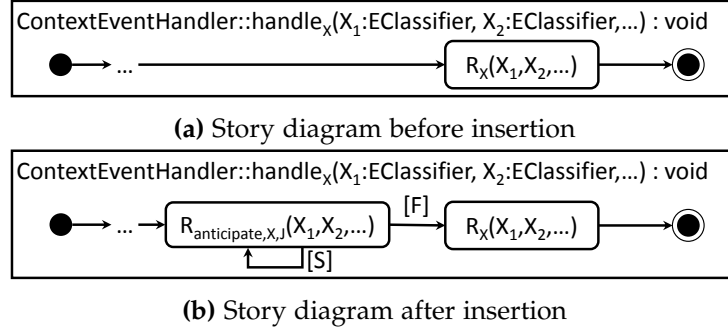


Figure 4.23: Insertion of anticipation loop in story diagram (schematic)

LOOP creates and inserts an anticipation loop that removes all violations of $AC_{X,J}$ for the given parameter list (X_1, X_2, \dots) .

At the beginning of the procedure SYNTHESIZEANTICIPATIONLOOP, a fresh anticipation rule $R_{anticipate,X,J}$ is created. This rule has the same parameter list as R'_X (line 13). The LHS pattern $LHS_{anticipate,X,J}$ matches if the topology contains a violation of $AC_{X,J}$ for the partial match resulting from the given parameters (X_1, X_2, \dots) (line 14). Thanks to the anticipability precondition, at least one link variable v_{YZ} has an associated marking constraint in $LHS_{anticipate,X,J}$ (line 15). If multiple candidates for v_{YZ} exist, any of them may be chosen because it suffices to unmark one link variable during the rule application to destroy the violation of $AC_{X,J}$. The RHS pattern $RHS_{anticipate,X,J}$ is derived from $LHS_{anticipate,X,J}$ by copying the pattern graph of $LHS_{anticipate,X,J}$ and inserting a single attribute constraint that ensures that v_{YZ} is unmarked after applying $R_{anticipate,X,J}$ (line 16).

Afterwards, each conclusion pattern $c_{X,J,K}$ of $AC_{X,J}$ is translated into a negative application condition of $R_{anticipate,X,J}$ (lines 17–20). The generated application is negative because each conclusion pattern $c_{X,J,K}$ represents a situation in which applying R'_X preserves $AC_{X,J}$ and, thus, unmarking v_{YZ} is unnecessary.

Finally, an application of the generated rule $R_{anticipate,X,J}$ is inserted in front of SN_e (lines 23–26, visualized by Figure 4.23). The $[S]$ -edge at $SN_{anticipate,X,J}$ forms a loop and, thereby, ensures that $R_{anticipate,X,J}$ is applied as long as possible (line 25). The execution may only continue to SN_e if all violations of $AC_{X,J}$ have been resolved.

Before we prove the correctness of Algorithm 4.3 in Section 4.4.3, we illustrate the algorithm using our running example kTC.

Example 4.26 (Anticipation loop synthesis step ③ for kTC). Figure 4.25a shows the kTC-specific link addition handler after the synthesis of two anticipation loops. The anticipation loops around the applications of $R_{anticipate,+e,2}$ (Figure 4.25b) and $R_{anticipate,+e,3}$ (Figure 4.25c) originate from the synthesized application conditions $NAC_{+e,2}$ and $NAC_{+e,3}$, respectively.

Figure 4.25d shows the kTC-specific link removal handler after the synthesis of two anticipation loops. The anticipation loops that apply $R_{\text{anticipate},-e,1}$ (Figure 4.25e) and $R_{\text{anticipate},-e,2}$ (Figure 4.25f) originate from the synthesized application conditions $NAC_{+e,2}$ and $NAC_{+e,3}$, respectively.

Figure 4.26a shows the kTC-specific link-weight modification handler after the synthesis of four anticipation loops. The anticipation loops that apply the rules $R_{\text{anticipate,mod-w},1}$ (Figure 4.26b), $R_{\text{anticipate,mod-w},2}$ (Figure 4.26c), $R_{\text{anticipate,mod-w},3}$ (Figure 4.26d), and $R_{\text{anticipate,mod-w},4}$ (Figure 4.26e) originate from the synthesized application conditions $NAC_{\text{mod-w},1}$, $NAC_{\text{mod-w},2}$, $PAC_{\text{mod-w},3}$, $PAC_{\text{mod-w},4}$, respectively.

Example 4.27 (Effect of application conditions of anticipation loops). This example illustrates how the negative application conditions of the anticipation rules $R_{\text{anticipate,mod-w},3}$ and $R_{\text{anticipate,mod-w},4}$ ensure that no links are being unmarked unnecessarily. Let's consider the example shown in Figure 4.24a (continued from Figure 4.19, kTC parameter $k = 1.3$). In this example, the anticipation loops only take effect when the context events at ℓ_{67} and ℓ_{76} are processed by the link-weight modification handler. For the remaining four pending link-weight modification events, the negative application conditions prevent an application of $R_{\text{anticipate,mod-w},3}$ and $R_{\text{anticipate,mod-w},4}$ because the topology is weakly consistent even after the link-weight modification events have been processed.

The negative application conditions $NAC_{\text{anticipate,mod-w},3,1}$ and $NAC_{\text{anticipate,mod-w},4,1}$ prevent that ℓ_{14} and ℓ_{41} are being unmarked unnecessarily due to the existence of an alternative φ_{kTC} -fulfilling triangle. Analogously, the negative application conditions $NAC_{\text{anticipate,mod-w},3,2}$ and $NAC_{\text{anticipate,mod-w},4,2}$ prevent that ℓ_{46} and ℓ_{64} are being unmarked unnecessarily because the affected triangle still fulfills φ_{kTC} after the link-weight modification event has been processed. The topology that results from handling all context events is shown in Figure 4.24b. In total, eight links need to be unmarked.

After applying the anticipation loop synthesis algorithm to our running example, we are now ready to conduct the proof of correctness for the kTC specification again.

Example 4.28 (Correctness of kTC specification after anticipation loop synthesis). After the anticipation loop synthesis step ③, the kTC specification consists of the (unchanged) TC algorithm specification shown in Figure 4.20 and the context event handler specification shown in Figures 4.25a, 4.25d and 4.26a. The criteria CC1, CC2, and CC3 need not be revisited because we did not modify the TC algorithm specification during the anticipation loop synthesis step. In contrast, the modifications of the context event handler diagrams may—and in case of CC6, should—affect the remaining correctness criteria.

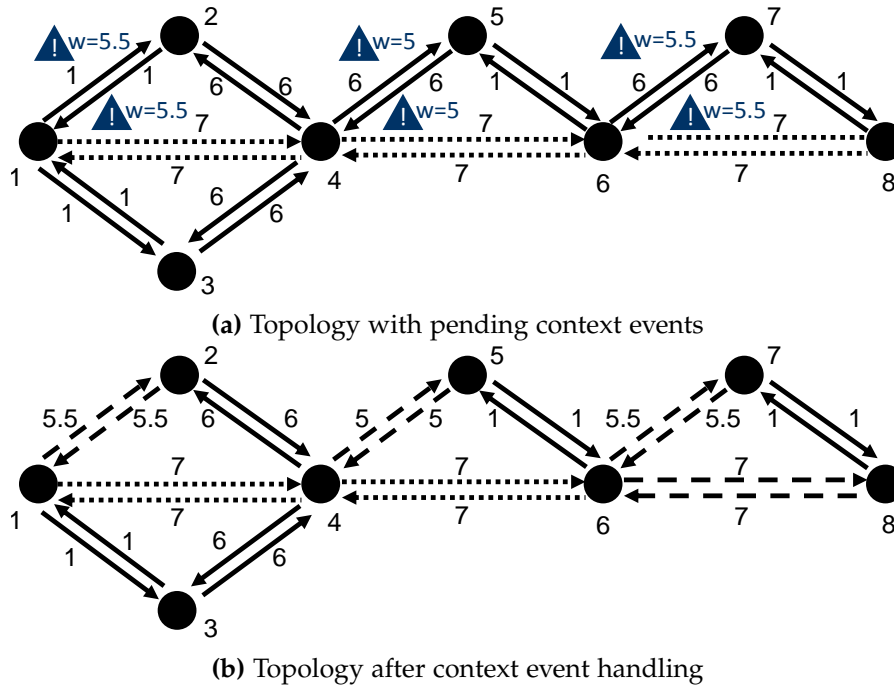


Figure 4.24: Example: Effect of negative application conditions of anticipation rules (kTC with $k = 1.3$)

Criterion CC4: The execution of each kTC-specific context event handler story diagram in Figures 4.25a, 4.25d and 4.26a terminates. *Proof idea:* The only reason why the context event handlers may not terminate are the freshly introduced anticipation loops. This may only happen if at least one anticipation rule $R_{anticipate,X,J}$ is applicable infinitely often. However, this can never occur because each successful application of an anticipation rule $R_{anticipate,X,J}$ unmarks a link that was marked previously. The topology contains only a finite number of marked links and, therefore, each context event handler terminates.

Criterion CC5: Each kTC-specific context event handler specification preserves weak consistency. *Proof idea:* The only possible violation of weak consistency could originate from the introduced anticipation rules $R_{anticipate,X,J}$. Still, the only modification performed by an anticipation rule is that a marked link is being unmarked. Unmarking a link never violates weak consistency (as discussed in Example 4.23). Therefore, each context event handler preserves weak consistency.

Criterion CC6: Each kTC-specific context event handler specification processes the corresponding context event. *Proof idea:* Upon termination of each anticipation loop, no violations of the corresponding synthesized application conditions of the context

event rules exist. A new violation of an application condition $AC_{X,J}$ may only be created if a link is being activated or inactivated. Still, no links are being marked during the execution of a context event handler. Therefore, no violation of any synthesized application condition of the context event rule exists when the execution of the story diagram arrives at SN_e .

Verdict: The kTC specification consisting of the TC algorithm specification in Figure 4.20 and the context event handlers in Figures 4.25a, 4.25d and 4.26a is correct.

4.4.3 Proof of correctness for anticipation loop synthesis algorithm

In the following, we prove that the anticipation loop synthesis algorithm (Algorithm 4.3) produces context event handlers that are correct according to the criteria CC4, CC5, and CC6 (Definition 4.7). We split the proof into two steps. First, we define correctness on the level of anticipation loops and prove that the synthesized anticipation loops are correct. Second, we use this result to prove that the refined context event handler specification is correct.

4.4.3.1 Correctness of synthesized anticipation loops

We begin with the characterization of correctness for anticipation loops in the following definition.

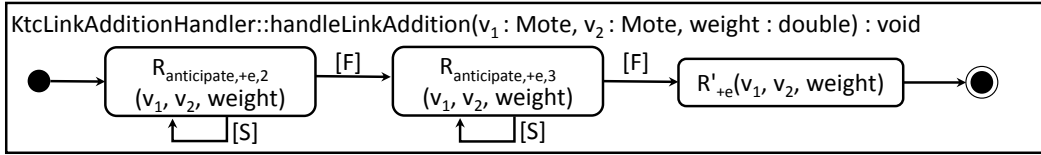
Definition 4.29 (Correctness of anticipation loop). Let $AC_{X,J}$ be a synthesized application condition of a GT rule R_X (from Algorithm 4.3), and $R_{\text{anticipate},X,J}$ the anticipation rule synthesized from $AC_{X,J}$ using Algorithm 4.3. An *anticipation loop w.r.t. $AC_{X,J}$* is correct if

- (i) $R_{\text{anticipate},X,J}$ is applicable if and only if a violation of $AC_{X,J}$ w.r.t. the context event that is currently processed exists (*least change*),
- (ii) the application of $R_{\text{anticipate},X,J}$ reduces the number of violations of $AC_{X,J}$ in the model (*violation resolution*), and
- (iii) the application of $R_{\text{anticipate},X,J}$ does not create additional violations of any application condition of R_X , including $AC_{X,J}$ (*consistency preservation*).

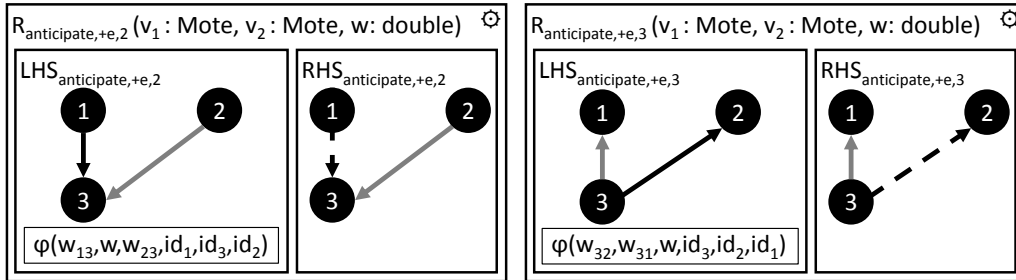
□

The intention behind Definition 4.29 is that an anticipation loop shall only resolve violations of application conditions that relate to the current context event (least-change and violation-resolution properties) and that anticipation loops can be arranged in any order without threatening weak consistency.

Our goal is now to prove the following correctness theorem for synthesized anticipation loops.

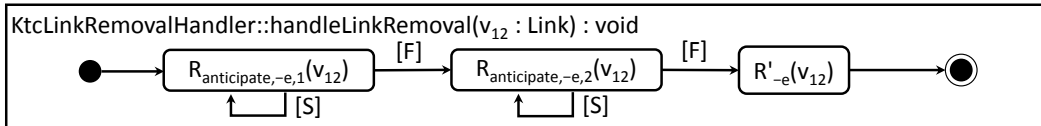


(a) kTC-specific link addition handler

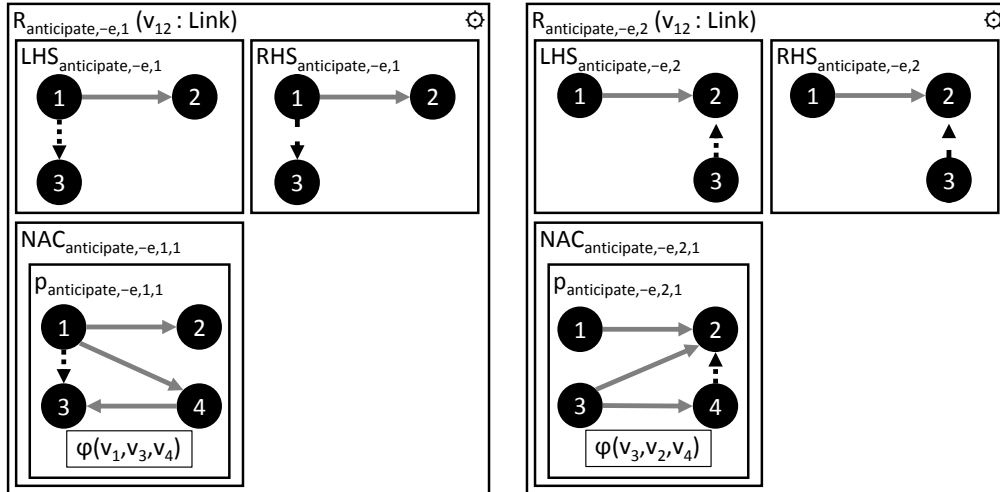


(b) $R_{anticipate,+e,2}$

(c) $R_{anticipate,+e,3}$



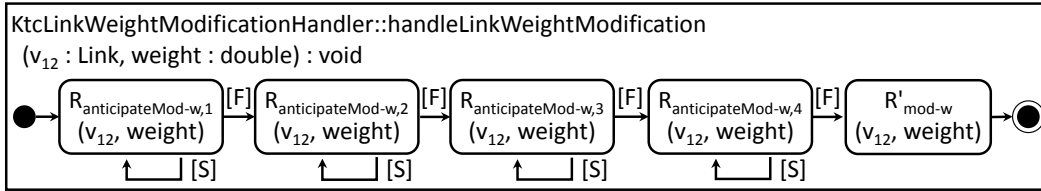
(d) kTC-specific link removal handler



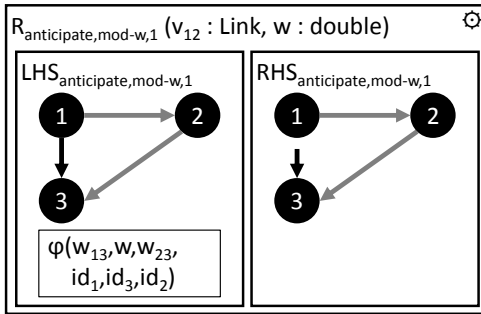
(e) $R_{anticipate,-e,1}$

(f) $R_{anticipate,-e,2}$

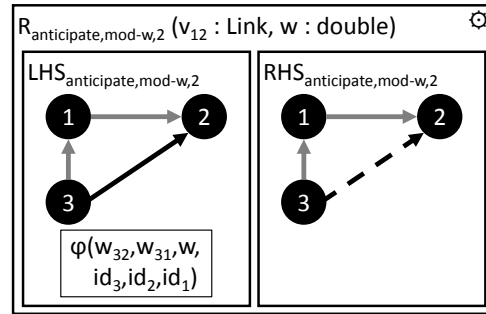
Figure 4.25: kTC-specific link addition and removal handlers with synthesized anticipation loops



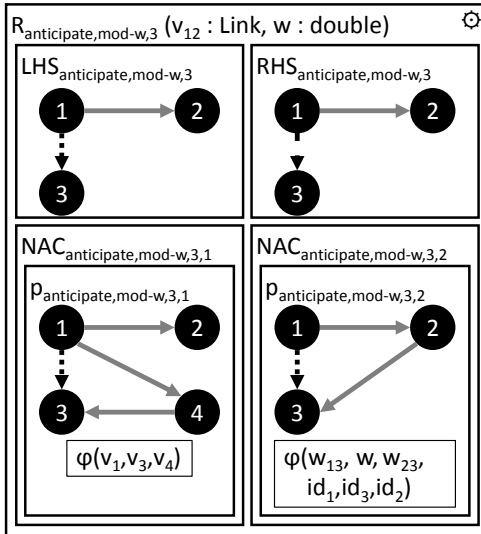
(a) kTC-specific link removal handler



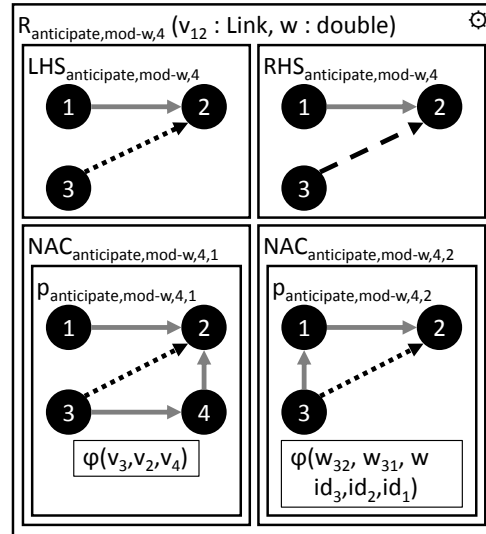
(b) $R_{\text{anticipate,mod-w},1}$



(c) $R_{\text{anticipate,mod-w},2}$



(d) $R_{\text{anticipate,mod-w},3}$



(e) $R_{\text{anticipate,mod-w},4}$

Figure 4.26: kTC-specific link-weight modification handler with refined link-weight modification rule $R'_{\text{mod-w}}$ and synthesized anticipation rules $R_{\text{anticipate,mod-w},1}$, $R_{\text{anticipate,mod-w},2}$, $R_{\text{anticipate,mod-w},3}$, and $R_{\text{anticipate,mod-w},4}$

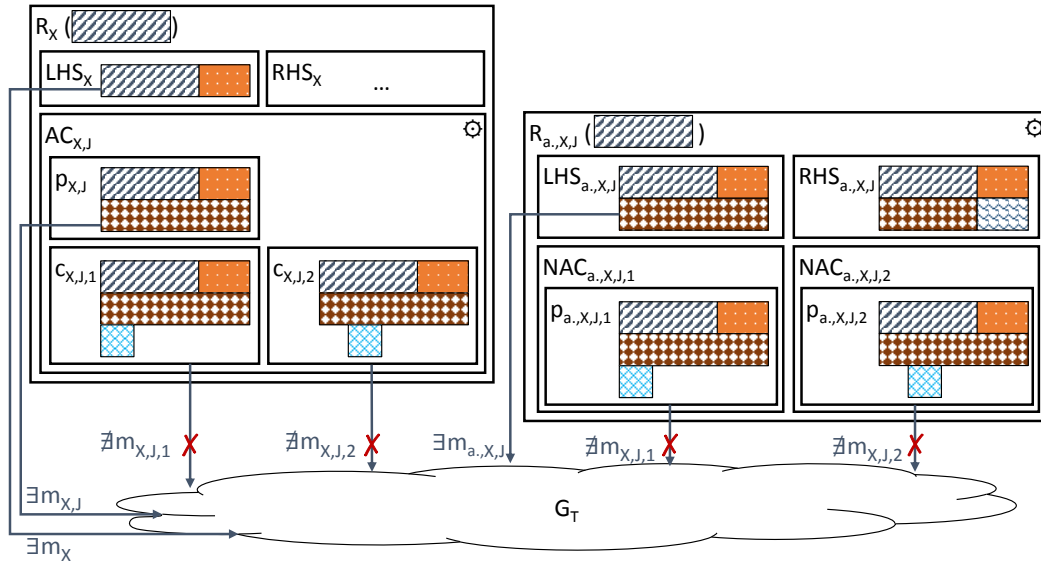


Figure 4.27: Sketch for proof of correctness of anticipation loop synthesis algorithm

Theorem 4.30 (Correctness of synthesized anticipation loops). An anticipation loop that originates from executing Algorithm 4.3 for a synthesized application condition $AC_{X,J}$ is correct w.r.t. Definition 4.29. \square

We split the proof of Theorem 4.30 into four paragraphs and begin with a preparatory sketch of the involved rules.

PRELIMINARIES FOR PROOFS Figure 4.27 shows a sketch of a GT rule R_X with a synthesized application condition $AC_{X,J}$ as well as the structure of and the relations among the involved patterns in R_X and $R_{anticipate,X,J}$. To improve the visual presentation, we abbreviate $R_{anticipate,X,J}$ as $R_{a,X,J}$ in the figure and represent identical pattern elements (i.e., mote variables, link variables, and attribute constraints) using rectangles with identical color and filling.

We shortly recapitulate the properties of the involved variables and patterns. The GT rule R_X has a list of input rule parameters, which uniquely characterize the pending context event that is processed by applying R_X (shown as diagonal lines). These rule parameters constitute a partial match of LHS_X . Furthermore, LHS_X may contain variables that are not bound by the partial match (shown as orange). To ensure that the context event characterized by the rule input parameters is unambiguous, the extension from the partial match determined by the rule parameters to a match of LHS_X is unique. The RHS pattern RHS_X of R_X is omitted here because it is not relevant for the synthesis of the anticipation rule $R_{anticipate,X,J}$. Now, let's consider the synthesized application condition $AC_{X,J}$. The premise $p_{X,J}$ of $AC_{X,J}$ is an extension of LHS_X (shown as $\text{orange and diagonal lines}$), and each conclusion pattern $c_{X,J,1}, c_{X,J,2}, \dots$ of $AC_{X,J}$ is an extension of $p_{X,J}$ (shown as $\text{orange and diagonal lines and blue}$). As

usual, we omit the mappings from a LHS pattern to the application condition premise patterns of a rule and from the premise to the conclusion patterns of an application condition to improve readability.

From the construction of $R_{\text{anticipate},X,J}$ (Algorithm 4.3) and the anticipability preconditions (Definition 4.22), we know that the following relations between R_X and $R_{\text{anticipate},X,J}$ hold.

- $R_{\text{anticipate},X,J}$ is invoked using the same parameters as R_X (▨).
- $\text{LHS}_{\text{anticipate},X,J}$ is identical to $p_{X,J}$ (▨, ▨, ▨) and contains a marking constraint $s(v_{YZ}) = A$ or $s(v_{YZ}) = I$ for some link variable v_{YZ} .
- $\text{RHS}_{\text{anticipate},X,J}$ results from $\text{LHS}_{\text{anticipate},X,J}$ by removing all attribute constraints and adding the unmarking attribute constraint $s(v_{YZ}) = U$. This difference is indicated by ▨.
- The premise $p_{\text{anticipate},X,J,K}$ of each NAC of the anticipation rule $R_{\text{anticipate},X,J}$ is identical to the corresponding conclusion pattern $c_{X,J,K}$ of the synthesized application condition $\text{AC}_{X,J}$ (▨).

To show that the synthesized application condition is correct, we consider a concrete violation of $\text{AC}_{X,J}$ caused by the pending context event (represented by the rule parameters of R_X). According to Definition 4.20, this violation is a match $m_{X,J}$ of $p_{X,J}$ that (i) is an extension of the match m_X of LHS_X and (ii) cannot be extended to a match of any conclusion pattern $c_{X,J,1}, c_{X,J,2}, \dots$ exists. In Figure 4.27, the existence and nonexistence of matches are shown as arrows and crossed arrows (✗) from a pattern to the topology G_T , respectively. We can now show that the three required correctness criteria for anticipation loops (Definition 4.29) hold.

PROOF OF LEAST-CHANGE CRITERION From the identified mappings between R_X and $R_{\text{anticipate},X,J}$, we conclude that the existence of a match m_X of $p_{X,J}$ implies that a match $m_{\text{anticipate},X,J}$ of $\text{LHS}_{\text{anticipate},X,J}$ exists and that both matches are identical. The GT rule $R_{\text{anticipate},X,J}$ is applicable for the given context event if and only if no extension of $m_{\text{anticipate},X,J}$ to a match $m_{\text{anticipate},X,J,K}$ of any of its NACs $\text{NAC}_{\text{anticipate},X,J,K}$ exists. Due to the fact that the premise patterns of these NACs are identical to the corresponding conclusion patterns in R_X , we can reformulate this statement as follows: $R_{\text{anticipate},X,J}$ is applicable for the given context event if and only if no extension of $m_{X,J}$ to a match $m_{X,J,K}$ of any conclusion pattern $c_{X,J,K}$ of $\text{AC}_{X,J}$ exists. This is only the case if $m_{X,J}$ constitutes a violation of $\text{AC}_{X,J}$. Therefore, $R_{\text{anticipate},X,J}$ is applicable for the given context event if and only if this context event entails a violation of $\text{AC}_{X,J}$.

PROOF OF VIOLATION-RESOLUTION CRITERION To show that the violation-resolution criterion holds, we prove that a match $m_{\text{anticipate},X,J}$ of $\text{LHS}_{\text{anticipate},X,J}$ that corresponds to a violation of an application condition no longer exists after the application of the anticipation rule $R_{\text{anticipate},X,J}$.

We know that $\text{LHS}_{\text{anticipate},X,J}$ contains at least one marking attribute constraint for a link variable v_{YZ} (i.e., $s(v_{YZ}) = A$ or $s(v_{YZ}) = I$). Therefore, a match of $\text{LHS}_{\text{anticipate},X,J}$ maps link variable v_{YZ} to a marked link. For the same link variable v_{YZ} , the RHS pattern $\text{RHS}_{\text{anticipate},X,J}$ contains an unmarking constraint (i.e., $s(v_{YZ}) = U$). This constraint ensures that the marked link that v_{YZ} maps to becomes unmarked during the rule application. Therefore, the original match $m_{\text{anticipate},X,J}$ no longer exists. This means that the violation that is represented by $m_{\text{anticipate},X,J}$ is resolved by applying the anticipation rule $R_{\text{anticipate},X,J}$ at $m_{\text{anticipate},X,J}$.

PROOF OF CONSISTENCY-PRESERVATION CRITERION The only modification that the anticipation rule $R_{\text{anticipate},X,J}$ performs is to set the state of a link to unmarked. According to the second anticipability precondition, unmarking a link preserves weak consistency (Definition 4.22). Therefore, applying an anticipation rule preserves weak consistency.

To sum up, all three correctness criteria for anticipation loops (Definition 4.29) are fulfilled by the synthesized anticipation loops and, therefore, Theorem 4.30 holds.

4.4.3.2 Correctness of refined context event handlers

We have already shown that each synthesized application condition is correct (Theorem 4.30). In the following, we build on this result to show that a context event handler that has been refined using Algorithm 4.3 is correct in the sense of Definition 4.7. As a reminder, we repeat the relevant correctness criteria for context event handlers in the following definition.

Definition 4.31 (Correctness of context event handler (excerpt of Definition 4.7)). A context event handler of a TC mechanism specification is *correct w.r.t. a given three-level consistency specification* (in the sense of Definition 3.22) if the following three correctness criteria are fulfilled.

- (i) For a given context event, the context event handler terminates (CC4).
- (ii) The context event handler preserves weak consistency (CC5).
- (iii) The context event handler processes the context event (CC6).

A context event handler processes a context event if it (i) removes the corresponding context event marker from the topology, and (ii) applies the corresponding operationalizing context event rule according to Table 3.2. \square

The following theorem is the main result of this section.

Theorem 4.32 (Correctness of refined context event handler specification). Let the operation handle_X be a context event handler that terminates (CC4) and preserves weak consistency (CC5). Then, after a refinement handle_X using Algorithm 4.3, handle_X is correct w.r.t. Definitions 4.7 and 4.31. \square

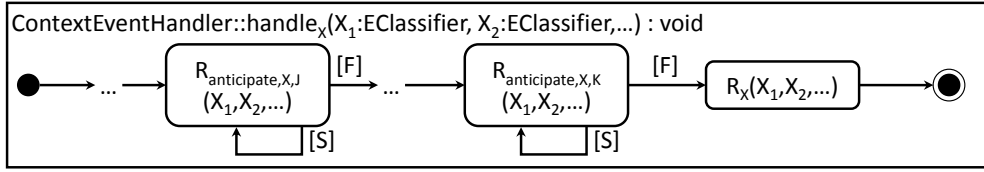


Figure 4.28: Sketch of refined context event handler for proof of correctness

In the subsequent paragraphs, we show all three correctness criteria for the refined context event handler.

PRELIMINARIES FOR PROOFS We first summarize the notation for the following proofs. As a reminder, Figure 4.28 shows the structure of a refined context event handler handle_x . The GT rule R_x is the considered context event rule that possess a number of synthesized application conditions (Algorithm 4.1). The rule and operation parameters X_1, X_2, \dots represent the context event that is processed by handle_x . The operation handle_x is the context event handler that corresponds to R_x and has been refined using Algorithm 4.3. With $AC_{x,j}$, we denote one of the synthesized application conditions of R_x . For each synthesized application condition $AC_{x,j}$, a corresponding anticipation loop with an anticipation loop $R_{\text{anticipate},x,j}$ exists. Due to Theorem 4.30, we know that each anticipation loop is correct. This means that applying $R_{\text{anticipate},x,j}$ produces no additional violations of application conditions of R_x and resolves at least one violation of $AC_{x,j}$.

PROOF OF TERMINATION Now we show that the execution of the refined context event handler specification terminates for a given context event (CC4) under the assumption that the unrefined context event handler terminates. The control flows of the unrefined and the refined context event handlers differ only in the additional anticipation loops that have been inserted in front of the story node containing R_x . The story node that contains R_x has an outgoing activity edge leading to the stop node. Therefore, it is sufficient to show that each anticipation loop terminates to prove that handle_x also terminates. Let's consider an arbitrary anticipation loop of an anticipation rule $R_{\text{anticipate},x,j}$. The anticipation loop does not terminate if and only if the contained GT rule $R_{\text{anticipate},x,j}$ is applicable infinitely often due to the looping [S]-edge.

The number of nodes and links in a topology is finite. Therefore, the number of violations of $AC_{x,j}$ is finite. The fulfilled violation-resolution criterion of $R_{\text{anticipate},x,j}$ guarantees that the number of violations of $AC_{x,j}$ decreases with each application of $R_{\text{anticipate},x,j}$. This implies that the number of iterations of the anticipation loop is finite. Therefore, the context event handler terminates.

PROOF OF PRESERVED WEAK CONSISTENCY Next, we show that the refined context event handler handle_X preserves weak consistency under the assumption that it preserved weak consistency prior to the refinement. Similar to the proof of termination, we only have to consider the anticipation loops because they constitute the only difference between unrefined and refined context event handler specification.

Due to the fulfilled consistency-preservation criterion of correct anticipation loops, we know that each anticipation rule preserves weak consistency. This means that weak consistency is an invariant of handle_X , and we conclude that handle_X preserves weak consistency.

PROOF OF APPLICATION OF CONTEXT EVENTS Finally, we show that the refined context event handler handle_X processes each context event. This is equivalent to the requirement that R_X is applicable for each context event (as stated in Table 3.2). A context event preserves the original (i.e., non-synthesized) application conditions of R_X . Therefore, the only reason for R_X to be inapplicable is a violation of one of the synthesized application conditions.

We show that R_X is applicable by contradiction. Therefore, let's assume that the execution of handle_X has arrived at the application of R_X and that a violation of a synthesized application condition $AC_{X,J}$ of R_X exists. The control flow of the refined context event handler ensures that the application of R_X can only be reached if the anticipation rule $R_{\text{anticipate},X,J}$ that corresponds to $AC_{X,J}$ was inapplicable previously. At this point, the topology was free of violations of $AC_{X,J}$ because the anticipation loop iterates as long as possible and resolves a violation of $AC_{X,J}$ in each iteration. This means that a violation of $AC_{X,J}$ must have been introduced by a GT rule application that followed the anticipation loop of $R_{\text{anticipate},X,J}$. Algorithm 4.3 inserts the sequence of anticipation loops immediately in front of the application of R_X . Therefore, all modifications of the topology between the application of $R_{\text{anticipate},X,J}$ and R_X result from a successful applications of another anticipation rule $R_{\text{anticipate},X,K}$. Still, this is impossible because the consistency-preservation criterion holds for each anticipation loop. Therefore, no violation of $AC_{X,J}$ w.r.t. the current context event may exist when the execution arrives at the story node containing R_X .

To sum up, a context event handler specification that has been refined using Algorithm 4.3 is correct and Theorem 4.32 holds.

4.5 APPLICABILITY OF ANTICIPATION LOOP SYNTHESIS ALGORITHM

In this section, we first discuss the applicability of the anticipation loop synthesis algorithm in general and, then, outline two additional scenarios where the anticipation loop synthesis algorithm allows to overcome the reduced applicability of GT rules. The first use case is the handling of parameter modifications of the enabled TC algorithm in adaptive WSNs. The second use case is to enforce the termination of the TC algorithm specification.

4.5.1 *Discussion of applicability*

The anticipation loop synthesis algorithm has multiple preconditions that could limit its applicability to further use cases beyond the discussed running example.

In Section 4.4, we already discussed that the assumed structure of context event handler operations—one story node containing the application of the context event rule with the same parameters as the surrounding context event handler operation that has one incoming activity edge—does not limit the applicability of Algorithm 4.3 because these restrictions can be alleviated by introducing appropriate intermediate operations and activity nodes, respectively.

The first anticipability precondition in Definition 4.22 requires that each application condition that shall be processed contains a link with a marking constraint. This precondition does not limit the applicability of Algorithm 4.3 because the TC algorithm can never produce a match of an application condition premise that lacks a marking constraint. The reason for this guarantee is that the only modifications that originate from GT rule applications inside the TC algorithm are the activation and inactivation of links. No other attributes or associations are modified by the TC algorithm.

The second anticipability precondition requires that unmarking a link never violates weak consistency. This property may exclude use cases for Algorithm 4.3 where the premise or a conclusion pattern of an application condition requires links to be in a particular state. Whether or not Algorithm 4.3 is still applicable in this case can be verified statically by applying the constructive approach to the unmarking rule R_u . If the constructive approach returns at least one synthesized application condition, Algorithm 4.3 is inapplicable because each synthesized application condition represents a situation where unmarking a link may violate weak consistency.

In Section 4.5.3, we discuss a scenario where the TC-algorithm-specific constraints contain additional marking attribute constraints and outline how Algorithm 4.3 can be extended to not require the second anticipability precondition.

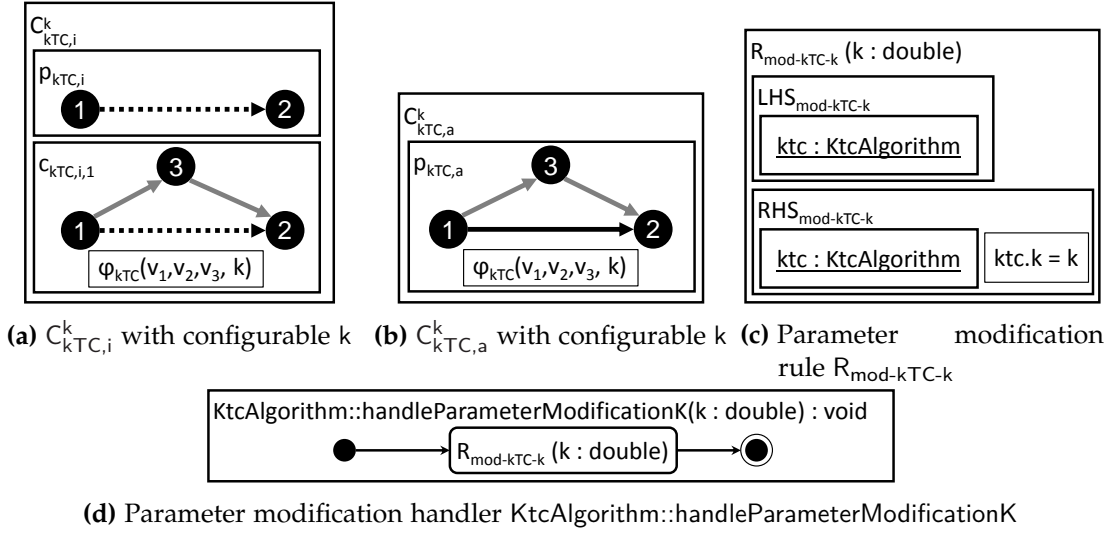


Figure 4.29: Extension of specification for TC algorithm parameter reconfiguration use case

4.5.2 Handling parameter modifications of topology control algorithms

Modern WSNs are adaptive in the sense that the configuration parameters of an individual TC algorithm can be reconfigured or entire TC algorithms can be exchanged at runtime (see also Definition 2.33). In the following, we show how to use the constructive approach (1) and (2) and our proposed anticipation loop synthesis algorithm (3) to ensure that the topology is weakly consistent after reconfiguring a parameter of the TC algorithm. For this, we have to widen our view of the TC mechanism specification. Until now, we worked with a fixed notion of weak consistency (i.e., an immutable constraint set \mathcal{C}_W). The just described scenario is different in that the constraint set that constitutes weak consistency changes at runtime due to the TC algorithm parameter modification. For the following steps, we use a shorthand kTC predicate φ_{kTC} that has four parameters. It is identical to the shorthand kTC predicate introduced in Equation (3.6) but allows to configure the parameter k . Furthermore, we annotate the constraint sets that determine weak and strong consistency with the current value of k . For instance, $C_{kTC,a}^{1.3}$ and $C_{kTC,i}^{1.3}$ denote that the k parameter of kTC is set to 1.3. The kTC -specific constraints with configurable k are shown in Figures 4.29a and 4.29b.

To investigate the effects of this extension, we proceed as follows. First, we introduce TC algorithm parameter modification GT rules as a new category of GT rules that are irrestrictable (like context event rules) because the modification of a TC algorithm parameter shall always be successful. Figure 4.29c shows the new GT rule $R_{mod-kTC-k}$ in object diagram notation. The GT rule specifies that the parameter k of the TC algorithm

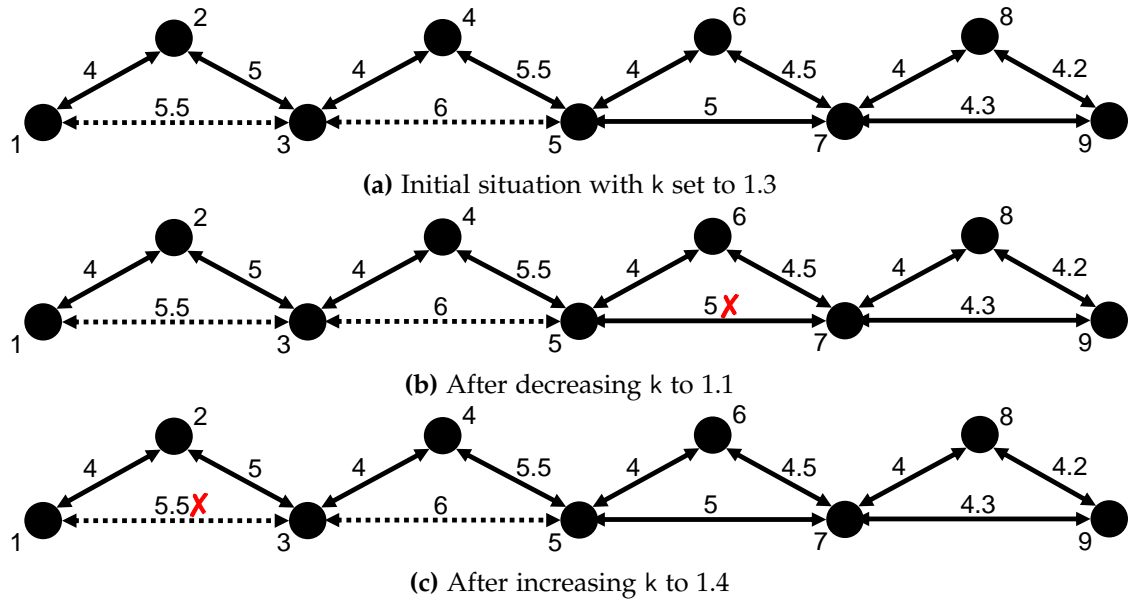


Figure 4.30: Example: Consistency violations (X) due to modification of k_{TC} parameter k

k_{TC} changes to a new value k (provided as input rule parameter). As usual, the implicit object variable for the unique Topology instance is hidden from the GT rule.

Second, we create a handler operation in the TC algorithm class within the meta-model and implement the operation as story diagram that contains a single story node that applies the TC algorithm parameter modification rule. Figure 4.29d shows the story diagram that implements the new operation `KtcAlgorithm::handleParameterModificationK`. The extension of the metamodel is not shown here for conciseness reasons. The operation has a single parameter k , which represents the new value of k and is passed as parameter to the GT rule in the single story node of the story diagram.

Figure 4.30 illustrates that the specification violates $C_{k_{TC},a}$ at this point. The initial topology in Figure 4.30a is strongly consistent if k is set to 1.3. The remaining subfigures illustrate the effect of decreasing k to 1.1 (Figure 4.30b) and increasing k to 1.4 (Figure 4.30c). In the former case, the links l_{57} and l_{75} violate $C_{k_{TC},a}$ because their weights are now larger than $k \cdot 4 = 4.4$. In the latter case, the links l_{13} and l_{31} violate $C_{k_{TC},i}$ because their weights are no longer large enough ($5.5 < k \cdot 4 = 5.6$). The red cross marks (X) indicates that only two of the four inactive links violate weak consistency (i.e., form a match of the negative graph constraint $C_{k_{TC},a}$). As in Figure 4.22c, the consistency violations could be resolved by unmarking l_{57} and l_{75} in the former case and l_{13} and l_{31} in the latter case.

Third, we apply the constructive approach (1 and 2) and the anticipation loop synthesis algorithm (3) to refine the TC algorithm parameter modification diagram as to ensure that weak consistency holds w.r.t. the modified constraint set after executing the

TC algorithm parameter modification operation. Figure 4.31a shows the single postcondition premise $p'_{\text{mod-kTC-k,a,1}}$, and Figure 4.31b the corresponding application condition premise $p_{\text{mod-kTC-k,a,1}}$ that result from the gluing substep 1.1 and the anticipation step 2 for $(R_{\text{mod-kTC-k}}, C_{\text{kTC,a}})$. Figure 4.31c shows the application condition $AC_{\text{mod-kTC-k,i,f}}$ or the rule-constraint pair $(R_{\text{mod-kTC-k}}, C_{\text{kTC,i}})$. In these figures, we use a mixed notation that represents node and link variables in compact notation and all other types of variables in object diagram notation. The glued part of $RHS_{\text{mod-kTC-k}}$ and $p_{\text{kTC,a}}$ is the hidden topology. Figure 4.31d shows the refined parameter modification rule $R'_{\text{mod-kTC-k}}$ after applying the constructive approach.

To ensure that $R'_{\text{mod-kTC-k}}$ is always applicable, the anticipation loop synthesis step 2 finally inserts two anticipation loops that correspond to the two synthesized application conditions into `KtcAlgorithm::handleParameterModificationK` (Figure 4.31e). Figures 4.31f and 4.31g show the corresponding anticipation rules.

4.5.3 Enforcing termination of topology control algorithm specification

In Example 4.19, the proof of termination for the kTC algorithm specification was arguably simple because the only two synthesized application conditions of R'_a and R'_i are “complementary” to each other, that is, one of the two rules R'_a and R'_i is always applicable to a given unmarked link v_{12} . This property ensures that the TC algorithm specification marks a link in each iteration of the loop. This observation does not hold in general. In [119], we show how to use the same algorithm presented here for refining the story diagram of the kTC algorithm specification to ensure that it always terminates.

In this thesis, we give a brief summary of the case study, which we discuss in detail in [118, 119, 122]. We begin with the modified TC-algorithm-specific constraint set $C_W^M = \{C_{\text{kTC,i}}^M, C_{\text{kTC,a}}^M\} \cup C_T$, whose kTC-specific constraints are shown in Figure 4.32. The considered GT rules are identical to our running example.

The shown active- and inactive-link constraints contain marking constraints for the link variables v_{13} and v_{32} . This means that the negative graph constraint $C_{\text{kTC,a}}^M$ is more “lenient” than $C_{\text{kTC,a}}$ because a topology that violates $C_{\text{kTC,a}}$ may fulfill $C_{\text{kTC,a}}^M$. In contrast, $C_{\text{kTC,i}}^M$ is “stricter” than $C_{\text{kTC,i}}$ because an extension of a premise match must additionally fulfill the marking constraints compared to $C_{\text{kTC,i}}$.

If we apply the constructive to the modified constraint set C_W^M , we obtain a different set of synthesized application conditions. For the moment, we focus on the refined TC rules R'_a and R'_i , which result from applying the constructive approach (1 and 2) and are shown in Figures 4.33a and 4.33b, respectively. The refined TC algorithm story diagram is identical to one shown in Figure 4.20. However, both refined TC rules possess three synthesized application conditions. The application conditions $PAC_{i,1}$ and $NAC_{a,1}$ are similar to before (apart from the additional marking constraints). The only differences are the new marking conditions $s_{13}, s_{32} \in \{A, I\}$. As in the preceding examples, $PAC_{i,1}$ and $NAC_{a,1}$ are complementary: A link v_{12} that fulfills $PAC_{i,1}$ does not fulfill $NAC_{a,1}$,

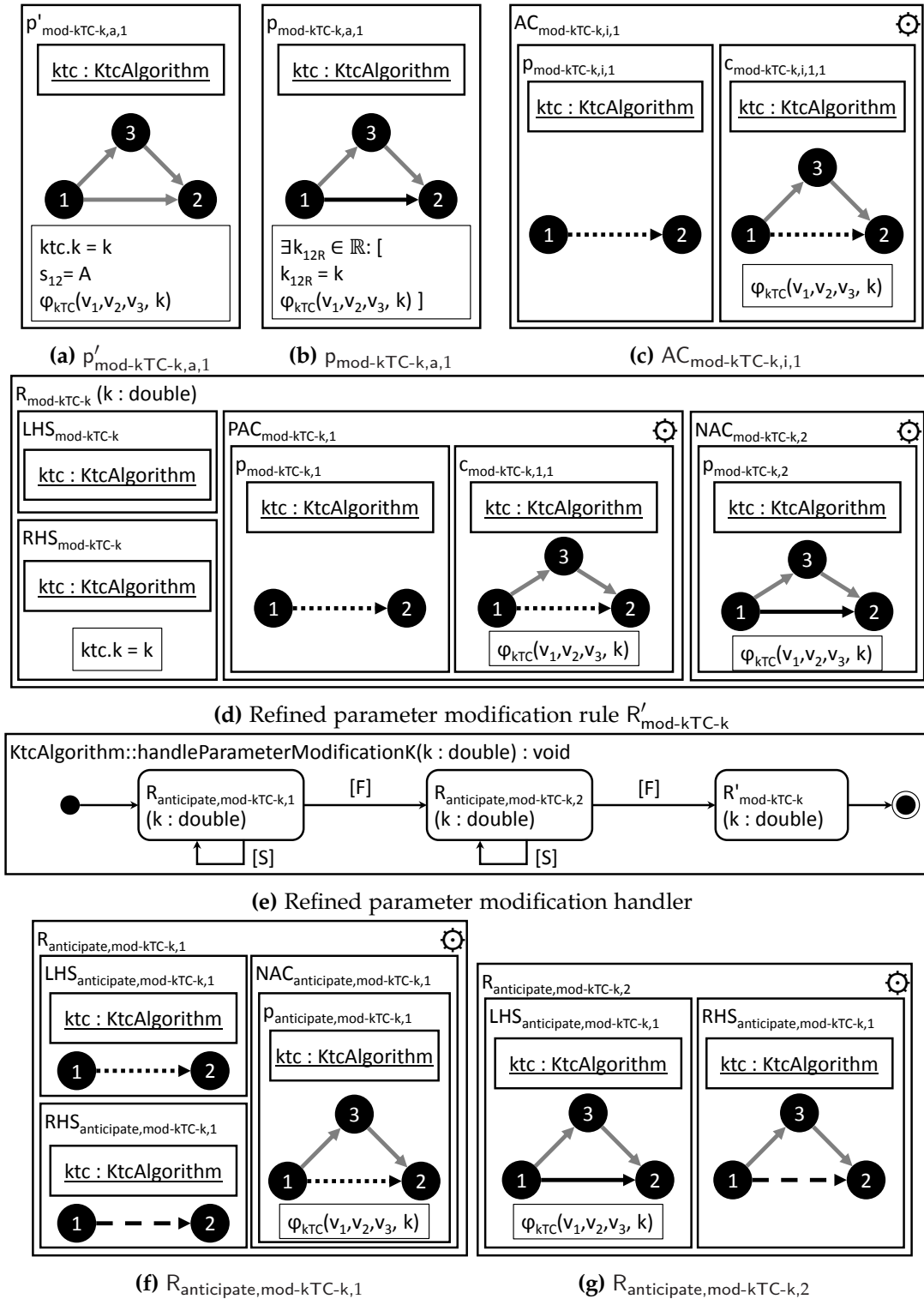


Figure 4.31: Example: Results of applying the constructive approach to $R_{\text{mod-kTC-k}}$

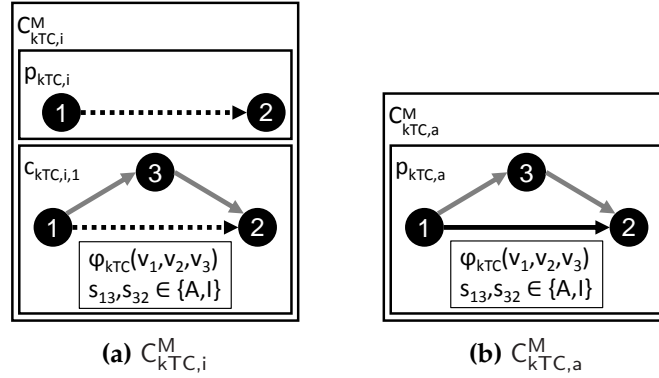
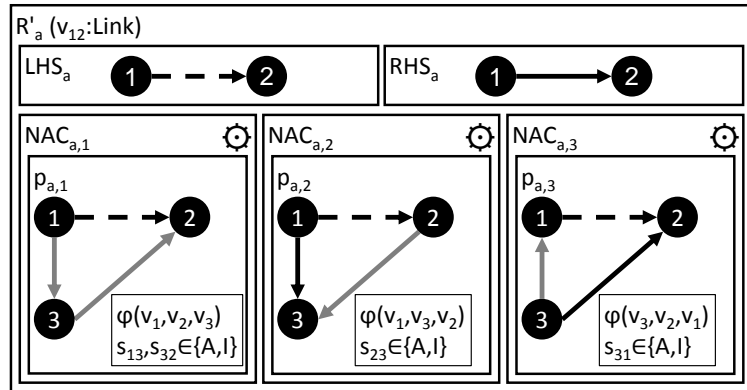


Figure 4.32: kTC-specific constraints with marking constraints for v_{13} and v_{32}

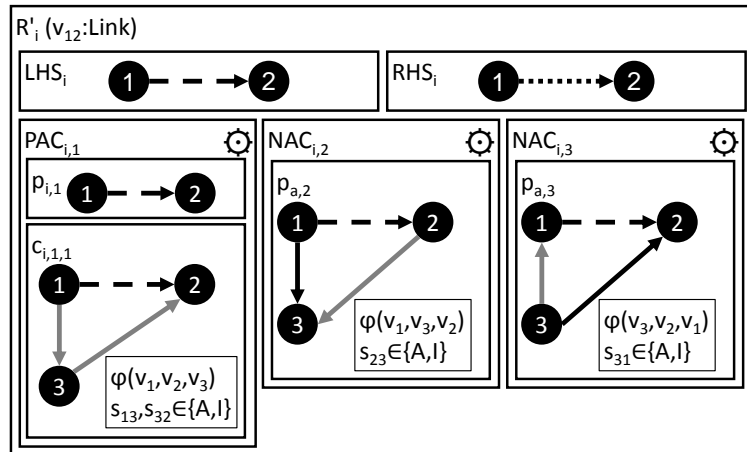
and vice versa. The four other application conditions $NAC_{i,2}$, $NAC_{i,3}$, $NAC_{a,2}$, $NAC_{a,3}$ result from the synthesis algorithm iterations for the rule-constraint pairs $(R_a, C_{kTC,a}^M)$ and $(R_i, C_{kTC,i}^M)$. Notably, the patterns of $NAC_{i,2}$ and $NAC_{a,2}$ as well as $NAC_{i,3}$ and $NAC_{a,3}$ are identical.

In fact, the identical pairs of application conditions may prevent the termination of the TC algorithm specification.

Example 4.33 (Nontermination of kTC with C_W^M). We illustrate this observation using the topology shown in Figure 4.34a (kTC with $k = 1.3$). In the shown weakly consistent topology, the four links l_{12} , l_{21} , l_{47} , and l_{74} are still unmarked. Still, both TC rules are inapplicable to the topology (among others) due to the identical pairs of negative application conditions $NAC_{a,2}$ - $NAC_{i,2}$ (for l_{12} , l_{47}) and $NAC_{a,3}$ - $NAC_{i,3}$ (for l_{21} and l_{74}). This means that the TC algorithm specification never terminates for the topology in Figure 4.34a. The corresponding strongly consistent topology is shown in Figure 4.34f. Table 4.6 summarizes the application conditions that prevent the application of R'_a and R'_i . The complementary application conditions $NAC_{a,1}$ and $PAC_{i,1}$ are not problematic in this context because the control flow of the TC algorithm specification ensures that both GT rules are tried if one of the GT rules is inapplicable. The solution in for the given topology is to retract marking decision systematically before trying to apply R'_a and R'_i . For example, l_{14} and l_{41} should be inactive because they are both the weight-maximal links in a φ_{kTC} -fulfilling triangle (as shown in Figure 4.34f). This modification, however, would produce a new violation of $C_{kTC,i}$ at l_{35} . In Figure 4.34b, the marking decisions for l_{14} and l_{41} have been retracted. In Figure 4.34c, the link l_{12} and l_{21} that caused the preceding retraction step have been activated. Throughout this example, we process pairs of reverse links jointly to reduce the number of intermediate steps. When generalizing the pre-



(a) Refined inactivation rule R'_i



(b) Refined activation rule R'_a

Figure 4.33: Refined TC rules after application of constructive approach based on C_W^M

Table 4.6: Reasons for inapplicability of R'_a and R'_i in Figure 4.34a (preventing link: violation of application condition of R'_a or R'_i , bold: problematic pairs of application conditions)

Link	Final state	Preventing link(s)	Reasons for R'_a	Reasons for R'_i
l_{12}	A	l_{14}	NAC_{a,2}	NAC_{i,2} , PAC _{i,1}
l_{21}	A	l_{41}	NAC_{a,3}	NAC_{i,3} , PAC _{i,1}
l_{47}	I	l_{46}, l_{36}	NAC _{a,1} , NAC_{a,2}	NAC_{i,2}
l_{74}	I	l_{64}, l_{63}	NAC _{a,1} , NAC_{a,3}	NAC_{i,3}

sented idea of retraction, we perform the retraction operation for each unmarked link separately (e.g., unmark l_{14} , then mark l_{12}). In Figure 4.34d, we see that the retraction step must be carried out recursively because unmarking a link may cause a violation of weak consistency. Therefore, to enable the marking of l_{47} and l_{47} , the four links l_{46} , l_{36} , l_{64} , and l_{63} have to be unmarked. In Figure 4.34e, the links l_{47} and l_{74} that caused the second retraction step have been inactivated. Finally, in Figure 4.34f, the remaining unmarked links have been marked.

The solution to the nontermination problem that we discussed in the preceding example can be generalized in a similar way as for context event handlers. However, the anticipation loop synthesis algorithm cannot be applied directly in this scenario because the second anticipability precondition is violated (see Definition 4.22): Unmarking a link (i.e., applying R_u) may violate weak consistency (which necessitates the recursive retraction step in Figure 4.34d).

For the sake of conciseness, we only sketch the three required modifications of our approach in the following. First, we refine R_u during the application of the constructive approach as usual (1) and (2) because R_u may violate weak consistency. The unmarking rule R_u is irrestrictable (like context event rules) because we use it to unmark links intentionally. Therefore, we introduce unmarking handler operations handle_u into the TC algorithm class (e.g., KtcAlgorithm) and the context event handler classes (e.g., $\text{KtcLinkAdditionHandler}$), which is also subject to the anticipation loop synthesis algorithm. The specifications of handle_u are identical in each class. Whenever a link v_{12} shall be unmarked in the specification, we insert an invocation of the unmarking handler operation handle_u : $\text{handle}_u(v_{12})$.

Second, the anatomy of the synthesized anticipation loops changes because of the violated anticipability precondition that R_u preserves weak consistency. Instead of unmarking a link that violates a synthesized application condition, we delegate the unmarking process to handle_u (see sketch in Figure 4.35) The synthesized anticipation rule $R_{\text{anticipate},X,J}$ now has equal LHS and RHS patterns and the identified unmarked

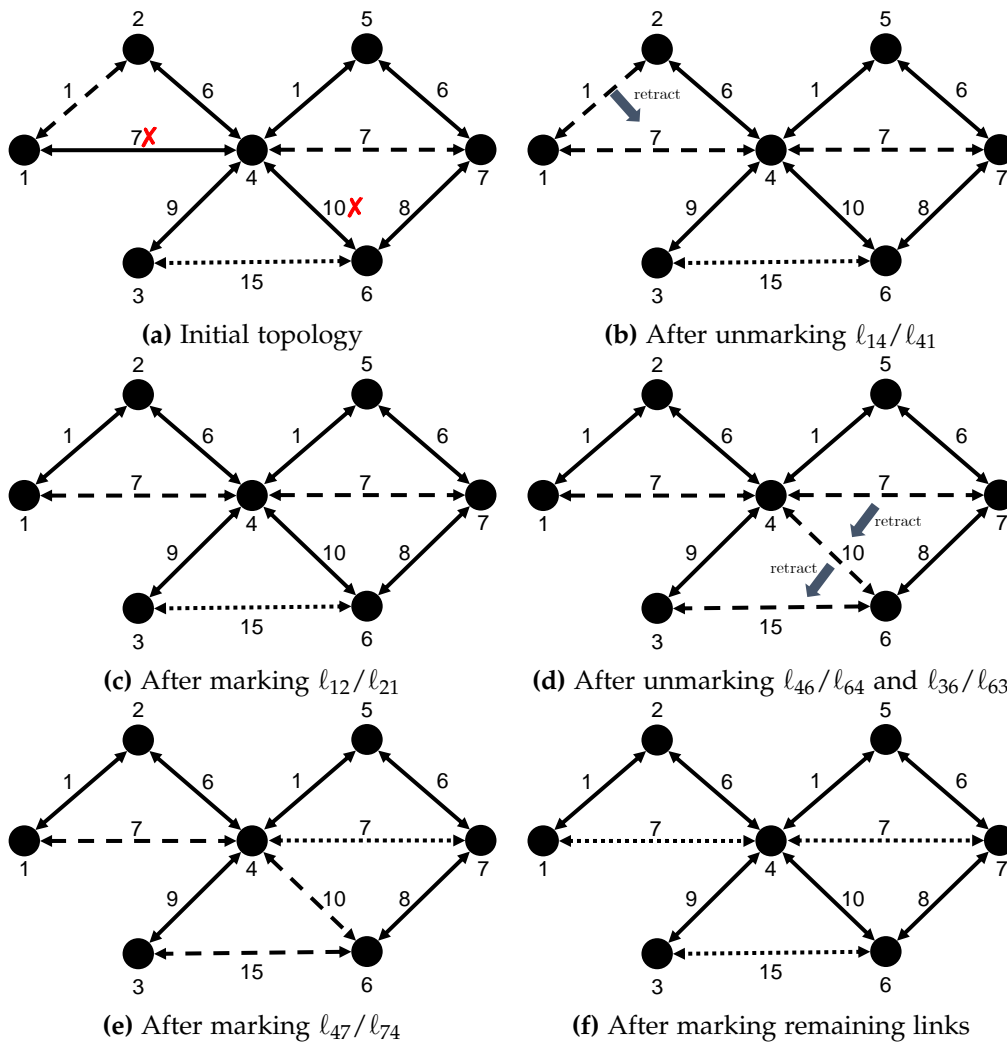


Figure 4.34: Example: Solution to nontermination of refined TC algorithm specification (X: violation of $NAC_{a,2}-NAC_{i,2}$ or $NAC_{a,3}-NAC_{i,3}$, kTC with $k = 1.3$)

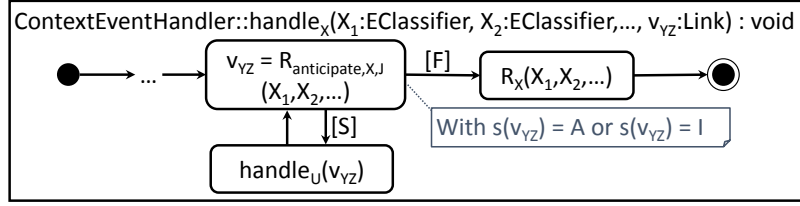


Figure 4.35: Structure of synthesized anticipation loops (generalized, v_{YZ} : link variable with marking constraint in $AC_{X,Y}$)

link is passed to $handle_u$. This also applies to the synthesized anticipation loops inside $handle_u$. In other words, the unmarking handler operation $handle_u$ contains recursive invocations of itself.

Example 4.34 (Unmarking handler). Figure 4.36 shows the refined unmarking rule R'_u and the refined unmarking handler $KtcAlgorithm::handle_u$. During the application of the constructive approach (1) and (2), the unmarking rule R_u receives two synthesized application conditions $PAC_{u,1}$ and $PAC_{u,2}$, which are similar to the synthesized application conditions of R'_e . The positive application condition $PAC_{u,1}$ ensures that each outgoing inactive link v_{13} of v_1 is still part of a φ_{kTC} -fulfilling triangle after unmarking v_{12} . Similarly, the positive application condition $PAC_{u,2}$ ensures that each incoming inactive link v_{32} of v_2 is still part of a φ_{kTC} -fulfilling triangle after unmarking v_{12} .

To ensure that R'_u is always applicable for a given link v_{12} , the unmarking handler operation $handle_u$ contains two synthesized anticipation loops that eliminate all violations of $PAC_{u,1}$ and $PAC_{u,2}$. For conciseness reasons, the actual anticipation rules are not shown here.

Third, to ensure that the TC algorithm specification terminates, we apply a modified version of the anticipation loop synthesis algorithm to each pair of shared application conditions $AC_{a,X}-AC_{i,Y}$ of R'_a and R'_i . A *retraction loop w.r.t. an unmarked link v_{12} and a pair of shared application conditions $AC_{a,X}-AC_{i,Y}$ of the activation and inactivation rules* is a story node that applies the retraction rule $R_{retract,aX,iY}$ that corresponds to $AC_{a,X}-AC_{i,Y}$ as long as possible. An application of a retraction rule identifies a marked link that is part of a violation of $AC_{a,X}-AC_{i,Y}$. In general, the retraction rule unmarks a link v_{YZ} that is part of a violation of $AC_{a,X}-AC_{i,Y}$. As for the anticipation loop synthesis algorithm, the application condition pair $AC_{a,X}-AC_{i,Y}$ must contain a marking constraint for v_{YZ} (e.g., $s(v_{YZ}) = A$ or $s(v_{YZ}) = I$) to ensure that each application of the retraction rule destroys a violation of $AC_{a,X}-AC_{i,Y}$. For each shared synthesized application condition of R'_a and R'_i , we insert a retraction operation invocation at the beginning of the loop body (i.e., in front of R_a when using the proposed template specification in Figure 4.2) Figure 4.37a sketches the structure of the inserted retraction loop. The link variable v_{YZ}

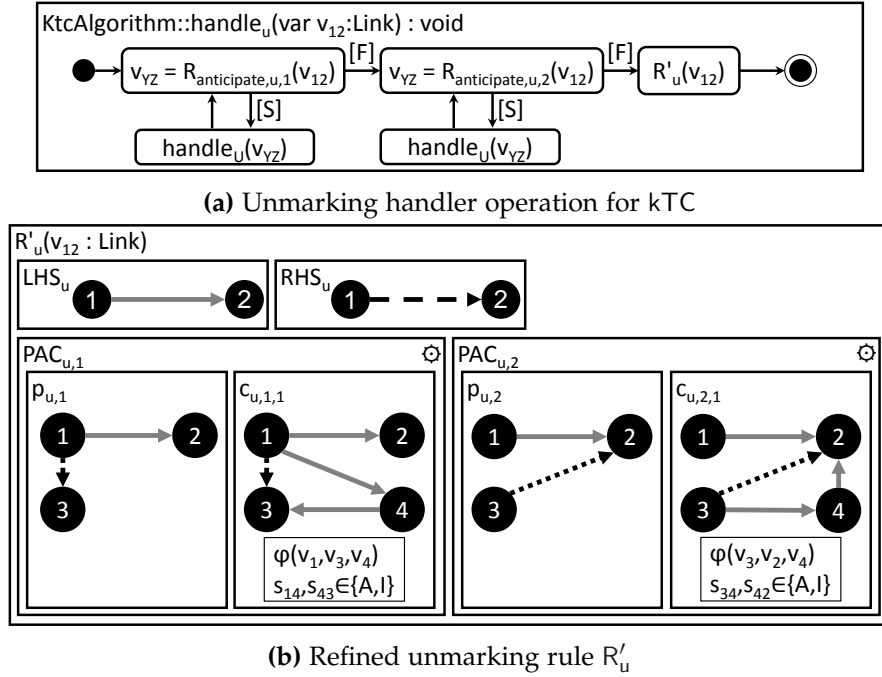


Figure 4.36: Refined unmarking handler and unmarking rule R'_u for kTC

must be chosen such that entire retraction process (including the recursive invocation of handle_u) terminates. The termination proofs for the TC algorithm and context event specifications (CC1 and CC4) need to consider the recursive behavior of handle_u .

A general strategy to conduct termination proofs for GT rule systems is to define a *potential function* ν (also called *Noetherian function*), which maps a topology G_T to a tuple of natural numbers $\nu(G_T) \in \mathbb{N}^N$ for some $N \in \mathbb{N}$. In our scenario, possible tuple elements could be the number of links per link state or the number of motes. To show that the retraction process terminates, we have to prove *descending chain condition* [150]. This property states that the value of $\nu(G_T)$ decreases strictly w.r.t. the canonical order on \mathbb{N}^N with each possible sequence of GT rule application. If this property cannot be shown for a certain GT rule, we can still try to leverage information from the control flow specification. The major challenge when applying this proof strategy in our scenario is the proper choice of the Noetherian function ν .

The following example illustrates an alternative idea for proving the termination of kTC.

Example 4.35 (Retraction loops for kTC). Figure 4.37 shows the concrete refined kTC specification with the two synthesized retraction rules $R_{\text{retract},a2,i2}$ (Figure 4.37c) and $R_{\text{retract},a3,i3}$ (Figure 4.37d). To ensure that the kTC specification terminates, we un-

mark the link variable that represents the weight-maximal link in a φ_{kTC} -fulfilling triangle (e.g., v_{13} in $R_{\text{retract},a2,i2}$ and v_{32} in $R_{\text{retract},a3,i3}$). We apply a similar strategy during the anticipation loop synthesis for the unmarking handler operation handle_u and the context event handler operations.

The following observation allows us to prove that all operations terminate (CC1 and CC4) and preserve weak consistency: First, whenever handle_u is invoked on a link v_{YZ} during the processing of another link v_{12} (e.g., inside handle_u in Figure 4.36), the unique weight $w'(v_{XY})$ is larger than the unique weight $w'(v_{12})$ of v_{12} : $w'(v_{XY}) \succ w'(v_{12})$.

Based on this observation, we can conduct a proof by induction that is similar to the proof of connectivity (Example 3.23) as sketched in the following:

Induction claim: For a given topology $G_T = (V_T, E_T)$, the invocation of $\text{KtcAlgorithm}::\text{handle}_u$ for a given marked link $l_j \in E_T : s(l_j) \in \{A, I\}$ terminates, preserves weak consistency, and unmarks the link l_j . Let $L = (l_1, l_2, \dots)$ be a list that contains all links in E_T and let L be sorted by decreasing w' (i.e., $w'(l_1) \succ w'(l_2) \succ \dots$).

Induction start ($j = 1$): During the execution of $\text{KtcAlgorithm}::\text{handle}_u(l_1)$, none of the anticipate rules is applicable. Otherwise, we would find a link l_k (matched by variable v_{YZ}) with $w'(l_k) \succ w'(l_1)$, which contradicts the fact that l_1 is the link with maximum unique weight in G_T . Thus, the execution of $\text{KtcAlgorithm}::\text{handle}_u(l_1)$ effectively consists of the application of R'_u to l_1 . The rule application preserves weak consistency by construction, and the operation always terminates. Furthermore, the application of R_u to l_1 is successful because both application conditions $\text{PAC}_{u,1}$ and $\text{PAC}_{u,2}$ are fulfilled. Therefore, the induction claim holds for $j = 1$.

Induction step ($j = N \rightarrow N + 1$): We now assume that the induction claim holds for all links l_j in L with $j \leq N$, and show that the induction claim also holds for link l_{N+1} (i.e., that the execution of $\text{KtcAlgorithm}::\text{handle}_u(l_{N+1})$ terminates, preserves weak consistency, and unmarks l_{N+1}). By construction of the anticipation rules $R_{\text{anticipate},u,1}$ and $R_{\text{anticipate},u,2}$, each link l_k (represented by variable v_{YZ}) that is passed to a recursive invocation of $\text{KtcAlgorithm}::\text{handle}_u$ is longer than l_{N+1} w.r.t. w' : $w'(l_k) \succ w'(l_{N+1})$. This means that k is smaller than or equal to N because of the sorting of L according to w' . Therefore, each recursive invocation of $\text{KtcAlgorithm}::\text{handle}_u$ terminates, preserves connectivity, and causes l_k to be unmarked because we assume that the induction claim holds for all l_j with $j \leq N$. This means that each anticipation loop terminates because the execution of the body of each anticipation loop destroys the match of the corresponding anticipation rule. Therefore, the execution eventually arrives at the application of R'_u to l_{N+1} . Furthermore, the application of R'_u to l_{N+1} is successful because both application conditions $\text{PAC}_{u,1}$ and $\text{PAC}_{u,2}$ are fulfilled by construction of the anticipation loops (3), and R'_u preserves weak consistency (1 and 2). Therefore, the induction claim holds for l_{N+1} .

We close the description of the second use case by discussing a potential improvement regarding the retraction of marking decisions. Our model of TC mechanisms (see

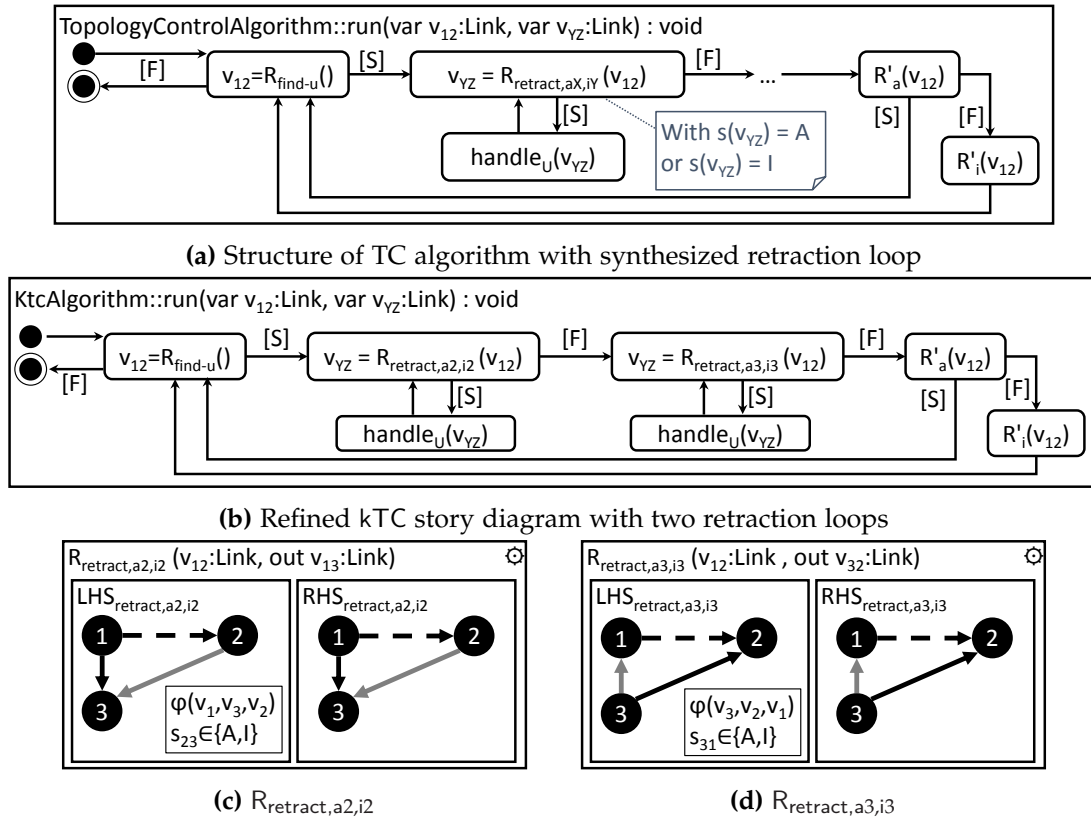


Figure 4.37: Sketch of refined TC algorithm specification in general and for kTC

Figure 2.8) distinguishes between two types of components that both set the link state attribute: The TC algorithm is responsible for enforcing strong consistency, that is, for deciding whether to activate or inactivate the currently unmarked links in the topology. The context event handlers are responsible for processing pending context events by unmarking links in a way that weak consistency is preserved. Inside a context event handler, anticipation loops unmark links (potentially recursively, see Section 4.5.3) to preserve weak consistency.

A complementary strategy would be to decide whether weak consistency can be preserved by toggling the state of a link (i.e., from active to inactive, or vice versa). Example 4.33 illustrates a situation where this strategy could save unnecessary unmarking operations: In Figure 4.34b, instead of retracting the marking decisions for l_{14} and l_{41} , we can inactivate l_{14} and l_{41} immediately. Afterwards, weak consistency still holds. This strategy also avoids the recursive retraction that is caused by the marking attempts for l_{47} and l_{74} (Figure 4.34d). Here, inactivating l_{46} and l_{64} results in a weakly consistent topology, and the states of l_{36} and l_{63} can be left unchanged. This idea could be generalized in future work.

4.6 RELATED WORK

In this section, we survey related work on (i) conceptual extensions of the constructive approach and their potential application to developing TC algorithms, (ii) works that build on the results of the constructive approach (similar to the anticipation loop synthesis algorithm described in this thesis), and (iii) alternate approaches to ensure that a specification (or implementation thereof) fulfills consistency properties.

4.6.1 *Conceptual extensions of the constructive approach*

Graphical consistency constraints (for short graph constraints), as introduced in [91] and as used throughout this paper, express the requirement that particular combinations of nodes and edges should be present in or absent from a graph. This formalism has been generalized to be applicable to High-Level Replacement (HLR) categories in [56]. Habel and Pennemann generalized the constructive approach to support declarative consistency specifications formulated as nested graph constraints, which are as expressive as first-order logic [83]. In his Ph.D. thesis [175], Pennemann provides details on his extension of the constructive approach to nested graph constraints. The (non-nested) graph constraints that we employ in this thesis are a special case of nested graph constraints (with nesting depth of one). As an extension of the original constructive approach, Deckwerth and Varró proposed to handle attribute constraints in an additional processing step [44]. In his dissertation [43], Deckwerth extends the results of [44] to a more general category of typed projective graph transformations (see [43, Ch. 7] for details). We make extensive use of their attribute handling approach in this thesis. In [250], Taentzer and Rensink show how to handle type graphs with type inheritance using the constructive approach.

Graph constraints provide the benefit that they can be used to constructively refine GT rules. Still, their expressiveness is relatively limited. For instance, global constraints such as connectivity cannot be expressed using graph constraints. For this reason, we needed to prove manually that the fulfillment of the TC-algorithm-specific graph constraints in our running example implies that the topology is AU-connected (see Example 3.23). In [84], Habel and Radke present HR^* constraints, a new type of graph constraints that allow to express path-related properties. They showed that the constructive approach is also applicable to HR^* constraints [196]. Details can be found in Radke's Ph.D. thesis [197]. We assume that the synthesis of anticipation loops is also possible for synthesized application conditions based on HR^* constraints. However, employing HR^* constraints in the context of developing TC algorithms could lead to practical issues. The size of the local view that is required to check whether an HR^* application condition is fulfilled may be unbounded. Still, HR^* constraints could be useful to represent tree-based TC algorithms (e.g., LMST [140]). In case of LMST, an HR^* constraint could be used to specify that no active link may be part of a cycle in the topology where this

link is the weight-maximal link (see cycle cancellation rule of Kruskal’s algorithm [127]). An extensive discussion of HR^* constraints can be found in Radke’s dissertation [197].

The constructive approach is a mechanical algorithm. Executing it manually entails considerable manual effort as the large amount of synthesis results for the our small running example show. Fortunately, tool support for translating nested graph constraints into application conditions of GT rules has been proposed recently¹ [164]. The OCL2AC tool translates OCL constraints first into nested graph constraints and then applies the specialization ① and anticipation step ② to the nested graph constraints to obtain application conditions. In the context of OCL2AC, the specialization step is called *shift step* and the latter step is called “left step”). OCL2AC builds on the HENSHIN [9] graph transformation tool. HENSHIN is an EMF-based tool that interprets GT rules at runtime. The OCL2AC tool builds on previous conceptual work by Radke et al. [198, 199]. Currently, OCL2AC is not capable of handling simple relational attribute constraints. For this reason, its application to the kTC example is not possible yet.

Plump and Poskitt followed another line of research rooted at the original constructive approach [91]. Their focus lies on synthesizing preconditions of graph programs in the GP (for “Graph Programs”) GT language [179]. For this purpose, they propose the *E-constraint* formalism, an extension to nested graph constraints that supports node attributes based on infinite domains (e.g., natural numbers) and attribute constraints. In [186, 187], they propose a Hoare calculus to derive preconditions of entire graph programs based on the preconditions of individual rules, which are in turn derived from a given set of E-constraints in the spirit of the constructive approach. In his Ph.D. thesis [185], Poskitt provides more details of the Hoare-style verification approach using E-constraints and presents an adoption to GP2 [12], the successor of GP. The approach by Poskitt and Plump is also applicable to our scenario. Their handling of node attributes is an alternative to the approach by Deckwerth and Varró [43, 44], whose approach to attribute handling should be applicable to nested graph constraints as well (thanks to the generalization of [44] in [43]). Poskitt and Plump’s approach even supports scenarios where the pre- and postconditions of the considered GT rules vary from rule to rule. In contrast, in this thesis, we ensure constructively that all refined GT rules preserve weak consistency. More precisely, using the conventional {precondition}-modification-{postcondition} notation, we ensure that $\{C_W\} R \{C_W\}$ holds for each TC and context event rule R and that $\{C_W\} \text{TopologyControlAlgorithm}::\text{run} \{C_S\}$ holds for the TC algorithm specification.

In [188], as an extension to E-constraints, Plump and Poskitt present *M-constraints*, which allow to express Monadic Second-order logic [39] constraint (e.g., to require connectivity, acyclicity, or two-colorability of graphs). Their construction of application conditions from M-constraints uses the same schema as the original constructive approach [91]. Still, the construction is more elaborate because the algorithm needs to keep track of potential intermediate paths between nodes. The expressiveness of M-

¹ OCL2AC page: <https://ocl2ac.github.io/home/> (visited: 2018-09-19)

constraints is lower than the expressiveness of the HR^* constraints by Radke and Habel because the latter formalism is capable of expressing node-counting Monadic Second-order constraints (e.g., to ensure that a graph always has even node count).

4.6.2 Further usage scenarios of constructive approach

The constructive approach by Heckel and Wagner [91] has been applied in various scenarios. In the following, we survey usage scenarios that build immediately on the constructive approach. For conciseness reasons, we omit usage that build, for instance, on nested graph constraints.

In [159], Mens et al. synthesize application conditions for rules that represent refactoring strategies [62]. In the same spirit, [27] presents an approach to refactoring using distributed GT to represent the architectural aspects of refactoring operations. In [81], the authors specify which elements shall be present in an abstract view of a base model in terms of positive and negative patterns, which can be interpreted as simplified positive and negative graph constraints. These graph constraints are then specialized into application conditions of GT rules that maintain the end-user view of a base model. In contrast to this thesis, the authors of the preceding works use the constructive approach unchanged and do not introduce additional steps.

In [26], propose an approach to refining a behavioral specification for ensuring that a set of positive and negative graph constraints, which they call *enforced generative patterns*, is fulfilled. The development of construction rules for Petri nets serves as illustrative example. Similar to [81], the authors represent integrity properties of Petri nets as positive (i.e., desired) and negative (i.e., forbidden) patterns instead of the precondition-conclusion structure in [91]. In contrast to positive constraints in the sense of [91], Bottoni et al. only support positive constraints with one conclusion pattern. The authors discuss several strategies for ensuring consistency of a GT-based specification w.r.t. a set of positive and negative patterns. The first strategy is to extend the effect of a GT rule (i.e., the RHS pattern). The second strategy is to extend the required context for applying a GT rule (i.e., the LHS pattern). The third strategy is to combine the first and the second strategy by extending both the effect and the required context of a GT rule. Their idea extends the concept of weakest preconditions [47], which forms the basis of the constructive approach, by not only synthesizing preconditions but also additional actions (e.g., the addition of missing required graph elements). In this thesis, synthesizing additional actions is often not an alternative because this would entail, for instance, the creation of nodes and/or links or changes to the value of the link state attribute during arbitrary rule applications. Still, we think that using enforced generative patterns according to the third (mixed) strategy, could be a suitable approach to shortcut unnecessary intermediate unmarking steps. Even though attribute handling is not supported in the approach presented in [26], the authors outline briefly that their approach can be extended to support attribute constraints as well.

In [174], Bottoni et al. present an approach to handle modifications of the constraint set for a specification. They distinguish between eight different modification types (e.g., increase/decrease premise/conclusion pattern graph size) and propose appropriate refinement strategies to ensure that the GT rule preserves the modified constraint set. The modification of constraint sets is a generalization of the discussed TC algorithm parameter modification in Section 4.5.2. In contrast to this thesis, Bottoni et al. only allow for GT rules that either only construct or only delete elements. Furthermore, they only support attribute assignment, but point out that an extension to a general attribute handling is probably possible.

4.6.3 Approaches to ensuring consistency properties

We conclude the discussion of related work in this chapter by surveying different techniques for ensuring that a piece of software fulfills formal requirements. In general, at least three major approaches exist for this purpose. *Correct-by-construction approaches* integrate the requirements during the construction of the software, which is the fundamental idea of this thesis. *Verification-based approaches* examine a system often based on finite subset of the state space (if no suitable abstractions exist). *Test-based approaches* exercises the algorithm by executing a set of representative test cases, each consisting of input data and the expected result and, for each test case, comparing the actual result with the expected result.

In [85], the authors highlight that combining these approaches is useful in realistic projects. In [69], the authors compare 23 verification approaches (incl. conceptual papers as well as model transformation tools) regarding their verification strategies. Amongst others, the authors distinguish between different types of modeling languages (e.g., graph-based vs. general UML models), analysis goals (e.g., behavioral vs. translation correctness) and employed verification techniques (e.g., theorem proving vs. model checking). The main results of the survey are presented as tabular feature model in [69, Tab. 1]. From the discussed verification approaches, the already discussed works by Poskitt and Plump [187] as well as Giese and Lambers [76] are closest in spirit to this thesis. The latter work presents results that have been extended by Dyck in [53] as part of the KORMORAN project². We already discussed the relation between [187] and this thesis. The work by Giese and Lambers [76] proposes to evaluate whether a model transformation specified as triple graph grammar (TGG) [224] preserves a behavioral specification given as set of GT rules using bisimulation.

CORRECTNESS BY CONSTRUCTION Correct-by-construction approaches are being studied extensively in the context of hardware design and software development processes (e.g., [14, 15, 190]). However, to the best of our knowledge, we are the first group to propose a correct-by-construction design methodology for TC algorithms. In the following,

² KORMORAN page: <https://hpi.de/giese/forschung/projekte/kormoran.html> (visited: 2018-09-19)

we discuss a prominent example of a correct-by-construction methodology in greater detail. In [2], the authors present EVENT-B, a correct-by-construction methodology for systems engineering that works by stepwise refinement. The rationale behind EVENT-B is that the developer starts with an initial coarse-grained specification of the system under development and then refines the specification gradually into a fine-grained specification. An EVENT-B model consists of contexts and machines, where a context specifies, for instance, the ranges of variables. A machine is a behavioral specification consisting of (i) a set of variables, which characterize the state of the system, (ii) a set of invariants that the machine shall fulfill permanently, and (iii) a set of events, which specify how the state of the machine evolves. The rationale behind the EVENT-B approach is now that the user starts building an EVENT-B model and receives feedback about which proof obligations have to be shown such that a specified event preserves the invariants of a machine. Based on the necessary proof obligations, the developer may either infer that the EVENT-B model is incomplete (e.g., because preconditions are missing) or prove that the proof obligation holds (semi-automatically or manually). RODIN provides an ECLIPSE -based interface for creating EVENT-B models. A concise summary of EVENT-B is available in [1] alongside with the presentation of RODIN. Additionally, [1, Sec. 4] illustrates the idea of incremental model refinement in EVENT-B using an access-control sample scenario. Using RODIN, the developer receives interactive feedback about pending proof obligations and may use automatic or interactive theorem proving as well as model checking (for models with a finite state space). Additionally, RODIN automatically hides proof obligations that are fulfilled trivially and allows to find out which invariants lead to which proof obligations (traceability). We pursue a similar goal as EVENT-B but use different modeling techniques. For instance, we employ graph constraints for specifying local consistency properties. If the consistency specification contains global consistency properties (e.g., connectivity), we are obliged to prove that the fulfillment of local consistency properties implies the preservation of the global consistency properties. In contrast to EVENT-B, our usage of the constructive approach allows us to refine the specification (semi-)automatically by synthesizing application conditions that ensure the preservation of local consistency properties (i.e., graph constraints). Similar to EVENT-B, our approach is not fully mechanistic because (i) we need to prove manually that the global consistency properties are fulfilled whenever the local consistency properties are fulfilled, and (ii) after applying the constructive approach and the anticipation loop synthesis algorithm, we need to prove manually that the specification terminates (due to reduced applicability of the refined GT rules).

In [41], de Lara et al. present an MDE correct-by-construction approach for refining triple graph grammar rules based on positive and negative patterns, which are interpreted as positive and negative graph constraints (similar to [81]). The authors specify valid output models in terms of pattern triples and provide a translation algorithm that is tailored to triple graph grammars.

In [16], Becker et al. specify unsafe states of a mechatronic system using negative graph patterns (i.e., negative graph constraints). They present an algorithm for verifying statically whether a specification of the mechatronic system as set of GT rules may violate the safety graph constraints using the specialization step of [91]. To reduce the number of false-positive results, they employ the anticipation step (called *ApplyBack* in the paper) to identify only those GT rules that turn a consistent model into an inconsistent model. The authors provide an implementation of the proposed verification algorithm that encodes the GT rules and the graph constraints into first-order logic expressions. These expressions can be evaluated using the tool CROCOPAT³ [23]. In this thesis, we extend the constructive approach for the scenario of local algorithms by enabling the anticipation of unavoidable consistency violations. To distinguish between constraints that need to hold permanently (like the safety constraints in [16]) and constraints that are necessary for ensuring that all links are classified, we introduce the dichotomy of weak and strong consistency.

In [38], Clarisó et al. propose to translate metamodel invariants formulated as OCL constraints [79] into weakest preconditions of model transformation rules. Their approach is not bound to GT rules. Instead, they refine the model transformation rule on the level of atomic operations (e.g., object deletion or creation). They propose to use their constructive approach also for automatic *transformation diagnosis*, where a model finder tool is used to generate models for which the refined model transformation rules are inapplicable. The metamodel invariants in [38] correspond to the structural consistency level introduced in Definition 3.22. In this thesis, we also employed (manual) transformation diagnosis for motivating why we need to introduce anticipation loops for irrestrictable GT rules (e.g., context event rules). In contrast to the graph constraints the constraint language OCL is at least as expressive as first-order logic. In contrast to the results of this thesis, the authors of [38] need not conduct any additional refinement steps (like the anticipation step) to re-establish the applicability of irrestrictable rules because the invariants specified as OCL need to hold permanently. Still, we think that our approach could be combined with [38] if irrestrictable rules exist.

An inspiration for us was the work by Fritsche et al. [64], who present an algorithm to derive shortcut rules in the context of model synchronization using triple graph grammars (TGGs) [223, 224]. From a pair of TGG rules, they synthesize a new TGG rule whose effect corresponds to the retraction of the first TGG rule and the application of the second TGG rule of the rule pair. This strategy can help to reduce information loss and increase efficiency in incremental model transformation scenarios.

In a recent work, Kosiol et al. [123] report on ongoing efforts to create a correct-by-construction development methodology for TGGs that follows a consistency-enforcing strategy. Instead of synthesizing application conditions to ensure that a TGG rule application preserves consistency, they propose to modify the actions of the TGG rule to

³ Website: <https://www.sosy-lab.org/people/beyer/CrocoPat/> (visited: 2018-09-12)

enforce that consistency is fulfilled after applying the TGG rule. They leverage that TGG rules are monotonic (i.e., only create and preserve model elements).

In [165, 166], Nassar et al. present an approach to repair invalid EMF models. For instance, an invalid model can contain superfluous model elements (i.e., objects and associations) due to violated upper multiplicity bounds or lack model elements due to violated lower multiplicity bounds. Consequently, their approach consists of a model-trimming and a model-completing phase, which remove superfluous and add missing elements, respectively. They derive appropriate repair operations from the metamodel, formalize their approach, prove that it is correct, and provide interactive tool support for model repair based on the model transformation tool HENSHIN [9]. Their focus lies on re-establishing the validity of models in a systematic way. In contrast, our approach is to start with weakly consistent (i.e., also valid) model and to derive a TC mechanism specification that preserves weak consistency and, in case of the TC algorithm, enforces strong consistency. In a preceding work, Taentzer et al. [249] propose to introduce the underlying concept of consistency-improving repair operations. Their goal is to repair an invalid model step by step. This means that the model may be invalid after a repair step, but its degree of invalidity should decrease. Their approach also takes the sequence of (user) edit operations that lead to the inconsistent model into account by identifying appropriate repair operations for each performed edit operation.

VERIFICATION One technique for verifying correctness properties of software is model checking [205]. Many model checking approaches are only suitable for verifying properties for models of a finite size. The benefit of the correct-by-construction approach, as applied in this thesis, is that it ensures correctness for models of arbitrary size because each refined GT rule is guaranteed to preserve the specified correctness properties.

In [167], Navarro et al. extend the logic of nested graph conditions by Habel and Pennemann [83] to paths of unbounded length. They present a technique for proving whether a given graph condition with paths is fulfilled for a given graph and, for instance, whether a given (sub-)expression in the logic of nested graph conditions with paths is always *true* or *false* (i.e., a tautology or a contradiction). In contrast to the idea behind this thesis, their approach is primarily suitable to verify the fulfillment of graph constraints with paths instead of refining a behavioral specification to ensure that the graph constraints are preserved.

In [280], Zave specifies the Session Initiation Protocol (SIP) in the specification language PROMELA to verify SIP using the model checker SPIN [96]. In PROMELA, a protocol is specified as a set of concurrent processes (e.g., server and client process), which exchange messages according to a predefined set of restrictions. SPIN is able to construct the state space for a given PROMELA specification up to a fixed size and determines whether a sequence of events in the state space exists that violates liveness or safety properties. Using SPIN, Zave was able to detect several race conditions in SIP.

In a spirit similar to [280], Zave investigated the frequently cited and built-upon CHORD protocol [243] for maintaining distributed hash tables in peer-to-peer networks in [281]. Using the model checker ALLOY [100], Zave was able to identify sequences of events that lead to an unrecoverable state of a CHORD network. Also in this case, the state space investigated using Alloy was finite, but obviously sufficient to detect several specification flaws of CHORD. In [282], Zave finally showed how to improve the CHORD specification to overcome the detected problems.

In [110], model checking is applied to detect bugs in the TC algorithm LMST [140], leading to an improved implementation thereof.

Graph abstractions [13] are a formalism that alleviates the problem that the verification of systems is often limited to a finite part of the state space. By means of graph abstractions symbolic representations of whole classes of models can be introduced. The benefit of the correct-by-construction approach, as applied in this paper, is that it ensures correctness for models of arbitrary size because each refined GT rule is guaranteed to preserve the specified correctness properties.

In Section 3.7, we already discussed examples of verification techniques for probabilistic and stochastic properties of GT systems [90, 124, 157].

TESTING In comparison to correct-by-construction and verification-based approaches, the goal of testing is to derive a finite number of representative test cases that cover a part of the space of possible input values of a piece of software that is as large as possible [163]. Testing is a complementary approach to the correct-by-construction approach that we present in this thesis. From a practical viewpoint, testing is always necessary, e.g., to ensure that the synthesized application conditions have been properly specified in an MDE tool. For an overview of testing techniques for communication system algorithms, we refer the reader to the survey article by Lai [133].

 TOOL SUPPORT FOR RAPID EVALUATION

In this chapter, we present tool support for evaluating TC mechanism specifications rapidly using a network simulator and hardware testbeds. Figure 4.1 locates the role of this chapter in the entire TC algorithm development process (see also Figure 1.2). One of the major goals of our approach is to provide the TC mechanism developer with an interface that allows to specify the TC mechanisms on the level of metamodels and GT rules. Therefore, we decided to build on an existing MDE tool for developing the TC mechanism specification. We chose the modeling tool `EMOFLON` [135] because it is extensible enough to compile the same TC mechanism specification into executable code for different target platforms (e.g., `JAVA`, `C`). We discuss reasons for choosing `EMOFLON` in Section 5.1.

In Section 5.2, we present `COBOLT`, a tool for evaluating a TC mechanism specification modeled with `EMOFLON` using the network simulator `SIMONSTRATOR` [208]. We chose the established network simulator `SIMONSTRATOR` because it provides the necessary components for simulating WSNs and is implemented in `JAVA`. Choosing `JAVA` as target language simplifies the implementation of an interface to the code generated using `EMOFLON`. We enabled the simulative evaluation of TC mechanism specifications by introducing a reusable TC component in `SIMONSTRATOR`. Similar to the proposed architecture of TC mechanisms in Figure 2.8, the TC component has extension points for (i) receiving updates from the input topology (e.g., WiFi), (ii) providing update events to the virtual topology (i.e., a reduced view of the WiFi topology), (iii) invoking the `JAVA`

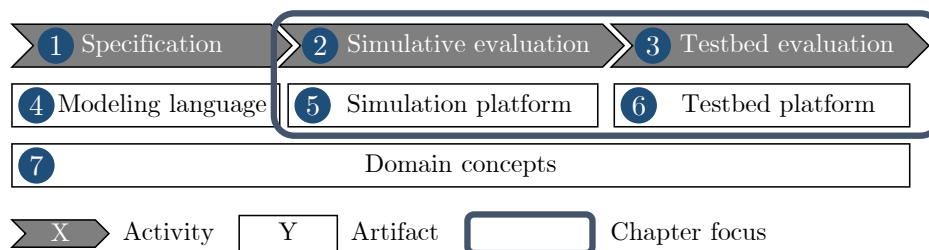


Figure 5.1: Location of Chapter 5 in TC algorithm development process

code that `EMOFLON` generates from the TC mechanism specification. Furthermore, the TC component allows to exchange or reconfigure the active TC algorithm.

In Section 5.3, we present `CMOFLON`, a tool for evaluating a TC mechanism specification on hardware motes running the sensor operating system `CONTIKI` [51]. We selected `CONTIKI` because it is a widely used operating system for IoT devices and the `ToCoCo` framework [239] provides a suitable programming interface for implementing TC algorithms in `CONTIKI`. `CMOFLON` generates C code that is tailored to the `ToCoCo` framework. We designed `CMOFLON` to reuse the `EMOFLON` build process where possible. One of our goals during the design of `CMOFLON` was to compare the generated C code with manual implementations of three TC algorithms (`kTC`, `l*kTC`, `LMST` [239]). For this purpose, we evaluated the generated TC algorithms in the `FLOCKLAB`¹ [149] testbed at ETH Zurich, which consists of 30 `TELOS B` [181] motes.

In Section 5.4, we survey related work on MDE and code generation approaches for network simulation and testbed evaluation.

As a preview of this chapter, Table 5.1 compares the developed tools `COBOLT` and `CMOFLON` for simulative and testbed evaluation w.r.t. key properties. `COBOLT` builds on the unmodified MDE tool `EMOFLON`, whereas `CMOFLON` is a variant of `EMOFLON` that generates embedded C code. The target platform of the TC algorithms generated using `COBOLT` is the `JAVA`-based network simulator `SIMONSTRATOR`, which runs on regular PCs or, for larger experiments, headless on compute servers. In contrast, the TC algorithms generated using `CMOFLON` run on hardware motes (e.g., `TELOS B`) and build on the `ToCoCo` framework for TC in `CONTIKI`. Even though the emulation of these TC algorithms using the `COOJA` emulator of `CONTIKI` is possible, such experiments are typically of a considerably smaller size compared to the simulation experiments with `SIMONSTRATOR` due to the computational effort of emulating the CPU of each mote. The TC algorithms generated with `COBOLT` can be supported easily, for instance, using the debugger view of `ECLIPSE`. A TC algorithm generated with `CMOFLON` can be debugged by inspecting logging messages (testbed and `COOJA`) or the `COOJA` debugger, which is documented only sparsely.

¹ `FLOCKLAB` page: <https://flocklab.ethz.ch/> (visited: 2018-09-17)

Table 5.1: Comparison of developed tools COBOLT and cMOFLON

Property	Simulation	Testbed
Goal	experiments based on numerous scenarios	experiments in selected realistic environment(s)
Typical size	up to 1000 motes	dozens of motes (e.g., 30 in FLOCKLAB [149])
Novel tool	COBOLT (Section 5.2)	cMOFLON (Section 5.3)
Our evaluation goals	Correctness, applicability, dynamic vs. batch TC (see Section 5.2.4)	Correctness, efficiency (see Section 5.3.4)
Integration	novel SIMONSTRATOR components and mapping to EMF	custom code generator
Software platform	eMOFLON [135], SIMONSTRATOR [208]	CONTIKI [51], ToCoCo [239]
Hardware platform	computer	hardware mote (e.g., TELOS B [181])
Language	JAVA	C
TC mode	batch, dynamic	batch
Evaluation type	simulation	testbed, emulation (COOJA)
Debugging support	++ (regular JAVA debugger)	o (logfile, CONTIKI debugger)

5.1 SELECTION OF MODELING TOOL

The selection of the modeling tool focuses on the specification phase of our approach. Ideally, the selected modeling tool should support all specification activities described in Chapter 3. These activities comprise (MT₁) metamodeling valid topologies (Section 3.1), (MT₂) specifying topology modifications as GT rules (Section 3.4) (MT₃) specifying local consistency properties as graph constraints (Section 3.2), (MT₄) specifying the control flow of topology modifications (Section 3.5), (MT₅) applying the constructive approach by refining the GT rules based on the weak consistency graph constraints, (MT₆) evaluating the TC mechanism specification in a network simulator and in hardware testbeds.

In the following discussion, we only consider modeling tools that are being developed actively. Only few model transformation tools allow for the specification of graph constraints (MT₃) (e.g., AGG [248]). Still, from the perspective of our approach, specifying graph constraints is only sensible if the tool supports applying the constructive approach (MT₅) (i.e., the synthesis of application conditions for GT rules based on graph constraints).

Regarding (MT₅), we are aware of two implementations: First, in his Ph.D. thesis [43], Deckwerth presented the prototype SyGRAV for synthesizing application conditions of GT rules from graph constraints. SyGRAV builds on eMOFLON [135] and supports the handling of attribute constraints in patterns, as described in [44]. Unfortunately, the tool was not released publicly, and its development is apparently discontinued. Second, Nassar et al. recently presented the tool OCL2AC [164], which allows to transform nested graph constraints into application conditions of GT rules. OCL2AC builds on the MDE tool HENSHIN [9]. Unfortunately, OCL2AC currently can only handle simple relational attribute constraints and only one attribute constraint per attribute of a variable. The attribute constraints that are required to express TC algorithms such as kTC are not supported (e.g., specifying attribute constraints that comprise arithmetic operations). Due to the generally missing sufficient tool support for applying the constructive approach, we exclude (MT₃) and (MT₅) from the list of selection criteria for modeling tools. Instead, for our running example, we conduct the constructive approach manually and specify the resulting GT rules directly using the chosen modeling tool. In the following, we discuss the remaining criteria (MT₁), (MT₂), (MT₄), and (MT₆) further.

Numerous tools for specifying and generating code from metamodels exist (MT₁) (e.g., ENTERPRISE ARCHITECT², ECLIPSE EMF³). To further restrict the search space, we take the specification of GT rules into account (MT₂). Several model transformation tools allow to specify metamodels and GT rules (e.g., ATL [104], GRGEN.NET [74], HENSHIN [9], eMOFLON [135], VIATRA [21], DEMOCLES [260], GP2 [185], GROOVE [204]). For a comprehensive overview of 60 model transformation tools, we refer to a recent survey [105]. A subset of the aforementioned tools support the specification of control flow.

² ENTERPRISE ARCHITECT page: <https://sparxsystems.com.au/products/ea/> (visited: 2018-10-14)

³ EMF page: <https://www.eclipse.org/modeling/emf/> (visited: 2018-09-19)

Model transformation tools with a specification language for the control flow comprise GRGEN.NET [74], GP2 [185], and EMOFLON [135]. These tools all fulfill the selection criteria (MT₁), (MT₂), and (MT₄). As discussed in Chapter 4, our approach is not tied to a specific control flow specification language. The anticipation loop synthesis algorithm (Algorithm 4.3) can certainly be adopted to other control flow languages (e.g., control-flow units in HENSHIN [9] or graph programs in GP2 [185]).

The only remaining selection criterion to discuss is how the model transformation specification can be evaluated in network simulators and on testbed platforms (MT₆). At this point, we refrain from presenting a comprehensive survey of network simulators and hardware testbeds. Our general observation is that the programming of hardware notes is often carried out in C-like languages (e.g., NESc [73]), whereas network simulators tend to use C/C++ (e.g., NS-3 [253], OMNET++ [259]) or JAVA (e.g., SIMONSTRATOR [208], SIDNET-SWANS [75]).

Ideally, the selected modeling tool should allow to generate code for multiple target platforms. If this is not possible, an adoption of the code generation process should be feasible. Still, not all model transformation tools generate code from the specification. For example, the main model transformation engine of HENSHIN [9] is an interpreter, even though an extension for generating code for bulk-synchronous execution exists [125]. Other model transformation tools follow an event-driven approach (e.g., VITRA [21]) and provide a model transformation engine that tracks all available matches for each pattern. Without major efforts for porting these data structures, they bind the respective tools to one target platform. Model transformation tools that support the generation of target-platform code comprise EMOFLON [135] (JAVA based on EMF), GP2 [185] (C), and GRGEN.NET [74] (C#, .NET). All of the compiler-based model transformation tools that we are aware of support the generation of code for only one target language. Also, given that sensor devices tend to be highly resource-constrained, it would be beneficial if the code generation templates could be adjusted and exchanged easily. Therefore, we needed to decide for a model transformation tool that supports at least one of the desired target languages C, C++, or JAVA. We would then extend the selected modeling tool to support the code generation for the missing target language. Eventually, we decided to build our work on EMOFLON [135] because its underlying pattern matching engine DEMOCLES was built to exchange the target platform without affecting other steps of the build process [260].

EMOFLON BUILD PROCESS In the following, we describe how EMOFLON generates code from a visual specification. Figure 5.2 provides an overview of the code generation workflow of EMOFLON. The workflow consists of the three major phases import, validation, and code generation. All major specification artifacts (i.e., metamodel, story diagrams, GT rules) are created using the EMOFLON add-in for the commercial modeling tool ENTERPRISE ARCHITECT⁴ (called as EMOFLON EA in Figure 5.2).

⁴ ENTERPRISE ARCHITECT page: <https://sparxsystems.com.au/products/ea/> (visited: 2018-10-14)

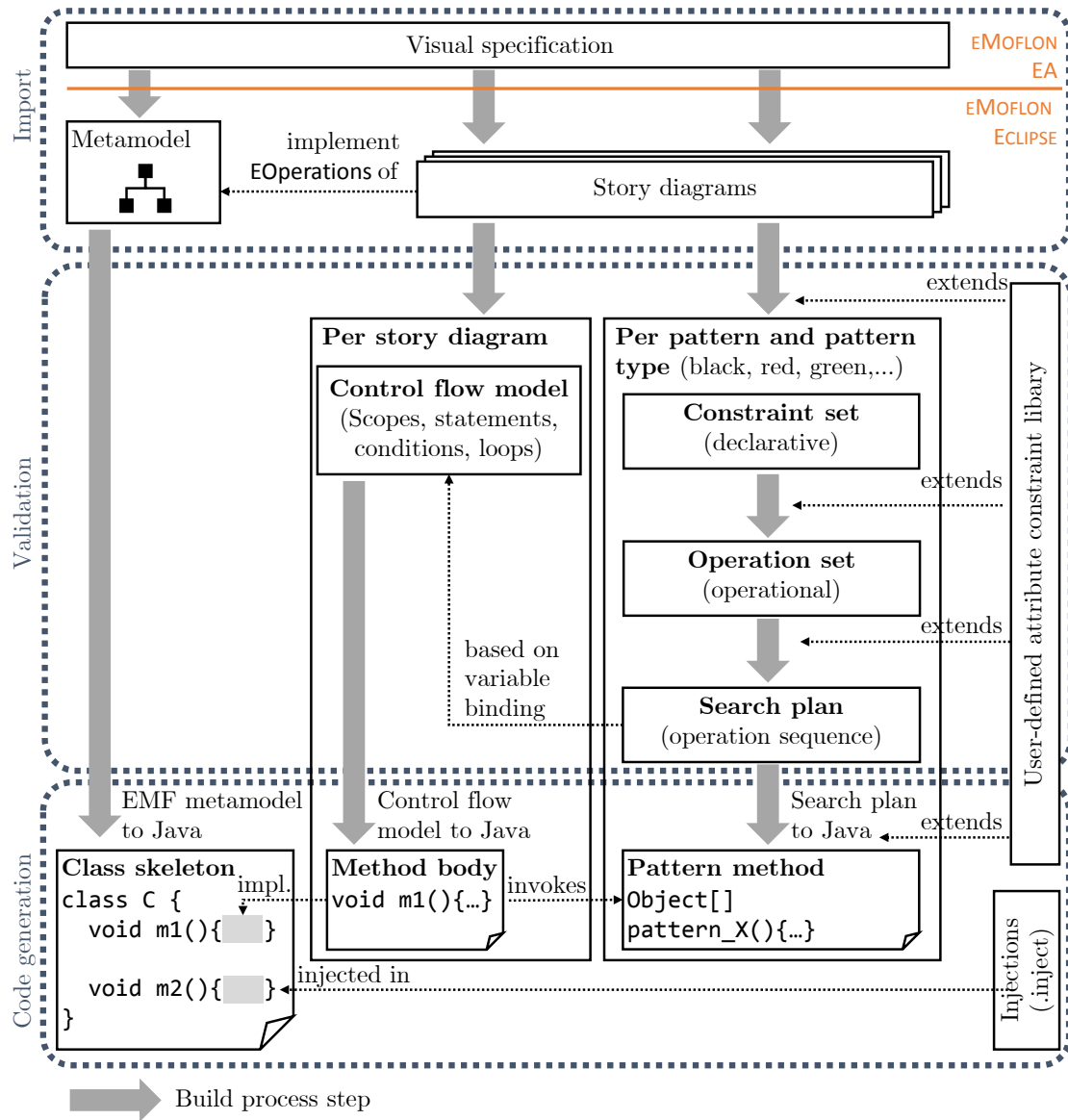


Figure 5.2: Build process architecture of eMOFLON

The validation of and code generation from the specification is performed using the EMOFLON plugin, which builds on ECLIPSE Modeling Components (called as EMOFLON ECLIPSE in Figure 5.2). For this purpose, EMOFLON EA exports the specification artifacts as an XMI file, which is then processed further by EMOFLON ECLIPSE. The import results in an Ecore file that contains the metamodel and the story diagrams. Each story diagram is attached to its corresponding EOperation in the metamodel.

To translate the Ecore-based metamodel, EMOFLON uses the existing EMF code generation process. This process produces for each EClass a Java class skeleton that contains (amongst others) default implementations for getters and setters of the attributes and references of a class, and empty method bodies for EOperations that shall be implemented by the user.

The code generator of EMOFLON extends the EMF build process as follows. Each story diagram is translated into the body of the Java method that corresponds to the EOperation of the story diagram. To decouple the representation of the control flow as story diagram from the actual code generation logic, each story diagram is translated first into a control flow model. The *control flow model of a story diagram* represents the structure of an imperative, `goto`-free method implementation and facilitates the mapping to control-flow structures in Java, C, or C++. These concepts comprise scoping, scoped variables, conditions, loop types (e.g., do-while, while-do, foreach). Story diagrams that cannot be translated into a control flow model are rejected. For this reason, the current step is called the *validation step*.

The construction of the control flow model is complemented by transforming the GT rule applications inside story nodes. Each GT rule is first transformed into up to five patterns of different types. The *pattern type* determines the role of the pattern during the application of a GT rule. The *black pattern of a GT rule* contains all variables that are part of the LHS pattern of the GT rule. A GT rule is only applicable, if a match of its black pattern exists. The *red pattern of a GT rule* contains all those elements that are deleted by the rule application (i.e., all variables that exist in the LHS but not in the RHS pattern). The red pattern is always a subpattern of the black pattern of a GT rule. Analogously, the *green pattern of a GT rule* contains all variables that exist in the RHS pattern but not in the LHS pattern of the GT rule. The remaining two types of patterns enrich story diagrams with additional language features that we did not discuss until now. In EMOFLON, a variable may be bound not only using pattern matching but also by assigning it from (i) another variable (possibly of a supertype to enable conditions based on subtype checks) or (ii) the invocation of an EOperation, which returns an instance of a compatible type. Such variables are collected in the *binding pattern of a GT rule*. Finally, EMOFLON handles the invocation of EOperations and the extraction of return values of EOperations from pattern matches uniformly as *expression patterns*. The latter two pattern types exist for technical reasons and will not be considered in the following.

The code generation per pattern is carried out using an *operation-driven pattern matching engine* [260]. In the following, we summarize the DEMOCLES code generation process for patterns and refer the reader to [260] for a detailed description.

As a first step, DEMOCLES decomposes a pattern into a set of declarative constraints. These constraints represent the attribute constraints of a pattern and the pattern graph uniformly. For example, an object variable v may be associated with a type constraint that requires that an object that is mapped by v in a match must have the type Mote. As another example, a pair (v_1, v_2) of object variables may be associated with a constraint that requires that a source-outgoing association from v_1 to v_2 exists in a match. Each constraint has a number of *constraint parameters* (e.g., one parameter in case of the Mote type constraint and two parameters in case of the source-outgoing association constraint).

As a second step, based on the Ecore metamodel, DEMOCLES then infers how each constraint can be satisfied based on different binding constellations of the constraint parameters. This step is called *operationalization*. For example, the type constraint can only be checked if the given variable v is already bound. In contrast, regarding the (v_1, v_2) object variable pair that is associated with a source-outgoing association, (i) if both variables are bound, then the corresponding operation checks whether a source-outgoing association exists between the mapped objects, (ii) if v_1 is bound and v_2 is unbound, then an iterator of the outgoing links of the match of v_1 is assigned to v_2 , each representing a partial match that needs to be complemented further, or (iii) v_1 is unbound and v_2 is bound, a mapping from v_1 to the unique source node of v_2 is added to the match.

As a third step, DEMOCLES creates the search plan for each pattern (if exists). The *search plan of a pattern* is a sequence of operations (calculated in the previous, second step) that lead to a complete match of the pattern based on an initial partial match. For a black pattern, DEMOCLES takes the partial match consisting of the rule input parameters as starting point to decide which operations to use. In contrast, in a red pattern, DEMOCLES expects all variables to be bound, and the generated search plan contains only removal operations, which delete all model elements that correspond to the variables of the red pattern. Finally, the generated search plan of a green pattern creates model elements for all free variables in the pattern and interprets bound elements as context. Therefore, the search plan of a green pattern contains only creation operations.

As a fourth and last step, DEMOCLES translates the search plan of each pattern into a static JAVA method using a hierarchical STRINGTEMPLATE⁵. The order of invocations of the pattern methods in the method body that corresponds to the control flow model ensures that the pattern methods are invoked in the proper order (e.g., black before red before green).

EMOFLON allows the user to provide the following two types of extensions of the code generation workflow. First, the user may specify user-defined attribute constraints. A *user-defined attribute constraint* consists of a constraint signature (i.e., a constraint name and parameter list) and a set of operationalizations. An operationalization is a code frag-

⁵ STRINGTEMPLATE page: <http://www.stringtemplate.org/> (visited: 2018-10-15)

ment with placeholders for the operation parameters. DEMOCLES consults the library of user-defined attribute constraints if it encounters an unknown constraint type. Second, the user may provide injections. An *injection* is platform-specific code that implements a method body or contributes additional methods and attributes to a class. Injections are stored in separate files having the extension *inject*. This technique allows users to provide platform-specific implementations of EOperations that are difficult to specify using GT rules.

5.2 MODEL-BASED SIMULATIVE EVALUATION WITH COBOLT

After the specification of a TC mechanism, the first step is usually to evaluate the TC mechanism using a network simulator. Using a network simulator is advantageous because it allows to test the TC mechanism rapidly and reproducibly in a large number of scenarios. To produce meaningful results regarding the applicability or performance of a TC mechanism, the network simulator needs to provide realistic models for all simulated aspects of the system (e.g., physical transmission medium, interference, message loss in relation to distance).

In this section, we present COBOLT, a tool for evaluating a specified TC algorithm rapidly using the network simulator SIMONSTRATOR. COBOLT stands for “Correct-by-construction development of topology control algorithms.” In the following, we discuss our reasons for choosing the network simulator SIMONSTRATOR and its important concepts and capabilities (Section 5.2.1). Afterwards, we describe the architecture behind COBOLT, an extension of SIMONSTRATOR that allows to integrate TC mechanisms generated with EMOFLON into existing network experiments (Section 5.2.2). In Section 5.2.3, we summarize implementation-related challenges that occurred during development of COBOLT. In Section 5.2.4, we illustrate the utility of our integration based on a simulative evaluation of the specification of the TC algorithm kTC.

RELATED PUBLICATIONS AND THESES We presented COBOLT in [119]. Complementary tool support for pure interpreted pattern matching in the SIMONSTRATOR simulator was developed in the Bachelor’s thesis by Lukas Neumann [168]. Dario Mirizzi designed and implemented a demonstrator for showcasing the effects of TC in a mobile video streaming scenario in his Bachelor’s thesis using SIMONSTRATOR [160]. His work influenced a PerCom demonstration paper by Stein et al. [237], which I also contributed to.

5.2.1 Selection of network simulator

The model transformation tool EMOFLON generates EMF-based JAVA code. Therefore, our first selection criterion for the network simulation environment was that it should integrate with TC algorithms implemented in JAVA. This criterion precludes several network simulators such as NS-3 or OMNET++, for which the TC algorithm needs to be implemented in C++. In 2015, when we started to develop COBOLT, we were aware of two mature JAVA-based simulation environments for wireless networks: PEERFACTSIM.KOM⁶ [242] and SIDNET-SWANS⁷ [75]. SIDNET-SWANS appeared to be discontinued already at that time, while PEERFACTSIM.KOM appeared to be supported and

⁶ PEERFACTSIM.KOM page: <http://peerfact.kom.e-technik.tu-darmstadt.de/> (visited: 2018-10-04)

⁷ SIDNET-SWANS website <http://www.ece.northwestern.edu/~ocg474/SIDnet.html> (visited: 2018-10-04)

extended actively⁸. In fact, PEERFACTSIM.KOM became part of the network simulation environment SIMONSTRATOR⁹ [208], which aimed at not only simulating network algorithms but also evaluating them on Android hardware. From now on, we refer to PEERFACTSIM.KOM as SIMONSTRATOR for simplicity.

Due to these observations and the prospected technical support during the development of COBOLT, we opted for SIMONSTRATOR as target platform for evaluating the TC mechanisms generated using EMOFLON.

5.2.2 Architecture

In the following, we summarize important SIMONSTRATOR concepts and, then, describe our design goals and decisions as well as the contributed components.

5.2.2.1 Structure of simulation setup in Simonstrator

Figure 5.3 illustrates a typical setup of a SIMONSTRATOR experiment. In SIMONSTRATOR, a device is represented by a *host*. The *host builder* is responsible for initializing all hosts based on an configuration file (XML). A host contains a number of *host components* (indicated with the \boxplus symbol). A host component possesses a lifecycle that is implemented using strategy methods for initialization and shutdown. Host components of a particular type are created by a *host component factory*. A *layer-specific host component* resides on a particular layer of the OSI reference model. The general structure of a host shown in Figure 5.3 resembles the organization of the OSI model shown in Figure 2.4. To ensure that a host component on a particular layer provides the necessary interfaces (e.g., end-to-end message transfer on the transport layer), SIMONSTRATOR requires components on all layers below the application layer to inherit from a corresponding *skeleton host component* (e.g., skeleton transport component). In Figure 5.3, the mnemonic of layer-specific components contains an additional character that indicates this component is a layer-specific component (e.g., \boxplus for transport-layer host components). All other host components are generic host components. SIMONSTRATOR was originally built for evaluating overlay protocols (i.e., above the transport layer) and provides implementations for widely used underlay protocols (i.e., physical, link, network, and transport layer). The host component for location and movement stores the current (usually two-dimensional) location as well as movement speed and direction of each host. This host component additionally decides how movement speed and direction evolve over time. Figure 5.3 lists examples for typical protocols on each layer.

⁸ PEERFACTSIM.KOM refs.: <http://peerfact.kom.e-technik.tu-darmstadt.de/de/publications/index.html> (visited: 2018-10-04)

⁹ SIMONSTRATOR page: <http://simonstrator.com> (visited: 2018-10-04)

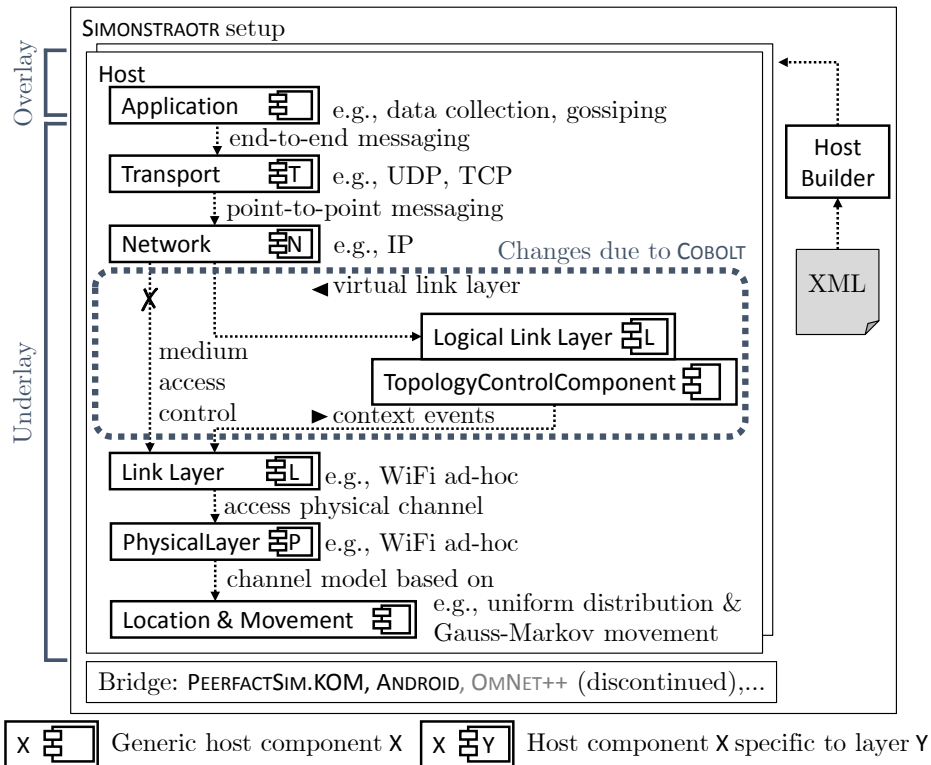


Figure 5.3: Structure of SIMONSTRATOR simulation setup with COBOLT-specific extensions

A particularity of SIMONSTRATOR is that the same application may be evaluated using the network simulators PEERFACTSIM.KOM [242] or OMNET++ [259] as well as on ANDROID¹⁰ devices. Unfortunately, the support for OMNET++ appears to be discontinued.

5.2.2.2 Design decisions

During the development of COBOLT, we took the following three major design decisions. First, an existing experimental setup should be reusable with minor changes. Therefore, we decided to introduce a new host-component type *TopologyControlComponent*. This component observes the configured *input (link) layer*, presents a *virtual (link) layer* to the network layer, manages the available TC algorithms and their context event handlers, and reconfigures the enabled TC algorithm.

Second, the developed components should be reusable (e.g., for arbitrary link layers or between other layers). Especially, the interfaces for monitoring the input topology and providing the virtual topology should not rely on the particularities of one network layer but rather use graph-based abstractions. To achieve the goal of reusability, we created

¹⁰ ANDROID page: <https://www.android.com/> (visited: 2018-10-22)

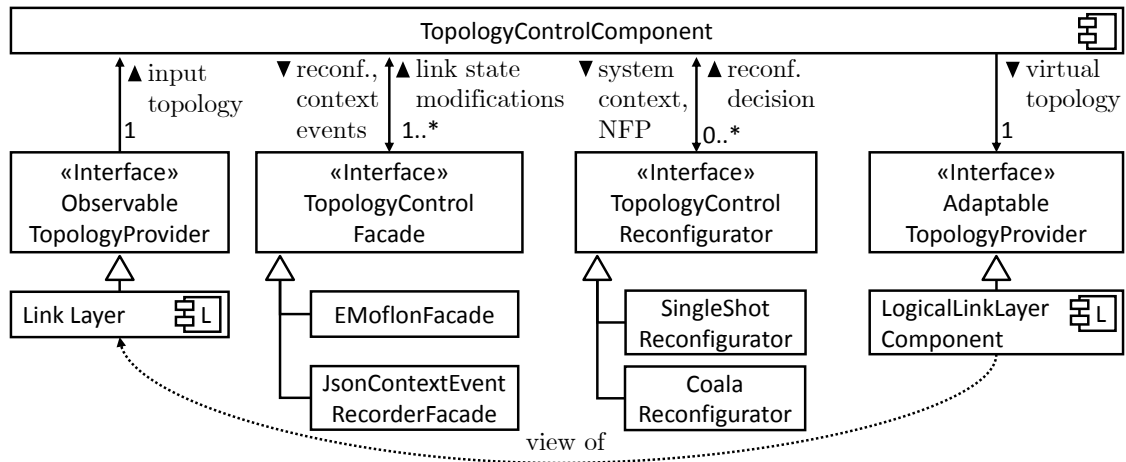


Figure 5.4: TopologyControlComponent with important interfaces

interfaces for all relevant interaction points between the `TopologyControlComponent` and other components.

Third, our extensions of `SIMONSTRATOR` should be as independent of `EMOFLON` as possible. This separation allows TC developers to use `COBOLT` for implementing TC algorithms manually and to generate TC algorithm code using different MDE tools. We achieved independence from `EMOFLON` by defining a generic interface for TC mechanism implementations.

5.2.2.3 Cobolt-specific components

We now discuss the internal architecture of the `TopologyControlComponent`, the interfaces and the implementations of each interface that correspond to our running example.

TOPOLOGY CONTROL COMPONENT Figure 5.4 shows the `TopologyControlComponent` along with the four major interfaces that we use to separate concerns regarding (i) the monitoring of the input topology, (ii) the invocation of the TC algorithm and context event handlers, (iii) the reconfiguration of the enabled TC algorithm, and (iv) the presentation of the virtual topology.

We now discuss each of these interfaces along with exemplary implementations in detail. The input topology of the `TopologyControlComponent` is retrieved via exactly one registered implementation of the `ObservableTopologyProvider` interface. The main functionality of this interface is to return a graph-based snapshot of the (surrounding) network. From the perspective of dynamic TC, it would be desirable to receive events about recent modifications of the network instead of snapshots. Unfortunately, these events are not provided by the link-layer components in `SIMONSTRATOR`. Therefore, we opted to derive the context events inside the `TopologyControlComponent` given a previous

state of the input topology and the current input topology. To extract input topologies from a link-layer host component, this host component needs to implement the interface `ObservableTopologyProvider`. The `ObservableTopologyProvider` interface and the required extensions of the link-layer host component were developed by Stein et al. during their work on [240].

The second major interface of `TopologyControlComponent` is the *TopologyControlFacade* interface. This interface hides one or more TC algorithms with their respective context event handlers. A `TopologyControlComponent` must have at least one registered `TopologyControlFacade`. From the perspective of `TopologyControlComponent`, subclasses of `TopologyControlFacade` allow to select an active TC algorithm, configure the parameters of an active TC algorithm, and receive context events. A `TopologyControlFacade` is responsible for tracking the current state of the TC mechanism topology that results from the context events that the `TopologyControlFacade` has received from the `TopologyControlComponent` so far and from the previous TC algorithm and context event handler invocations. The subclass `EMoflonFacade` of `TopologyControlFacade` maintains an EMF-based topology model and encapsulates one or more TC algorithms with their corresponding context event handlers specified using `EMOFLON`. Another use case of the `TopologyControlFacade` is to record context events for visualization purposes. This use case is covered by the class `JsonContextEventRecorderFacade`, which is not described in detail here.

The third major interface of `TopologyControlComponent` is the *TopologyControlReconfigurator* interface. A `TopologyControlComponent` can consult zero or more registered `TopologyControlReconfigurators` to determine when to adjust the parameters of the active TC algorithm (e.g., k of kTC) or to switch to another TC algorithm. These decisions are based on context information that is provided by the `TopologyControlComponent` (e.g., host movement speed, host density, communication pattern(s) of the application). The class `SingleShotReconfigurator` allows to configure a single reconfiguration decision. This class allows the user to observe the varying system performance when enabling or disabling TC during a simulation run. The name of the subclass `CoalaReconfigurator` alludes to the name of the reconfiguration engine `COALA` [271]. `COALA` uses a linear regression algorithm trained using supervised learning to map a given system context and non-functional property (NFP) to the TC algorithm configuration that promises the best performance w.r.t. the NFP. For details on `COALA`, we refer the reader to [271], where `COBOLT` has been used in a case study.

The fourth major interface of `TopologyControlComponent` is *AdaptableTopologyProvider*. Similar to `ObservableTopologyProvider`, this interface provides a graph-based view of a component. The `AdaptableTopologyProvider` interface extends `ObservableTopologyProvider` by means to add elements to and remove elements from the graph view. A `TopologyControlComponent` uses exactly one `AdaptableTopologyProvider` to present its virtual topology. After running the TC algorithm or a context event handler via the `TopologyControlFacade`, the collected link state modifications are propagated to the `AdaptableTopologyProvider`. As TC-specific subclass of `AdaptableTopologyProvider`, we implemented the

LogicalLinkLayer component, which is a link-layer host component and is aware of (i) the link layer component that provides the input topology and (ii) the link state modifications communicated by the *TopologyControlComponent*. Internally, the logical link layer component maintains a blacklist of inactive links. The logical link layer component was developed by Stein et al. in the context of [240].

EMOFLONFACADE Due to its central role in COBOLT, we now present the details of *EMoflonFacade*, the *EMOFLON*-specific implementation of *TopologyControlFacade*. Figure 5.5 shows that *EMoflonFacade* encapsulates two synchronized perspectives of the TC mechanism topology and a set of TC algorithms with their corresponding context event handlers. The first view builds on the *SimGraph* graph data structure, which is used for graph-based algorithms throughout *SIMONSTRATOR*. This view is visible to other host components for analysis and visualization purposes.

The second view is an EMF-based TC mechanism topology model, which conforms to the topology metamodel (see Figure 3.4). The topology metamodel also defines the interfaces of the TC algorithm and context event handler specifications. The control logic of *EMoflonFacade* ensures that incoming context events lead to updates of both TC mechanism topology views. For practical reasons, an *EMoflonFacade* can contain multiple TC algorithms with their corresponding context event handlers. Based on requests from the *TopologyControlComponent*, the *EMoflonFacade* selects, configures, and invokes the active TC algorithm. Therefore, a *EMoflonFacade* reflects all aspects of a TC multi-mechanism (see also Definition 2.33), apart from the following properties: (i) The reconfiguration logic resides outside the *EMoflonFacade* in subclasses of *TopologyControlReconfigurator*, and (ii) The TC multi-mechanism maintains only one topology for all TC mechanisms.

Currently, the single implemented strategy for reconfiguring TC mechanism parameters or switching from TC mechanism A to a TC mechanism B is to first unmark all links, then perform the parameter modification or reconfiguration to TC mechanism B, and, finally, run the reconfigured TC algorithm. More sophisticated transition strategies could transform a weakly consistent topology w.r.t. TC mechanism A into a weakly consistent topology w.r.t. TC mechanism B using less unmarking operations. In Section 4.5.2, we sketched how to handle parameter modifications of *kTC*.

VISUALIZATION SUPPORT We found it helpful to get a visual impression of the resulting virtual topology. *SIMONSTRATOR* provides components for visualizing the network topology and metric plots. The visualization of the current logical link layer component has been contributed by Michael Stein, originally. During our work on COBOLT, we extended the visualization component and build convenience methods for plotting metrics of a simulated WSN (e.g., latency, energy consumption, packet drop rate). Figure 5.6 provides an impression of how a running simulation can be inspected using a visualization of the virtual topology (left-hand side), the current simulation statistics (standard *SIMONSTRATOR* progress view), and metric plots (right-hand side). The topol-

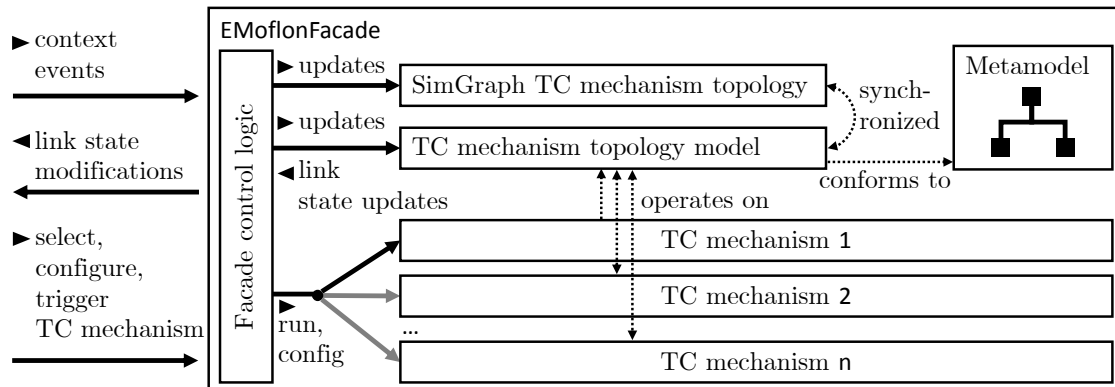


Figure 5.5: Architecture of EMoflonFacade

ogy visualization shows a WSN with 100 motes, distributed uniformly onto an area of $800\text{ m} \times 800\text{ m}$.

We experienced that the TC visualization component in SIMONSTRATOR was difficult to extend (e.g., with pan and zoom). Therefore, we created a novel visualization that presents the input and the output topology of a TC mechanism in one view, alongside with plots for metrics. This visualization is part of the COALAVIZ tool [176, 177] and implemented as a web user interface based on VAADIN¹¹. Figure 5.7 shows a screenshot of the COALA user interface. The left-hand side panel shows the current state of the WSN. The blue circles denote motes (e.g., n_{39}) and the single green circle (labeled with 40) in the center of the panel represents the base station. Active links are marked black (e.g., $l_{39\ 38}$) and inactive links are marked gray (e.g., $l_{39\ 30}$). The top panel on the right-hand side shows a plot of the average latency of all packets within the last three simulated minutes. The panel below the metric plot illustrates the current system state as feature diagram (see also Section 3.6). Rectangular boxes indicate either system features (e.g., possible TC algorithms) or context features (e.g., mote density, current application). Rounded rectangular boxes represent feature attributes (e.g., TC algorithm parameters, TC interval). Gray rectangular and rounded rectangular boxes denote the feature model instance representing the current system configuration. The panel below the feature diagram lists the available performance goals and their weights (e.g., end-to-end drop rate, end-to-end latency, Jain fairness of the energy consumption, and mean energy consumption over all motes).

¹¹ VAADIN page: <https://vaadin.com/> (visited: 2018-10-11)

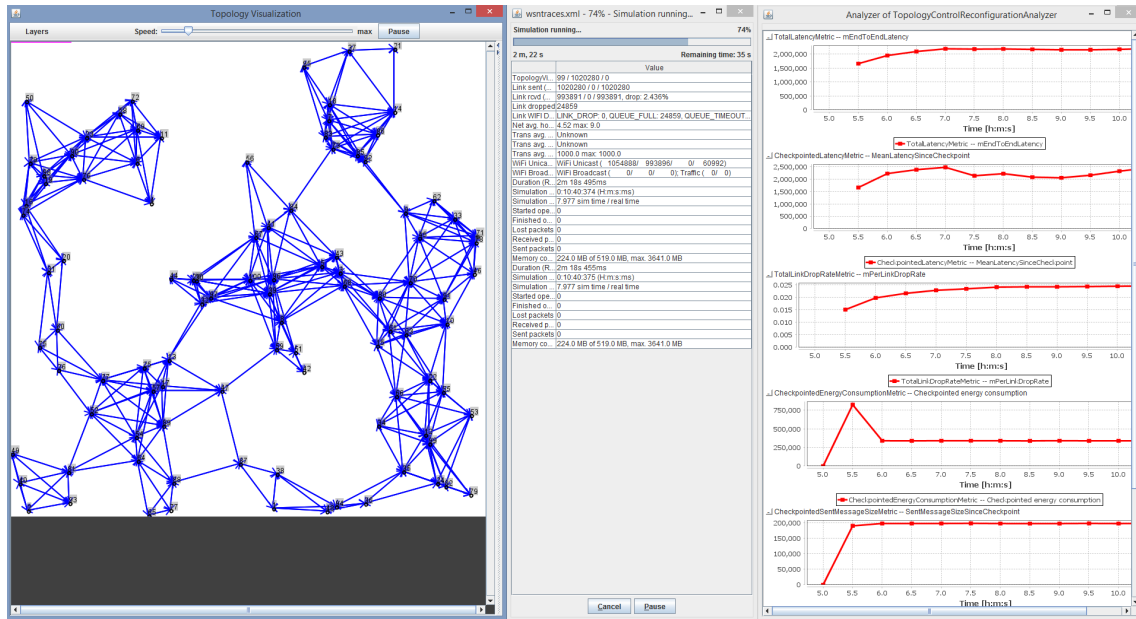


Figure 5.6: Topology visualization, simulation progress, and metric plots in SIMONSTRATOR

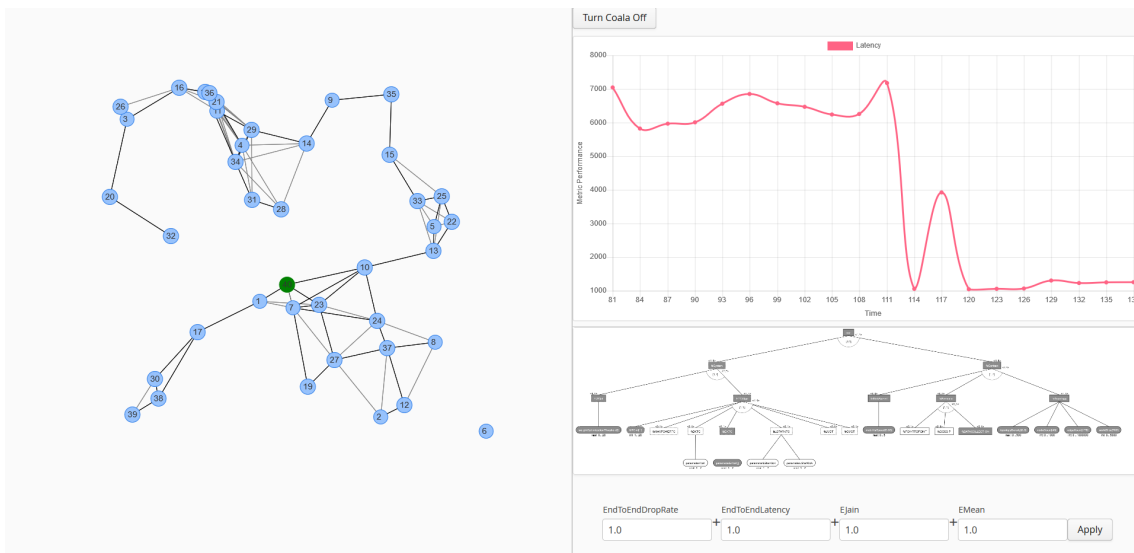


Figure 5.7: COALA visualization of network, metrics, and configuration (space)

5.2.3 Implementation aspects

In the following, we discuss the technical aspects of integrating EMOFLON and SIMONSTRATOR as well as the specification of GT rules with multiple application conditions in EMOFLON.

5.2.3.1 Integration of eMoflon and Simonstrator

We implemented COBOLT for SIMONSTRATOR 2. For a given specification, EMOFLON produces one or more ECLIPSE plugins¹² that contain the generated code. In contrast, SIMONSTRATOR builds on APACHE MAVEN¹³. Both frameworks are not compatible natively. The code generated from the TC algorithm and context event handler specifications could be transferred into MAVEN artifacts, but this would slow the specify-generate-test development cycle considerably. To bridge the gap between ECLIPSE bundles generated by EMOFLON and the MAVEN-based infrastructure of SIMONSTRATOR, we use the ECLIPSE TYCHO framework¹⁴.

5.2.3.2 Specifying multiple application conditions in eMoflon

Another implementation aspect is that the specification frontend of EMOFLON supports only one NAC per GT rule and no PACs. To specify the GT rules for our running example using EMOFLON, we adapted the specification according to the rules described in the following.

If a GT rule has one NAC, this NAC can be specified directly in EMOFLON. If a GT rule has one PAC with a single conclusion pattern, we can replace the LHS with the conclusion pattern because the premise and the conclusion pattern both extend the original LHS pattern. If a GT rule R_X has more than one positive or negative application condition, respectively, or a PAC with more than one conclusion pattern, we transform a story node containing an application of R_X (see Figure 5.8a) into a larger SDM fragment (see Figure 5.8d).

We discuss the transformation for a sample GT rule R_X that has one negative application condition $NAC_{X,1}$ and one positive application condition $PAC_{X,2}$ with two conclusion patterns $c_{X,2,1}$ and $c_{X,2,2}$ (Figure 5.8b). Figure 5.8a shows an original SDM fragment that consists of an application of R_X with a number of input parameters and output parameters (indicated by text in normal font enclosed by square brackets, e.g., *[Input parameters]*). Figure 5.8d shows the resulting SDM fragment that can be specified in EMOFLON. The resulting SDM fragment in Figure 5.8d possesses the same incoming activity edges, an outgoing activity edge labeled with [End] (explanation follows), and

¹² ECLIPSE PDE page: <https://www.eclipse.org/pde/> (visited: 2018-10-10)

¹³ MAVEN page: <https://maven.apache.org/> (visited: 2018-10-10)

¹⁴ TYCHO page: <https://www.eclipse.org/tycho/> (visited: 2018-10-10)

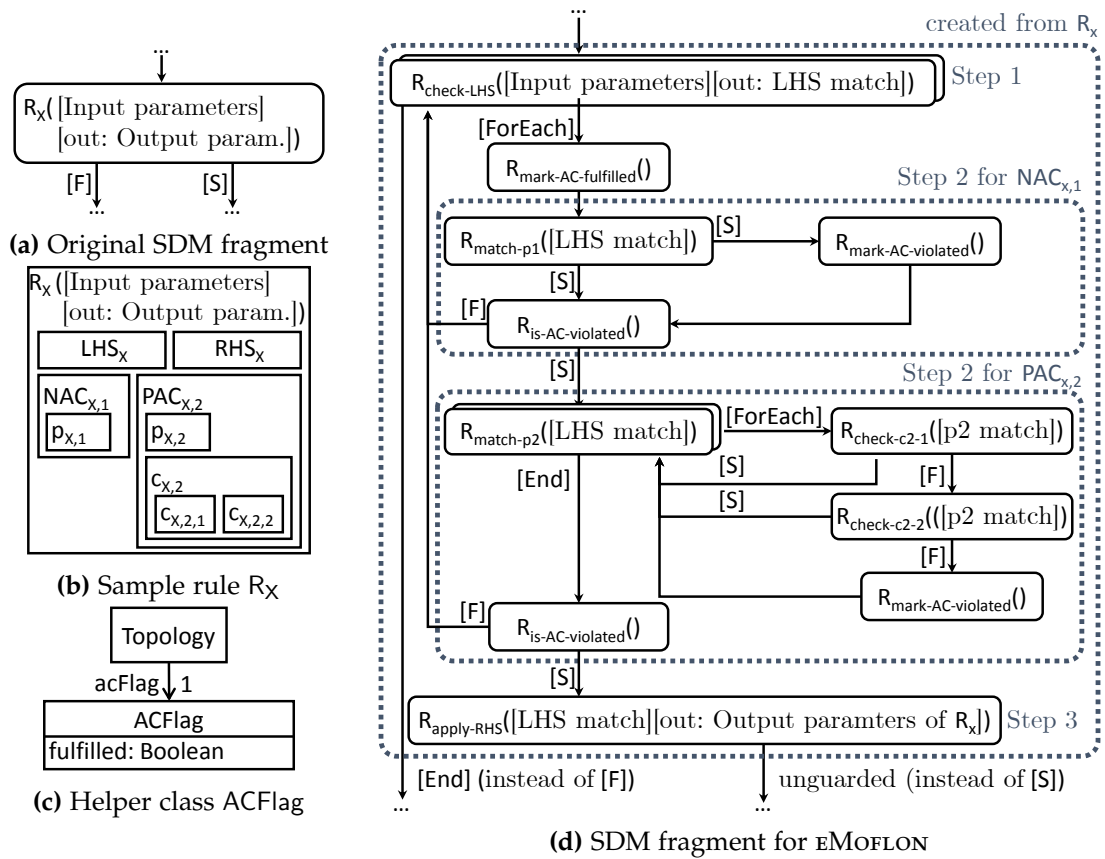


Figure 5.8: Sketch of transformation of application conditions into eMOFLON

one [S]-edge. For specifying arbitrary many application conditions of a GT rule using EMOFLON, we split the application of R_X into the following three steps.

STEP 1: FIND LHS MATCH In the first step, we iterate over all matches of the LHS pattern LHS_X of R_X (see $R_{\text{check-LHS}}$ in Figure 5.8d). This rule application is the target of the original incoming activity edge of R_X , receives the original input parameters of R_X as input, and returns a match of LHS_X (represented using multiple rule output parameters, *[out: LHS match]*). In the subsequent steps, we evaluate for each match of LHS_X whether it fulfills all application conditions, and apply R_X to the first such match of LHS_X .

To collect all matches of LHS_X , we employ a foreach story node. A *foreach node* is a special type of story node in EMOFLON that collects and caches *all* matches of the LHS pattern of the contained rule when the execution arrives at the foreach node. In figures, a foreach node is indicated by two stacked rounded rectangles. For each cached match, the execution continues along the *[ForEach]-edge*. Only after each cached match has been processed, the execution continues along the *[End]-edge*. When the execution continues along the *[End]-edge*, the cache that corresponds to the foreach node is cleared.

STEP 2: CHECK APPLICATION CONDITIONS In the second step, we check whether all application conditions are fulfilled for a given match of LHS_X . We transform each application condition into an SDM fragment (nested dashed rounded rectangles in Figure 5.8d) that has an outgoing [S]-edge to indicate that the application condition is fulfilled and an outgoing [F]-edge to indicate that the application condition is violated. An outgoing [F]-edge returns to the outermost foreach node, which iterates over all matches of LHS_X . An outgoing [S]-edge leads either to the SDM fragment for the next application condition to check (e.g., after the SDM fragment of $NAC_{X,1}$) or, if all application conditions have been checked, to the actual application of R_X (e.g., after the SDM fragment of $PAC_{X,2}$).

To indicate whether an application is fulfilled or not, we introduce the helper class ACFlag into the metamodel (Figure 5.8c), which is attached to Topology for technical reasons. As for Topology, we assume that a single instance of ACFlag exists and is globally accessible via the global Topology object to simplify the notation. The ACFlag class possesses a Boolean attribute, that is set to *true* if the application conditions are fulfilled tentatively. Before starting to check the application conditions for the current LHS match, the AC flag is set to *true* (see $R_{\text{mark-AC-fulfilled}}$). At the end of each SDM fragment that corresponds to the check of an application condition, the application of $R_{\text{is-AC-violated}}$ checks for a detected violation of the application condition.

To check a negative application condition, we try to find a match of the premise of the application condition (e.g., $R_{\text{match-p1}}$ for $p_{X,1}$). If such a match exists, we know that this application condition is violated and set the AC flag to *false* (via $R_{\text{mark-AC-violated}}$).

To check a positive application condition, we need a larger SDM fragment. For each match of the premise (e.g., *[p2 match]* in Figure 5.8d), we check whether the premise

match can be extended to a match of at least one conclusion pattern. If no such extension exists, the AC flag is set to *false* and the execution returns to the outermost foreach node. For iterating over all extensions of the current LHS_X match to a match of $p_{X,2}$, we also use a foreach story node.

STEP 3: APPLY RHS The third step is the actual application of R_X . The application of the GT rule $R_{\text{apply-RHS}}$ receives the match of LHS_X that fulfills all application conditions as input parameter (i.e., *[LHS match]*) and binds the output parameters according to the original rule R_X (denoted by *[out: Output parameters of R_X]*). The LHS and RHS patterns of $R_{\text{apply-RHS}}$ are identical to LHS_X and RHS_X , respectively, and, therefore, $R_{\text{apply-RHS}}$ is always applicable. For this reason, the single outgoing activity edge is unguarded.

5.2.3.3 Consistency checking

COBOLT allows to check that local and global consistency properties hold after each invocation of the TC algorithm and context event handler specification. These consistency checks serve as assertions that the generated code is correct w.r.t. the specification. COBOLT already provides a number of consistency checks (e.g., for AU- and A-connectivity). For novel TC algorithms the developer may provide her own consistency checks. Due to the fact that evaluating consistency checks after each TC algorithm and context event handler invocation increases the execution time of experiments, the user may also decide to disable consistency checks entirely.

5.2.3.4 Code and tool availability

COBOLT is open source and available on GITHUB¹⁵.

5.2.4 Evaluation of Cobolt

In this section, we present an evaluation of COBOLT. In Section 5.2.4.1, we state the research questions according to the goals of this thesis. In Section 5.2.4.2, we describe the evaluation setup, and in Sections 5.2.4.3 to 5.2.4.5, we present and discuss the results of this evaluation.

5.2.4.1 Research questions

We conduct our evaluation based on the kTC algorithm to answer a number of research questions regarding the correctness and applicability of COBOLT and the efficiency of dynamic TC in comparison to batch TC. Due to a lack of published and open source independent manual implementations of TC algorithms in SIMONSTRATOR, we cannot

¹⁵ COBOLT repository: <https://github.com/eMoflon/cobolt> (visited: 2018-10-11)

evaluate, for instance, the efficiency of the generated kTC algorithm in comparison to a manually implemented and tuned variant of kTC.

RQ1-CORRECTNESS A major goal of this thesis is that the TC mechanism specification *and* the resulting generated evaluation artifacts are correct (see Goal 1 in Section 1.2). Therefore, RQ1 relates to the correctness of COBOLT. To this end, we evaluate the following subquestions qualitatively and quantitatively.

- How is the correctness of the generated TC algorithms ensured?
- How is the correctness of the generated code for kTC ensured?

RQ2-EFFICIENCY COBOLT shall be applicable in practice (Goal 3 in Section 1.2). We want to identify whether a rapid development process is possible using COBOLT and how the execution time of the generated TC algorithms contributes to the execution time of the entire simulation study. To this end, we evaluate the following subquestions quantitatively.

- How does the computational effort of a generated TC algorithm relate to the simulation duration?
- Is this computational effort reasonable?

RQ3-DYNAMIC TC In the final part of this evaluation, we investigate in which scenarios dynamic TC or batch TC are more efficient (Goal 2 in Section 1.2). We investigate the following subquestions quantitatively.

- In which experiments is dynamic TC more efficient than batch TC?
- For which classes of scenarios is dynamic TC more efficient than batch TC?

As classes, we consider dense and sparse topologies, large and small topologies (in terms of mote count), and stationary and mobile WSNs.

5.2.4.2 Evaluation setup

All experiments were performed on a 64-bit workstation with an Intel i7-2600 CPU¹⁶ (2 cores, 3.4 GHz) and 8 GB of RAM. We reserved a maximum of 2 GB of RAM for the JAVA Virtual Machine that ran the simulations. The operating system was Windows 7 Professional (64 bit). We use the JAVA JDK 1.8 Update 171 (64 bit), ECLIPSE IDE 2018-09, EMOFLON 3.5.1, SIMONSTRATOR 2.5.0 (modified), and COBOLT 1.0, ENTERPRISE ARCHITECT 11.

In several parts of this evaluation, we use data from a collection of 240 experiments that we conducted. In the following, we describe how we designed these experiments. One of our goals was to cover a large range of application scenarios of WSNs. From the numerous possible dimensions that determine each application scenario, we chose

¹⁶ Intel Core i7-2600 specification: <https://ark.intel.com/products/52213/Intel-Core-i7-2600-Processor-8M-Cache-up-to-3-80-GHz-> (visited: 2018-11-19)

Table 5.2: Summary of configuration parameters for simulation experiments

Category	Parameter	Value
Simulation	Duration	1 h of simulated time
	Seed count	10
	World size	{300 m, 600 m, 900 m}
	Node count	{100, 250}
	Movement	Gauss–Markov [31] ($\alpha=0.2$, $v \in \{0 \frac{\text{m}}{\text{s}}, 1.4 \frac{\text{m}}{\text{s}}\}$)
	Placement (initial)	uniform at random
	Energy (initial)	uniform at random in [30 %, 100 %] of 1300J
Underlay	Transmission radius	130 m
	Link layer	IEEE 802.11 Ad-Hoc
	TC algorithm	kTC with $k = 1.41$
	Incrementality	{batch, dynamic}
	Link weight	Euclidean distance
	TC interval	1 min
	Routing protocol	global-knowledge routing
	Transport protocol	UDP
Overlay	Application	data collection application (many-to-one)
	Message size	1 kB
	Transmission interval	uniform at random in 100 ms to 200 ms

the following variable dimensions: mote count (few vs. many), network extent (small area vs. large area), mobility (stationary vs. mobile motes). To assess the effect of batch TC in comparison to dynamic TC, we introduce the fourth dimension of incrementality (i.e., batch vs. dynamic). To keep the extent of this evaluation reasonable, we refrained from varying the following possible additional dimensions: physical channel model, link layer, routing protocol, transport protocol, communication pattern (e.g., many-to-many).

Table 5.2 summarizes the fixed and variable parameters of the simulation experiments that we conducted. Variable parameters (i.e., mote count, world size, mobility speed, and TC incrementality mode) are shown in set notation. The choices of the fixed parameter values and variable parameter ranges are inspired by the following works from the TC literature: [34, 72, 109, 130, 142, 143, 211, 227, 238, 270].

In the following, we explain the chosen parameters briefly. We terminated each simulation run after 1 h of simulation time. A simulation starts with a set of either 100 or 250 motes. These motes are distributed uniformly at random onto a square area with a

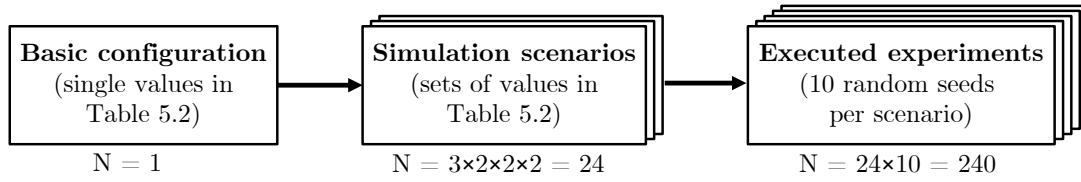


Figure 5.9: Simulation setup with scenario-specific parameter values and random seeds

fixed side length (either 300 m, 600 m, or 900 m). The base station is located in the center of the simulation area.

As movement model, we chose a Gauss–Markov model [31], in which motes can move freely within the boundaries of the simulated area. The movement speed is Gaussian and the movement direction is Markovian (see [31] for details). We set the movement speed in one half of the experiments to $0 \frac{\text{m}}{\text{s}}$ (i.e., the WSN is stationary) and in the other half to $1.4 \frac{\text{m}}{\text{s}}$, which corresponds to a typical walking speed [137].

For simulating the energy consumption, we chose a state-based energy model. A mote can be in one of four states (i.e., sending, receiving, idle, off). In each state, it consumes a fixed power. The initial energy of each mote is chosen uniformly at random within 30% to 100% of 1300 J. The maximum energy of 1300 J corresponds to the typical capacity of a single AAA battery. The base station has a static power supply in our experiments (i.e., its energy level is infinite).

In the following, we describe the components of the network stack. On the physical and link layer, we chose the IEEE 802.11 Ad-Hoc protocol [278] because it was readily available in SIMONSTRATOR and has been used in other works (e.g., [239]). The weights of the links in the underlay are equal to their Euclidean distance. On the network layer, we chose a routing algorithm that uses global knowledge because investigating the interactions of TC and a realistic routing algorithm (e.g., RPL [276]) were not in the focus of this evaluation. On the transport layer, we selected the packet-oriented User Datagram Protocol (UDP) protocol [189]. This protocol is typically used for small data units.

On the application layer, we used a data collection application. Each mote regularly transmits a message with 1 kB of payload to the base station. We chose 1 kB to avoid fragmentation on the transport layer. To reduce the number of collisions, the waiting time between transmission is chosen uniformly at random in the range from 100 ms to 200 ms.

As TC algorithm, we chose kTC with $k = 1.41$. We chose k according to the experiments in [227]. kTC is executed every 1 min. In half of the experiments, we executed kTC in batch and dynamic mode, respectively.

The chosen battery model and application lead to the depletion of a small number of motes (i.e., 1 to 13 motes) during 138 of the 240 experiments. Together with the movement model that was active in 120 of the 240 experiments, all five types of context

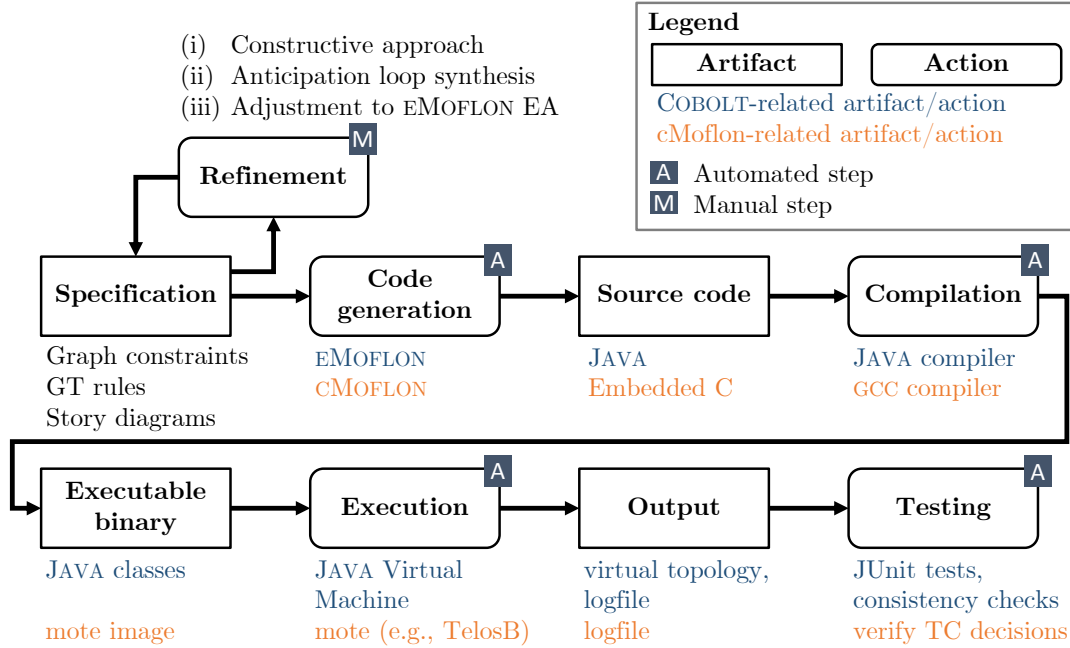


Figure 5.10: Steps in correct-by-construction methodology for discussion of correctness (jointly for COBOLT and cMOFLON, rectangle: artifact, rounded rectangle: process step)

events were covered (i.e., node addition, node removal, link addition, link removal, and link weight change).

To reduce the effects of randomness (e.g., due to the initial mote position distribution), we repeated each scenario 10 times with different random seeds. SIMONSTRATOR provides a factory for pseudo-random numbers per host component that ensures that, for a fixed seed, the sequence of random numbers is predetermined. This handling of random numbers makes our results reproducible (except for the CPU time measurements, which depend on the hardware).

In total, we obtain $n_{\text{mote}} \times n_{\text{world size}} \times n_{\text{speed}} \times n_{\text{TC mode}} = 24$ simulation scenarios. As illustrated in Figure 5.9, using 10 seeds per scenario, we obtain 240 experiments.

5.2.4.3 Results and discussion for RQ1 (correctness)

In the following, we investigate how we ascertained correctness of COBOLT by investigating possible faults during each step in our methodology, starting from the specification and ending with the execution of the TC algorithm.

Figure 5.10 depicts the artifacts (rectangles) and steps (rounded rectangles) in our methodology. This figure serves as orientation for the discussion of correctness of COBOLT and cMOFLON, as indicated by the captions below the artifacts and process steps. The major steps in our methodology are the refinement of the GT rules and the

story diagrams based on the local consistency constraints (as described in Chapter 4), the generation of platform-specific code from this specification, the compilation of the source code to an executable binary, and the execution of the binary, which results in the (simulation) output. In this evaluation, we consider the correctness both on the specification level and of the resulting source code and executable binary.

Conceptually, the specification refinement results in a correct TC mechanism specification. Still, different types of faults may occur during the practical application of the constructive approach and the subsequent code generation. First, due to the lacking tool support, the specification refinement needs to be carried out manually. Second, the code generator (i.e., `EMOFLON`) may contain faults that lead to generated platform-specific source code that fails to conform to the specification. Third, the compiler that translates the platform-specific code into an executable binary file (here, the `JAVA` compiler) may lead to an executable representation that does not conform to the language specification of the source-code language. Fourth, the execution environment (here, the `JAVA VIRTUAL MACHINE`) may execute the binary file in an erroneous manner.

Given that the chosen target language `JAVA` has been used in practice for more than 20 years now, we rely on its proper behavior in the following. Therefore, we will focus on the specification refinement and code generation step in the following because these steps are either carried out manually or performed automatically by a tool that is not as mature as the `JAVA` compiler or `JAVA VIRTUAL MACHINE`.

SPECIFICATION REFINEMENT The manual refinement of the TC mechanism specification based on the constructive approach and the anticipation loop synthesis algorithm is an elaborate task. Additionally, due to the lacking support for multiple application conditions per GT rule in `EMOFLON`, we need to adjust the specification a third time, as discussed in Section 5.2.3.2. We systematized the three involved refinement steps as far as possible by providing exact algorithms for synthesizing application conditions (Algorithm 4.1) and anticipation loops (Algorithm 4.3). The result of the specification refinement cannot be tested in isolation.

Nevertheless, the involved graph constraints can be used to derive potential consistency-violating situations. For example, by inspecting the premise pattern of a negative constraint, we can determine possible modifications of the topology that could lead to a match of the premise and, therewith, a violation of the constraint (e.g., adding a link that completes a φ_{kTC} -fulfilling triangle). The identified sample sequences of topology modifications can be translated into unit tests.

QUANTITATIVE DISCUSSION FOR `kTC` During the specification of `kTC`, we performed three major test runs, which we describe in the following.

In the first test run, we defined 24 unit tests for `kTC` using the aforementioned strategy of investigating a subset of the potential violations of the `kTC`-specific graph constraints. These unit tests address the execution of `kTC` in batch and dynamic mode. We manu-

ally derived these test cases to cover possible sequences of topology modifications that could lead to a consistency violation. To assess the effectiveness of our test, we measured the instruction and branch coverage for the classes that implement kTC achieved by the 24 unit tests. For this purpose, we used the tool ECLEMMA¹⁷ (version 1.3.1). The implementation of kTC is split into two classes. The abstract class `TriangleBasedTopologyControlAlgorithm` contains, for instance, the control flow for invoking the TC algorithm and the context event handlers within a TC mechanism. The concrete subclass `PlainKTC` contains the actual implementation of the kTC-specific patterns. For the abstract class `TriangleBasedTopologyControlAlgorithm`, the test suite achieved 71.2 % instruction and 62.4 % branch coverage (1427 of 2005 instructions). For the kTC-specific class `PlainKTC`, the test suite achieved 90.9 % instruction and 66.2 % branch coverage (438 of 482 instructions). This sums up to a total instruction coverage of 75.0 % (1865 of 2487 instructions).

In the second test run, we used the capability of COBOLT to check the weak and strong consistency constraints after each execution of a context event handler and the TC algorithm, respectively (see also Section 5.2.3.3). During the 240 experiments that we conducted 288 000 of these checks were performed and no consistency violation was detected. The number of consistency checks corresponds to the 600 invocations of the TC algorithm and the context event handling, respectively (i.e., $600 \times 2 \times 240 = 288\,000$).

In the third test run, we created a dedicated simulation setup to measure the instruction and branch coverage of the generated code for kTC. This simulation setup consists of the base configuration shown in Table 5.2 with 250 motes, 900 m world size, a mobility speed of $1.4 \frac{\text{m}}{\text{s}}$ (to increase the number of context events), and kTC running in dynamic mode. In contrast to Table 5.2, we executed the simulation until all 250 motes had run out of energy (after 98 min of simulated time). For the abstract class `TriangleBasedTopologyControlAlgorithm`, the test suite achieved 77.9 % instruction and 67.4 % branch coverage (1561 of 2005 instructions). For the kTC-specific class `PlainKTC`, the test suite achieved 72.4 % instruction and 54.8 % branch coverage (349 of 482 instructions). This sums up to a total instruction coverage of 76.8 % (1910 of 2487 instructions).

The merged coverage results of the 24 unit tests and the simulation run show the following coverage values. For the abstract class `TriangleBasedTopologyControlAlgorithm`, the 78.0 % instruction and 68.5 % branch coverage could be achieved (1564 of 2005 instructions). For the kTC-specific class `PlainKTC`, 90.9 % instruction and 67.7 % branch coverage could be achieved (438 of 482 instructions). This amounts to a total instruction coverage of 80.5 % (2002 of 2487 instructions).

Remarkably, the simulation run led to lower instruction and branch coverage values for `PlainKTC`, but to larger coverage values for `TriangleBasedTopologyControlAlgorithm`. In total, the joint testing of the generated code of kTC based on the `JUNIT` test suite and the simulation run increased the instruction and code coverage compared to the individual results. These results indicate that kTC is well tested.

¹⁷ ECLEMMA page: <https://www.eclemma.org/> (visited: 2018-10-31)

QUANTITATIVE DISCUSSION FOR BUILD PROCESS COBOLT relies on the unmodified build process of EMOFLON. Therefore, we investigated to which degree the code generator of EMOFLON is tested. EMOFLON has been under development since 2012. Since then, a public tests repository¹⁸ has been built up. Apparently, EMOFLON is proven in use. Version 3.5.1 of EMOFLON has been tested against a public test suite consisting of 179 projects and a total number of 372 JUNIT¹⁹ unit tests. These tests consist of small to medium-sized specifications resulting from research projects and student theses. To quantify in how far the test suite certifies that the EMOFLON build process is well-tested, we measured the instruction coverage of the build process.

As experimental setup, we created a fresh EMOFLON developer workspace. This workspace contains the source code of all 64 projects that constitute the code base of EMOFLON ECLIPSE. Afterwards, we started a runtime ECLIPSE workspace²⁰. In this runtime workspace, we triggered the code generation for the 179 projects that constitute the aforementioned test suite. The used test suite can be found on GITHUB²¹. We assess the quality of the test suite by measuring the instruction coverage of the relevant projects. We determine a set of 20 EMOFLON projects that are relevant for the SDM code generation process. The names of the relevant projects are shown in the first column of Table 5.3. As before, we used ECLEMMA to measure the code coverage in the developer workspace.

The instruction coverage of all EMOFLON projects is 28.2% (375422 of 1329766 instructions). For the 20 projects that are relevant for the SDM build process, we obtain an instruction coverage of 57.6% (116469 of 202077 instructions). Table 5.3 summarizes the coverage per project. The coverage ranges from 2.2% to 93.3%. As a result of this observation, we first performed a manual inspection of the projects with the lowest coverage.

First, the reason for the small coverage of 2.2% for the project *org.moflon.core.moca-processing* originates from the fact that this project contains a large amount of generated EMF code that serves for processing XMI data structures. We think that the low coverage is not critical here.

Second, the coverage for project *org.moflon.sdm.compiler.eclipse* was 30.2%. We found out that 1217 of 2770 instructions serve to conduct a preprocessing of the code-generating class. This task is not meant to be executed by an EMOFLON instance. Furthermore, 95 instructions were dead code. Therefore, only 1401 instructions were relevant for the EMOFLON build process. The coverage for this part of the project was 59.6% (836 of 1401 instructions).

¹⁸ EMOFLON test suite: <https://github.com/eMoflon/emoflon-tests> (visited: 2018-10-12)

¹⁹ JUNIT page: <https://junit.org/> (visited: 2018-10-12)

²⁰ For details on ECLIPSE runtime workspaces, see: https://help.eclipse.org/oxygen/topic/org.eclipse.pde.doc.user/guide/tools/launchers/eclipse_application_launcher.htm (visited: 2018-11-19)

²¹ ECLIPSE Team Project Set file on GITHUB: https://github.com/eMoflon/emoflon-tool/blob/emoflon-tie_3.5.1/org.moflon.ide.workspaceinstaller.psf/resources/psf/tests/AllTests.psf (visited: 2018-11-19)

Table 5.3: Coverage for SDM-related EMOFLOn projects (Project name: o.m.s = org.moflon.sdm, $N_{ins}^{cov}/N_{ins}^{total}$: number of covered/total instructions)

Project name	Coverage	N_{ins}^{cov}	N_{ins}^{total}
org.moflon.core.moca.processing	2.2 %	210	9723
o.m.s.compiler.eclipse	30.2 %	836	277
MocaTree	37.0 %	1411	3816
o.m.s.constraints.democles	47.8 %	379	793
org.moflon.core.mocatoflon	52.6 %	24 586	46 764
o.m.s.runtime.democles	53.6 %	5433	10 131
o.m.s.constraints.operationspecification	53.7 %	2062	3838
o.m.s.compiler.democles.validation.controlflow	54.4 %	16 527	30 387
o.m.s.compiler.democles.validation.result	54.6 %	930	1703
o.m.s.democles.literalexpressionsolver	56.4 %	3621	6419
org.moflon.core.dfs	56.5 %	2429	4301
SDMLanguage	60.1 %	13 953	23 224
o.m.s.constraints.constraintstodemocles	61.0 %	384	6296
o.m.s.constraints.scopevalidation	65.4 %	1301	1989
o.m.s.compiler.democles	67.4 %	4771	7074
org.moflon.ide.core	67.6 %	4819	713
o.m.s.compiler.democles.validation.scope	67.9 %	19 719	29 027
o.m.s.compiler.democles.pattern	77.1 %	11 622	15 078
o.m.s.controlflow.reversenavigation	92.4 %	375	406
o.m.s.constraints.codegenerator	93.3 %	1101	118
Summary	57.6 %	116 469	202 077

Third, the coverage values of 37.0% and 47.8% for *MocaTree* and *org.moflon.sdm.constraints.democles* originate from the fact that both projects contain only generated data structures for representing XML trees and constraints of patterns. The same argument applies to the 53.6% coverage of *org.moflon.sdm.runtime.democles*, which represents the control flow metamodel.

From a coarse-grained inspection of the remaining relevant projects, we conclude that it is difficult to achieve a high coverage of code generated from EMF metamodels because this code contains, for instance, numerous methods that provide reflection capabilities. Furthermore, thanks to our experiments, we were able to identify a considerable amount of code that is not used within EMOFLON. This code is, therefore, uncritical for the build process but contributes to the total number of noncovered instructions.

To gather a deeper understanding of how the coverage is distributed, we conducted an in-depth analysis of the project *org.moflon.sdm.compiler.democles.validation.scope*. This project is responsible for transforming story diagrams into control flow models and consists mainly of generated EMF code. The actual transformation is implemented as story diagrams as well because EMOFLON has been bootstrapped [135]. ECLEMMA reported an instruction coverage of 67.9% (19 719 of 29 027 instructions). The majority of the 569 generated pattern methods have a coverage between 88% and 100% (488 methods, ca. 85.7%). Further 45 pattern methods (ca. 7.9%) have a coverage of 0%. This can be explained by the fact that these methods mainly serve for error reporting, which is not tested by the EMOFLON test suite. From our analysis, we conclude, that 6 entirely untested methods (i.e., with 0% coverage) exist that should be covered by further tests.

To sum up this in-depth analysis of the coverage of one of the core project of EMOFLON, we can conclude that the test coverage for the relevant transformation methods is already at a good level with more than 85% of the pattern methods of the transformation having a coverage of more than 88%.

To summarize the discussion of the build process of EMOFLON, our analyses reveal that the coverage is already decent in general. We found that dead code and code of preprocessing components lead to a falsely reduced code coverage.

We answer RQ1 as follows. The specification refinement steps of our methodology are correct by construction, as shown by proofs in the related work [44, 91] (regarding the constructive approach) or in this thesis (regarding the anticipation loop synthesis, see Section 4.4.3). In contrast, the correctness of neither the involved code generator (i.e., EMOFLON) nor the execution environment (i.e., the JAVA VIRTUAL MACHINE) can be proved formally. Regarding the JAVA VIRTUAL MACHINE, we think that it is proven in use. Regarding EMOFLON, it is only possible to certify that the generated code for concrete specifications returns correct results for concrete input models. This strategy is used to test (i) each EMOFLON release based on 372 unit tests in 179 realistic test projects, and (ii) the kTC specification for our experiments. To determine whether these tests are effective, we analyzed in how far (i) the first set of tests achieves a reasonable coverage of the build process components in EMOFLON, and (ii) the second set of tests

achieves a reasonable coverage of the generated code of kTC. For eMOFLON, we found that the instruction coverage is decent in general (57.6%). For kTC, we achieved a good instruction coverage of 80.5%. These results are reasonable because the eMOFLON and the generated code of kTC contain boilerplate EMF code (e.g., to provide reflection on the metamodel level), which is unnecessary in our scenario. Additionally, some of the build process components of eMOFLON contain either dead code or code that is not relevant for the actual SDM build process but contributes to the fraction of noncovered instructions (e.g., developer code or irrelevant utility classes).

THREATS TO VALIDITY A major threat to construct validity of this part of the evaluation is that we could not evaluate the code coverage of (i) the DEMOCLES code generation engine, which is integrated as binary library in eMOFLON, and (ii) the ENTERPRISE ARCHITECT add-in, which is implemented in C# (instead of JAVA). Measuring the code coverage of both components is feasible technically but required excessive manual effort, which is beyond the scope of this evaluation. Furthermore, this threat is mitigated by the fact that the generated code in the runtime workspace is exercised by 372 unit tests. These unit tests constitute integration tests for the entire build process.

5.2.4.4 Results and discussion for RQ2 (efficiency)

In the following, we investigate the applicability of COBOLT in the context of realistic simulation setups. We use the experimental setup described in Section 5.2.4.2.

METRICS We assess the computational cost of using COBOLT in terms of the CPU time for each invocation of the TC algorithm and the context event handlers and compare these values with the CPU time required for the entire simulation.

RESULTS Figure 5.11 compares the CPU time for the TC algorithm execution, the context event handling, and the entire simulation in all 240 experiments as three boxplots. The boxplots follow the convention that (i) the horizontal line indicates the median value, (ii) the lower and upper caps mark the 25%- and 75%-percentile (i.e., the first and third quartile), (iii) the whiskers enclose all values within the 1.5-inter-quartile range, and (iv) outliers are shown as circles. For a given ordered sequence of values $x_1 < x_2 < \dots < x_N$, the p %-percentile is $x_{\lfloor N \cdot p + 1 \rfloor}$ if N uneven and the mean value of $x_{N \cdot p}$ and $x_{N \cdot p + 1}$ if N is even. The *inter-quartile range* (IQR) is equal to $x_{N \cdot 75\%} - x_{N \cdot 25\%}$. If the range of values does not exceed the 1.5-IQR interval, the upper and lower ends of the whiskers reflect minimum and maximum data points. The lower whisker marks the minimum data point that at least as large as $x_{N \cdot 25\%} - \text{IQR}$. Conversely, the upper whisker marks the largest data point that is at most as large as $x_{N \cdot 75\%} + \text{IQR}$.

In Figure 5.11, each boxplot is labeled with the median value. No outliers were observed. For each experiment, we calculated the sum over the execution time of the repeated execution of the TC algorithm and the context event handlers. The plot shows

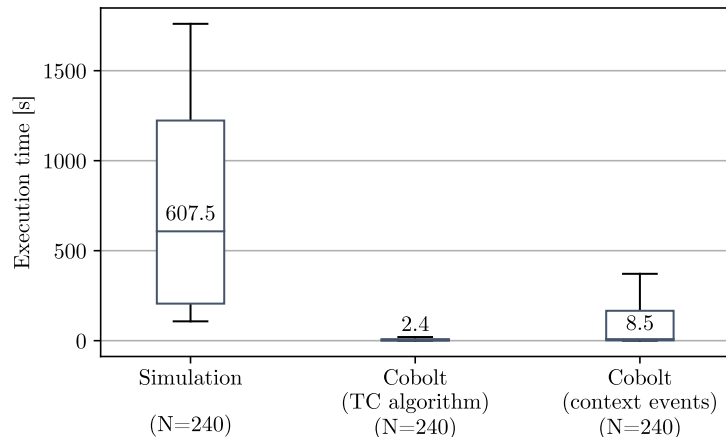


Figure 5.11: Execution time comparison of simulation, TC algorithm, and context event handling (label per boxplot: median execution time, **N**: number of data points per boxplot)

that the median simulation duration is 607.5 s. The execution of the TC component requires 10.7 s in the median case (i.e., 1.8 % of the median simulation execution time).

The plot also shows that the simulation execution time varies between 107 s (lower whisker) and 1760 s (upper whisker). The corresponding ranges for the TC algorithm is 0.061 s to 20.2 s and for the context event handlers 0.36 s to 371 s.

DISCUSSION The preceding results show that the code generated by COBOLT contributes a small fraction to the entire simulation duration during the experiments. Therefore, we conclude that COBOLT is well applicable for practical network simulations.

The reason for the large inter-quartile ranges in Figure 5.11 can be explained by the fact that each boxplot summarizes the total execution time values for 240 experiments and that the experiments cover a wide range of topologies. This can be quantified using the initial topology density (i.e., the initial link count divided by the initial mote count). The median density of all 240 experiments is 22.7 with first and third quartiles equal to 12.2 and 40.8. The maximum initial density is 104, which is an extreme value in comparison to the experimental setups in the consulted TC works. These numbers also indicate that our experiments cover a representative range of topology sizes.

To sum up, we answer RQ2 as follows. With a median fraction of 1.8 % of the total simulation execution time, the computational overhead of the code generated using COBOLT is low. This allows us to conclude that COBOLT is applicable in realistic WSN simulation scenarios

THREATS TO VALIDITY A major threat to external validity is the choice of the simulation scenarios. Regarding the experimental setup, we mitigated this threat by extracting typical evaluation setups from 11 works on TC. Even though we used a single TC al-

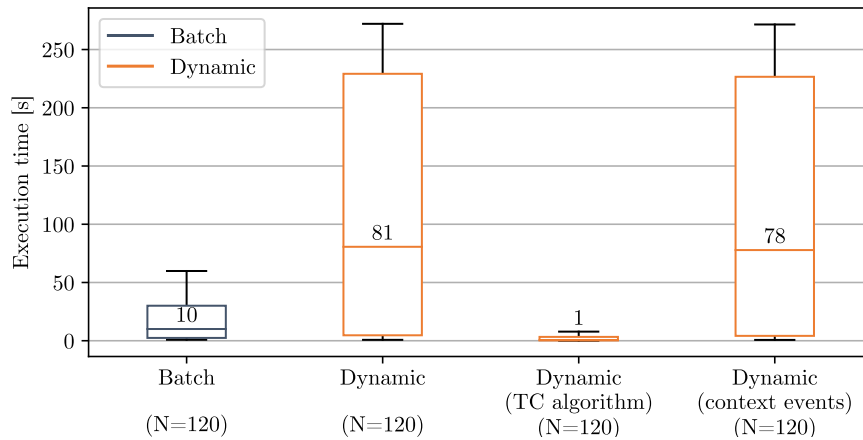


Figure 5.12: Execution time comparison batch vs. dynamic (label per boxplot: median execution time, **N**: number of data points per boxplot)

gorithm, kTC, in this evaluation, we observed that kTC is a typical representative of TC algorithms based on studying prominent TC survey papers [219, 264, 267]. In this context, “typical” means that the computational complexity of several state-of-the-art TC algorithms is comparable to kTC, for instance, because they also use a triangular pattern graph. A detailed discussion of our investigation of the aforementioned surveys follows in the subsequent Chapter 6.

A second threat to external validity is our choice of the hardware platform for developing the TC algorithm and conducting the experiments because the CPU execution time strongly depends on the chosen hardware platform. This threat does not apply to this evaluation because we used a workstation with a CPU that was released in 2011 and 2 GB of working memory, which should be available in today’s workstations. Therefore, we can be sure that the execution time of the generated code as well as the entire simulation will be smaller when the experiments are conducted on a more recent hardware. Furthermore, it is not probable that the proportions of the execution time values differ drastically because both the execution of the generated code as well as the other simulation processes are mainly CPU intensive tasks.

A major threat to the internal validity of our results is that numerous processes within a simulation run are random (e.g., node movement, inter-transmission waiting time, initial energy distribution). To mitigate the influence of randomness, we evaluated each of the 24 simulation scenarios with 10 different random seeds.

5.2.4.5 Results and discussion for RQ3 (dynamic topology control)

In the following, we compare the efficiency of dynamic and batch TC in the context of the conducted experiments. For this research question, we reuse the experimental setup Section 5.2.4.2.

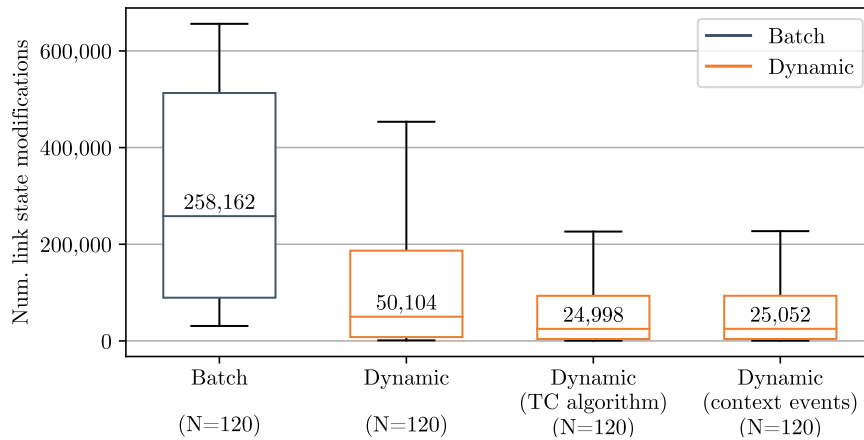


Figure 5.13: Link state modification comparison batch vs. dynamic (label per boxplot: median link state modification count, **N**: number of data points per boxplot)

METRICS We quantify the effort of batch and dynamic TC in terms of the required (i) execution time (as before), and (ii) number of link state modifications. A link state modification is the change of the state of a link (e.g., from U to A or I during the TC algorithm execution). We measure these metrics per execution of the TC algorithm and each context event handler.

RESULTS The leftmost two boxplots in Figure 5.12 compare the execution time (in s) of the 120 experiments for which kTC was executed in batch and dynamic mode, respectively. The rightmost two boxplots show the fraction of time that is consumed by the execution of the TC algorithm and the context event handlers, respectively.

For each experiment, we calculated the sum of the respective execution time values. This means that each boxplot aggregates 120 data points.

The median execution time for batch TC is 10 s, and the median execution time for dynamic TC is 81 s. The ranges for batch and dynamic TC are 0.82 s to 59.9 s and 0.75 s to 272 s, respectively.

The detailed boxplots for the dynamic scenario reveal that the major fraction of the time for dynamic TC is spent in context event handling (78 s in the median case).

As a second metric for assessing the cost of batch vs. dynamic TC, we consider the link state modification count. The x axis of Figure 5.13 is identical to Figure 5.12. In contrast, the y axis of the plot shows the total number of link state modifications per experiment. Again, each boxplot summarizes 120 experiments with batch and dynamic TC enabled, respectively. Each data point corresponds to the added link state modification counts of all TC and context event handler executions per experiment.

The plots shows that batch TC requires 258 162 link state modifications in the median case, whereas dynamic TC requires 50 104 link state modifications. The link state modi-

fication counts for the execution of the TC algorithm and the context event handling are almost identical (24 998 vs. 25 052).

DISCUSSION The results shown in Figures 5.12 and 5.13 allow for the following conclusions. First, batch TC is considerably faster than dynamic TC regarding CPU time in the median case. The results regarding execution time reveal that a large fraction of the time that a context event handler consumes is spent in analyzing the topology (and not in link state modifications). This can be seen when comparing the median number of link state modifications for the TC algorithm and the context event handlers, which are almost equal in Figure 5.13, with the required execution time, which differs by a factor of 78 (Figure 5.12). One reason of the large amount of time for analyzing the topology is that the state of a link is stored as attribute value. This means that identifying a link having a particular state entails a linear search through the entire list of links (either of the incident links of a mote or of all links in the topology). This behavior could be improved in the future by allowing a faster access to all links having a given state. For instance, this could be achieved using dedicated associations in the metamodel or by employing an incremental pattern matcher (e.g., VIATRA [21]), which keeps track of all matches of relevant patterns in a time-efficient manner. Still, the result in Section 5.2.4.4 show that the TC algorithm and context event handler executions contribute only 1.8% of the entire simulation duration. Therefore, it is doubtful whether further optimizations lead to a perceivable improvement.

Second, dynamic TC clearly outperforms batch TC regarding the link state modification count. In the median case, batch TC requires 5.2 times more link state modifications (i.e., 50 104 instead of 258 162 link state modifications). A link state modification usually entails an adaptation of the transmission power of a mote. This means that the second result is the more salient one compared to the execution time comparison.

To sum up, we answer RQ₃ as follows. In the median case, (i) dynamic TC requires a larger CPU execution time during simulations compared to batch TC, and (ii) dynamic TC is considerably more efficient regarding the hardware-independent link state modification count.

THREATS TO VALIDITY The discussion of threats to validity from Section 5.2.4.4 applies analogously to this part of the evaluation and is not repeated here for conciseness reasons.

5.3 MODEL-BASED TESTBED EVALUATION WITH CMOFLON

With COBOLT, we presented tool support for the model-based simulative evaluation of TC algorithms. In this section, we describe cMOFLON, a complementary tool for the testbed evaluation of TC algorithms. cMOFLON is a variant of eMOFLON that generates embedded C code for the IoT operating system CONTIKI [51]. The prefix “c” of cMOFLON alludes to the target platform CONTIKI that replaces the target platform EMF of eMOFLON. Additionally, cMOFLON is also a case study that showcases how eMOFLON can be adjusted to generate code for new target platforms.

DESIGN GOALS Our foremost design goal during the development of cMOFLON was to use the same TC algorithm specification for simulation and testbed evaluation. This is especially important because, with COOJA, only an emulator (instead of a simulator) for CONTIKI exists. COOJA emulates the CPU of a mote. This allows the user to evaluate, for instance, the energy consumption of TC algorithms precisely. Still, the emulation of the CPU of each mote leads to computationally expensive experiments even for small mote counts. Therefore, it is important that the same specification can be evaluated using cMOFLON and COBOLT to increase the user’s confidence in the correctness of the specification.

One of the major goals of this thesis is to ensure the correctness of the generated TC algorithms (see Goal 1 in Section 1.2). Ensuring the correctness of a code generator is difficult in general. To profit from the extensively tested build process of eMOFLON and its continuing development, cMOFLON should reuse build process components from eMOFLON where possible.

WSN platforms tend to be resource constrained. This applies especially to the available program memory (the so-called *text segment*). Therefore, the code generated using cMOFLON shall map directly to internal data structures of the target platform where possible. We decided to generate C code for ToCoCo, an existing framework for implementing TC algorithms based on the CONTIKI operating system [239]. This choice allows us to profit from a well-tested framework for TC and to compare the TC algorithms generated using cMOFLON with the three manually implemented TC algorithms that ship with ToCoCo.

Covering the requirements of all existing TC algorithms immediately is impossible. Therefore, we decided to develop cMOFLON based on the requirements of the three TC algorithms kTC, l*kTC and LMST. To support further TC algorithms, cMOFLON provides extension points regarding custom attribute constraints and auxiliary data structures.

CHOICE OF l*kTC AND LMST We chose the TC algorithms l*kTC and LMST as additional running example for this section on cMOFLON because of the following two reasons. The first reason is that kTC, l*kTC, and LMST have been developed manually and independently from cMOFLON for ToCoCo. Their implementations in ToCoCo

have been published in [239], and their source code is available online²². This allows us to compare the TC algorithms generated using cMOFLON with their manually implemented counterparts using the same operating system (i.e., CONTIKI) and TC library (i.e., ToCoCo). In contrast, no published and independent manual implementation of a TC algorithm in SIMONSTRATOR exists. Therefore, we did not consider these examples in the evaluation of COBOLT (Section 5.2.4).

The second reason is that the selected algorithms are also representative for a larger class of TC algorithms. kTC and l*kTC are representatives of the class of triangle-based TC algorithms, which comprises well-known TC algorithms such as RNG [109], GG [211], XTC [270], or DG [48]. The TC surveys in [219, 264, 267]) provide further evidence of this claim because, for each of the mentioned triangle-based TC algorithms, they present additional variants that have been developed over the years. LMST lends itself as example because it is a widely known and built-upon TC algorithm: The corresponding INFOCOM conference paper has been cited at least 241 times²³, and the corresponding journal paper has been cited at least 181 times²⁴ until today. This indicates that LMST is a representative TC algorithm.

Therefore, if we are able to specify and generate code for kTC, l*kTC, and LMST using cMOFLON, this is a strong argument for its applicability.

RELATED PUBLICATIONS AND THESES We presented cMOFLON 1.0.0 in [121]. David Giessing designed and implemented the first prototype of cMOFLON in his Bachelor's thesis [77].

5.3.1 Architecture

We begin the description of cMOFLON with an architectural overview. To understand the architecture of cMOFLON, we briefly describe the operating system CONTIKI and ToCoCo. Then, we outline how we built the cMOFLON as a variant of eMOFLON.

5.3.1.1 Contiki

Our choice of CONTIKI as underlying sensor operating system was mainly determined by the fact that it is a widely used operating system for sensor devices. Since its publication in 2004, the corresponding LCN paper [51] has been cited 815 times²⁵. Furthermore, with ToCoCo [239], a convenient abstraction layer for implementing TC algorithms based on CONTIKI exists. The kernel of CONTIKI provides non-preemptive multi-process execution with preemptive multi-threading available as application library. The code size of CONTIKI on an MSP430 microcontroller is ca. 3.9 kB.

²² ToCoCo repository (original): <https://github.com/steinmic/ToCoCo> (visited: 2018-10-12)

²³ According to <https://ieeexplore.ieee.org/document/1209193/> (visited: 2018-11-25)

²⁴ According to <https://ieeexplore.ieee.org/document/1427709/> (visited: 2018-11-25)

²⁵ Citation count according to <https://ieeexplore.ieee.org/document/1367266> (visited: 2018-10-16)

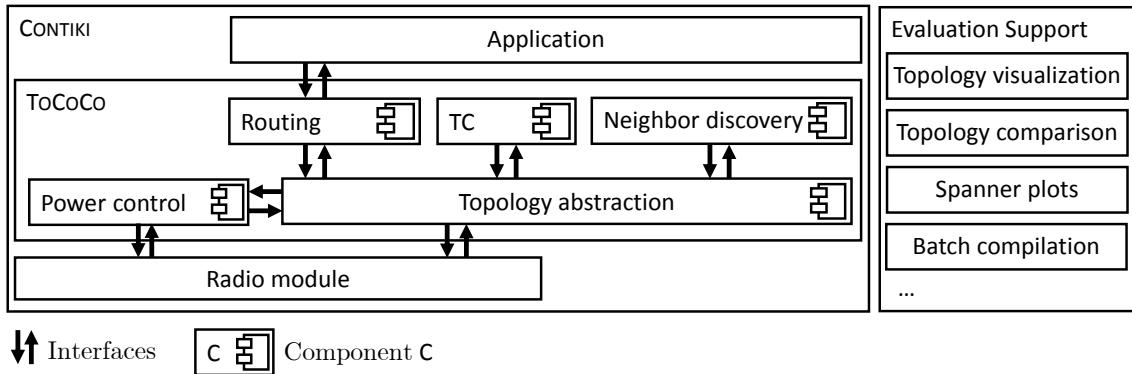


Figure 5.14: Architecture of the ToCoCo evaluation framework

5.3.1.2 ToCoCo

Instead of developing a cMOFLON-specific framework for TC in CONTIKI, we built on the existing ToCoCo framework [239]. ToCoCo eases the implementation of novel TC algorithms by hiding the low-level technical details from the TC algorithm developer. Figure 5.14 illustrates the architecture of ToCoCo, which defines a number of different *component types*. The interfaces of all component types build on the central topology abstraction component. The *topology abstraction component* provides a graph-based view of the local view of a mote and stores the virtual topology as a blacklist of ignored neighbors. For each *component type*, ToCoCo ships with one or more existing implementations and allows the TC algorithm developer to add further implementations. We describe the role of each component type in the following. In this context, Figure 5.15 summarizes the described component configuration options as feature diagram (see also Section 3.6). The figure also highlights the partial configuration that we used for the evaluation of cMOFLON.

An optional *TC component* controls the execution of the active TC algorithm and decides which neighbors shall be added to or removed from the virtual topology. Choosing a TC algorithm is optional. ToCoCo ships with three TC algorithms: kTC [227, 228], l*kTC [130, 239], and LMST [140]. For each of these TC algorithms, we provide a generated counterpart.

An optional *neighbor discovery component* provides the weighted input topology to the topology abstraction component. The responsibility of this component is the monitoring of immediate neighbors to identify incident links and the message exchange with other motes to increase the size of the local view according to the needs of the selected TC algorithm. ToCoCo currently provides only a single neighbor discovery protocol, which builds up a two-hop local view using broadcast messages. The weight of links is either set to the measured RSSI or to the mote distance based on preconfigured posi-

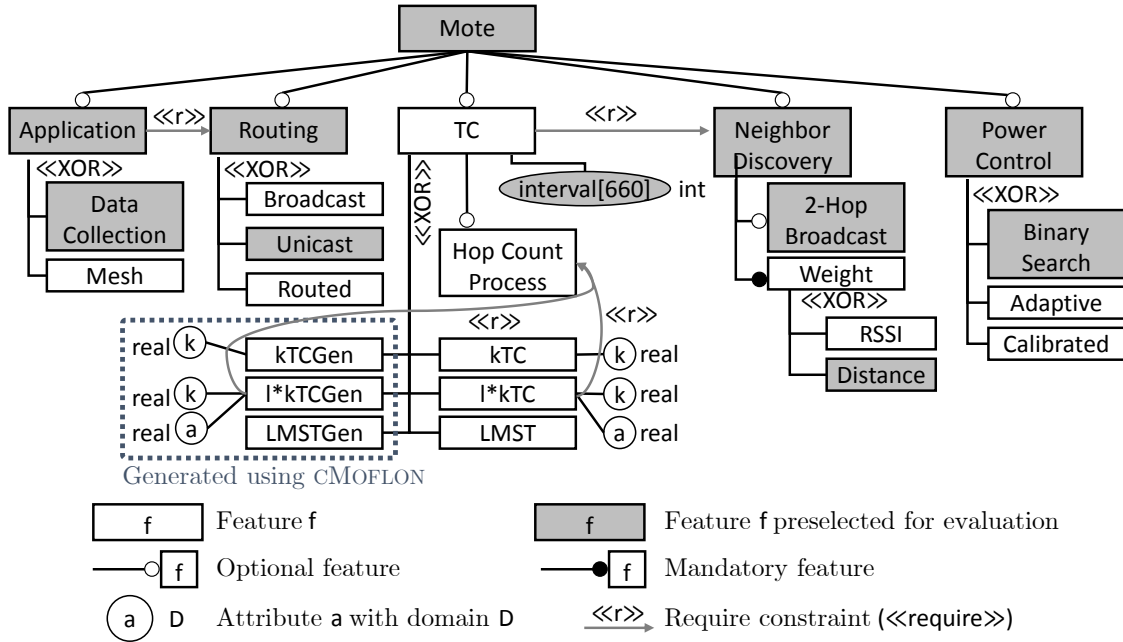


Figure 5.15: ToCoCo configuration space and selected options

tion information per mote. Neighborhood discovery is mandatory if a TC component is active.

An optional *power control component* adjusts the transmission power of the radio module to ensure that all active links can be used by the routing component. Three different power-control algorithms are available: binary search, adaptive, or calibrated.

An optional *routing component* provides a convenient interface to the application to exchange messages among motes. ToCoCo reuses routing algorithms of the RIME [52] stack (e.g., broadcast, unicast, routed message forwarding).

The *application* is not a component per se but is specific to the current use case. ToCoCo ships with four sample applications: data collection, random walk, power calibration, and a wireless mesh protocol.

Typically, the business logic of a ToCoCo component is implemented as one or more CONTIKI processes and contained in a single pair of header (.h) and implementation (.c) file. The configuration options of a component are registered as preprocessor constants in a central configuration file. The original version of ToCoCo [239] is available open source on GITHUB²⁶. For our work on cMOFLON, we created an open-source fork²⁷ and extended the existing suite of evaluation scripts (e.g., for plotting time series of topologies and for the batch compilation of TC algorithms).

26 ToCoCo repository (original): <https://github.com/steinmic/ToCoCo> (visited: 2018-10-12)

27 ToCoCo repository (cMOFLON fork): <https://github.com/eMoflon/ToCoCo> (visited: 2018-10-12)

5.3.1.3 Integration with eMoflon build process

Figure 5.16 shows the build process of cMOFLON. This figure highlights which components of eMOFLON we reused (see also Figure 5.2) and which parts we implement specifically for cMOFLON. All build steps that are modified in cMOFLON compared to eMOFLON are highlighted in blue and marked with a “C”. We reused the ENTERPRISE ARCHITECT add-in (eMOFLON EA) as well as the import and validation components of eMOFLON without modifications. At this point, we provide a summary of the code generation workflow. Detailed descriptions of how individual artifacts are mapped to C code can be found in Section 5.3.3.

First, instead of generating class skeletons from each EClass in the metamodel, we decided to distinguish between concepts that can be mapped to internal data structures of ToCoCo and CONTIKI, and user-specified concepts (e.g., helper data structures). Therefore, cMOFLON expects to find certain EClasses and certain attributes of these classes in the metamodel (e.g., Mote, Link, Link::weight). Second, the transformation from control flow model to C code is similar compared to JAVA because both languages share most control structures (scopes, scoping of variables, conditions, do-while loops, while-do loops). Third, the transformation from search plan to C code is also structurally similar to eMOFLON. Here, we reused the strategy of mapping each search plan to a hierarchical STRINGTEMPLATE.

5.3.2 Specification

We now describe the particularities of cMOFLON regarding the specification perspective (this section) and the code generation (Section 5.3.3). To highlight the capabilities of cMOFLON, we extend our running example by two additional TC algorithms: (i) l*kTC [239] is a variant of kTC that requires user-defined attribute constraints, and (ii) LMST [140] is a tree-based TC algorithm that possesses a more complex control flow specification compared to kTC and requires auxiliary data structures.

5.3.2.1 The l*kTC algorithm

Until now, we considered as running example the TC algorithm kTC [227], which is an application-agnostic TC algorithm. An *application-agnostic TC algorithm* is unaware of the active application (e.g., data collection or pairwise message exchange). In [239], Stein et al. point out that considering application-related information in a TC algorithm may help to improve the quality of the virtual topology (e.g., w.r.t. latency). To showcase this advantage, they propose l*kTC, a variant of kTC that is tailored to data collection scenarios, where all motes report to a dedicated base station. In such scenarios, each mote n is aware of its *hop-count* $h(n)$, which is the number of hops that are necessary to reach the base station. In the following, we assume that $h(n) < 0$, if hop-count information for a mote n is missing. l*kTC inactivates a link l_{XY} if the link (i) l_{XY} is

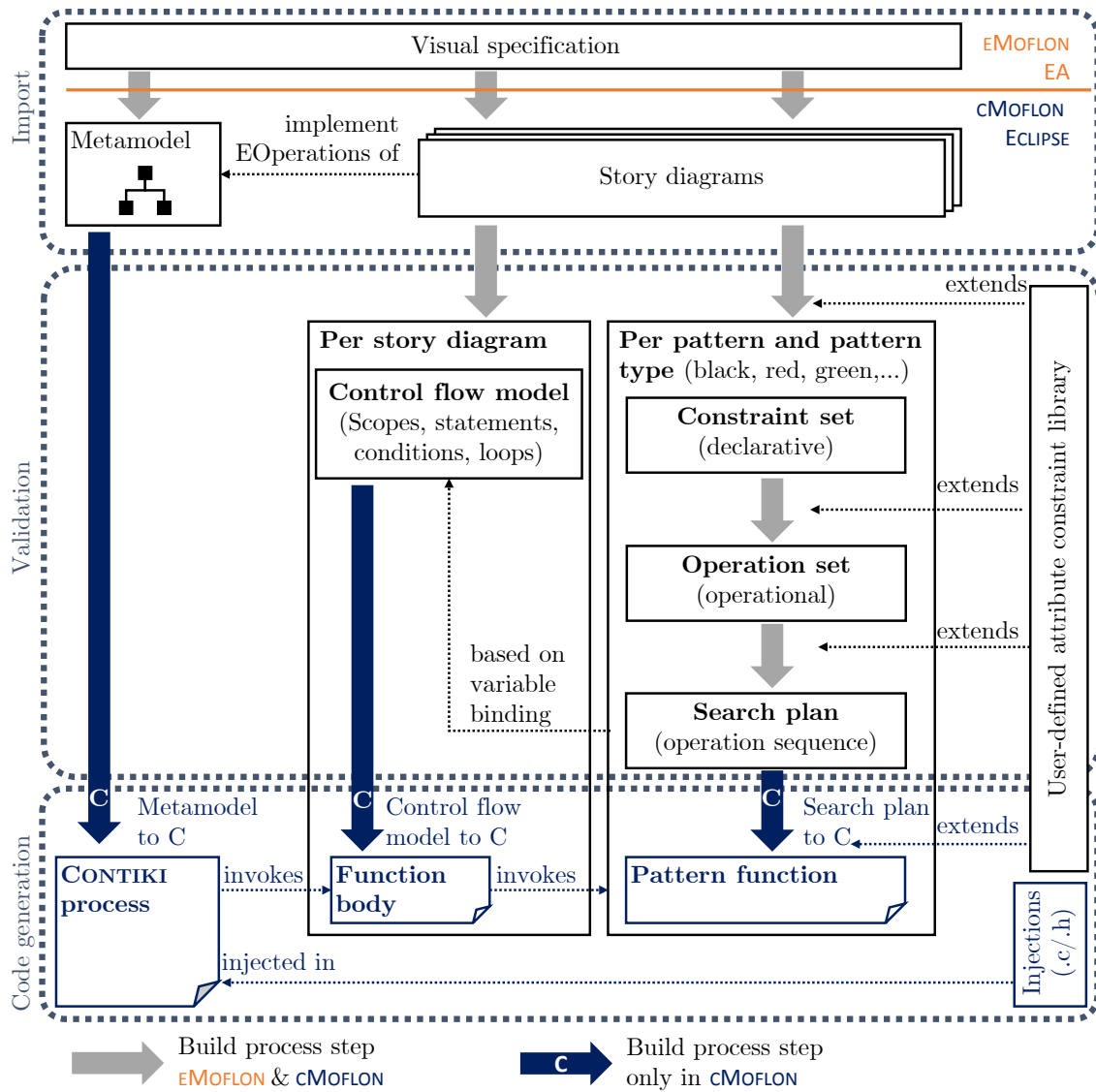


Figure 5.16: cMOflon build process with reuse from eMOflon

the weight-maximal link in a triangle (l_{XY}, l_{XZ}, l_{ZY}) , and (ii) the weight of l_{XY} is at least k times larger than the minimal weight in the triangle, and (iii) inactivating l_{XY} keeps the increase in the hop count of n_X and n_Y below a given *stretch factor* a compared with the input topology. The first two conditions are identical to the kTC-specific conditions (see Equation (3.4) in Section 3.2), while the last condition is specific to l*kTC. The third, l*kTC-specific condition can be expressed using first-order logic with arithmetic expressions as follows (h_X, h_Y, h_Z denote the hop count of motes n_X, n_Y, n_Z):

$$\varphi_{l^*kTC}(w_{XY}, w_{XZ}, w_{ZY}, h_X, h_Y, h_Z, X, Y, Z) : \Leftrightarrow \quad (5.1)$$

$$\varphi_{kTC}(w_{XY}, w_{XZ}, w_{ZY}, X, Y, Z) \quad (5.2)$$

$$\wedge \min(h_X, h_Y, h_Z) \geq 0 \quad (5.3)$$

$$\wedge (h_X = h_Y \Rightarrow \text{true}) \quad (5.4)$$

$$\wedge \left(h_X \neq h_Y \Rightarrow \frac{h_Z + 1}{\max(h_X, h_Y)} \leq a \right) \quad (5.5)$$

Clause 5.2 mandates that a link may only be inactivated if it fulfills the kTC-specific conditions. Clause 5.3 requires that a link may only be inactivated if hop-count information for all involved motes is available. Clause 5.4 states that, if the hop counts of n_X and n_Y are equal, then the link shall be inactivated because the path of neither n_X nor n_Y to the base station becomes longer by inactivating l_{XY} . Clause 5.5 covers the complementary case when one of the motes n_X or n_Y is closer to the base station compared to the other mote. In this case, the fraction of the enlarged routing path length $(h_Z + 1)$ and the maximum current hop count of n_X and n_Y may not exceed the stretch factor a .

The l*kTC algorithm is a heuristic TC algorithm: The calculation of the assumed increase in routing path length is performed locally and, therefore, independently of the decisions of all other motes. Therefore, the increase of the routing path length of a given mote to the base station in the virtual topology is not limited by the factor a in general.

Example 5.1 (l*kTC). Figure 5.17 shows a sample output topology of l*kTC for the kTC parameter $k = 1.3$ and the stretch factor $a = 1.5$. Mote n_1 acts as base station, and each mote is labeled with the current hop-count distance to n_1 in parentheses.

The links l_{47} and l_{74} are inactive because their weights are maximal and at least k times larger than the minimal weight in the triangles (l_{47}, l_{45}, l_{57}) and (l_{74}, l_{75}, l_{54}) , respectively. Additionally, the hop-count increase is $\frac{h(n_5)+1}{\max(h(n_4), h(n_7))} = \frac{3}{2} = 1.5$, which does not exceed a .

In contrast, the links l_{14} and l_{41} are active even though they are part of the φ_{kTC} -fulfilling triangles (l_{14}, l_{12}, l_{24}) and (l_{41}, l_{42}, l_{21}) , respectively. The reason is that the increase in hop count is $\frac{h(n_2)+1}{\max(h(n_1), h(n_2))} = \frac{2}{1} = 2$, which exceeds a .

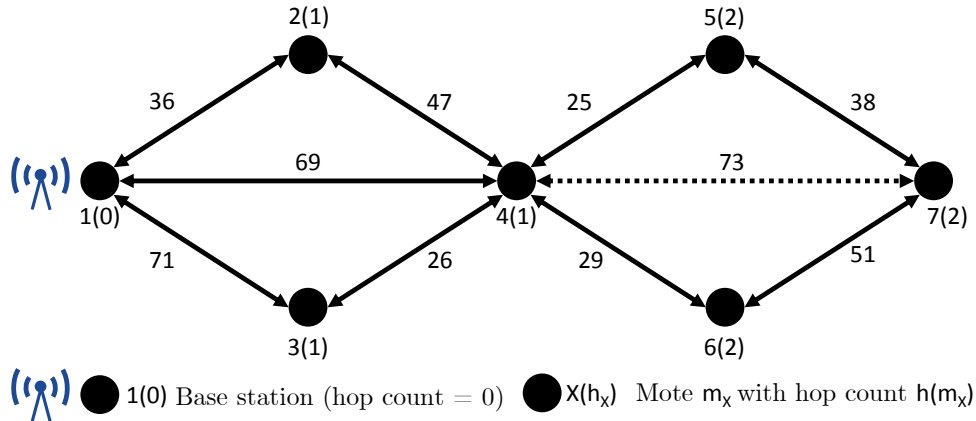


Figure 5.17: Example: Topology of l^*kTC with $k = 1.3, a = 1.5$

5.3.2.2 The LMST algorithm

The third considered TC algorithm is LMST [140]. We already discussed this algorithm briefly to highlight the limitations of graph constraints without path expressions in Section 3.7. The idea behind LMST is that each mote calculates a *local minimum spanning tree*, which is a minimum spanning tree within its local view. Then, the mote activates all outgoing links that are part of the local minimum spanning tree. In general, the resulting topology is not necessarily symmetric because a link that is part of the local minimum spanning tree in one local view could be excluded from the local minimum spanning tree in another local view [140]. Therefore, the LMST algorithm drops unidirectional links in the virtual topology of LMST (i.e., active links for which the reverse link is inactive). In [140], the authors show that this step does not violate connectivity.

LMST is an exciting further example for our methodology because its virtual topology is a minimum spanning tree, which must be acyclic. Similar to connectivity, acyclicity is a property that cannot be expressed using first-order logic [59]. Therefore, to develop LMST using our methodology, we have to provide a set graph constraints whose joint fulfillment implies that the active links in the virtual topology form a minimum spanning tree. We refrain from describing the detailed construction process for LMST at this point because our goal is to use LMST for illustrating the capabilities of cMOFLON. The following example concludes the introduction of LMST.

Example 5.2 (LMST). Figure 5.18 shows the result of applying LMST to the same input topology as in Example 5.1. In contrast to l^*kTC , no dedicated base station is marked because LMST is an application-agnostic TC algorithm. Three pairs of links are inactive. The links l_{14} and l_{41} are the weight-maximal links on the cycle

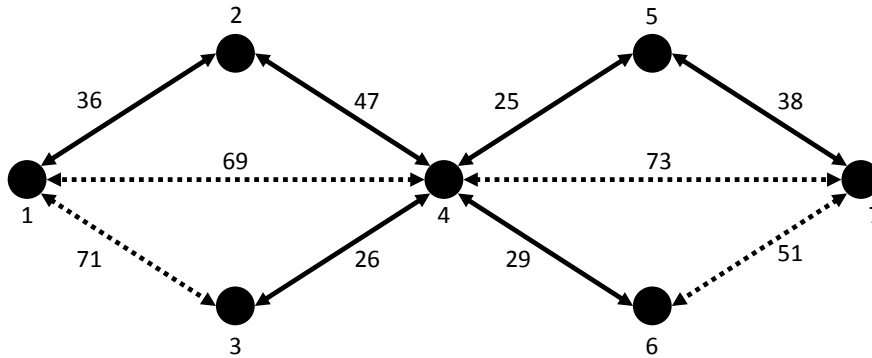


Figure 5.18: Example: Virtual topology of LMST

consisting of the links l_{14} , l_{42} , and l_{21} as well as l_{41} , l_{12} , l_{24} , respectively. Analogous cycles justify the inactivation of l_{13} , l_{31} , l_{47} , l_{74} , l_{67} , and l_{76} .

As illustrated in this example, the virtual topology of LMST always contains at least as many inactive links as the virtual topology of kTC because each φ_{kTC} -fulfilling triangle constitutes a cycle that justifies the inactivation of the weight-maximal link on this cycle.

5.3.2.3 Basic metamodel of cMoflon

After introducing the three TC algorithms that serve as running examples for cMOFLON, we now present the *basic metamodel* of cMOFLON. Figure 5.19 shows the basic metamodel together with the necessary extensions for kTC, l*kTC, and LMST (see dotted frames). The basic metamodel consists of all metamodel elements that are *not* enclosed by a dotted frame. cMOFLON expects to find these metamodel elements, which are handled specially by cMOFLON. As in the general TC metamodel (Figure 3.4), we interpret the topology as directed graph, whose nodes represent the motes of the topology (Mote) and whose links possess properties to store the state and weight of each link (Link::state, Link::weight). A TC algorithm is represented by a concrete subclass of TopologyControlAlgorithm, whose run operation contains the TC algorithm logic.

What distinguishes the cMOFLON metamodel from the general TC metamodel is that the TC algorithm is attached to a particular mote (instead of the entire topology). The reason is that TC algorithms specified with cMOFLON are always executed as localized algorithm based on a local view of the topology. The mote that is attached to the TC algorithm instance is the *self-mote*, which represents the mote that owns the local view.

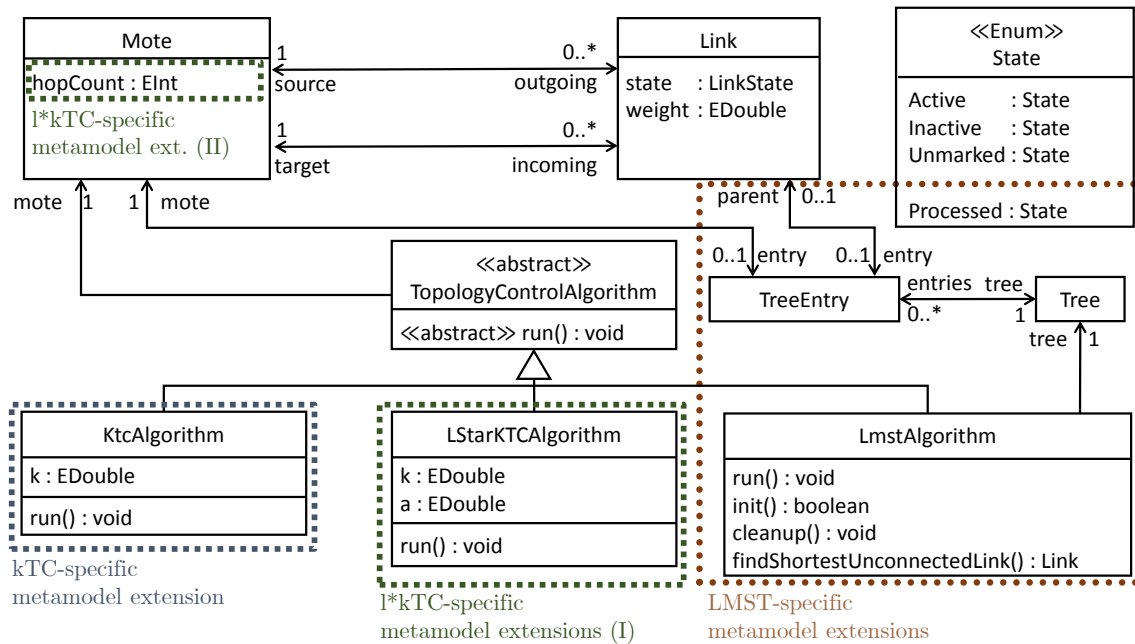


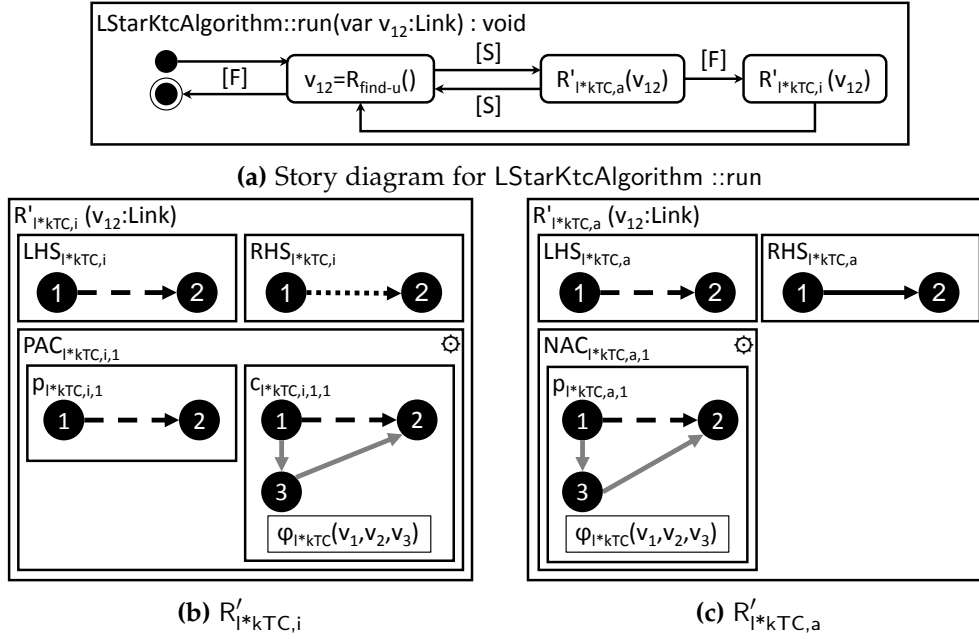
Figure 5.19: Basic cMOFLON metamodel with extensions for kTC, l*kTC, and LMST

5.3.2.4 Specifying kTC

The kTC algorithm can be specified without modifying **Mote** or **Link** because kTC only requires information about the weight of links. The actual TC algorithm is modeled by introducing the new subclass **KtcAlgorithm** of **TopologyControlAlgorithm**. The control flow of the **KtcAlgorithm::run** operation is specified in an analogous way as shown earlier in Chapter 4. The ToCoCo framework does not provide notifications about changes to the input topology. Therefore, we only consider the TC algorithm specification and neglect context event handlers throughout this chapter.

5.3.2.5 Specifying l*kTC

In contrast to kTC, the TC algorithm l*kTC required additional information about the hop count of each mote. We represent this property by adding a corresponding attribute to the **Mote** class (see Figure 5.19). As for kTC, we also create a subclass **LStarKtcAlgorithm** of **TopologyControlAlgorithm** to specify the control flow of l*kTC. Figure 5.20a shows the corresponding story diagram, which possesses the same structure as the kTC and Maxpower algorithms that we specified earlier. The GT rule $R_{\text{find-u}}$ serves to identify an unmarked link v_{12} and is identical to the corresponding GT rules in the story diagrams for kTC and Maxpower. The GT rules $R'_{l*kTC,a}$ and $R'_{l*kTC,i}$ conduct the actual marking of the identified unmarked link v_{12} . Both rules could result from applying the constructive approach and are, therefore, decorated with a gear symbol at


 Figure 5.20: Specification of I^*kTC

the synthesized application conditions. In the figure, we use a shorthand version of the I^*kTC -specific predicate:

$$\varphi_{I^*kTC}(v_1, v_2, v_3) \Leftrightarrow \varphi_{I^*kTC}(w_{12}, w_{13}, w_{32}, h(v_1), h(v_2), h(v_3), id(v_1), id(v_2), id(v_3))$$

The I^*kTC -specific clauses Equations (5.3) to (5.5) of the I^*kTC constraint are implemented as a single user-defined constraint `hopcountOK` in `cMOFLON`. The `hopcountOK` constraint has four parameters corresponding to the three hop-count values h_x , h_y , and h_z as well as the stretch factor a . Regarding the operationalization, the only supported adornment mandates that all four parameters are bound. The evaluation of the constraint is implemented by the helper function shown in Listing 5.1. In the code, an implication $a \Rightarrow b$ is implemented in the usual way as $\neg a \vee b$.

5.3.2.6 Specifying LMST

For the specification of LMST in `cMOFLON`, we follow the same approach as Stein et al. in [239]. They separate the implementation of LMST into a tree-construction phase and a marking phase. During the *tree-construction phase*, the local minimum spanning tree is built up using Prim's algorithm [191] and stored in an additional tree data structure. The tree structure stores for each mote the *parent link*, which is the link that connects the mote to the tree. Prim's algorithm works by iterating over all links in increasing order of

```

1 static bool hopcountOK(EInt hX, EInt hY, EInt hZ, EDouble a) {
2     if (min(hX, min(hY, hZ)) < 0)
3         return false;
4     bool result = true;
5     result &= (!(hX == hY) || true);
6     result &= (!(hX != hY) || ((hZ + 1) * 1.0 / max(hX, hY) < a));
7     return result;
8 }

```

Listing 5.1: C code for evaluating the user-defined hopcountOK constraint

unique weight and adding all those links to the minimum spanning tree whose addition does not lead to a cycle in the tree.

The classes `Tree` and `TreeEntry` represent the tree data structure in the metamodel. The tree is accessible from `LmstAlgorithm` and points to a list of tree entries (entries-Tree association). Each tree entry represents one mote (mote-entry association) and has at most one parent link.

Additionally, we introduce a fourth link state `Processed` for technical reasons. This state serves for marking the current candidate link for insertion into the minimum spanning tree.

To specify the control flow of the LMST algorithm, we create a subclass `LmstAlgorithm` of `TopologyControlAlgorithm`. Besides the obligatory `run` operation, we specify additional helper operations. The Boolean `init` operation initializes the tree data structure by creating a single `Tree` object and, for each `Mote` object, a `TreeEntry` object. The return value indicates whether the initialization was successful (i.e., enough memory could be allocated). The `cleanup` operation serves to tear down the tree data structure. The `findShortestUnconnectedLink` identifies a link of minimal weight that has one incident mote with a parent link and another incident mote without a parent link. We provide injections for the `init` and `cleanup` operations and specified the `run` and `findShortestUnconnectedLink` operations using story diagrams. For the sake of conciseness, we omit the story diagram of `findShortestUnconnectedLink` here.

Figure 5.21a shows the story diagram that implements the `run` operation of `LmstAlgorithm`. First, the `init` operation is invoked. If the invocation is successful (i.e., returns *true*), the execution continues to the tree-construction phase. Otherwise, the execution continues to the invocation of `cleanup`. The tree-construction phase consists of a loop that continues as long as `findShortestUnconnectedLink` returns a (weight-minimal) link that connects a previously unconnected mote to the tree. A mote is connected to the tree by establishing a parent-entry association (see $R_{\text{connect-to-tree}}$ in Figure 5.21b). If no more suitable links can be found, the execution enters the marking phase, which is a loop

over all unmarked outgoing links of the self-mote (see $R_{LMST,find-u}$ in Figure 5.21c). The GT rule $R_{LMST,a}$ specifies that an unmarked link v_{12} shall be activated if it is the parent link of its target mote. If $R_{LMST,a}$ is inapplicable, v_{12} is inactivated by applying $R_{LMST,i}$. If no more outgoing unmarked links of the self-mote exist, the execution continues to the invocation of cleanup and, finally, terminates.

5.3.2.7 Supported metamodeling concepts and extension points

After describing the specification of the three considered TC algorithms, we now describe which extensions of the basic metamodel are supported in cMOFLON. cMOFLON supports the following types of extensions on the metamodel level: adding (E1) custom attributes and associations (e.g., `Mote::hopCount`, `entries-Tree`), (E2) custom attribute constraints (e.g., `hopcountOK`), and (E3) custom classes (e.g., `TreeEntry`). In our running example, `!*kTC` required (E1) and (E2), and `LMST` required (E3).

cMOFLON does not support inheritance relations except for inheritance from `TopologyControlAlgorithm` to indicate that a TC algorithm exists.

5.3.3 Code generation

In the following, we list a number of detailed implementation decisions that we took during the development of cMOFLON regarding the code generator. As mentioned earlier, we did not develop a general-purpose code generator but tailored the generated code to the available data structures of the target platform.

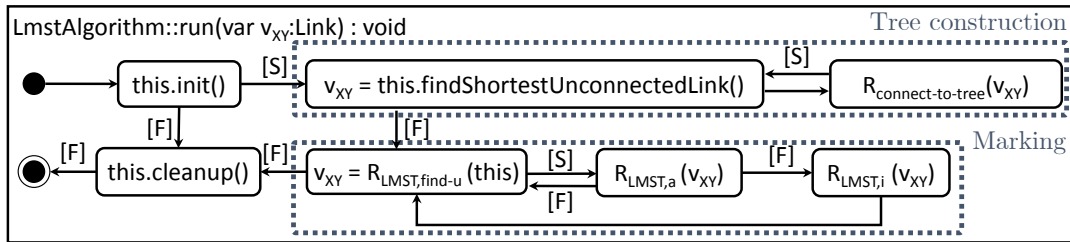
5.3.3.1 Mimicking object orientation

To provide a systematic mapping from classes and their operations to functions in C, we use a typical scheme that composes function name of the class name and method name and inserts a synthetic pointer parameter `this`. For example, the function prototype that corresponds to `KtcAlgorithm::run` is

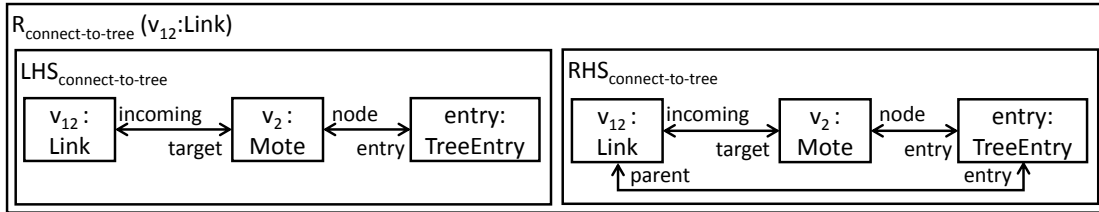
```
1 void ktcAlgorithm_run(KTC_ALGORITHM_T* this).
```

This example also illustrates how we map EClass names systematically to C type names. The C type name results from splitting the class name at uppercase letters, inserting underscores, converting all letters to uppercase, and appending the suffix `_T`. We generate for each attribute getter and setter function prototypes and, for each association type, we generate getter function prototype returning an object of type `list_t`. The type `list_t` is a generic list data structure provided by the CONTIKI standard library²⁸. For the attributes `Link::state` and `Link::weight` and the source-outgoing and target-incoming associations, we not only provide prototypes but also mappings to the internal data structures of `ToCoCo`, as described later.

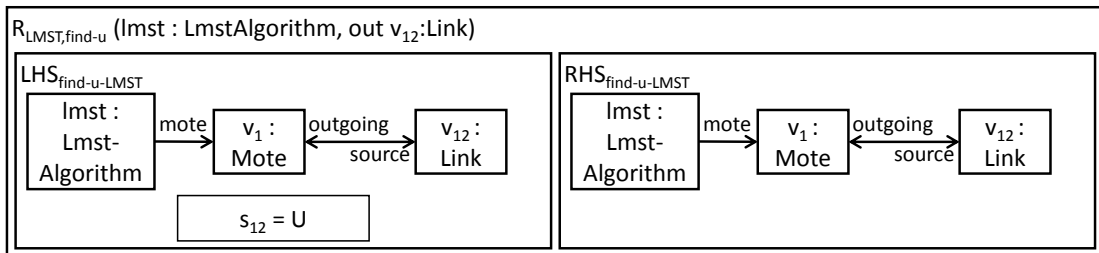
²⁸ CONTIKI linked list library <http://contiki.sourceforge.net/docs/2.6/a01682.html> (visited: 2018-10-16)



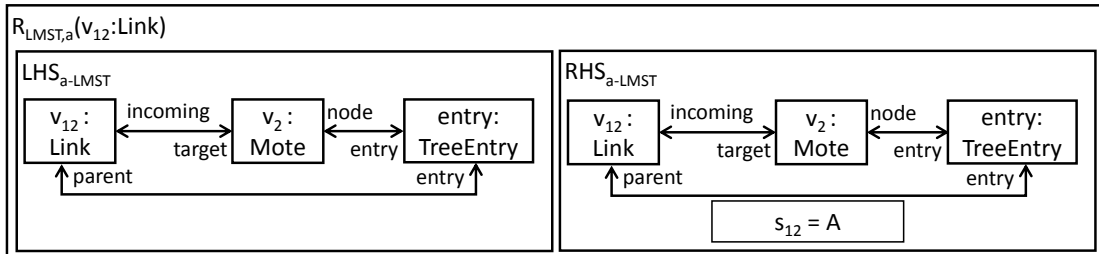
(a) Story diagram for `LmstAlgorithm::run`



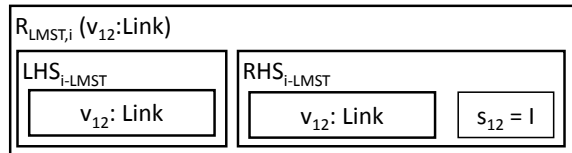
(b) $R_{\text{connect-to-tree}}$



(c) $R_{\text{LMST,find-u}}$



(d) $R_{\text{LMST,a}}$



(e) $R_{\text{LMST,i}}$

Figure 5.21: Specification of LMST

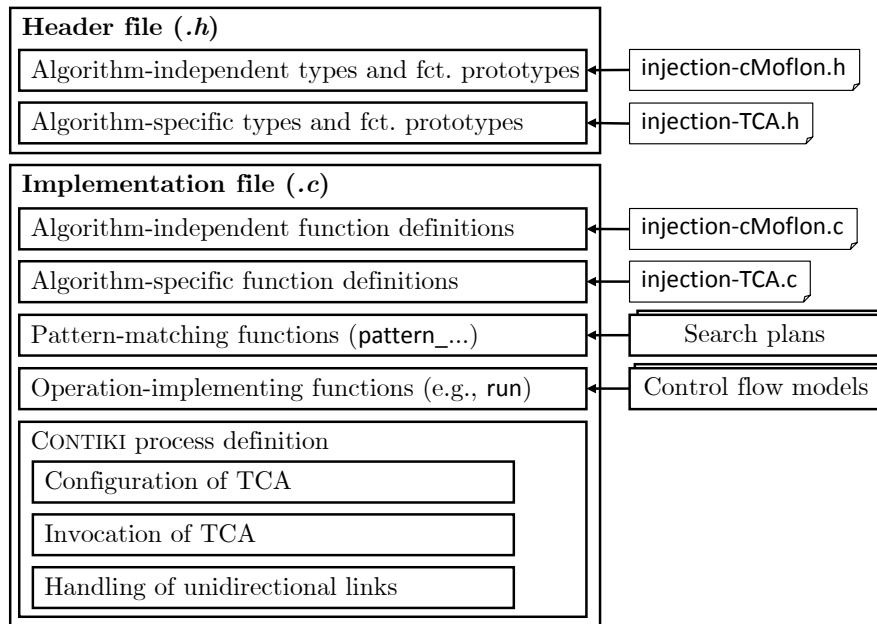


Figure 5.22: Anatomy of a generated TC algorithm in cMOFLON

5.3.3.2 Representing a topology control algorithm

Figure 5.22 shows the structure of the header (*.h*) and implementation file (*.c*) that cMOFLON generates for each subclass of `TopologyControlAlgorithm`. The header file is composed of TC-algorithm-independent type definitions and function declarations (e.g., standard `ECORE` data types and their comparators) and TC-algorithm-specific type definitions and function declarations (e.g., for the tree data structure of LMST).

The implementation file starts with the corresponding definitions of TC-algorithm-independent and TC-algorithm-specific functions. Afterwards, the definitions of the pattern-matching and operation-implementing functions are inserted. At the end of the implementation file follows the CONTIKI process that contains the configuration and invocation of the TC algorithm.

5.3.3.3 Representing motes and links

We map the class `Mote` to the ToCoCo data structure `networkaddr_t`, which basically stores the unique address of a mote. For consistency reasons, we use the type `MOTE_T` in the generated source, which is a `typedef` for `networkaddr_t`.

We map the class `Link` to the ToCoCo data structure `neighbor_t`. For consistency reasons, we use the type `LINK_T` in the generated source, which is a `typedef` for `neighbor_t`.

In ToCoCo, both directions of a link are stored in one instance of `neighbor_t`. This reduces the memory footprint compared to linking two different structures for each direction. This strategy avoids replicating the pointer to each incident mote and the linking between reverse links. Inside `neighbor_t`, each direction of a link pair is associated with a weight. A negative link weight indicates that the link in the corresponding direction is missing. In this case, the link is asymmetric. For identifying asymmetric links, cMOFLON provides the attribute constraint `isWeightDefined`. For a given double value, the constraint evaluates to *true* if the value of weight indicates that the weight is present and *false* if the link in this direction is missing.

ToCoCo guarantees that all links incident to the self-mote have the self-mote as first mote. For all other links, the order is undefined. Currently, the developer needs to consider this property on the specification level by checking links that are not incident to the self-mote in both directions. This results from technical reasons because the code generator of DEMOCLES does not support multiple bodies per pattern, which could be used to evaluate a link in both directions. An alternate solution to this problem would have been to reflect the internal structure of `neighbor_t` in the metamodel, but this would mean that we pollute the specification with details of the implementation. Furthermore, such a platform-specific modification would break the interoperability with COBOLT.

5.3.3.4 *Representing data types and helper classes*

As shown in Table 5.4, cMOFLON provides mappings for standard Ecore data types as well as for the Mote, Link, and (as an example) for the LMST-specific classes `LmstAlgorithm`, `Tree`, `TreeEntry`. The LMST-specific classes are provided by the developer as injections. For instance, the C type that corresponds to the `TreeEntry` class, must have the name `TREE_ENTRY_T`. Regarding the standard Ecore types, the developer may either use the corresponding C type (e.g., `char`) or the `typedef` of the Ecore type (e.g., `EChar`).

5.3.3.5 *Representing link states*

The state enumeration is represented by the enumeration type `LinkState`. For memory efficiency reasons, we extended the `neighbor_t` data structure of ToCoCo by an additional attribute. This additional attribute can be switched off using a preprocessor flag to ensure backwards compatibility with ToCoCo.

We provide a custom setter function `link_setMarked` for the link state, which updates the blacklist of the topology abstraction component of ToCoCo. Inactivating a link corresponds to adding the target mote to the blacklist and activating a link corresponds to removing the target mote from the blacklist. The function only handles link state modifications that affect outgoing links of the self-mote because modifying the state of all non-incident links has no effect.

Table 5.4: Type mappings in cMOFLON (incl. LMST-specific ones, [X]: EMF type name)

Metamodel elem.	Type mapping	Built in?
Mote	<code>typedef networkaddr_t NODE_T</code>	✓
Link	<code>typedef neighbor_t LINK_T</code>	✓
boolean	<code>typedef bool EBoolean</code>	✓
double	<code>typedef double EDouble</code>	✓
float	<code>typedef float EFloat</code>	✓
int	<code>typedef int EInt</code>	✓
long	<code>typedef long ELong</code>	✓
char	<code>typedef char EChar</code>	✓
char	<code>typedef short EShort</code>	✓
byte	<code>typedef char EByte</code>	✓
string	<code>typedef const char* EString</code>	✓
LmstAlgorithm	<code>typedef struct { NODE_T* node; struct TREE_T* tree; } LMSTALGORITHM_T;</code>	–
Tree	<code>typedef struct { LMSTALGORITHM_T* algo; list_t entries; // Memory for tree entries struct memb* mem; } TREE_T;</code>	–
TreeEntry	<code>typedef struct { struct TREEENTRY_T* next; NODE_T* node; LINK_T* parent; TREE_T* tree; bool isInTree; } TREEENTRY_T;</code>	–

5.3.3.6 Representing object creation and deletion

cMOFLON supports the specification of GT rules that modify attributes of variables and create or delete elements. For structural modifications (i.e., creation or deletion of ob-

jects), cMOFLON creates prototypes for the corresponding red and green patterns but leaves the actual memory management to the developer.

5.3.3.7 *Representing matches and collections of matches*

An important implementation decision was how to represent matches and collections of matches. The first question was whether to generate a dedicated match type for each pattern (as in, e.g., VIATRA [21]) or to use an untyped, generic representation of matches (as in, e.g., EMOFLON [135]).

The first alternative allows for returning matches by value because, for each pattern-matching function, the memory layout of the match data structure is known in advance. In this case, no additional memory management is necessary. The major disadvantage of this alternative is that we would need to create custom data type per pattern, which consumes additional memory in the text section of the compiled binary. This increased text section size reduces the available memory for the stack and heap sections, which is precious on resource-constrained devices.

The second alternative alleviates this problem by representing a match uniformly as generic array. For example, in EMOFLON, a match has the type `Object []`, and a collection of matches has the type `Object [] []`. The corresponding types in C are `void**` for matches and `void***` for collections of matches. Each entry of a generic array is a reference (JAVA) or pointer (C) to an element of the match. The code generator keeps track of the dynamic type of each element and inserts statements that extract and cast the match elements to the proper type in the method body implementation. The major disadvantage of this alternative is that it requires to perform memory management for the matches. Due to importance of reducing the memory consumption as much as possible on nodes, we opted for the second alternative. This entails that we have to manage the memory of the involved match data structures.

A match can be stored either in static or dynamic memory. If we use static memory management (i.e., global variables), the corresponding memory region is reserved permanently for the match and unavailable to the WSN application or other ToCoCo component. If we use dynamic memory management (i.e., `malloc`, `free`), we only acquire as much memory as needed to store the current match or collection of matches. After the termination of the TC algorithm, the memory is freed and available again to other processes. In both cases, the execution of TC may fail if insufficient memory is available. In case of static memory management, the global memory that has been reserved at compile time may turn out to be too small to store the required number of matches. In case of dynamic memory management, the heap may be too small, which results in a failure of TC at runtime.

The preceding discussion shows that no best solution exists. After preliminary experiments with both types of memory management, we decided to follow a hybrid strategy. For all types of story nodes but foreach nodes, we use a global data structure for storing the current match. The size of this data structure is determined by the maximum

```

1 static void* _result[7]; // Match data structure
2 // Omitted code...
3 static void** pattern_KtcAlgorithm_FindU_blackBFF(NODE_T* n1) {
4     LINK_T* e12;
5     list_t e12_n1_outgoing = node_getOutgoingLinks(n1);
6     for(e12=list_head_pred(e12_n1_outgoing,n1,node_isOutgoing);
7         e12!=NULL;
8         e12=list_item_next_pred(e12, n1, node_isOutgoing)) {
9         NODE_T* n2 = link_getTarget(e12);
10        if (!node_equals(n2, n1)) {
11            LinkState e12_marked = link_getMarked(e12);
12            if(linkState_equals(e12_marked, UNCLASSIFIED)){
13                // Match found: Store pointers in match
14                _result[0]= n1;
15                _result[1]= e12;
16                _result[2]= n2;
17                return _result;
18            }
19        }
20    }
21    return NULL; // Signal that no match exists
22 }

```

Listing 5.2: Example: Memory management for single-match pattern

number of variables that occur in a pattern in the specification. This number can be calculated at compile time. For example, for each of the three TC algorithms kTC, l*kTC and LMST, the maximum number of variables per pattern is seven. On the TELOS B platform [181], a pointer requires 2 B. Therefore, the size of the global match data structure was $7 \cdot 2 \text{ B} = 14 \text{ B}$ in total.

For foreach story nodes, we use the dynamic memory to hold the matches and a list in global memory for storing pointers to these matches. The size of the collection of matches of a pattern in a foreach story node cannot be determined at compilation time because it usually depends on the size model at runtime. Therefore, the processing of a foreach node may fail at runtime due to a lack of dynamic memory or insufficient reserved memory in the global match collection list.

For illustration purposes, Listing 5.2 shows an excerpt of the generated code of kTC. Line 1 contains the declaration of the global match data structure. During the search plan generation, cMOFLON determined that all patterns have at most seven vari-

ables. Given that a pointer requires 2B on the platform, this data structure requires $7 \cdot 2B = 14B$ in global memory. Each function that corresponds to a single-match pattern, returns a pointer of type `void**`. This also allows to return `NULL` if no match could be found (Line 21). If a match has been found (e.g., an unmarked link in Listing 5.2), the required number of entries is used in the match data structure (Line 13). For the specification of `kTC`, `l*kTC` and `LMST`, we refrained from using `foreach` nodes entirely. To test that `foreach` nodes are translated into proper C code, we used a dummy algorithm that inactivates all links whose weight is below a given threshold.

5.3.3.8 Providing hop-count information

The `l*kTC` algorithm requires information about the hop count of each mote. `cMOFLON` provides a configuration option that inserts a *hop-count propagation process* into the generated `CONTIKI` process. The original implementation of this process stems from the `ToCoCo` framework [239]. The process provides configuration to control the minimum and maximum intervals between broadcasts of the hop count of a mote. Other TC algorithms specified with `cMOFLON` that require access to hop-count information can reuse this hop-count propagation process.

5.3.3.9 Binary image size

The size of the generated machine code influences the available dynamic memory on the mote. Therefore, we experimented with some techniques to reduce the size of the compiled machine code. In a first attempt, we reduced the size of the resulting binary image by making all generated methods `static`. Using `static` functions allows the linker to omit symbols for external linkage.

The binary image size can also be reduced on the specification level. One example is the handling of pairs of [S] and [F]-edges compared to unlabeled activity edges. For the latter type of activity edges, the code generator inserts an error-handling routine that is when the pattern matching fails. In the former case, the execution continues unconditionally to the next activity node. For example, in Figure 5.20a, the activity edge that returns from the application of $R_{l*kTC,i}$ to the loop head is unguarded. Still, we know that a rule application always succeeds at this point. Therefore, we could insert a second activity edge and label the activity edges with [S] and [F], respectively.

Finally, the build process of `EMOFLON` inserts null-pointer checks whenever a single-valued `EReference` is traversed (i.e., with multiplicity `0..1` or `1..1`). In the latter case, this check can be omitted if we know that the model conforms to the metamodel (which should be a valid assumption if no GT rule application violates this property and we start with a valid model). For example, a link always possesses exactly one source and target mote. This is ensured by the corresponding data structure `neighbor_t` in `ToCoCo`, which has always two `networkaddr_t` that correspond to the incident motes. Therefore, we could omit null-pointer checks in the C code whenever we access the

source and target mote of a link. This analysis is currently unimplemented in `EMOFLON` and could lead to further improvements in the binary image size.

5.3.3.10 *Tool and code availability*

In this thesis, we present `cMOFLON 2.0.0`, an extension of `cMOFLON 1.0.0`, which we presented in [117]. `cMOFLON` is open-source and available on GITHUB²⁹. In an additional repository³⁰, we provide the specifications of the TC algorithms `Maxpower`, `kTC`, `l*kTC`, and `LMST`.

5.3.4 *Evaluation of cMoflon*

In the following, we present evaluation results for `cMOFLON`. We state the research questions regarding correctness and efficiency in Section 5.3.4.1, describe the experimental setup in Section 5.3.4.2, and discuss the results for each research question in Sections 5.3.4.3 and 5.3.4.4.

5.3.4.1 *Research Questions*

RQ1-CORRECTNESS Our foremost goal is to ensure that the C code generated using `cMOFLON` is correct. Similar to the evaluation of `COBOLT`, we answer the following subquestions.

- How can we ensure the correctness of a TC algorithm generated using `cMOFLON`?
- How can we ensure the correctness of the three generated TC algorithms `kTC`, `l*kTC`, and `LMST`?

We evaluate the first subquestion based on metrics for reuse of `EMOFLON` components in `cMOFLON`, and the second subquestion based on an inspection of the resulting topologies of the generated TC algorithms in 84 testbed experiments.

RQ2-EFFICIENCY In the second part of the evaluation, we focus on efficiency. In contrast to the evaluation of `COBOLT`, we have access to manual implementations for each of the three TC algorithms, which were developed independently of `cMOFLON`. This allows us to compare the generated with the manual variant of each TC algorithm.

The scarcest resource of a mote is often the available memory at runtime. The amount of runtime memory (i.e., stack and heap) is determined by the total working memory and the size of the memory segments that store the machine code (text section) as well as the global variables and literals (data and BSS). Therefore, we evaluate the generated TC algorithms w.r.t. both code memory and runtime memory efficiency. The available memory is, amongst others, determined by the memory consumption of the machine code in the binary image. Therefore, it is important that a binary executable that results

²⁹ `cMOFLON` repository: <https://github.com/eMoflon/cmoflon> (visited: 2018-10-11)

³⁰ `cMOFLON` examples repository: <https://github.com/eMoflon/cmoflon-examples> (visited: 2018-10-16)

from code generated using cMOFLON is compact enough to be deployed on a memory-constrained mote.

A typical dimension for evaluating runtime efficiency is execution time. However, the execution time is not of major importance in the context of a batch TC scenario because the TC algorithm is executed infrequently. Nevertheless, an excessive execution time would threaten the applicability of cMOFLON in the envisioned future scenarios that employ dynamic TC. To this end, we also evaluate the execution time of the generated TC algorithms. In summary, we will answer the following subquestions.

- How large is the code memory consumption of each generated TC algorithm?
- Is the code memory consumption reasonable?
- How large is the runtime memory consumption of the generated TC algorithms?
- How does the execution time of the generated TC algorithms compare to their manually implemented counterparts?

We evaluate these subquestions quantitatively based on the measured size of the sensor images and the execution time of the generated TC algorithms in testbed experiments.

In comparison to COBOLT, we only consider two instead of three research questions in the evaluation of cMOFLON. The reason is that we cannot perform a comparison of dynamic and batch TC because ToCoCo provides no support for the propagation of context events. Therefore, we focus on investigating the correctness of cMOFLON (RQ1) and the efficiency of the generated TC algorithms (RQ2).

5.3.4.2 *Experimental setup*

We conduct the quantitative part of the evaluation based on the three TC algorithms kTC, l*kTC, and LMST, which serve as running example for cMOFLON. As baseline, we consider the manually implemented variants of all algorithms, which are available in ToCoCo [239].

We executed the generated and manually implemented TC algorithms on the ca. 30 TELOS-B [181] motes of the FLOCKLAB testbed [149] at ETH Zurich. Figure 5.23 shows the state of FLOCKLAB on 2018-11-28 at 1500 (CET)³¹. In total, we conducted 14 experiments for each of the six manually implemented and generated TC algorithms, resulting in 84 experiments. The experiments took place from Friday, 2018-11-09 until Monday, 2018-11-12.

As operating system, we used CONTIKI 3.0. We used the same basic configuration as in [239] (see highlighted configuration in Figure 5.15). Each experiment starts with a neighbor-discovery phase, during which each mote builds up a two-hop local view of its neighborhood. In FLOCKLAB, the local neighborhood of each mote has stabilized after ca. 10 min. Therefore, the TC algorithm runs once after 10.5 min. Each experiment stops after 21 min. The WSN application is a data collection application (many-to-one communication). Mote 1 (top-left in Figure 5.23) is the base station in all experiments.

³¹ Origin of FLOCKLAB map: <https://user.flocklab.ethz.ch/user/testbedstatus.php>



Figure 5.23: FLOCKLAB location map with floor plan and mote status

The FLOCKLAB input topology changes frequently, even though the motes are stationary. One possible reason is that the motes are placed in an office building of the university where staff moves along the floors. Additionally, motes get maintained on a regular basis and, therefore, are sometimes unavailable for testbed experiments. For example, mote 28 was unavailable during four experiments (with a corresponding maintenance notice on the website) and mote 11 was unavailable in one experiment (without official announcement).

Due to the observed fluctuations in the testbed topology, we performed the experiments in collated order. This means that we executed 14 batches of experiments. Within each batch, the six algorithms ran in the following order: kTC (manual), l*kTC (manual), LMST (manual), kTC (generated), l*kTC (generated), and LMST (generated).

5.3.4.3 Results and discussion for RQ1 (correctness)

In Section 5.2.4.3, we already discussed the possible reasons for faults during the different development phases of a TC algorithm when using our methodology. All aspects regarding the specification refinement carry over to cMOFLON because cMOFLON reuses the visual specification frontend of eMOFLON without modifications. Regarding the target platform, we use the programming language C and the compiler MSP430-GCC³², a cross-compiler variant of GCC³³ for the MSP430 microcontroller of the TELOSb motes, which are the platform for executing the compiled binaries. Similar to the JAVA compiler

³² MSP430-GCC tool chain: <https://github.com/contiki-os/contiki/wiki/MSP430X> (visited: 2018-11-23)

³³ GCC page: <https://www.gnu.org/software/gcc/> (visited: 2018-11-23)

and the `JAVA VIRTUAL MACHINE` in the context of `COBOLT`, we assume that the employed `MSP430-GCC` compiler and the `TELOS B` motes work correctly.

Therefore, the major difference between `COBOLT` and `cMOFLON` is the code generator. `COBOLT` relies on the unmodified build process of `eMOFLON`, whereas `cMOFLON` is a dedicated code generator from `SDM` to embedded C for `CONTIKI`.

This exchange could introduce new faults in the build process. Due to time constraints, it was not possible for us to build a test suite for `cMOFLON` that is comparable in size to the test suite of `eMOFLON` (179 test projects, 372 unit tests). To reduce the potential of introducing faults, we reused as many components of `eMOFLON` as possible. Regarding the running examples of this section, we validated that the virtual topologies of the generated TC algorithms fulfill the TC-algorithm-specific conditions. Unfortunately, for technical reasons, we were unable to unit-test the generated TC algorithms in `CONTIKI`.

METRICS FOR BUILD PROCESS We determined the number of lines of code and the project count in both `eMOFLON` and `cMOFLON`. For this purpose, we used the tool `CLOC`³⁴ (Version 1.81). We used the source code of `eMOFLON 3.5.1`³⁵, `cMOFLON 1.5.0`³⁶, and `DEMOCLES 1.4.0`.

`eMOFLON` is a bootstrapped tool. This means that parts of the `eMOFLON` source code are generated using itself. The following measurements comprise this generated code, which is not available on `GITHUB`, but requires an `eMOFLON` developer workspace.

`eMOFLON` and `cMOFLON` build on the template language `STRINGTEMPLATE`³⁷ for model-to-text generation. The line-counting tool `CLOC` currently lacks support for `STRINGTEMPLATE`. Therefore, we counted the amount of `STRINGTEMPLATE` code separately. In `eMOFLON`, we considered all files having the extension `stg` (for “`STRINGTEMPLATE` group”) that are relevant for the `SDM` build process, and, in `cMOFLON`, we considered all `stg` files.

RESULTS FOR BUILD PROCESS Table 5.5 summarizes the number of projects and the number of lines of code for `eMOFLON`, `DEMOCLES`, and `cMOFLON`. The main programming languages for `eMOFLON` are `JAVA` and `C#`. The 35 622 lines of `C#` code originate entirely from the `eMOFLON` add-in for `ENTERPRISE ARCHITECT`. Due to the fact that `eMOFLON` is a bootstrapped tool, 89.9% of its `JAVA` code is generated code (i.e., 510 801 of 568 124 lines of code). `DEMOCLES` and `cMOFLON` mainly use `JAVA`.

The amount of `JAVA` code in `cMOFLON` corresponds to 0.67% of the total amount of `JAVA` code and to 6.6% of the manually implemented code in `eMOFLON`.

³⁴ `CLOC` page: <https://github.com/AlDanial/cloc> (visited: 2018-11-21)

³⁵ `eMOFLON 3.5.1` source code: https://github.com/eMoflon/emoflon-tool/releases/tag/emoflon-tie_3.5.1 (visited: 2018-11-21)

³⁶ `cMOFLON 1.5.0` source code: https://github.com/eMoflon/cmoflon/releases/tag/cmoflon_1.5.0 (visited: 2018-11-21)

³⁷ `STRINGTEMPLATE` page: <http://www.stringtemplate.org/> (visited: 2018-10-15)

Table 5.5: Size comparison eMOFLON vs. cMOFLON (ST: STRINGTEMPLATE, LOC: lines of code)

Tool	Projects	LOC Java	LOC C#	LOC ST
eMOFLON	64	568 124	35 622	373
DEMOCLES	11	8676	0	0
cMOFLON	5	3797	0	987

The amount of STRINGTEMPLATE code in cMOFLON is 2.6 times larger compared to eMOFLON. The primary reason is that cMOFLON also generates the header and implementation files using STRINGTEMPLATE. In contrast, eMOFLON uses a JAVA string factory that has been generated from JET³⁸ templates for the same purpose.

DISCUSSION FOR BUILD PROCESS Judging by the lines of source code, we conclude that cMOFLON is a lean variant of eMOFLON. The considerable architectural reuse of eMOFLON build process components in cMOFLON is also apparent in Figure 5.16. This degree of reuse, especially of complex components (e.g., the transformation from SDM to control flow model) reduces the potential for introducing faults into cMOFLON compared to eMOFLON.

METRICS FOR CONCRETE TOPOLOGY CONTROL ALGORITHMS For assessing whether the generated code produces a correct virtual topology, we inspected the reported input and virtual topology that each mote reports on a regular basis in a logfile. We inspected the virtual topology at the end of each testbed experiment (i.e., after 20 min) to ensure that the TC algorithm execution had terminated on all motes.

RESULTS AND DISCUSSION FOR CONCRETE TOPOLOGY CONTROL ALGORITHMS The inspection of the logfiles of the motes in the 84 experiments revealed no errors in the virtual topology. Therefore, we are confident that the generated TC algorithms are correct.

ANSWER TO RQ1 To sum up, we answer RQ1 as follows. As for COBOLT, the specification is correct by construction, and we rely on the correctness of the compiler and execution environment. In contrast to COBOLT, unit-testing the cMOFLON build process and the generated code was not possible for technical reasons. Instead, we inspected the generated code of the three representative TC algorithms manually and ensured that the corresponding binary executables returned correct results in 84 testbed experiments.

38 JET page: <https://www.eclipse.org/modeling/m2t/?project=jet> (visited: 2018-11-21)

THREATS TO VALIDITY The major threat to validity is the choice of metrics for investigating the correctness of cMOFLON. One alternate option would be to compare the output of cMOFLON with a reference code generator. Unfortunately, we are not aware of any reference code generator for TC in CONTIKI. Still, the evaluated metrics indicate that the space for introducing faults into the build process (compared to EMOFLON) is limited. Additionally, the analysis of the three TC algorithms that serve as running example in this section increase our confidence in the correctness of cMOFLON.

5.3.4.4 Results and discussion for RQ2 (efficiency)

In the second part of the evaluation of cMOFLON, we assess the efficiency of cMOFLON and the generated code for the three considered TC algorithms kTC, l*kTC, and LMST. We assess the efficiency of cMOFLON in terms of the required code generation time, and we assess the efficiency of the resulting TC algorithms in terms of code memory consumption and execution time.

METRICS FOR CODE GENERATION For measuring the code generation time, we installed cMOFLON in an ECLIPSE IDE 2018-09 (with Modeling Components) and set up a fresh ECLIPSE workspace. Then, we imported the cMOFLON example projects³⁹ containing the specifications of kTC, l*kTC, and LMST. From the user perspective, the code generation consists of two steps. First, the user exports her specification from ENTERPRISE ARCHITECT into an XMI file. We measured the duration of this step manually. Second, inside ECLIPSE, the code is generated from the exported XMI file. For this step, we used the execution time reported by EMOFLON.

RESULTS FOR CODE GENERATION To reduce warm-up effects, we measured the duration of each code generation step ten times. The median export duration from ENTERPRISE ARCHITECT was 3.9 s (min. 3.7 s, max. 4.5 s). The medium build time in ECLIPSE was 0.94 s (min. 0.56 s, max. 2.63 s). We observed the maximum build time during the first execution of the EMOFLON builder. All subsequent executions took at most 1.05 s.

DISCUSSION FOR CODE GENERATION The preceding results show that the median (4.9 s) and maximum (7.1 s) execution time of the code generation in our experiments are reasonable and do not restrict the applicability of cMOFLON.

METRICS FOR BINARY IMAGE SIZE The number of generated lines of code is not salient to assess the code memory consumption because the compiler may conduct optimizations (e.g., removal of unused functions, inlining). Therefore, we use the UNIX tool `SIZE`⁴⁰ to examine the size of the sections in the compiled binary images. For a given binary image, `SIZE` reports four values. The *text section size* (in B) is the size of the memory

³⁹ cMOFLON examples repository: <https://github.com/eMoflon/cmoflon-examples> (visited: 2018-10-16)

⁴⁰ `SIZE` page: <https://linux.die.net/man/1/size> (visited: 2018-10-31)

Table 5.6: Binary image size comparison (Δ_M^G : Manual-to-generated size increase)

Algorithm	Text[B]	Δ_M^G	Data[B]	Δ_M^G	BSS[B]	Δ_M^G	Total[B]	Δ_M^G
MP	37 843	–	234	–	9212	–	47 289	–
kTC _M	39 161	–	234	–	9224	–	48 619	–
kTC _G	41 091	4.9 %	242	3.4 %	9440	2.3 %	50 773	4.4 %
l*kTC _M	40 305	–	244	–	9474	–	50 023	–
l*kTC _G	42 587	5.7 %	252	3.3 %	9690	2.3 %	52 529	5.0 %
LMST _M	39 357	–	242	–	9436	–	49 035	–
LMST _G	41 745	6.1 %	244	0.8 %	9444	0.1 %	51 433	4.9 %

that is consumed by the machine code in the memory of the running system. The *data section size* (in B) is the size of the memory that is consumed by (string) literal values and initialized global variables. The *BSS section size* (in B) is the size of the memory that is consumed by uninitialized global variables. We measure the size of these sections for seven binary images: three binary images for the generated TC algorithms, three binary images for their manual counterparts, and one binary image with disabled TC as baseline.

RESULTS FOR BINARY IMAGE SIZE Table 5.6 provides an overview of the sizes of the compiled binary images for the baseline Maxpower algorithm (abbreviated as MP) and the three TC algorithms kTC, l*kTC, LMST (manual and generated). The table shows the total image size as well as the size of the text, binary, and BSS sections (in B). Each column that contains absolute size values is accompanied by a subsequent column labeled with Δ_M^G , which shows the increase in size for a generated TC algorithm compared to a manually implemented TC algorithm.

We observe that l*kTC has the largest manual-to-generated size increase (5.0%). Still, this increase is only 0.6 percentage points above the lowest manual-to-generated size increase for kTC.

DISCUSSION FOR BINARY IMAGE SIZE The preceding results indicate that the increase in binary image size is moderate and relatively homogeneous for all three considered TC algorithms. It is not surprising that LMST and l*kTC result in larger images compared to kTC. The former algorithm possess a considerably more complex control flow (e.g., comprising multiple EOperations) and the latter algorithm required additional source code for propagating the hop-count information to other notes.

METRICS FOR RUNTIME MEMORY CONSUMPTION The code generated by cMOFLON consumes static memory for storing matches of GT rules in regular story nodes. The required memory is determined and reserved at compilation time. Additionally, to store the auxiliary tree data structure for LMST, we used dynamic memory. The required heap memory varies with the actual size of the neighborhood of a mote.

RESULTS FOR RUNTIME MEMORY CONSUMPTION For all three TC algorithms, the maximum number of variables per pattern was seven. Each variable is represented by a pointer variable. On the TELOS platform, a pointer requires 2 B of memory. Therefore, the total memory consumption for the match data structure is 14 B for all considered TC algorithms.

Regarding the size of the tree data structure for the generated variant of LMST, the consumed heap memory consists of one tree object (instance of `Tree` in Figure 5.19) having a size of 8 B, and as many tree entries (instances of `TreeEntry`) as motes exist in the two-hop neighborhood. Each tree entry has a size of 10 B. In the 14 simulation runs, the total size of the tree data structure ranged from 128 B to 278 B with a median size of 218 B. These numbers correspond to a minimum, median, and maximum size of the two-hop neighborhood of 12, 21, and 27 motes, respectively.

DISCUSSION FOR RUNTIME MEMORY CONSUMPTION Given a total working memory of 48 kB on the TELOS platform, the memory consumption for matches and the tree of LMST is reasonable.

The memory efficiency of LMST could certainly be improved, for example, by encoding the state of each link (i.e., “in tree”, “connected to tree forward”, “connected to tree backward”) more concisely. Still, to establish comparability with the manual implementation of LMST in [239], we chose the same representation of the tree data structure in this case. Furthermore, in all evaluation runs, the heap was large enough to store the tree data structure, and the required memory was only reserved during the execution of LMST.

The global match data structure could also be placed in dynamic memory with only minor changes to the build process. However, for the considered TC algorithms, the amount of memory that could be saved in the course of this optimization (i.e., 14 B) could be outweighed by the resulting increased size of the text section because we would have to insert statements for allocating and deallocating the reserved dynamic memory, along with the necessary error handling if insufficient dynamic memory is available.

METRICS FOR EXECUTION TIME To evaluate the execution time, we measure the CPU time (in ms) that the execution of the run method requires on each mote.

RESULTS FOR EXECUTION TIME Figure 5.24 summarizes the execution time for the manual and generated variant of each TC algorithm. The differences in the number of

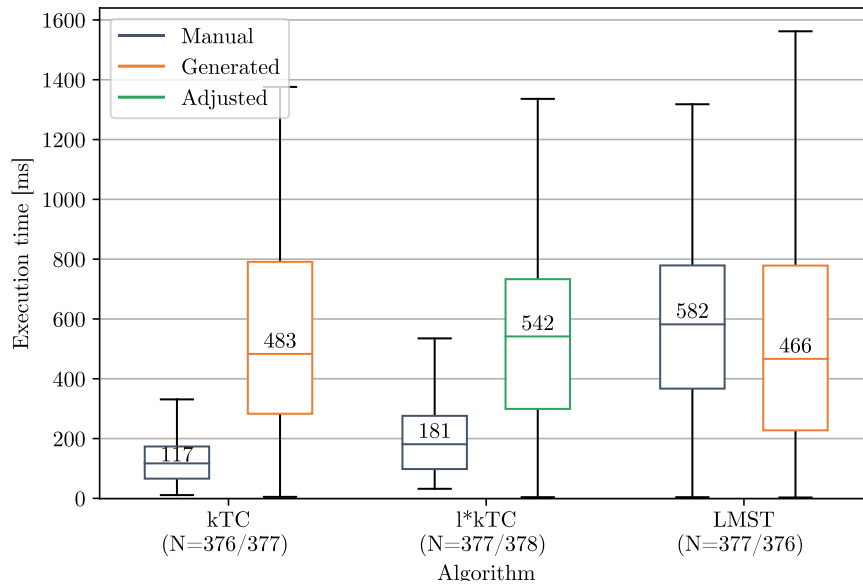


Figure 5.24: Execution time comparison of manual and generated TC algorithms (label per boxplot: median execution time, fourth boxplot: generated l*kTC after adjustment of search plan, **N**: number of data points per boxplot (left/right))

data points per boxplot in Figures 5.24 and 5.25 originate from the fact that the number of available motives during the experiments varied between 26 and 27.

A special case is the fourth boxplot, which corresponds to an adjusted variant of the generated l*kTC algorithm. In the following, we explain how we derived the adjusted variant of l*kTC from the generated l*kTC. During the evaluation of the original 84 experiments, we determined that the median execution time of the generated l*kTC was ca. 64 s, which is more than 350 times larger than the execution time of the manual variant of l*kTC (see Figure 5.25).

We observed that this excessive execution time originates from the fact that the l*kTC-specific predicate is evaluated early in the search plan generated by CMOFLON. The problem is that the auxiliary hop-count information is not stored inside the data structure that represents the Mote class but in a separate list. Each time that the hop count of a mote is required, this list is scanned using linear search to identify the list entry that belongs to this mote. The original search plan evaluated the l*kTC-specific predicate after matching the first and second of the three link variables of the triangle pattern because, at this point, all three motives of the potential match are bound.

Multiple solutions to this problem are possible. First, the hop-count information could be stored inside the `struct networkaddr_t`, which corresponds to the Mote class. This is an undesirable solution because it entails a modification of a core data structure of ToCoCo only for the purpose of supporting hop-count-based TC algorithms. Second,

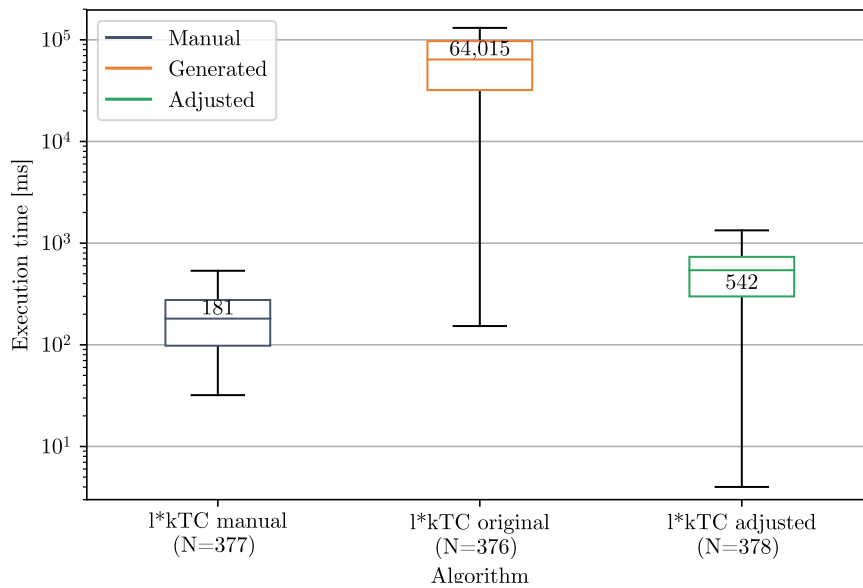


Figure 5.25: Execution time comparison of manual, generated, and adjusted l*kTC (logarithmic y axis, label per boxplot: median execution time, **N**: number of data points per boxplot)

the search plan generation algorithm of DEMOCLES allows to assign weights to each operation to signal that certain operations are costly. Unfortunately, these search plan weights cannot be influenced by the user, currently. Attribute constraints always have a low weight and are, therefore, evaluated as early as possible.

To investigate how much the execution time could be reduced if the search plan weights could be influenced, we moved the evaluation of the l*kTC-specific constraint to the end of the search plan. We executed the adjusted l*kTC algorithm in another batch of 14 experiments (without intermediate execution of the manual variant of l*kTC). The results are shown as the fourth boxplot in Figure 5.24.

In the median case, the adjusted variant of l*kTC requires 542 ms. This is 361 ms more than the manual variant of l*kTC, and ca. 118 times faster compared with the unmodified generated variant of l*kTC. A similar situation can be observed for kTC. In this case, the manual variant requires 366 ms more time than the manual variant in the median case.

Surprisingly, the execution time of the generated LMST algorithm is 116 ms smaller compared to the manual variant.

DISCUSSION FOR EXECUTION TIME The evaluation results concerning execution time are encouraging. The execution times for the generated kTC and l*kTC algorithm are larger compared to their manual counterparts. Still, this increase is moderate given

that the TC algorithms are executed in batch mode. We think that the execution time could be lowered even more in a (future) dynamic TC scenario with cMOFLON.

A key insight of the preceding results is that a highly desirable feature for cMOFLON (and eMOFLON in general) is to make search plan weights configurable. Otherwise, the user may be forced to modify the generated code manually to fix unfavorable decisions of the search plan generation or to change core data structures of the ToCoCo framework, which is certainly error prone.

ANSWERS TO RQ2 To sum up, we answer RQ2 as follows for the investigated three TC algorithms. The increase in the size of the compiled binary image ranges from 4.4% to 5.0%. This increase is relatively homogeneous and reasonable. Further improvements (e.g., the reduction of null-pointer checks) could lead to a decreased binary image size. With 14 B of permanently reserved global memory for all three TC algorithms, the generated code has a small memory footprint regarding static memory. A large enough amount of dynamic memory for the auxiliary data structures of LMST was available in all 14 runs of the generated LMST variant.

The generated kTC and l*kTC variants showed an increase and the generated LMST variant showed a decrease in execution time compared to the manual variants. In the former two cases, the absolute increase was in the order of several hundred milliseconds, which is reasonable for batch TC. For l*kTC, a manual adjustment of the search plan was necessary due to the missing configuration options for search plan weights in eMOFLON.

THREATS TO VALIDITY A major threat to external validity of our results is that we evaluated the efficiency of the generated code only using one sensor platform (i.e., TELOS_B) and one hardware testbed (i.e., FLOCKLAB). Still, we think that both choices are representative. TELOS_B is a rather resource-constrained mote platform, has been first proposed in 2005, and is still widely used [194]. These observations allow us to generalize our findings regarding the memory consumption and the execution time of the TC algorithms generated using cMOFLON. Our observation is also that FLOCKLAB is heavily used by the research community and, therefore, constitutes a representative (research) testbed.

A major threat to internal validity is that the input topology of the testbed varies continuously (e.g., due to moving obstacles), which could lead to unsystematic errors in the measurements. To mitigate this threat, we evaluated each TC algorithm repeatedly (i.e., 14 times) over a period of four days.

5.4 RELATED WORK

In this section, we survey related work first on existing approaches to integrate modeling tools with network simulators or to use GT tools for network simulation and, then, on code generation approaches for WSNs.

5.4.1 *Network simulation and model-driven engineering*

Kulcsár et al. presented a predecessor of COBOLT in [130]. Their tool served to evaluate properties of virtual topology snapshots resulting from a TC algorithm specified using EMOFLON. Their focus was on highlighting that MDE is suitable to prototype novel TC algorithms rapidly. In contrast to COBOLT, their tool was not able to play back the virtual topology into a running simulation. Instead, they captured snapshots of the input topology from a running simulation in PEERFACTSIM.KOM [242] and transformed this snapshot once into a corresponding EMF-based model. This model was then processed by the generated TC algorithm prototype.

Several works in the GT research area evaluate their approaches based on communication and cyber-physical systems. Our impression is that many of these works represent few aspects of the system to highlight the capabilities of their modeling approach. For example, in [157], Maximova et al. use the scenario of small communicating autonomous rail-mounted vehicles to illustrate how timed probabilistic GT can be used to estimate the probability of emergency breaks. We discussed this example already in detail in Section 4.6. Still, their goal was not to use their approach for simulating larger networks or to derive implementations for hardware testbeds.

In the line of research [112, 113] that led to his Ph.D. thesis [111], Ajab Khan developed a formal and technical framework for the stochastic simulation of voice-over-IP overlay communication networks using graph transformation. Their graph-based network model represents the clients (resource-constrained peers and powerful super-peers) and their connections as model. They employ stochastic GT rules [90] to model context events (e.g., peers joining and leaving the network). Instead of connecting to an existing simulator, they build a standalone simulation environment by modeling all relevant aspects of the system as GT rules. Technically, they use the stochastic GT simulation engine GRASS (Graph-based Stochastic Simulation [256]), whose development has been discontinued. In contrast to Khan et al., we decided to connect to an established network simulator instead of trying to model all system dynamics using GT rules. Our first reason for this is that developing algorithms for the simulation of underlay dynamics (e.g., interference, physical obstacles, mote movement) is an elaborate task. The resulting algorithms should be thoroughly tested. For example, SIMONSTRATOR reuses underlay models from the NS-3 simulator [253]. Another related reason is that we expected that the acceptance of our tool support would be larger if we use a network simulator that is used in publications in communication systems engineering. A third

reason is that SIMONSTRATOR is being extended continuously with functionality such as the batch execution of experiments, the visualization of the network topology, and the plotting of metrics. Therefore, we decided to only represent the business logic of TC algorithms using programmed GT and, for all other aspects, built on an existing simulation infrastructure.

Real-time MAUDE [171] is a prominent example of a simulator for communication systems. In a case study, Ölveczky et al. could show that MAUDE produces comparable or even more reliable results for the simulation of coverage-based TC algorithms [152]. Similar to using GT rules for specifying TC algorithms, MAUDE uses rewrite rules to model transitions between system states in general. These transitions can be further annotated with timing constraints to allow for evaluating, for instance the minimum or maximum execution time of a trace of transitions. Even though MAUDE appears to be suitable to simulate WSNs, we are not aware of any works that allow for refining a MAUDE specification constructively based on weakest preconditions to ensure that required consistency properties hold. This property, however, is important for our scenario. As a consequence, MAUDE can be understood as complementary tool to the tool support that we describe in this chapter.

5.4.2 Code generation for wireless sensor networks

A recent systematic mapping study summarizes 21 MDE approaches for WSNs: 15 approaches allow generating code for C/C++ or NeSC and 11 approaches support topologies, but none allows to explicitly specify TC algorithms [58]. Most MDE approaches focus on architectures for WSNs.

The SCATTERCLIPSE [4] framework is an ECLIPSE-based toolkit for generating and testing WSN applications. In SCATTERCLIPSE, the developer specifies the implementation of a WSN algorithm as activity diagram. In contrast to SDM, the actions of these activity diagrams correspond to low-level instructions (e.g., the extraction of a sensor sample) rather than applications of GT rules. Still, SCATTERCLIPSE has a formal foundation because it uses only those modeling languages of fUML, the foundational UML, which is a subset of the UML for which formal semantics have been standardized.

The AGILLA [20] framework is an agent-based MDE platform for WSNs. As in SCATTERCLIPSE, AGILLA provides an instruction-based view of the specification of the developed WSN algorithm.

The SAMSON framework [184] provides an architecture description language of a WSN and allows to generate code for CONTIKI. All of these approaches (would) represent TC as a software component, concealing the concrete TC implementation. This makes our approach complementary to many existing MDE middleware approaches: In this context, TC can be seen as a service component that should be configured according to the demands of the active application (e.g., concerning robustness, path lengths). As part of the SAMSON tooling environment, a code generator for the WSN operat-

ing system CONTIKI has been developed. To the best of our knowledge, no other MDE approaches for generating code of TC algorithms exist in the literature.

In [12], a C code generator for the Graph Programming Language 2 (GP2) [11] is presented. Similar to DEMOCLES [260], the code generation and pattern matching back-end of EMOFLON, GP2 transforms the graph patterns into a depth-first search plan with matching operations. Similar to cMOFLON, each operation corresponds to a particular (hierarchical) code template and the pattern matching is rooted, that is, the pattern matching starts at a given model element to limit the search space. In contrast to GP2, cMOFLON uses story diagrams for specifying the control flow. Story diagrams support the invocation of arbitrary user-defined operations, and the search plan generation can be easily configured using modules for search plan weights and strategies [260]. While EMF has emerged as de-facto target language for MDE tools, a number of tools support other target platforms (e.g., GRGEN.NET [74] for the .Net platform or PROGRES [225] for Modula2 and C code). EMF4CPP [57]⁴¹ aimed to provide full support for creating C++ code from EMF models, but appears to be discontinued.

Bur et al. [29] implemented a C++ code generator for executing graph queries specified in VIATRA in a distributed manner. They built the MoDES3 demonstrator for safety-critical cyber-physical systems [262]. The system consists of a simulated train network with resource-constrained embedded devices. Each embedded device is responsible for monitoring safety properties in a certain region. As target devices, they used ARDUINO⁴², RASPBERRY PI⁴³, and BEAGLEBOARD⁴⁴.

In [229], Schwichtenberg et al. present a prototype of the code generation framework CROSSCORE. CROSSCORE generates source code from an Ecore metamodel for the target languages C#, SWIFT, TYPESCRIPT, and JAVASCRIPT. CROSSCORE allows to exchange the code templates for classes, OCL constraints, and unit tests. In contrast to SDM in EMOFLON, the user cannot specify operation implementations in a platform-independent manner.

To sum up, we know neither of a tool for generating embedded C code from programmed GT nor of an MDE methodology that constructively integrates integrity properties into the development of TC algorithms and targets the evaluation of the resulting TC algorithms in simulation and testbed environments.

⁴¹ Website: <https://github.com/catedrasaes-umu/emf4cpp>

⁴² ARDUINO page: <https://www.arduino.cc/> (visited: 2018-09-19)

⁴³ RASPBERRY PI page: <https://www.raspberrypi.org/> (visited: 2018-09-19)

⁴⁴ BEAGLEBOARD page: <https://beagleboard.org/> (visited: 2018-09-19)

DEVELOPMENT OF FAMILIES OF TOPOLOGY CONTROL ALGORITHMS

In the preceding chapters, we discussed and showed which modeling techniques are suitable to specify the building blocks of a TC mechanism (Chapter 3), how the existing constructive approach and our novel anticipation loop synthesis algorithm can be used to obtain correct-by-construction TC mechanisms (Chapter 4), and how the specified TC mechanisms can be evaluated rapidly in a network simulator and hardware testbeds (Chapter 5). In this chapter, we show that our approach is applicable based on a set of 32 TC algorithms. The majority of these TC algorithms originates from the survey articles [219, 267], the book [264], and the TC comparison paper [66]. We selected all major TC algorithms and some of the presented variants from the aforementioned works. The set of TC algorithms is complemented by further existing variants of kTC (l-kTC [130], g-kTC [130], l*kTC [239]), and e-kTC, an energy-aware variant of kTC that we presented first in [119]. Figure 6.1 locates the role of this chapter in the entire TC algorithm development process (see also Figure 1.2).

In [219, 264, 267], TC algorithms are categorized based on the required information (e.g., as neighbor-based or location-based TC algorithms). We observed that TC algorithms also form families based on common structural characteristics of their local consistency properties (e.g., a triangle pattern). The individual TC algorithms in a family refine this family constraint. To the best of our knowledge, this observation has not been captured systematically before. Therefore, as one contribution of this chapter, we

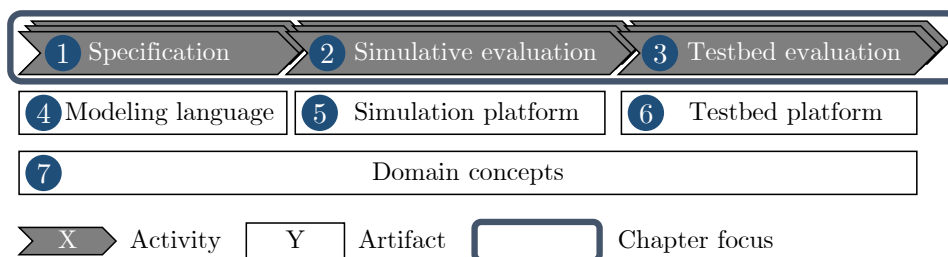


Figure 6.1: Location of Chapter 6 in TC algorithm development process

propose to specify the variability within a family of TC algorithms using feature models (see also Section 3.6). This representation also enables the derivation of TC transitions at runtime in adaptive WSNs (see also Definition 2.34).

The structure of this chapter follows the same steps as in Chapters 2 and 3. In Section 6.1, we present the considered TC algorithms. Due to the large number of TC algorithms, we only present and discuss a subset of these TC algorithms. For the selected TC algorithms, we state the TC-algorithm-specific conditions in terms of first-order logic with arithmetic expressions in Section 6.2. We show that the conditions of certain TC algorithms can be formulated as conjunction of two parts: a common (structural) condition and a condition that is specific to the particular TC algorithm. In Section 6.3, we show how feature models and extensions of graph constraints allow us to represent the relations of TC algorithms within a family. In Section 6.4, we leverage the commonalities within a TC algorithm family to prove the preservation of connectivity jointly for all considered triangle-based TC algorithms. After illustrating the applicability of our methodology based on a larger set of TC algorithms, we discuss which types of TC algorithms can be developed using our methodology in general (Section 6.5). We conclude this chapter with a survey of related work on variability modeling in communication systems engineering (Section 6.6).

RELATED PUBLICATIONS AND THESES In [119], we presented the approach discussed in this chapter based on a reduced set of TC algorithms. Maximilian Herbst investigated further TC algorithms and proposed a fine-grained categorization into triangle- and cone-based TC algorithms in his Bachelor's thesis [93].

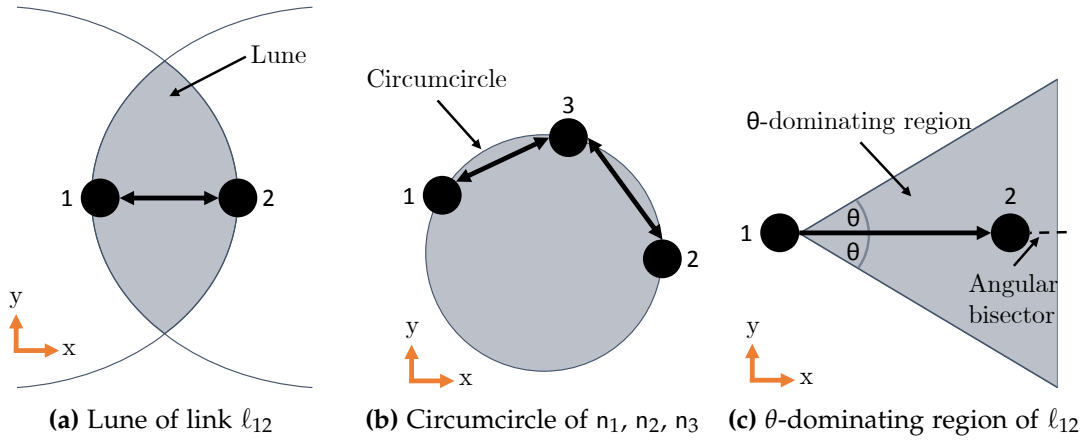


Figure 6.2: Illustration of lune, circumcircle, and θ -dominating region

6.1 LANDSCAPE OF TOPOLOGY CONTROL ALGORITHMS

In this section, we describe how we determined the set of TC algorithms and grouped them into families. We selected all major TC algorithms and variants of these major TC algorithms from the [66, 219, 264, 267]. Finally, we added the following variants of the kTC [227] algorithm: l-kTC [130], g-kTC [130], l*kTC [239], e-kTC [119]. In total, we collected 32 TC algorithms.

Figure 6.3 and Table 6.1 provide a visual and a detailed summary of the collected TC algorithms, respectively. The following definitions introduce concepts that occur for the first time in Table 6.1.

Definition 6.1 (Lune). Let n_1 and n_2 be two nodes that are embedded in the Euclidean plane and connected by a link ℓ_{12} . The *lune of a link ℓ_{12}* is the intersection of the two circles that are centered at the nodes n_1 and n_2 , respectively, and have a radius equal to the length of ℓ_{12} (Figure 6.2a). \square

Definition 6.2 (Circumcircle of three nodes). Let n_1 , n_2 , and n_3 be three nodes that are embedded in the Euclidean plane and form a connected graph. The *circumcircle of nodes n_1 , n_2 , n_3* is the unique circle that runs through n_1 , n_2 , and n_3 (Figure 6.2b). \square

Definition 6.3 (θ -dominating region). For a given angle $\theta \in [0, 180^\circ]$, the *θ -dominating region* of a link ℓ_{xy} is the 2θ -cone emanating from n_x and having ℓ_{xy} aligned with its angular bisector (Figure 6.2c). \square

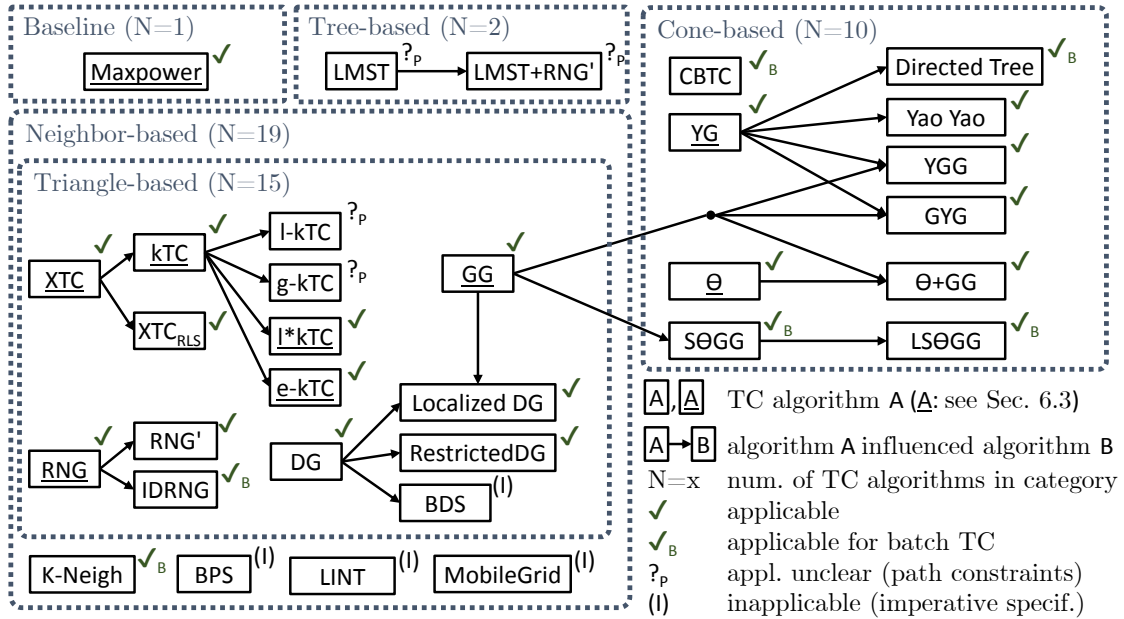


Figure 6.3: Visualization of landscape of collected TC algorithms

According to [219, 267], the collected algorithms can be categorized into three groups: 19 neighbor-based, two tree-based, and ten cone-based TC algorithms. The Maxpower algorithm is not part of any of these categories because it activates all links unconditionally. We observed that a subset of 15 neighbor-based TC algorithms use a triangular pattern in their specification. Regarding the applicability of our approach, we made the following observations. We found that our approach supports the development of batch and dynamic variants of 17 TC algorithms (marked with a green check mark, \checkmark). For further six TC algorithms only batch TC is supported (marked with a green check mark bearing a subscript “B”, \checkmark_B). We explain the reasons for this categorization later in Section 6.5.

Our methodology is inapplicable to eight TC algorithms due to the following two reasons. The first reason is that certain TC algorithms are presented using a purely imperative characterization from which we could not extract a declarative graph-based specification of a valid virtual topology (marked with (I)). This applies to four TC algorithms in total: BDS, BPS, LINT, and MobileGrid. The second reason is that certain algorithms comprise calculations that evaluate path conditions (e.g., to calculate exact routing path lengths, marked with $?_P$). This reason applies to four TC algorithms: I-kTC, g-kTC, LMST, and LMST+RNG'. In Section 6.2, we will discuss nine of the fully compatible TC algorithms (underlined in Figure 6.3 and Table 6.1). In Section 6.5, we explain why our approach is partly applicable to six and inapplicable to eight TC algorithms.

Table 6.1: Details of TC algorithm landscape (**Algorithm:** A: discussed in Section 6.2. [X]: Ref. to origin. **Type:** N: neighbor-based, Δ : triangle-based, T: tree-based/sink structures, C: cone-based. **Description:** [\checkmark , \checkmark_B , $?_P$, (I)]: applicable, applicable for batch TC, applicability unclear due to path constraints, inapplicable due to imperative specification.)

Algorithm	Type	Description
<u>Maxpower</u> [66]	–	[\checkmark] Activates all links. Common baseline. Also called all-links graph (ALG). Details: Section 6.2.2.1.
Triangle-based algorithms		
<u>RNG</u> [109] [219, 264, 267]	Δ	[\checkmark] Relative neighborhood graph. Inactivates a link ℓ_{XY} if a mote n_Z in the lune of n_X and n_Y exists. Details: Section 6.2.2.3.
<u>RNG'</u> [141] [267]	Δ	[\checkmark] Modified relative neighborhood graph. Inactivates a link if it fulfills the RNG-specific condition <i>and</i> if there is no mote n_Z on the boundary of the lune such that (i) $\text{id}(n_Z) < \text{id}(n_X)$ and $w(\ell_{XZ}) < w(\ell_{XY})$, or (ii) $\text{id}(n_Z) < \text{id}(n_Y)$ and $w(\ell_{YZ}) < w(\ell_{XY})$, or (iii) $\text{id}(n_Z) < \min(\text{id}(n_X), \text{id}(n_Y))$ and $w(\ell_{XZ}) < w(\ell_{XY})$
<u>IDRNG</u> [34] [66]	Δ	[\checkmark_B] Inclusive directed RNG. Inactivates a link ℓ_{12} only if no longer link exists that must be active.
<u>XTC</u> [270] [66, 219, 264]	Δ	[\checkmark] Inactivates the weight-maximal link in each triangle. Produces subgraph of RNG that is equal to RNG in the absence of ties. ID-based tie breaking. Details: Section 6.2.2.4
<u>XTC_{RLS}</u> [66]	Δ	[\checkmark] XTC with restricted link strength (RLS). Inactivates a link if it is inactivated by XTC and has a worse signal-to-noise ratio than r . The parameter r is configurable and set to -86 dB in [66] based on experiments.
<u>kTC</u> [227] [66]	Δ	[\checkmark] Inactivates a link if it is the weight-maximal link in a triangle and if its weight is at least k times larger than the minimal weight in the same triangle. The parameter k is configurable in the range $[1, \sqrt{2})$. Running example. Details: Sections 2.4.4 and 6.2.2.5.

Table 6.1 (continued): Details of TC algorithm landscape (**Algorithm:** A: discussed in Section 6.2. [X]: Ref. to origin. **Type:** N: neighbor-based, Δ : triangle-based, T: tree-based/sink structures, C: cone-based. **Description:** [\checkmark , \checkmark_B , $?_P$, (I)]: applicable, applicable for batch TC, applicability unclear due to path constraints, inapplicable due to imperative specification.)

Algorithm	Type	Description
l-kTC, g-kTC [130, 238]	Δ	[$?_P$] Inactivates link l_{12} if it fulfills the kTC-specific condition and if inactivation does not increase routing path length of n_1, n_2 beyond a stretch factor a . Increase in routing path length evaluated based on the assumption that only the current candidate link (l-kTC) or all candidate links (g-kTC) are inactive. Parameter a is configurable. Both algorithms require global view of topology.
l*kTC [239]	Δ	[\checkmark] Localized version of l-kTC that estimates the increase in routing path length based on the hop counts of motes in the neighborhood. Running example in Section 5.3. Details: Sections 5.3.2.1 and 6.2.2.6.
e-kTC [119]	Δ	[\checkmark] An energy-aware variant of kTC. Uses the inverse estimated remaining lifetime of a mote as weight. Inspired by CTCA [35]. Details: Section 6.2.2.9.
GG [211] [66, 219, 264, 267]	Δ	[\checkmark] Gabriel graph. Inactivates a link l_{12} if the circle with diameter l_{12} contains a mote n_3 . Details: Section 6.2.2.2.
DG [48] [219, 264, 267]	Δ	[\checkmark] Delaunay graph. Inactivates a link l_{12} if the mote n_2 is inside the circumcircle of a triangle that n_1 is part of. Requires global information and position information of each mote [267, p.119].
Localized DG [142] [264, 267]	Δ	[\checkmark] Localized version of DG. Only considers Delaunay triangulation of neighborhood. Activates all links that GG activates.
Restricted DG [72] [267]	Δ	[\checkmark] Localized version of DG. Activates a link l_{12} if l_{12} is in the Delaunay triangulations of n_1 and n_2 . Two steps: Determining local Delaunay triangulation and communicate the decisions to all neighbors to ensure symmetric links.

Table 6.1 (continued): Details of TC algorithm landscape (**Algorithm:** A: discussed in Section 6.2. [X]: Ref. to origin. **Type:** N: neighbor-based, Δ : triangle-based, T: tree-based/sink structures, C: cone-based. **Description:** [\checkmark , \checkmark_B , $?_P$, (I)]: applicable, applicable for batch TC, applicability unclear due to path constraints, inapplicable due to imperative specification.)

Algorithm	Type	Description
Neighbor-based algorithms		
BDS [25] [267]	N	[(I)] Bounded-degree spanner algorithm based on Delaunay triangulation. Ensures that each mote has degree of at least three and that routing path lengths are bounded. Requires global position information of motes. Pseudocode specification.
BPS [268] [267]	N	[(I)] Bounded-degree spanner algorithm based on Delaunay triangulation. Localized algorithm. Pseudocode specification.
k -Neigh [24] [66, 219, 264]	N	[\checkmark_B] Activates the k nearest neighbors. Optionally, asymmetric links can be removed. No correctness guarantee w.r.t. connectivity possible, but experiments show that, for $k = 9$, the virtual topology is connected with high probability.
MobileGrid [151] [219]	N	[(I)] Adjusts the transmission range of a mote to keep the number of motes within the transmission range (called the <i>contention index</i>) around a desired value extracted from a lookup table. Lookup table is populated based on simulation experiments. No proof of correctness (e.g., w.r.t. connectivity) is provided.
LINT [201] [219]	N	[(I)] Centralized algorithm with TC as constrained optimization problem with (bi-)connectivity as constraints and minimizing the energy consumption as performance goal. Requires global knowledge. No local characterization of consistency. Our methodology is inapplicable.
Cone-based algorithms		
YG [146] [66, 219, 264, 267]	C	[\checkmark] Yao graph. Separates area around mote into c equally-separated cones and activates only the weight-minimal outgoing link in each cone. Details: Section 6.2.2.7

Table 6.1 (continued): Details of TC algorithm landscape (**Algorithm:** A: discussed in Section 6.2. [X]: Ref. to origin. **Type:** N: neighbor-based, Δ : triangle-based, T: tree-based/sink structures, C: cone-based. **Description:** [\checkmark , \checkmark_B , $?_P$, (I)]: applicable, applicable for batch TC, applicability unclear due to path constraints, inapplicable due to imperative specification.)

Algorithm	Type	Description
Reverse Yao [146] [66, 219, 264, 267]	C	[\checkmark] Variant of YG that considers incoming instead of outgoing links.
GYG [146] [267]	C	[\checkmark] Applies YG, then applies GG.
YGG [146] [267]	C	[\checkmark] Applies GG, then applies YG. In [267], the imperative algorithms for obtaining YGG are called OrdYaoGG, SYaoGG.
Directed tree [145] [267]	C	[\checkmark_B] First calculates the Yao graph in the local view of a mote n_1 having c cones. Then, creates directed tree of a Yao graph by recursively investigating each neighbor mote n_2 in the Yao graph as follows. Divides the area around n_2 into c cones, adds the nearest neighbor n_3 in each cone, and inactivates the link ℓ_{13} (if active).
θ -graph [192] [264, 267]	C	[\checkmark] Separates area around mote into c equally-separated cones (of angle $\theta = \frac{360^\circ}{c}$). Activates the link with the weight-minimal projection on the angular bisector in each cone. Configurable parameter c . Details: Section 6.2.2.8.
θ -graph+GG [269] [267]	C	[\checkmark] Applies θ -graph, then apply GG.
Yao Yao [206] [219, 267]	C	[\checkmark] Combination of YG and Reverse Yao to bound in- and outdegree. Also called Sparsified Yao.
Symmetric Yao [144] [267]	C	[\checkmark] Activates link ℓ_{12} only if its reverse link ℓ_{21} is also active according to YG on n_2 .
S θ GG [143] [267]	C	[\checkmark_B] Processes links in increasing order of unique weight. Activates link if it is not covered by θ -dominating region of already active link. Configurable parameter θ .
LS θ GG [143]	C	[\checkmark_B] Sparsens topology of S θ GG by inactivating links in mote quadruples in 2-hop neighborhood. Link processing order relevant.

Table 6.1 (continued): Details of TC algorithm landscape (**Algorithm:** A: discussed in Section 6.2. [X]: Ref. to origin. **Type:** N: neighbor-based, Δ : triangle-based, T: tree-based/sink structures, C: cone-based. **Description:** [\checkmark , \checkmark_B , $?_P$, (I)]: applicable, applicable for batch TC, applicability unclear due to path constraints, inapplicable due to imperative specification.)

Algorithm	Type	Description
CBTC [139] [219, 264]	C	[\checkmark_B] Cone-based TC. Imperative. Processes links in increasing order of unique weight. Activates links until θ -cones of active links jointly cover angular range of the mote. Configurable parameter θ . Also called cone-covering graph.
Tree-based algorithms		
LMST [140] [66, 219, 267]	T	[$?_P$] Local minimum spanning tree. Constructs minimum spanning tree in local view. Running example in Section 5.3. Applicability unclear due to acyclicity constraints.
LMST+RNG' [147] [267]	T	[$?_P$] Local minimum spanning tree on top of an RNG'. Same considerations regarding graph constraints as for LMST. Applicability unclear due to acyclicity constraints.

6.2 SPECIFICATION OF TOPOLOGY CONTROL ALGORITHMS

In this section, we first present additional element properties that are necessary to understand the nine TC algorithms that we discuss afterwards in detail.

6.2.1 Element properties and context events

The algorithms in Table 6.1 partly require access to element properties that we introduce now.

PROPERTY DEFINITIONS Cone-based TC algorithms require estimations of the (relative) angles of their neighbors. Table 6.1 contains ten cone-based TC algorithms. Usually, the angle of a neighbor is based on the location of a mote, but can also be determined using directional antennas. Neighbor-based TC algorithms may also use location information to improve estimations of link weights. The *position* $\text{pos}(n_1)$ of a mote n_1 is a pair $(\text{lng}(n_1), \text{lat}(n_1))$ consisting of the real-valued *longitude* $\text{lng}(n_1)$ (also known as x coordinate) and the real-valued *latitude* $\text{lat}(n_1)$ (also known as y coordinate) of n_1 . In the following, we denote the longitude $\text{lng}(n_1)$ and latitude $\text{lat}(n_1)$ of a mote n_1 concisely as lng_1 and lat_1 .

The *angle* $a(\ell_{12})$ of a link ℓ_{12} can be calculated if the positions of its incident motes are known. In the following, we denote the angle $a(\ell_{12})$ of a link ℓ_{12} concisely as a_{12} .

$$a_{12} = \arctan \left(\frac{\text{lat}_1 - \text{lat}_2}{\text{lng}_1 - \text{lng}_2} \right).$$

A cone-based TC algorithm usually calculates the weight of a link from the distance of its incident motes because information about mote positions is available anyway. The *length* $\text{len}(\ell_{12})$ of a link ℓ_{12} can be calculated if the positions of its incident motes are known. We use the Euclidean distance of motes as an estimate for the actual geographic distance (i.e., the distance on the globe) because the maximum length of a link is bounded by the maximum transmission radius of a mote (e.g., a few hundred meters for WiFi). Therefore, using the Euclidean distance instead of the geographic distance is usually sufficient. In the following, we denote the length $\text{len}(\ell_{12})$ of a link ℓ_{12} concisely as len_{12} .

$$\text{len}(\ell_{12}) = \sqrt{(\text{lng}_1 - \text{lng}_2)^2 + (\text{lat}_1 - \text{lat}_2)^2}.$$

Energy-aware TC algorithms require information about the remaining energy of a mote. The only energy-aware TC algorithm in Table 6.1 is e-kTC. The real-valued *energy* E_n of a mote n is the remaining energy stored in the battery of n . In the following, we usually neglect the concrete unit.

The application-aware TC algorithm l*kTC [239] is applicable in scenarios with a dedicated data sink (e.g., a base station) The integer-valued *hop count* $h(n_1)$ of a mote n_1 is the

number of hops that are required to reach the base station. If no hop-count information is available for a mote n_1 (e.g., because the network is disconnected), we assume that $h(n_1) < 0$. In the following, we denote the hop count $h(n_1)$ of a mote n_1 concisely as h_1 .

METAMODEL EXTENSIONS The element properties introduced in the preceding paragraphs can be represented in the topology metamodel in a similar way as the mote identifier and link weight properties introduced in Section 2.2. We represent (i) the mote position as two attributes `Mote::longitude` and `Mote::latitude` of type `double`, (ii) the mote hop count as the attribute `Mote::hopCount` of type `int` (as in Figure 5.19), (iii) the mote energy as the attribute `Mote::energy` of type `double`, and (iv) the link length and angle as the attributes `Link::length` and `Link::angle` of type `double`. For conciseness reasons, we do not show the extended metamodel at this point.

CONTEXT EVENTS For each new property, we introduce corresponding context events rules and context event handler story diagrams. The context events for mote longitude, latitude, and energy are immediate context events, whereas the context events for mote position, link length, and link angle are all derived from modifications of the longitude and latitude of a mote. We assume that the hop-count attribute is updated when necessary by the routing.

The position and energy level of a mote typically change continuously. We assume that the corresponding context events are detected by the topology monitoring at discrete points in time and indicated as context event markers.

6.2.2 Presentation of topology control algorithms

Next, we provide a formal specification for each of the nine of 24 TC algorithms to which our approach is applicable (Sections 6.2.2.1 to 6.2.2.9). As in Chapter 3, we state for each TC algorithm the condition that determines when a link may be inactive in the virtual topology using first-order logic with arithmetic expressions. In Section 6.2.2.10, we additionally introduce the generic weight-based predicate $\phi_{\text{min-weight}}$. This predicate is inspired by XTC_{RLS} [66] and can be combined freely with all discussed TC algorithms.

6.2.2.1 Maxpower algorithm (revisited)

For completeness, we shortly revisit the TC algorithm `Maxpower`, which inactivates no links. Therefore, the set of inactive links in the virtual topology G_T of `Maxpower` can be characterized using first-order logic with arithmetic expressions as follows:

$$\forall G_T = (V_T, E_T) : \forall \ell_1 \in E_T : s(\ell_1) = I \Leftrightarrow \text{false}.$$

6.2.2.2 Gabriel graph algorithm

The virtual topology of the Gabriel Graph (GG) algorithm [211, 267] is built as follows. A link l_1 is inactive in the virtual topology of GG if and only if the circle with diameter l_1 and center between n_1 and n_2 contains no motes apart from n_1 and n_2 [70]. The formulation is location-based because each mote requires knowledge about its latitude and longitude to assess whether another mote is inside the described circle. Using Thales' theorem [3, p. 50], we obtain the following equivalent formulation that only uses link weights. A link is inactive if it is part of a triangle and its squared weight is smaller than the sum of the squared weights of the other links l_2 and l_3 . The set of inactive links in the virtual topology G_T of GG can be characterized using first-order logic with arithmetic expressions as follows:

$$\begin{aligned} \forall G_T=(V_T, E_T) : \forall l_1 \in E_T : \\ s(l_1) = \mathbf{I} \Rightarrow \exists l_2, l_3 \in E_T : \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{\text{GG}}(l_1, l_2, l_3) \\ \text{with} \\ \varphi_{\text{GG}}(l_1, l_2, l_3) = w_1^2 > w_2^2 + w_3^2. \end{aligned} \quad (6.1)$$

In the preceding condition, we reuse the *triangle predicate* $\varphi_{\text{triangle}}$ (see Equation (3.3)), which takes three links as parameters and evaluates to *true* if these links form a triangle. As a reminder, we repeat the definition of $\varphi_{\text{triangle}}$ here:

$$\varphi_{\text{triangle}}(l_1, l_2, l_3) \Leftrightarrow \text{src}(l_1) = \text{src}(l_2) \wedge \text{src}(l_3) = \text{trg}(l_2) \wedge \text{trg}(l_3) = \text{trg}(l_1).$$

6.2.2.3 Relative neighborhood algorithm

In the virtual topology of the Relative Neighborhood Graph (RNG) algorithm [109, 267], the lune (Definition 6.1) of each active link l_{12} contains no motes. Under the assumption that the triangle inequality holds for the link weight attribute, we can reformulate this condition as follows. A link l_1 is inactive in the virtual topology of RNG if it is the weight-maximal link in a triangle (l_1, l_2, l_3) in which l_2 and l_3 have smaller weights. The set of inactive links in the virtual topology G_T of RNG can be characterized using first-order logic with arithmetic expressions as follows:

$$\begin{aligned} \forall G_T=(V_T, E_T) : \forall l_1 \in E_T : \\ s(l_1) = \mathbf{I} \Rightarrow \exists l_2, l_3 \in E_T : \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{\text{RNG}}(l_1, l_2, l_3) \\ \text{with} \\ \varphi_{\text{RNG}}(l_1, l_2, l_3) = w_1 > \max(w_2, w_3). \end{aligned} \quad (6.2)$$

6.2.2.4 XTC algorithm

The idea behind XTC [270, Sec. 3] is that a large link weight indicates a low link quality. A link l_1 is inactive in the virtual topology of XTC if and only if two links of higher

quality (i.e., smaller weight) exist that connect the source with the target of ℓ_1 , possibly via multiple intermediate links. This is equivalent to the following formulation. A link in the output topology of XTC is inactive if it is the weight-maximal link in a triangle. In [270, Sec. 4], the authors refine the formulation of XTC based on unique link weights (see also Definition 2.24). The set of inactive links in the virtual topology G_T of XTC can be characterized using first-order logic with arithmetic expressions as follows:

$$\begin{aligned} \forall G_T=(V_T, E_T) : \forall \ell_1 \in E_T : \\ s(\ell_1) = \mathbf{I} : \exists \ell_2, \ell_3 \in E_T : \varphi_{\text{triangle}}(\ell_1, \ell_2, \ell_3) \wedge \varphi_{\text{XTC}}(\ell_1, \ell_2, \ell_3) \\ \text{with} \\ \varphi_{\text{XTC}}(\ell_1, \ell_2, \ell_3) :\Leftrightarrow w'_1 > \max(w'_2, w'_3). \end{aligned} \quad (6.3)$$

This condition is apparently similar to the condition of RNG. This may lead to the impression that the output topology of XTC is identical to RNG. However, this is true only if link weights are unique because RNG applies the $>$ -operator based on (raw) link weights while XTC applies the $>$ -operator for unique link weights.

6.2.2.5 kTC algorithm (revisited)

We already introduced kTC [227] in Section 2.4.4. Still, based on this formulation of the XTC predicate in Equation (6.3), we can rewrite the kTC-specific predicate φ_{kTC} as a refinement of φ_{XTC} as follows:

$$\varphi_{\text{kTC}}(\ell_1, \ell_2, \ell_3) :\Leftrightarrow \varphi_{\text{XTC}}(\ell_1, \ell_2, \ell_3) \wedge w'_1 > k \cdot \min(w'_2, w'_3).$$

This reformulation of φ_{kTC} explicates the following relation between kTC and XTC. Each link that is inactivated by kTC is also inactivated by XTC and each link that is activated by XTC is also activated by kTC.

6.2.2.6 l*kTC algorithm (revisited)

We introduced l*kTC in Section 5.3.2.1 and repeat its description here briefly to discuss its relation to the other triangle-based TC algorithms. l*kTC [239] is an application-aware variant of kTC that is tailored to many-to-one communication scenarios (e.g., data collection). In such scenarios, the hop count h_1 of each mote n_1 is defined as the number of hops on the routing path to the base station (see also Section 6.2.1). The intention behind the l*kTC algorithm is to inactivate only a subset of the links that would be inactivated by kTC to avoid stretching routing paths excessively. A link ℓ_1 is inactive in the virtual topology of l*kTC if (i) ℓ_1 fulfills the kTC-specific condition, and (ii) inactivating ℓ_1 keeps the increase in the hop count of the incident motes of ℓ_1 below a configurable *stretch factor* a compared with the input topology. Evaluating the second condition precisely requires global knowledge to determine in how far the routing path to the base station changes due to an inactivation of ℓ_1 . To keep the required knowledge

local, l*kTC estimates the increase in routing path length based only on the triangle that fulfills the first, kTC-specific condition. The set of inactive links in the virtual topology G_T of l*kTC can be characterized using first-order logic with arithmetic expressions as follows:

$$\forall G_T=(V_T, E_T) : \forall l_1 \in E_T : \\ s(l_1) = I \Rightarrow \exists l_2, l_3 \in E_T : \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{l^*kTC}(l_1, l_2, l_3)$$

with

$$\begin{aligned} \varphi_{l^*kTC}(l_{XY}, l_{XZ}, l_{ZY}) : \Leftrightarrow \\ & \varphi_{kTC}(l_{XY}, l_{XZ}, l_{ZY}) \\ & \wedge \min(h_X, h_Y, h_Z) \geq 0 \\ & \wedge (h_X = h_Y \Rightarrow \text{true}) \\ & \wedge \left(h_X \neq h_Y \Rightarrow \frac{h_Z + 1}{\max(h_X, h_Y)} \leq a \right). \end{aligned}$$

The formulation of φ_{l^*kTC} based on φ_{kTC} allows us to conclude that, for a given input topology, l*kTC inactivates a subset of the links that kTC inactivates. Conversely, each link that is activated by l*kTC is also activated by kTC.

6.2.2.7 Yao Graph Algorithm

The TC algorithms that we discussed until now are all neighbor-based TC algorithms. Additionally, each of these TC algorithms builds on the triangle predicate $\varphi_{\text{triangle}}$. To investigate a second major family of TC algorithms, we consider the YG and θ -graph algorithms in the following.

The YG algorithm [279] is a cone-based TC algorithm, which requires information about the relative angles of each neighbor mote. In this thesis, we assume that the angle of each link is determined based on information about mote positions, which can be configured statically (e.g., in the FLOCKLAB testbed [149]) or determined at runtime (e.g., using the Global Positioning System, WiFi triangulation, or Bluetooth beacons [209]).

In the YG algorithm, the area around a mote is separated into $c \in \mathbb{N}$ cones of equal angle. Within each cone, YG activates the link to its closest neighbor in this cone. The parameter c can be configured and should be larger than 6. Without loss of generality, we assume that c is a divisor of 360. As a preparation for the definition of the YG-specific predicate, we introduce the following variables. The *cone size* a_{cone} is the angle that a cone covers. For c cones, we obtain a cone size of $a_{\text{cone}} = \frac{360^\circ}{c}$. The x^{th} cone (with $x \in \{1, 2, \dots, c\}$) covers the angular range $[(x-1) \cdot a_{\text{cone}}, x \cdot a_{\text{cone}})$. The *cone function* $\text{cone} : E_T \times \mathbb{R} \rightarrow \mathbb{N}$ determines for a given link l_{12} and cone count c the number $x \in \{1, 2, \dots, c\}$ of the cone emanating from n_1 that contains n_2 :

$$\text{cone}(l_{12}, c) = \left\lfloor \frac{a_{12}}{a_{\text{cone}}} \right\rfloor = \left\lfloor \frac{a_{12}}{\frac{360^\circ}{c}} \right\rfloor.$$

The set of inactive links in the virtual topology G_T of YG can be characterized using first-order logic with arithmetic expressions as follows:

$$\begin{aligned}
\forall G_T = (V_T, E_T) : \forall l_1 \in E_T : \\
s(l_1) = I \Rightarrow \exists l_2 \in E_T : \varphi_{\text{sameSrc}}(l_1, l_2) \wedge \varphi_{\text{sameCone}}(l_1, l_2, c) \wedge \varphi_{YG}(l_1, l_2) \\
\text{with} \\
\varphi_{\text{sameSrc}}(l_1, l_2) : \Leftrightarrow \text{id}(\text{src}(l_1)) = \text{id}(\text{src}(l_2)) \wedge \text{id}(\text{trg}(l_1)) \neq \text{id}(\text{trg}(l_2)), \\
\varphi_{\text{sameCone}}(l_1, l_2, c) : \Leftrightarrow \text{cone}(l_1, c) = \text{cone}(l_2, c), \text{ and} \\
\varphi_{YG}(l_1, l_2) : \Leftrightarrow w'_1 > w'_2. \tag{6.4}
\end{aligned}$$

6.2.2.8 Theta-graph algorithm

The θ -graph algorithm [192] is related to the YG algorithm. The algorithm also activates exactly one link in each of the c cones of equal size. The θ in the name of the algorithm designates the cone size: $\theta := a_{\text{cone}} = \frac{360^\circ}{c}$. In contrast to the YG algorithm, the θ -graph algorithm activates a link in a cone if its *projected length* on the angular bisector of the cone is minimal among all links in the same cone. We determine the projected length of a link on the angular bisector of the x^{th} cone using the function $\text{proj} : E_T \times \mathbb{R} \rightarrow \mathbb{R}$, which is defined as:

$$\begin{aligned}
\text{proj}(l_{12}, c) &= \text{len}_{12} \cdot \cos(a_{12} - a_{\text{bisector}}) \\
&= \text{len}_{12} \cdot \cos(a_{12} - (\text{cone}(l_{12}, c) + 0.5) \cdot a_{\text{cone}}) \\
&= \text{len}_{12} \cdot \cos \left(a_{12} - \left(\left\lfloor \frac{a_{12}}{\frac{360^\circ}{c}} \right\rfloor + 0.5 \right) \cdot 360^\circ \right).
\end{aligned}$$

The set of inactive links in the virtual topology G_T of the θ -graph algorithm can be characterized using first-order logic with arithmetic expressions as follows:

$$\begin{aligned}
\forall G_T = (V_T, E_T) : \forall l_1 \in E_T : \\
s(l_1) = I \Rightarrow \exists l_2 \in E_T : \varphi_{\text{sameSrc}}(l_1, l_2) \wedge \varphi_{\text{sameCone}}(l_1, l_2, c) \wedge \varphi_{\theta\text{-graph}}(l_1, l_2) \\
\text{with} \\
\varphi_{\theta\text{-graph}}(l_{XY}, l_{XZ}, c) : \Leftrightarrow \text{proj}'(l_{XY}, c) > \text{proj}'(l_{XZ}, c). \tag{6.5}
\end{aligned}$$

Similar to the unique weight $w'(l)$ of a link l , we introduce the unique projected length $\text{proj}'(l)$ of a link l as tie breaker.

6.2.2.9 e-kTC algorithm

We used the CTCA algorithm [35] as motivating example in Chapter 1 to highlight the difficulty of ensuring traceability between specification and implementation. In the following, we present e-kTC, a variant of kTC that is energy-aware and inspired by CTCA.

First, we introduce our definition of network lifetime. Afterwards, we discuss an example that shows that kTC may lead to a reduced network lifetime in comparison to the Maxpower algorithm. Subsequently, we present e-kTC.

In the WSN community, extending the lifetime of a network is a key optimization goal. Numerous possible definitions of network lifetime exist [33]. We apply a definition that is tailored to the per-mote lifetime. A mote n_1 is *alive* if its remaining energy is positive (i.e., $E_1 > 0$), and *dead* if its battery is empty (i.e., $E_1 = 0$). This definitions allows us to characterize the lifetime of a topology

Definition 6.4 (Lifetime of a network). Let $G_T^{t_1}, G_T^{t_2}, \dots$ be a sequence of topology snapshots at discrete points in time t_1, t_2, \dots . The *lifetime* L_T of the network is the first point in time t_x at which one mote is dead in $G_T^{t_x}$. \square

The following example illustrates how applying kTC may reduce the lifetime of a network compared to the Maxpower algorithm.

Example 6.5 (Reduced network lifetime with kTC). Figure 6.4 shows a sample topology and the link states that result from applying kTC with $k = 1.3$. The label at a link ℓ_{12} denotes its weight w_{12} , which is set to its length len_{12} . We assume that (i) the value of the power $P(\ell_1)$ that is required to use a link ℓ_1 is the squared length of this link (i.e., $P(\ell_1) = \text{len}_1 \cdot \text{len}_1$), and (ii) mote n_1 periodically transmits data continuously to mote n_5 .

These messages are relayed using motes n_2, n_3 , and n_4 . The lifetime of the network is determined by ℓ_{23} . This link has the smallest fraction of remaining energy and required transmission power among the links on the only possible routing path: $L_T = \frac{E_2}{\text{len}_{23} \cdot \text{len}_{23}} = \frac{125}{6 \cdot 6} = 3.47$. This means that the lifetime of the network is four units.

If we use Maxpower instead of kTC in this example, all links are active. We may assume that the messages from n_1 to n_5 are routed using links ℓ_{13} and ℓ_{35} because this is the shortest routing path from n_1 to n_5 . In this case, the lifetime of the network is determined by the energy of n_5 and the energy consumption of ℓ_{35} : $L_T = \frac{1000}{10 \cdot 10} = 10$. This means that, under this simple model, the death of the first mote can be postponed by six time units when using Maxpower instead of kTC.

The problem of unbalanced energy consumption, which is apparent in Example 6.5, is well-known in the WSN literature [114]. In the following, we derive an energy-aware variant of kTC that estimates the remaining lifetime of each mote and uses the inverse estimated lifetime as inverse weight. We call this variant *e-kTC*.

For an energy-aware TC algorithm, estimating the remaining lifetime of a mote is important. This estimate allows the TC algorithm to relieve a mote proactively that would otherwise fail soon. The *expected transmission power* $\hat{P}(\ell_{12})$ of a link ℓ_{12} represents the minimum estimated power that is necessary to reach n_2 from n_1 . According to Friis'

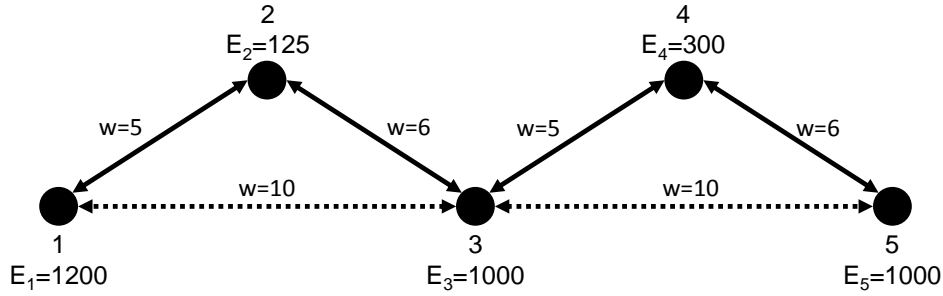


Figure 6.4: Example: Reduced network lifetime with kTC

free space propagation model [63], $\hat{P}(\ell_1)$ is proportional to a power of the length of a link ℓ_1 with an *attenuation exponent* $\beta \geq 2$:

$$\hat{P}(\ell_1) \propto \text{len}^\beta(\ell_1)$$

The expected transmission power per link can be used to estimate the remaining lifetime of a mote w.r.t. this link.

Definition 6.6 (Expected remaining lifetime w.r.t. a link). Let ℓ_{12} be a link with source mote having energy level E_1 and an estimated transmission power expected transmission power $\hat{P}(\ell_{12})$ of ℓ_{12} . The *expected remaining lifetime* $\hat{L}(\ell_{12})$ of n_1 w.r.t. ℓ_{12} is defined as:

$$\hat{L}(\ell_{12}) = \frac{E_1}{\hat{P}(\ell_{12})}. \quad (6.6)$$

□

The expected remaining lifetime of a mote w.r.t. a particular link is equal to the remaining time until the death of the mote if we assume that only this link is used for transmitting messages. Based on the expected remaining lifetime of a mote w.r.t. an individual link (Definition 6.6), we can estimate the remaining lifetime of the mote as follows.

Definition 6.7 (Expected remaining lifetime w.r.t. a link). Let n_1 be a mote in the topology $G_T = (V_T, E_T)$. The *expected remaining lifetime* $\hat{L}(n_1)$ of n_1 is the minimum expected remaining lifetime of its outgoing active links:

$$\hat{L}(n_1) = \min_{\ell_{12} \in E_T : s_{12}=A} \hat{L}(\ell_{12}).$$

□

Definition 6.6 and Definition 6.7 implicitly presume that transmitting a message consumes energy only at the sending mote. However, on real hardware, transmitting a

message also consumes energy at the receiving mote. We neglect this additional cost for the moment because it can be modeled by refining the calculation of $\widehat{P}(l_1)$ [35].

For the other neighbor-based TC algorithms that we discussed until now, we assumed that pairs of reverse links have the same weights. In practice, a binary aggregation function ensures these link weights are identical. Any binary function that is symmetric in its parameters is suitable as aggregation function (e.g., min, max, mean). To calculate such an aggregated weight, we need access to the reverse link of each link in the triangle. An *undirected triangle* consists of six links $l_1, l'_1, l_2, l'_2, l_3, l'_3$ where (i) $l_1, l_2,$ and l_3 form a regular triangle, and (ii) $l'_1, l'_2,$ and l'_3 are the inverse links of $l_1, l_2,$ and $l_3,$ respectively. In case of e-kTC, we consider undirected triangles because the expected remaining lifetimes of two links l_{12} and l_{21} differ if the energy levels E_1 and E_2 of the incident motes n_1 and n_2 differ. We use the minimum function to aggregate the potentially different values $\widehat{L}(l_{12})$ and $\widehat{L}(l_{21})$ in this case. For e-kTC, we use the minimum function because the incident mote with the lower energy level determines the point in time when a link disappears.

Definition 6.8 (Aggregated expected remaining lifetime). Let l_{12} be a link that has a reverse link l_{21} . The *aggregated expected remaining lifetime* \widehat{L}_{12}^{\min} of l_{12} is defined as:

$$\widehat{L}_{12}^{\min} = \min(\widehat{L}(l_{12}), \widehat{L}(l_{21})).$$

Note that $\widehat{L}_{12}^{\min} = \widehat{L}_{21}^{\min}$. □

Definition 6.8 allows us to characterize the virtual topology of e-kTC as follows. In the virtual topology of e-kTC, a link is inactive if and only if (i) this link is part of an undirected triangle, (ii) it has the minimum aggregated expected remaining lifetime among the links in the triangle, and (iii) its aggregated expected remaining lifetime is at least k times shorter than the maximum aggregated expected remaining lifetime of the other links in the triangle. As for kTC, the parameter k can be used to control the aggressiveness of the link-inactivation behavior. The lower k is, the more links are inactivated, in general.

The set of inactive links in the virtual topology G_T of e-kTC can be characterized using first-order logic with arithmetic expressions as follows:

$$\begin{aligned}
& \forall G_T = (V_T, E_T) : \forall l_1 \in E_T : \\
& \quad s(l_1) = I \Rightarrow \exists l'_1, l_2, l'_2, l_3, l'_3 \in E_T : \\
& \quad \quad \varphi_{\text{unTriangle}}(l_1, l'_1, l_2, l'_2, l_3, l'_3) \wedge \varphi_{\text{e-kTC}}(l_1, l'_1, l_2, l'_2, l_3, l'_3) \\
& \text{with} \\
& \quad \varphi_{\text{unTriangle}}(l_1, l'_1, l_2, l'_2, l_3, l'_3) :\Leftrightarrow \\
& \quad \quad \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{\text{rev}}(l_1, l'_1) \wedge \varphi_{\text{rev}}(l_2, l'_2) \wedge \varphi_{\text{rev}}(l_3, l'_3), \quad (6.7) \\
& \quad \varphi_{\text{rev}}(l_1, l_2) :\Leftrightarrow \text{src}(l_1) = \text{trg}(l_2) \wedge \text{trg}(l_1) = \text{src}(l_2), \text{ and} \quad (6.8) \\
& \quad \varphi_{\text{e-kTC}}(l_{XY}, l_{YX}, l_{XZ}, l_{ZX}, l_{ZY}, l_{YZ}) :\Leftrightarrow \\
& \quad \quad \widehat{L}_{X,Y}^{\text{min}} < \min(\widehat{L}_{X,Z}^{\text{min}}, \widehat{L}_{Z,Y}^{\text{min}}) \wedge \widehat{L}_{X,Y}^{\text{min}} < k \cdot \max(\widehat{L}_{X,Z}^{\text{min}}, \widehat{L}_{Z,Y}^{\text{min}}). \quad (6.9)
\end{aligned}$$

The predicate $\varphi_{\text{unTriangle}}$ is *true* if the given links form an undirected triangle (Equation (6.7)). It is defined using the auxiliary *reverse-link predicate* φ_{rev} , which evaluates to *true* if the second link is the reverse of the first link, and vice versa (Equation (6.8)). The predicate $\varphi_{\text{e-kTC}}$ captures the e-kTC-specific condition (Equation (6.9)). As usual, we employ mote identifiers as tie breakers and compare the unique aggregated expected remaining lifetime of the links in the triangle. The following example shows how e-kTC helps to extend the lifetime of a network.

Example 6.9 (Extended network lifetime with e-kTC). We use the same input topology as in Example 6.5 and make the same assumptions regarding energy consumption. Each link is annotated with its weight (w , equal to its length) and the aggregated expected remaining lifetime w.r.t. this link (L). The resulting aggregated value is highlighted in bold.

We apply e-kTC with $k = 1.0$. This means that e-kTC inactivates each link with minimal aggregated expected remaining lifetime in each triangle. In contrast to the decision of kTC in Example 6.5, e-kTC inactivates links l_{23} , l_{32} , l_{45} , and l_{54} . This means that messages from n_1 to n_5 are still transmitted using the energy-intensive links l_{13} and l_{35} because motes n_1 and n_3 have a large remaining energy compared to n_2 and n_4 .

Let's assume now that n_1 is transmitting messages continuously to n_5 . We simulate this behavior for two time units and obtain the topology shown in Figure 6.5b. In this topology, the aggregated expected remaining lifetime values of l_{35} and l_{53} have dropped to 8, which is the new minimum in the undirected triangle consisting of motes n_3 , n_4 , and n_5 . Therefore, e-kTC activates l_{45} and l_{54} , and inactivates l_{35} and l_{53} .

If we assume that e-kTC is invoked periodically every time unit, then the network lifetime is 15 time units. Mote n_2 is the first mote to run out of energy. This means

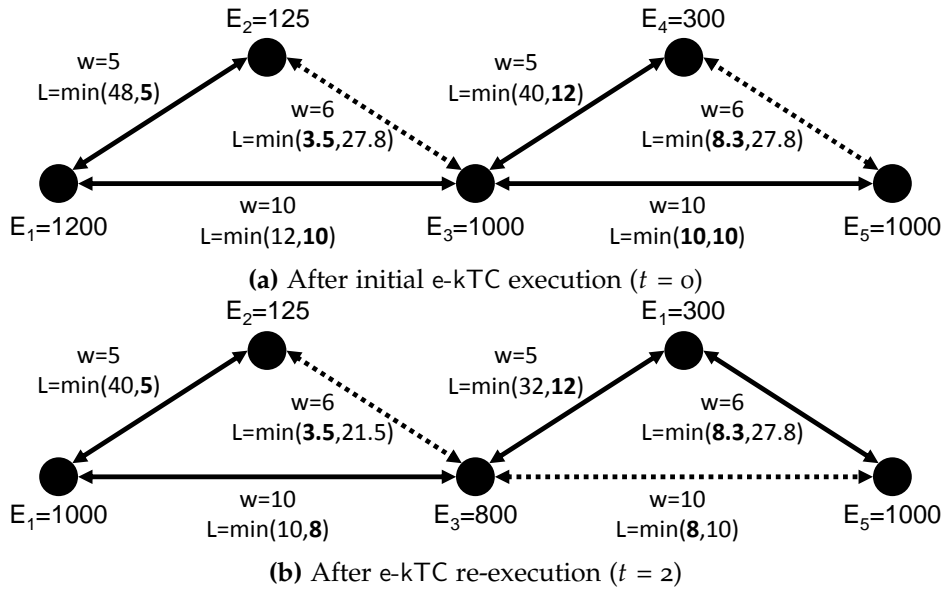


Figure 6.5: Example: Extended network lifetime with e-kTC

that e-kTC is able to extend the network lifetime by five time units compared to the Maxpower algorithm.

The preceding example illustrates that it is highly important to execute e-kTC dynamically to keep the network alive for as long as possible. In [119], we assessed e-kTC in a simulative evaluation. For the sake of conciseness, we do not repeat this evaluation here.

6.2.2.10 The minimum-weight predicate

Inspired by the idea of XTC_{RLS} [66], which (re-)activates links that provide a sufficiently large signal-to-noise ratio, we propose an auxiliary predicate that can be combined with any neighbor-based TC algorithm.

Working memory is a highly limited resource on motes. For this reason, keeping the entire neighborhood of a mote in working memory may be infeasible if the topology is dense. Fortunately, it is often unnecessary to store low-weight links because the energy consumption of a mote is typically predominated by links with large weight. Additionally, reducing the size of the processed neighborhood may reduce the execution time of a TC algorithm.

The *minimum-weight predicate* $\varphi_{\text{min-weight}}$ formalizes this reduction step. In the following condition, the parameter w_{thres} represents the configurable minimal weight of a link to be included in the considered neighborhood.

$$\begin{aligned} \forall G_T = (V_T, E_T) : \forall l_1 \in E_T : \\ s(l_1) = \mathbf{I} \Rightarrow \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{\text{min-weight}}(l_1, l_2, l_3, w_{\text{thres}}) \\ \text{with} \\ \varphi_{\text{min-weight}}(l_1, l_2, l_3, w_{\text{thres}}) = \min(w_1, w_2, w_3) \geq w_{\text{thres}}. \end{aligned} \quad (6.10)$$

This predicate can be used to compose new variants of the previously specified TC algorithms. For instance, a modified version of kTC with minimum-weight predicate can be specified in first-order logic with arithmetic expressions as follows:

$$\begin{aligned} \forall G_T = (V_T, E_T) : \forall l_1 \in E_T : \\ s(l_1) = \mathbf{I} \Rightarrow \varphi_{\text{triangle}}(l_1, l_2, l_3) \wedge \varphi_{\text{kTC}}(l_1, l_2, l_3) \wedge \varphi_{\text{min-weight}}(l_1, l_2, l_3, w_{\text{thres}}). \end{aligned}$$

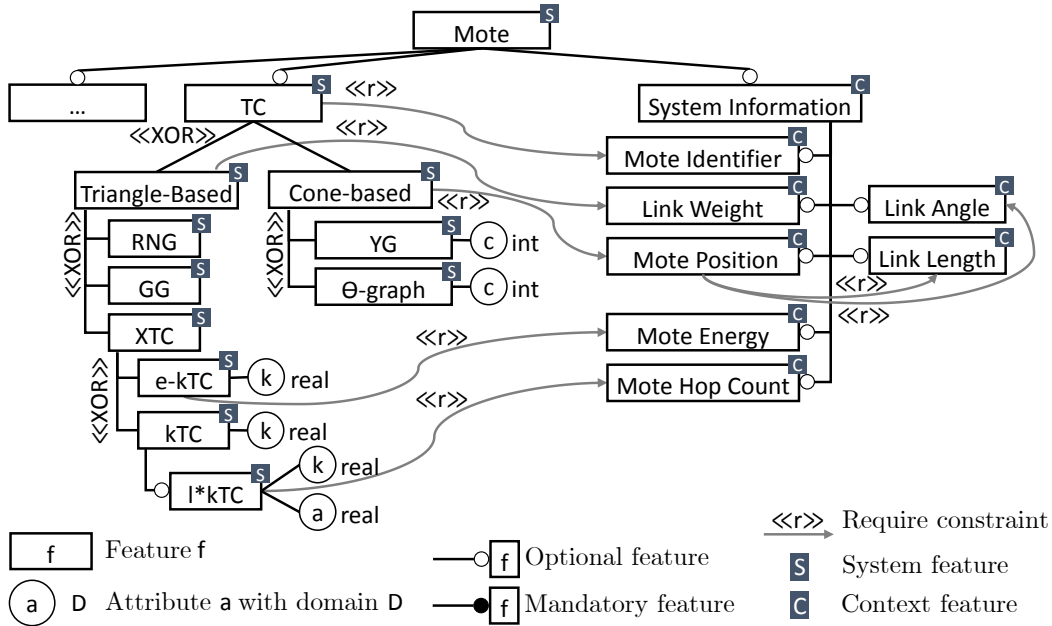


Figure 6.6: Feature diagram of with two TC algorithm families

6.3 SPECIFICATION OF FAMILIES OF TOPOLOGY CONTROL ALGORITHMS

In this section, we propose how to specify families of TC algorithms using feature models and extensions of graph constraints.

6.3.1 Specification using feature models

In Section 6.2, we observed that all of the discussed TC algorithms share certain predicates (e.g., the triangle predicate $\varphi_{\text{triangle}}$ or the same-cone predicate $\varphi_{\text{sameCone}}$). We propose to capture the variability within families of TC algorithms using context feature models (as introduced in Section 3.6).

Figure 6.6 depicts a context feature model that captures the following properties. First, a mote may access different types of information, which is represented by the subtree below the System Information context feature. Depending on the mote hardware and operating system, the child features of System Information are selected at compilation time, at configuration time, or even at runtime (see also [30]). This branch of the feature model also captures that information about link angles and link lengths can be accessed as soon as mote position information is available (specified as «require» constraint).

Second, a mote has an optional, configurable TC algorithm (TC). If TC is deselected, the Maxpower algorithm is active. The two child features of TC correspond to the two families of TC algorithms that we discuss in this chapter: Triangle-based is the (direct or

indirect) parent feature of all TC algorithms that employ the triangle predicate $\varphi_{\text{triangle}}$: RNG, GG, XTC, kTC, l*kTC, e-kTC. Similarly, Cone-based is the parent feature of the two cone-based TC algorithms YG and θ -graph, which share the predicates φ_{sameSrc} and $\varphi_{\text{sameCone}}$. Furthermore, the feature kTC is a child of XTC to indicate that φ_{kTC} builds on φ_{XTC} . For the same reason, l*kTC is a child of kTC. We placed e-kTC below XTC instead of kTC because its predicate is inspired by but not the same as the kTC-specific predicate φ_{kTC} .

Using a feature model, we can also specify formally which TC algorithms access which types of system information using «require» constraints. For example, a triangle-based TC algorithm require access to the weight of a link, and a cone-based TC algorithm uses the position of motes in its local view. Additionally, l*kTC needs access to the hop count of each mote, and e-kTC requires information about the energy level of a mote.

The feature in the top-left corner that contains an ellipsis ($\langle \dots \rangle$) indicates that we omitted further features that represent, for instance, routing mechanisms or WSN applications. Figure 5.15 provides a more comprehensive overview of possible system features of a mote.

Using feature models for representing TC algorithm families has several advantages. First, the relations of TC-algorithm-specific predicates are visualized in the feature diagram. Second, we can identify all TC algorithms that are compatible with a particular mote type based on «require» constraints from TC algorithms to context features. Third, feature models form the basis for specifying possible TC transitions of a TC multi-mechanism.

Finally, we discuss how to represent TC algorithms that combine predicates of two different families in the feature model. Examples of such TC algorithms are Yao Yao, YGG, GYG, and θ +GG. In all of these examples, a two-level virtual topology is constructed. For example, the YGG algorithm applies first GG and then YG. We propose to model such a TC algorithm as child-feature of the first-level TC algorithm (e.g., GG) and add a «require» constraint to the second-level TC algorithm (e.g., YG).

6.3.2 Specification using constraint refinement

Representing TC algorithms using feature models helps the developer to understand the configuration space of a mote and the variability within families of TC algorithms. Still, this specification does not provide enough information to apply our correct-by-construction methodology. One reason is that possible individual topology modifications and their control flow cannot be represented in a feature model.

Therefore, we specify the commonalities of the local consistency properties of a TC family as a *family graph constraint*. All members of a family refine the family graph constraint according to the following definition.

Definition 6.10 (Constraint refinement). A constraint C_Y refines a constraint C_X if C_Y can be obtained by performing one or more of the following modifications to C_X : (i) addition of object or link variables to a conclusion pattern, (ii) addition of object or link variable to the premise pattern (possibly entailing an extension of the conclusion patterns), (iii) addition of attribute constraints to a conclusion pattern, (iv) addition of attribute constraints to the premise pattern (possibly entailing an extension of the conclusion patterns), or (v) addition of conclusion patterns. \square

This definition defines a refinement relationship between graph constraints on a syntactic level. Depending on the performed modification, the set of models that fulfill the refined constraint C_Y can be larger than or smaller than the refined constraint C_X . The following example illustrates Definition 6.10 for the families of triangle and cone-based TC algorithms.

Example 6.11 (Constraint refinement). Figure 6.7 shows the two family constraints $C_{\text{triangle},a}$ and $C_{\text{triangle},i}$ for active and inactive links of triangle-based TC algorithms, and how the XTC-specific and kTC-specific constraints refine the family constraints. The family constraints reflect a link l_1 in the virtual topology of a triangle-based TC algorithm may only be inactive if it is part of a triangle (l_1, l_2, l_3) :

$$\forall G_T=(V_T, E_T) : \forall l_1 \in E_T : s(l_1) = I \Rightarrow \exists l_2, l_3 \in E_T : \varphi_{\text{triangle}}(l_1, l_2, l_3)$$

We recognize that the constraint $C_{kTC,i}$ is stricter than its family constraint $C_{\text{triangle},i}$ in the sense that all topologies that fulfill $C_{kTC,i}$ also fulfill $C_{\text{triangle},i}$, but not vice versa. A link that is part of a φ_{kTC} -fulfilling triangle is always part of a triangle, but not every triangle fulfills φ_{kTC} . In contrast, $C_{kTC,a}$ is less strict than its family constraint $C_{\text{triangle},a}$. Any active link in a triangle violates $C_{\text{triangle},a}$, whereas a violation of $C_{kTC,a}$ contains an active link that fulfills the kTC-specific attribute constraints.

In this example, only attribute constraints are added during the refinement. In case of e-kTC, fresh link variables v_{21} , v_{23} , and v_{31} are added to obtain an undirected triangle together with the e-kTC-specific attribute constraints during the refinement of $C_{\text{triangle},a}$ and $C_{\text{triangle},i}$.

Figure 6.8 shows the family constraints $C_{\text{cone},a}$ and $C_{\text{cone},i}$ for active and inactive links of cone-based TC algorithms, and how the YG-specific constraints refine these family constraints. The attribute constraint sc corresponds to the predicate $\varphi_{\text{sameCone}}$. The predicate φ_{sameSrc} is represented by the two link variables that share a common source mote variable.

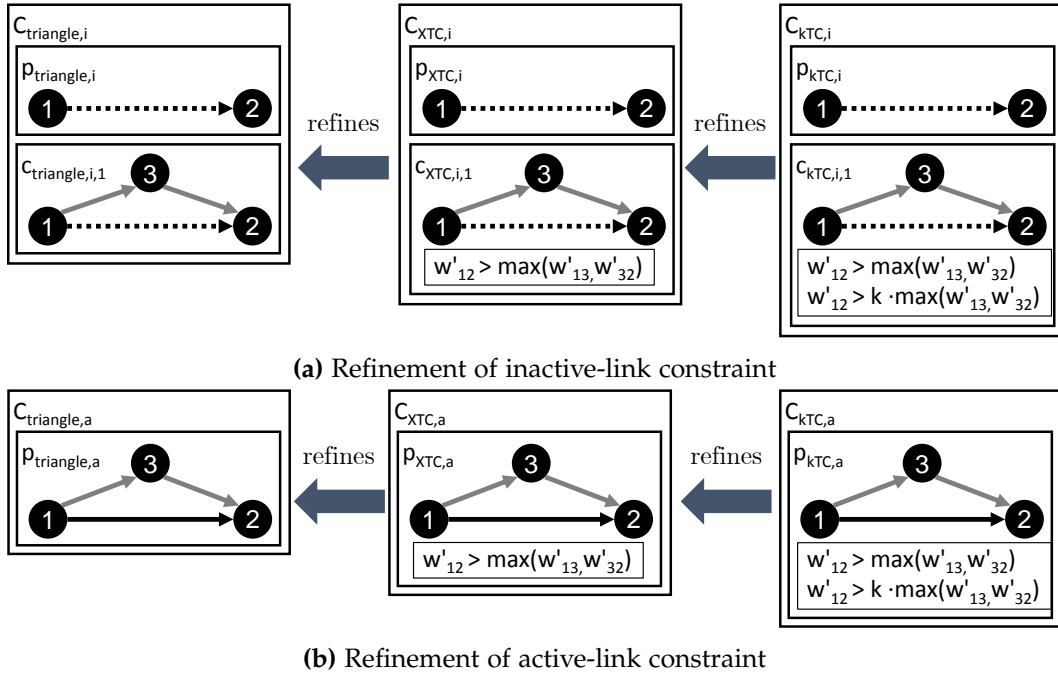


Figure 6.7: Example: Refinement in the family of triangle-based TC algorithms

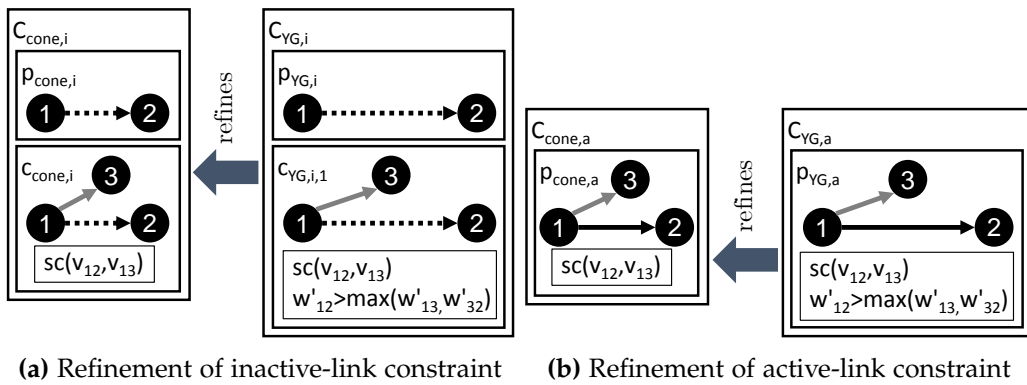


Figure 6.8: Example: Refinement in the family of cone-based TC algorithms

6.4 PROVING GLOBAL CONSISTENCY PROPERTIES

In this section, we illustrate how to prove global consistency properties for a family of TC algorithms. We use the family of triangle-based TC algorithms and the global consistency property of required connectivity of the virtual topology as primary example. Afterwards, we point out how the corresponding proof could be conducted for cone-based TC algorithms. We focus on the preservation of connectivity here because it is a crucial consistency property that each TC algorithm must fulfill.

The general proof idea is to show that a triangle-based TC algorithm preserves connectivity if a strict ordering of the link variables in the family constraints exists. For a concrete triangle-based TC algorithm, we then have to prove that it ensures the required strict ordering. The following proof is a generalization of the corresponding proof for the concrete triangle-based TC algorithm kTC (Example 3.23).

Example 6.12 (Proof of connectivity for triangle-based TC algorithms). In this example, we consider a triangle-based TC algorithm A . This means that the TC-algorithm-specific constraints $C_{A,a}$ and $C_{A,i}$ refine the family constraints $C_{\text{triangle},a}$ and $C_{\text{triangle},i}$. A topology is weakly consistent w.r.t. A if it fulfills the constraint set $\{C_{\text{no-loops}}, C_{\text{no-parallel-links}}, C_{A,a}, C_{A,i}\}$, and it is strongly consistent w.r.t. A if it additionally fulfills C_u .

An A -specific link order \prec_A is a total ordering of the links of a topology $G_T = (V_T, E_T)$ such that each triangle (v_{12}, v_{13}, v_{32}) that corresponds to a match of $p_{\text{triangle},a}$ and $c_{\text{triangle},i,1}$ (see Figure 6.7) fulfills the following inequalities:

$$v_{13} \prec_A v_{12} \text{ and } v_{32} \prec_A v_{12}.$$

For the kTC example, we defined \prec_{kTC} based on the unique weight of each link:

$$v_{13} \prec_{kTC} v_{12} :\Leftrightarrow w'_{13} < w'_{12}.$$

Based on these preliminaries, we now state the claim that we have to prove.

Claim: Let A be a triangle-based TC algorithm for which a A -specific link order exists. The virtual topology of A is connected if the input topology of A is connected and the TC mechanism topology is weakly consistent w.r.t. the A -specific graph constraints.

Proof sketch: The virtual topology of a TC mechanism is the AU -view of the TC mechanism topology and, therefore, is connected if each pair of motes is connected by an AU -path (i.e., a path of active or unmarked links) in the TC mechanism topology (see Definition 2.6). The mote and link sets of the input topology and TC mechanism topology are equal. This means that the TC mechanism topology is connected if the input topology is connected.

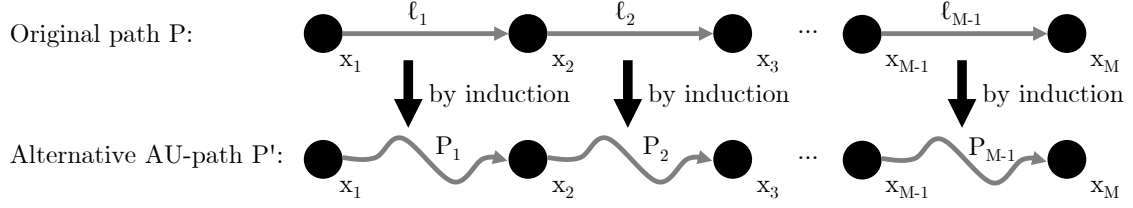


Figure 6.9: Construction of alternative AU-path P' from n_{x_1} to n_{x_M} in proof of preserved connectivity for triangle-based TC algorithms (identical to Figure 3.16a)

Therefore, it is sufficient to show the following claim. For a given path P between two motes n_{x_1} and n_{x_M} , an alternative AU-path P' between these motes can be constructed. In the following, we show for each link l_j on P (by induction) that its incident motes are connected by an AU-path P_j . The alternative path P' can be constructed by concatenating the per-link alternative AU-paths, as illustrated by Figure 6.9. This idea is the same as for the proof of connectivity for kTC (Example 3.23).

Induction claim: If the topology is connected and weakly consistent, then the incident motes of each link l_j are connected by an AU-path P_j . We show that this claim holds by induction over the links of the topology $G_T = (V_T, E_T)$, sorted according to the A -specific link order \prec_A . Let $L = (l_1, l_2, \dots)$ be a list that contains all links in E_T and let L be sorted according to \prec_A .

Induction start ($j = 1$): Let's first assume that the first link $l_j = l_1 = l_{XY}$ in L is inactive. In this case, the fulfillment of the inactive-link constraint $C_{A,i}$ implies that there are links l_{XZ} and l_{ZY} with $l_{XZ} \prec_A l_{XY}$ and $l_{ZY} \prec_A l_{XY}$. This means that both l_{XZ} and l_{ZY} appear before l_{XY} in L , which contradicts the assumption that l_1 is the first link in L . Therefore, l_1 is either active or unmarked and constitutes an AU-path.

Induction step ($j = N \rightarrow N + 1$): We now assume that the induction claim is true for all links l_j in L with $j \leq N$. We have to show that the claim also holds for link l_{N+1} . We distinguish between the three possible states of l_j , as illustrated in Figure 6.10. If l_j is active (Case 1) or unmarked (Case 2), the induction claim holds. If l_j is inactive, the fulfillment of $C_{A,i}$ implies that there are links l_{XZ} and l_{ZY} with $l_{XZ} \prec_A l_{XY}$ and $l_{ZY} \prec_A l_{XY}$. Therefore, $l_{XZ} = l_{j_1}$ and $l_{ZY} = l_{j_2}$ for some $j_1, j_2 \leq N$, and the induction claim holds for l_{XZ} and l_{ZY} . This implies that AU-paths P_1 and P_2 exist from n_X to n_Z and from n_Z to n_Y , respectively. The concatenation of P_1 and P_2 is an AU-path from n_X to n_Y , and the induction claim follows for $j = N + 1$. \square

For conciseness reasons, we omit the detailed proof for the family of cone-based TC algorithms at this point. The idea, however, is similar, even though the family constraint only contains two links. Still, cone-based TC algorithms usually build on position information. If we assume that the input topology is a UDG, we can conclude that the missing link l_{32} in Figure 6.8 exists and conduct the proof of correctness similar to Example 6.12.

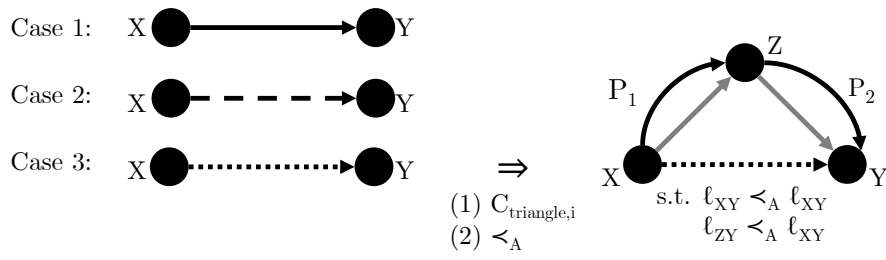


Figure 6.10: Case distinction in induction step in proof of preserved connectivity for kTC

6.5 DISCUSSION OF APPLICABILITY

In this section, we discuss the applicability of our methodology. First, we discuss the applicability based on further examples the TC landscape. Then, we present and discuss a decision diagram that serves as guideline for evaluating whether our approach is applicable to a particular domain.

6.5.1 Discussion based on examples from the topology control landscape

From the collected 32 TC algorithms, we discussed nine TC algorithms that can be developed systematically using our approach in Sections 6.2 to 6.4. For the sake of conciseness, we refrained from deriving the story diagrams of the TC algorithm and context event handler specifications.

FURTHER FULLY SUPPORTED TC ALGORITHMS We claim that the remaining TC algorithms marked with a green check mark (✓) in Figure 6.3 and Table 6.1 can be developed in an analogous way for the following reasons.

First, RNG' and XTC_{RLS} are refinements of RNG and XTC , respectively, that add further attribute constraints. The localized versions Restricted DG and Localized DG of DG are triangle-based TC algorithms that refine the family constraint by adding a fourth link variable v_{14} and attribute constraints that check whether v_{12} is inside the circumcircle of n_1 , n_3 , and n_4 .

Second, each TC algorithm that is a combination of two other fully supported TC algorithms can be represented by combining the attribute constraints, extending the pattern graphs, or both. This argument applies to Yao Yao, YGG, GYG, and $\theta+GG$. In the following example, we briefly explain the resulting graph constraints for YGG.

Example 6.13 (Constraints for YGG). Figure 6.11 shows the graph constraints for YGG. The positive inactive-link constraint $C_{YGG,i}$ specifies that a link v_{12} may be inactive in two cases. First, v_{12} may be inactive if the YG-specific condition is fulfilled (i.e., v_{12} is the weight-minimal link in a cone). Second, v_{12} may be inactive if it is part of a triangle that fulfills the GG-specific condition. In contrast to the original GG algorithm, the two link variables v_{13} and v_{32} must both be active. These additional marking constraints ensure that both links have been selected within their respective cone.

The two active-link constraints $C_{YGG,i}$ and $C_{YGG,i}$ complement the inactive-link constraint $C_{YGG,i}$ in the usual way. They prevent the activation of a link that should actually be inactive.

When applying our approach to YGG, the additional marking constraints in $C_{YGG,i,2}$ and $C_{YGG,i}$ require the application of the generalized variant of the anticipation loop synthesis algorithm (as in Section 4.5.3).

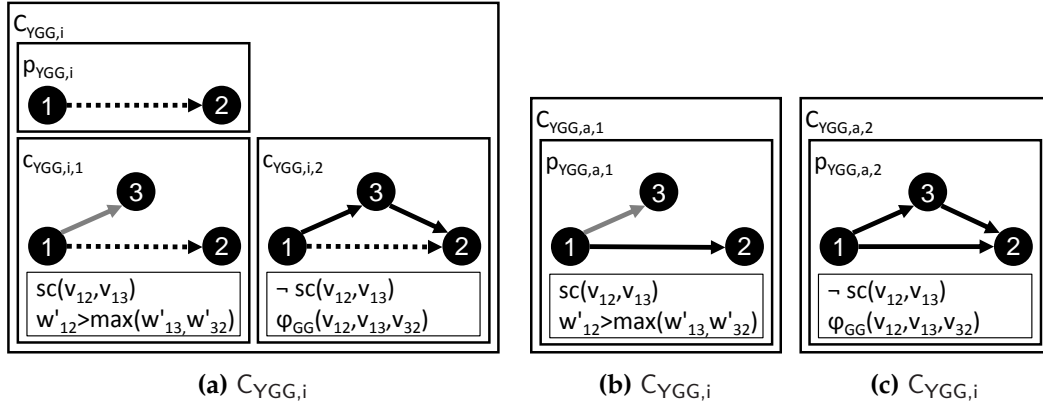


Figure 6.11: Example: Graph constraints for YGG

TC ALGORITHMS SUPPORTED IN BATCH MODE A subset of six TC algorithms processes unmarked links in a particular order (marked with \checkmark_B in Figure 6.3 and Table 6.1): IDRNG, k -Neigh, CBTC, Directed Tree, $S\theta GG$, and $LS\theta GG$. To specify these TC algorithms, we need to add an application condition to the rule $R_{\text{find-u}}$ that ensures that the identified unmarked link is minimal (e.g., w.r.t. its unique weight) among all unmarked links. The TC-algorithm-specific graph constraints of these TC algorithms contain more than one marking constraint. In Section 4.5.3, we discussed how the anticipation loop synthesis algorithm can be extended to such use cases. Therefore, we think that applying our approach to scenarios with inherent batch processing logic is possible, but needs further investigation.

UNCLEAR APPLICABILITY It is currently unclear in how far the four TC algorithms l-kTC, g-kTC, LMST and LMST+RNG' are supported by our approach (see $?_P$ in Figure 6.3 and Table 6.1). We discussed the case of LMST already in Section 4.6. For a fixed maximum neighborhood size, this restriction could be circumvented by deriving graph constraints for each possible path length. This approach would be infeasible in practice due to the excessive amount of application conditions that would need to be checked. In Section 5.3.2.2, we followed another approach by introducing auxiliary data structures and separating the control flow of LMST into a tree construction and marking phase. Similar to how we proved connectivity (which is inexpressible using nested graph constraints) based on triangle-based local patterns, we should investigate in how far the acyclicity constraints of LMST can be expressed using a set of graph constraints. Finally, a third approach for specifying TC algorithms with unbounded path expressions could be to employ graph constraints that are enriched with path conditions. To this end, we discussed several of the existing approaches in Section 4.6.

INAPPLICABILITY DUE TO MISSING DECLARATIVE SPECIFICATION For four TC algorithms, either no declarative characterization of the output topology exists (applies to MobileGrid and LINT, also noted by [219]) or the pseudocode formulation was so extensive that we were not able to extract the TC-algorithm-specific constraints (applies to BDS and BPS).

TOOL SUPPORT We do not provide an evaluation of applicability using COBOLT or cMOFLON at this point. The reason is that this chapter mainly serves to underpin the applicability of our approach from the specification perspective. We think that it is evident that WSN topologies can be modeled using metamodeling and that the corresponding topology modification types can be described using GT. Therefore, as soon as we are able to characterize the local consistency properties of a TC algorithm using graph constraints and show that these local consistency properties jointly imply the global consistency properties, our approach is applicable to a TC algorithm.

To this end, Section 6.2 provides exact specifications for nine algorithms, and this section additionally sketches how eight more TC algorithms can be developed using our approach.

For further results, we refer to the following publications that evaluate several of the considered algorithms using COBOLT. In [119], we conducted a comparative simulation study of kTC and e-kTC in combination with the minimum-weight predicate $\varphi_{\text{min-weight}}$. In [271], we used COALA to evaluate for nine classes of system contexts which of the following TC algorithms performs best w.r.t. mean latency: kTC, YG, e-kTC, and LMST. An evaluation of the discussed TC algorithms using cMOFLON should be possible because both position and energy information is available on the testbed nodes.

6.5.2 Discussion based on modeling techniques

Figure 6.12 shows a decision diagram that helps us to answer the question whether our proposed correct-by-construction methodology is applicable in a particular domain. We deliberately leave the domain open and do not restrict ourselves to TC, which serves as running example throughout this thesis. In the following, we will briefly discuss the significance of each question in Figure 6.12.

GRAPH-BASED MODEL A fundamental question for the applicability of our approach is whether the required view of the algorithm under development can be represented as attributed graph. In this thesis, we chose metamodeling as a graph-based specification technique, but our approach is not limited to this technique. Given the prominence of object-oriented software engineering paradigm, metamodeling is certainly not limiting the applicability of our approach.

CONSISTENCY PROPERTIES To be compatible with our methodology, the consistency properties of the domain need to be expressible at least partially using nested graph constraints. If not all consistency properties are expressible using nested graph constraints, we can still try to prove that the fulfillment of the nested graph constraints implies the fulfillment of the remaining consistency properties. For our running example, we could show that the triangle-based graph constraints that characterize kTC imply that the kTC specification also preserves connectivity, which is a global consistency property. The example of LMST shows that formulating local consistency properties appropriately to prove the global consistency properties (e.g., that the virtual topology is a tree in case of LMST) is an elaborate task.

Even though we used graph constraints without nesting in this thesis, the constructive approach can be applied to nested graph constraints as well [83, 164]. In Section 4.6, we also discussed formalisms that are more expressive than graph constraints. For our methodology, it is important that an adoption of the constructive approach exists for these formalisms and that the construction process is possible with reasonable manual effort or supported by tools.

MODIFICATION TYPES The second major prerequisite for our development methodology is that possible types of modifications can be specified as GT rules. Under the assumption that the view of the algorithm can be modeled as attributed graph, it is reasonable that all relevant elementary modifications of the topology can be specified as GT rules. For each modification type specified as GT rule, we can then ensure constructively that this GT rule preserves all consistency properties.

Even if certain types of modifications cannot be modeled using GT, we can try to prove and (if necessary) ensure that these types of modifications preserve weak consistency. Whether or not this is possible depends on the concrete use case. The major advantage of this conceptual extension point is that complex modification types need not be specified using GT rules. The main drawback, however, is that all modification types that are not specified using GT need to be implemented for all desired target platforms, which makes evaluating the same algorithm on multiple platforms difficult.

CONTROL FLOW We employed SDM in this thesis to specify the control flow of an algorithm. A story diagram can represent all control flow concepts of typical imperative programming languages (e.g., JAVA, C, C++). It supports specifying conditions (if-then-else), loops (while-do, do-while), operation-local variables, invoking arbitrary EOperations inside a story diagram (including recursion). Therefore, we think that SDM is expressive enough to cover most use cases. In contrast to invoking GT rules from a host language (e.g., JAVA), using a control flow specification language has the advantage that the control flow specification can be translated into multiple target languages. Furthermore, to obtain meaningful results, we need to prove that the algorithm speci-

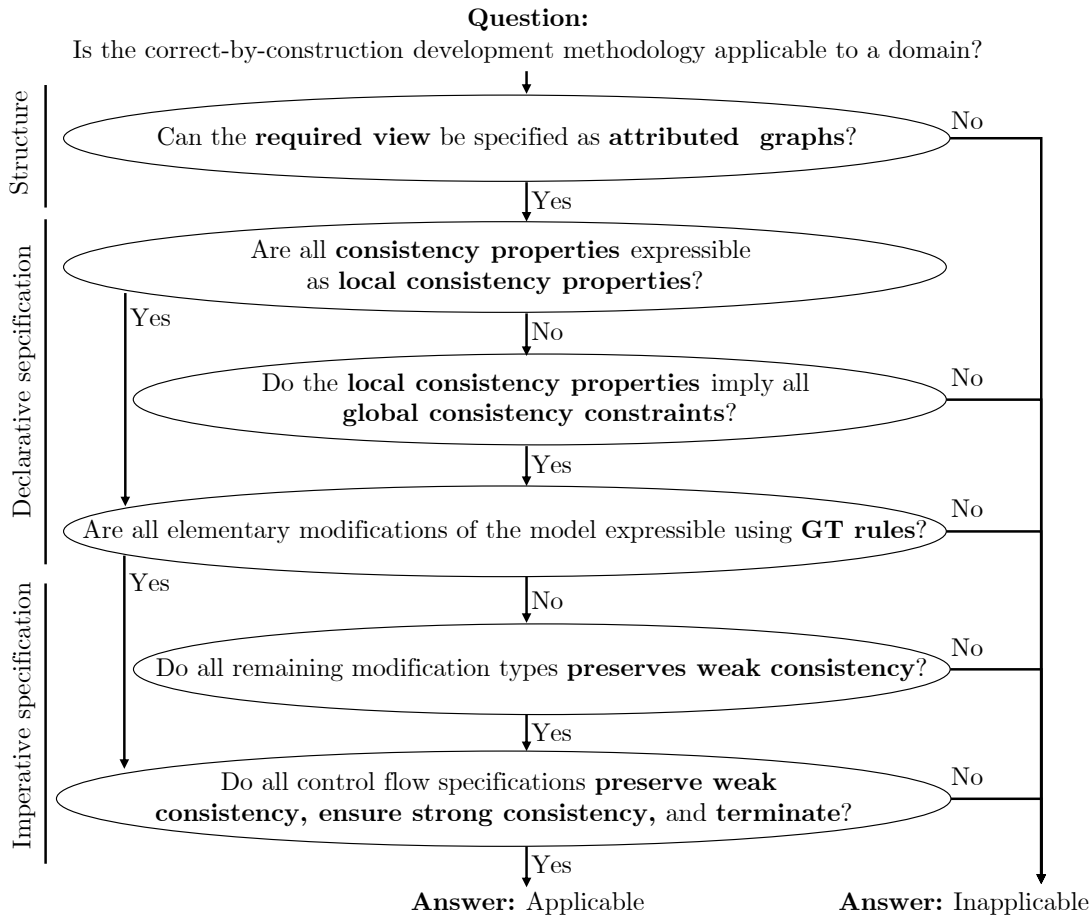


Figure 6.12: Decision diagram for applicability of correct-by-construction methodology

fication enforces strong consistency. This is arguably easier if we work with a suitable abstraction of the control flow rather than source code.

Regarding the specification refinement, the constructive approach by Heckel and Wagner [91] does not require the user to use a particular control flow specification language. Instead, each GT rule is refined individually based on a set of graph constraints.

In contrast, the anticipation loop synthesis algorithm modifies the control flow of the context event handlers. In this thesis, we specify the control flow of context event handlers using SDM. Still, our approach is not limited to SDM. A closer view at the anticipation loop synthesis algorithm reveals that it requires the following properties to be applicable in the context of a different control flow specification language.

First, sequential rule application must be supported. This is necessary to prepend the application of the anticipation rules to the application of the context event rule.

Second, it should be possible to specify the application of GT rules based on a set of rule input parameters. This is necessary to ensure that an anticipation rule has access to the pending context event that (potentially) violates the synthesized application condition of the anticipation rule. Still, even if the control flow language does not provide means to pass parameters to rule applications, the information about the pending context event could be encoded as additional model elements (e.g., similar to the context event markers that we used in Chapter 2 for illustration purposes).

Third, the control flow specification language should have support for specifying the exhaustive (i.e., “as-long-as-possible”) application of a GT rule. This is necessary because an anticipation loop shall only terminate if all violations of the corresponding application condition have been resolved. This requirement is necessary to reuse the structure of the correctness proofs in Section 4.4.3.

Our observation is that state-of-the-art GT languages (e.g., in HENSHIN [9] or GP2 [11]) provide all three required modeling capabilities. Therefore, an adoption of the anticipation loop synthesis algorithm to these control flow specification languages should be feasible with only minor (i.e., technical) adjustments to the correctness proofs in Section 4.4.3.

In the context of the modified kTC-specific graph constraints in Section 4.5.3 and the combined TC algorithms Yao Yao, YGG, GYG, and θ +GG in Section 6.5.1, we presented the generalized anticipation loop synthesis algorithm, which builds on weaker assumptions regarding the involved graph constraints. For such graph constraints, a more elaborate termination proof may be necessary (as in Example 4.35). In Section 4.5.3, we sketched in general and showed for kTC that a termination proof can be performed based on the descending chain property of a potential function [150].

CONCLUSION The preceding discussion shows that neither programmed GT nor meta-modeling constitute major limitations of our methodology. If at all, the expressiveness of nested graph constraints and GT rules limit the applicability of our approach to other domains.

6.6 RELATED WORK

In this section, we discuss related work on variability modeling in combination with GT and as a research challenge in the communication systems domain.

We proposed to organize families of TC algorithms in a notation similar to feature diagrams in [119]. In this thesis, we employed context feature models to indicate which features are selected by the environment and which features are selected by the TC multi-mechanism. Furthermore, we discussed our modeling approach based on the additional TC algorithm family of cone-based TC algorithms.

Only few contributions connect GT with techniques from DSPL engineering. For instance, in [245], the authors propose to model families of GT rules by merging multiple related GT rules into one GT rule whose variables are annotated with presence conditions. A presence condition specifies for each annotated variable in which variant of the GT rule it is contained. Based on this approach, the authors present tool support for automatically deriving variability-based GT rules from a set of traditional GT rules [244] and for editing the derived variability-aware rules [246]. In future work, this approach should be investigated w.r.t. its applicability to the WSN domain.

Several works apply DSPL techniques to support the development of adaptive communication systems. In [18], the authors specify the reconfiguration space of a flood warning WSN as a DSPL. In contrast to this thesis, their focus lies on modeling the different communication interfaces (e.g., WiFi or Bluetooth) of a mote and the conditions under which the respective interfaces should be enabled. In [172], the authors specify a product family of devices that act as environment monitoring and guidance system in a museum at the same time. In contrast to this thesis, their focus lies not on TC but on modeling variability w.r.t. the following four dimensions: communication scope (i.e., unicast vs. anycast communication), measured metrics (e.g., humidity or temperature), actuation (e.g., visual or acoustic), and localization technology (e.g., RFID-based localization). In [46], the possible configuration dimensions of wireless sensor-actor network (WSAN) motes are outlined using feature models. The authors propose a middleware that allows to instantiate the possible configurations in a memory- and energy-efficient way on WSAN motes. One of the considered components reflects the topology of the WSAN motes. In contrast to this thesis, the authors of [46] focus on surveying the typical complexity of the WSN domain. They treat the two considered TC algorithms (i.e., flat tree and hierarchical tree) as black boxes. In [68] (an extension of [46]), an SPL engineering process is presented that allows to configure resource-efficient middleware systems for WSAN motes with dedicated tasks based on a user-selected configuration (e.g., to build vehicular area networks [45] or intelligent living spaces [67]). The mapping between configuration space and the solution space is performed via model transformation and code generation engine (e.g., for JAVA 2 ME). Finally, [183] presents a variability-aware reference architecture that builds on [68].

In fact, DSPLs are also applied to model *self*-adaptive systems. Such systems monitor their environment, analyze the monitored data, plan appropriate measures and execute them to adapt to changing contextual environments. In [217], a framework for precalculating possible or probable configurations for resource-constraint devices is proposed. Such techniques are also useful in the WSN domain because the resources of WSN nodes are typically highly limited. While traditional WSNs were typically configured at deployment time using a fixed TC algorithm (if any) and a fixed parameter set, self-adaptive WSNs can be suitable (e.g., for environmental monitoring to detect wildfires or floods). For instance, in [7], the authors present a framework that may be used, for example, for reconfiguring parameters of WSN nodes in a wildfire detection network. In their case study, the framework is able to predict critical situations (e.g., imminent wildfires) and react appropriately by increasing the sampling rate. Use cases such as flood or wildfire detection show that switching between TC algorithms is a sensible use case and should be a future line of research.

To sum up, existing works on adaptive WSNs mainly focus on specifying variability and reconfigurations on the architectural level of WSNs [7, 18, 46, 67, 172, 183, 216, 217]. In contrast, in this thesis, we focus on leveraging variability of TC algorithm families constructively. To this end, our work is complementary to the related work in this area.

CONCLUSION

In this thesis, we present a practical model-based methodology for developing correct-by-construction dynamic TC algorithms. Such algorithms have been an important subject of research in the WSN research community in the past and are gaining even more importance with the advent of the IoT, which heavily builds on wireless networks.

Our research is driven by the initial observation that the development process of TC algorithms (see Figure 7.1) suffers from a gap between the specification phase and the evaluation phase. The specification phase serves to prove that the specified TC algorithm fulfills crucial consistency properties, and the evaluation phase serves to assess the applicability and performance of the TC algorithm implementation. Traditionally, the gap between the specification and evaluation phases arises from the fact that TC algorithms are implemented manually and, often, using low-level programming languages. This approach is obviously error prone and time consuming, and—worst of all—hinders the traceability between specification and implementation.

We tackle these shortcomings by proposing an MDE methodology for developing TC algorithms. In the following, we summarize the goals and corresponding contributions of this thesis briefly and end with an outline of future research directions.

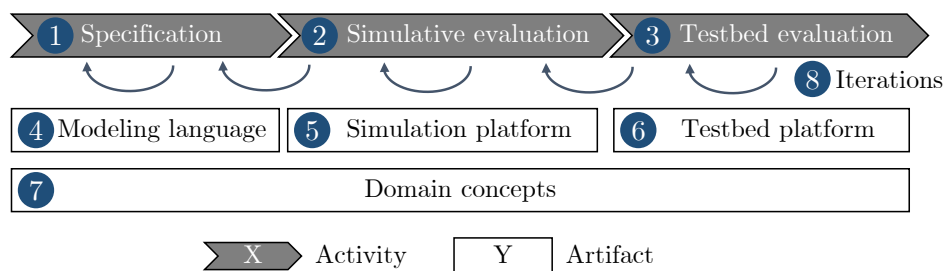


Figure 7.1: Overview of development phases of a TC algorithm (repeated from Figure 1.2)

7.1 SUMMARY

The TC literature lacks a standard representation of the input and output topologies of a TC algorithm. We propose to represent the input topology and virtual topology of a TC algorithm jointly as one attributed graph. The membership of a link in the virtual topology is encoded as a link property. This uniform representation allows us to develop a batch TC algorithm, which processes entire topology snapshots and neglects previous decisions, as well as a dynamic TC algorithm, which accepts individual change events of the underlying topology as input and outputs corresponding updates of the virtual topology (Goal 2). We extended the established TC terminology with concepts for representing elementary modifications of the topology (as context events), the architecture of a dynamic TC algorithm (as TC mechanism, consisting of a TC algorithm and context event handlers), and the architecture of a TC multi-mechanism, which supports the reconfiguration and exchange of TC mechanisms at runtime (TC transitions).

We build our approach on the following formal methods. We specify the set of valid input and virtual topologies of a TC algorithm using a type graph with attributes (i.e., a metamodel). We capture local consistency properties as graph constraints [91] and global consistency properties using second-order logic predicates on the virtual topology of a TC algorithm. We require that the local consistency properties jointly imply the global consistency properties. To enable dynamic TC algorithms, we distinguish between the following two levels of consistency: (i) weak consistency, which must hold permanently, and (ii) strong consistency, which must hold eventually after each TC algorithm execution. Each level of consistency is characterized by a set of local graph constraints. We formalize possible modifications of the topology using GT rules [56, 214], and the order of topology modifications using programmed GT [60]. Based on this foundation, we define six criteria for evaluating whether a TC mechanism specification is correct (Definition 4.7).

To ensure that the TC mechanism specification is correct, we propose a two-step process. In the first step, we employ the constructive approach [44, 91] for refining each GT rule to ensure that the refined GT rule preserves weak consistency. The refinement is carried out by synthesizing weakest application conditions for the GT rules. In the second step, we use the novel anticipation loop synthesis algorithm to refine the control flow of context event handlers to ensure that inevitable violations of synthesized application conditions are resolved prior to handling a context event. In general, this technique can also be used to ensure that the TC algorithm execution always terminates. Using these two refinement steps, we achieve that a TC mechanism specification is correct by construction (Goal 1).

We also aimed at a practical development methodology, which supports the specification, simulative evaluation, and testbed evaluation phase (Goal 3). To bridge the gap between the first and the latter two phases, we propose tool support for the rapid evaluation of a specified TC algorithm. For simulation experiments, we present COBOLT, a

tool integration of the MDE tool `EMOFLON` [135] and the network simulator `SIMONSTRATOR` [208]. For testbed evaluation, we present `cMOFLON`, a lean variant of `EMOFLON` that generates embedded C code for the IoT operating system `CONTIKI` [51]. We present evaluation results for assessing the applicability and efficiency of both tools. For `COBOLT`, we chose the TC algorithms `kTC` [227] as case study. For `cMOFLON`, we added `l*kTC` [239] and `LMST` [140] as further examples.

Finally, we show that our methodology is capable of representing state-of-the-art TC algorithms based on a collection of 32 TC algorithms from which 17 TC algorithms are supported in batch and dynamic mode, and six TC algorithms are supported in batch mode only due to the required ordered processing of links. The TC simulator component that we developed as part of `COBOLT` provides generic, reusable interfaces. `cMOFLON` is a blueprint for developing future domain-specific code generators on top of `EMOFLON`. These results show that our methodology is reusable within and beyond the TC research area (Goal 4).

7.2 OUTLOOK

EXPRESSIVENESS OF GRAPH CONSTRAINTS For specifying local consistency properties, we use premise–conclusion graph constraints [91]. Since the first proposal of the constructive approach in 1995, numerous formalisms have been proposed that allow for expressing additional consistency properties. Examples are nested graph constraints and extensions of graph constraints based on path expressions (as discussed in Section 4.6). To evaluate our methodology based on further types of (graph) constraints, the following two prerequisites must be fulfilled. First, an adoption of the constructive approach for the constraint type must exist. For example, nested graph constraints can also be translated into application conditions [199]. Second, the chosen constraint type should be capable of representing attribute constraints. For premise–conclusion graph constraints, a corresponding extension of the constructive approach exists [44]. Unfortunately, to the best of our knowledge, no published works on handling this type of attribute constraints exist for nested graph constraints or constraints with path expressions.

USAGE SCENARIOS A key consistency property of a TC mechanism is that it maintains connectivity. This means that the virtual topology of the TC mechanism must be connected as long as the input topology is connected. Another common consistency property of TC algorithms is coverage (see also Section 2.4). Coverage requirements are relevant if each mote monitors a certain geographic region (e.g., for intrusion detection). A recurring coverage requirement is that the boundary of a given area should be covered by the joint monitoring areas of all motes. To save energy, a mote may switch into an idle state if the remaining active motes still fulfill the coverage requirements. A promising line of research is the application of our approach to scenarios with coverage requirements. To incorporate coverage requirements, we need to extend the topology metamodel by attributes that represent the operation mode of a mote (i.e., active or idle) and the monitored area (e.g., the radius in meters). This extension allows us to encode the following local consistency property as graph constraint: A mote may only be idle if the monitored area of the active motes in its local view fulfill the coverage requirements. For tie breaking, we can use additional attribute constraints that relate to the remaining energy of the involved motes. The modification of the operation mode of a mote can be modeled using GT rules. Under these assumptions, we can use our correct-by-construction methodology to derive an algorithm for preserving coverage while reducing the number of active motes. This algorithm can also be combined with a link-based TC algorithm.

Another possible extension of our approach relates to the development of probabilistic TC algorithms. A possible probabilistic variant of kTC inactivates a link that fulfills the kTC condition with a certain probability p and activates it with probability $1 - p$. This probabilistic choice avoids the common problem of triangle-based TC algorithms that

the length of routing paths in the virtual topology can grow drastically in comparison to the input topology. Probabilistic TC algorithms can be modeled using probabilistic GT [124]. A probabilistic GT rule has one or more RHS patterns and a probability distribution over these patterns. For the described probabilistic kTC, we obtain an inactivation rule with two RHS patterns: the original RHS, which mandates the inactivation of the link, and an alternative RHS pattern, which activates the link. The described probabilistic kTC violates the original consistency constraints of kTC. The reason is that the TC mechanism topology may contain active links that fulfill the kTC condition. A solution to this problem is to introduce an additional link state (e.g., P). This state indicates that a link was processed by the probabilistic inactivation rule and, nevertheless, shall be active. Furthermore, we could add probabilistic rules for reevaluating the marking of I- and P-links. Such reevaluation rules could alleviate another problem of triangle-based TC algorithms: Due to the inactivation of a link in a triangle, the intermediate node tends to carry additional load because it serves as relay.

Besides these exemplary WSN application scenarios, our approach is applicable in further domains, as sketched in the following. The constructive approach has been used in several works in the security domain (e.g., on rule-based access control [175]). As discussed earlier, applying the pure constructive approach prevents the application of consistency-violating modifications of the system state, for example, due to unfavorable earlier decisions. In our scenario, this leads to a potential nontermination of the TC algorithm (see also Section 4.5). In the security domain, we would like to ensure that certain modifications of the system are always conducted, even if this leads to the revocation of earlier decisions. The anticipation loop synthesis algorithm can be used to revoke only those low-priority decisions that block the pending high-priority modification.

In the cloud computing domain, a common challenge is to map virtual networks in a data center network. This task is called virtual network embedding [255]. The sequence of embedding requests cannot be predicted. This means that the embedding of a virtual network with lower priority may prevent the embedding of a virtual network having a higher priority that arrives later. If the graph constraints are sufficiently expressive to characterize a valid embedding (e.g., in terms of available computing capacity and storage of the substrate network), our approach can be used to revoke earlier low-priority embedding decisions in favor of high-priority embedding decisions.

TOOL SUPPORT With OCL2AC [164], tool support for the synthesis of application conditions of GT rules based on nested graph constraints exists. OCL2AC handles nested graph constraints [83], which are, from a structural point of view, more expressive than premise–conclusion constraints [91]. Unfortunately, the current version of OCL2AC does not support the handling of complex attribute constraints (e.g., involving arithmetics), which are crucial from the perspective of the TC domain. After the addition of this feature, OCL2AC could be used to automate the transformation step from graph constraints to application conditions that we conducted manually.

EMOFLON does not support the specification of multiple application conditions per GT rule (see also Section 5.2). This technical limitation requires a manual transformation of the TC mechanism specification before evaluating it with COBOLT or cMOFLON. It is desirable to extend EMOFLON to support multiple application conditions per GT rule to avoid this error-prone manual step.

The ToCoCo framework, which we use as a target platform for testbed experiments, only supports the implementation of batch TC algorithms. Extending ToCoCo to provide access to context events would alleviate this limitation.

TOPOLOGY CONTROL RECONFIGURATION The focus of this work lies on the correct-by-construction development of individual TC mechanisms. In Section 2.5.4, we also modeled the reconfiguration inside a TC multi-mechanism to changing user requirements (e.g., from minimizing energy consumption to reducing latency) and system contexts (e.g., mote density, active communication patterns) as TC multi-mechanism. We experimented with deriving reconfiguration decisions of a TC multi-mechanism based on performance-influence models using the tool COALA [271]. Given a reconfiguration decision that demands a change from a source TC mechanism to a target TC mechanism, an open question is how to transform the source into the target topology such that a minimal number of links needs to be re-marked. In Section 4.5.2, we showed how to derive such transformation sequences using the anticipation loop synthesis algorithm. This approach can be used to conduct TC transitions from a source TC mechanism to a target TC mechanism as well. From the viewpoint of virtual topology consumer mechanisms (e.g., the routing algorithm), it is desirable to perform TC transitions gradually. This means that that a source TC mechanism topology should fade into the target TC mechanism topology. A TC multi-mechanism has to communicate to consumer mechanisms which links are about to vanish from the virtual topology (e.g., using a property that encodes the time to live of a link).

Besides the temporal coexistence, we can also investigate the regional coexistence of TC mechanisms. Let's imagine that the network is partitioned into geographic or logic regions (e.g., according to different mobility patterns or applications) and that each region shall be able to use a different TC mechanism (e.g., determined using COALA [271]). Then, a crucial question is how the behavior of motes at the border of two regions shall be specified. For example, the constraints of border motes can be a conjunction (in case of the inactive-link constraint) or disjunction (in case of the active-link constraint) of the graph constraints of the adjacent regions. Investigating questions regarding the consistency-preserving composition of graph constraints in regionalized networks is a worthwhile future line of research.

MODELING MESSAGE TRANSFER Our approach focuses on the correctness of the developed TC algorithm based on the local view of a mote. Hereby, we neglect the interaction among the local views of motes. A further line of research focuses on extending our

approach to incorporate message exchange. The first step is to extend the specification with means to describe possible message types as well as operations for sending and receiving messages. Regarding message transfer, we need to specify in how far messages are transferred reliably. An important question here is how to model the probability of message loss if reliable communication cannot be ensured. This could be achieved using probabilistic GT as discussed earlier. Another important aspect is the notion of parallelism and sequentiality of events. This means that we have to specify whether distinct events may occur at the same point in time (true concurrency) or always with a guaranteed intermediate time (parallelism). Finally, as usual in distributed systems, causality constraints are important to verify whether a distributed TC algorithm always leads to a stable state. These considerations already indicate that an extension of our approach from a localized to a distributed view produces a huge number of exciting new research questions.

BIBLIOGRAPHY

- [1] Jean-Raymond Abrial, Michael Butler, Stefan Hallerstede, Thai Son Hoang, Farhad Mehta, and Laurent Voisin. Rodin: an open toolset for modelling and reasoning in Event-B. *International Journal on Software Tools for Technology Transfer (JSTTT)*, 12(6):447–466, 2010. URL: <https://doi.org/10.1007/s10009-010-0145-y>. (Cited on p. 164.)
- [2] Jean-Raymond Abrial and Stefan Hallerstede. Refinement, Decomposition, and Instantiation of Discrete Models: Application to Event-B. *Fundamenta Informaticae*, 77(1–2):1–28, 2007. URL: <https://content.iospress.com/articles/fundamenta-informaticae/fi77-1-2-02>. (Cited on p. 164.)
- [3] Ilka Agricola and Thomas Friedrich. *Elementary Geometry*. American Mathematical Society (AMS), 2008. ISBN: 9780821843475 . URL: <https://bookstore.ams.org/html-43>. (Cited on p. 250.)
- [4] Mohammad Al Saad, Elfiede Fehr, Nicolai Kamenzky, and Jochen Schiller. ScatterClipse: A Model-Driven Tool-Chain for Developing, Testing, and Prototyping Wireless Sensor Networks. In *International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, pages 871–885, 2008. URL: <https://doi.org/10.1109/ISPA.2008.22>. (Cited on pp. 11, 90, and 236.)
- [5] Hande Alemdar and Cem Ersoy. Wireless Sensor Networks for Healthcare: A Survey. *Computer Networks (ComNet)*, 54(15):2688 – 2710, 2010. URL: <https://doi.org/10.1016/j.comnet.2010.05.003>. (Cited on p. 2.)
- [6] Bastian Alt, Markus Weckesser, Christian Becker, Matthias Hollick, Sounak Kar, Anja Klein, Robin Klose, Roland Kluge, Heinz Koeppel, Boris Koldehofe, Wasiur R. KhudaBukhsh, Manisha Luthra, Mahdi Mousavi, Max Mühlhäuser, Martin Pfannenmüller, Amr Rizk, Andy Schürr, and Ralf Steinmetz. Transitions: A Protocol-Independent View of the Future Internet. *Proceedings of the IEEE*, pages 1–12, 2019. URL: <https://doi.org/10.1109/JPROC.2019.2895964>. (Cited on p. 14.)
- [7] Ivan Dario Paez Anaya, Viliam Simko, Johann Bourcier, Noël Plouzeau, and Jean-Marc Jézéquel. A Prediction-Driven Adaptation Approach for Self-adaptive Sensor Networks. In *International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 145–154, New York, NY, USA, 2014. ACM. URL: <https://doi.org/10.1145/2593929.2593941>. (Cited on pp. 40 and 274.)

- [8] Anthony Anjorin, Erhan Leblebici, Roland Kluge, Andy Schürr, and Perdita Stevens. A Systematic Approach and Guidelines to Developing a Triple Graph Grammar. In *International Workshop on Bidirectional Transformations*, volume 1396 of *CEUR*, pages 81–95, 2015. URL: <http://ceur-ws.org/Vol-1396/bx2015.pdf>. (Cited on p. 14.)
- [9] Thorsten Arendt, Enrico Biermann, Stefan Jurack, Christian Krause, and Gabriele Taentzer. Henshin: Advanced Concepts and Tools for In-Place EMF Model Transformations. In *International Conference on Model Driven Engineering Languages and Systems (MODELS)*, volume 6394 of *LNCIS*, pages 121–135. Springer, Berlin Heidelberg, 2010. URL: https://doi.org/10.1007/978-3-642-16145-2_9. (Cited on pp. 161, 166, 172, 173, and 272.)
- [10] Christel Baier, Boudewijn Haverkort, Holger Hermanns, and Joost-Pieter Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Transactions on Software Engineering (TSE)*, 29, 2003. URL: <https://doi.org/10.1109/TSE.2003.1205180>. (Cited on p. 88.)
- [11] Christopher Bak. *GP 2: Efficient Implementation of a Graph Programming Language*. PhD thesis, Department of Computer Science, The University of York, 2015. URL: <http://etheses.whiterose.ac.uk/12586/>. (Cited on pp. 237 and 272.)
- [12] Christopher Bak and Detlef Plump. Compiling Graph Programs to C. In *International Conference on Graph Transformation (ICGT)*, pages 102–117. Springer International, 2016. URL: https://doi.org/10.1007/978-3-319-40530-8_7. (Cited on pp. 161 and 237.)
- [13] Paolo Baldan, Andrea Corradini, and Barbara König. A Framework for the Verification of Infinite-State Graph Transformation Systems. *Information and Computation*, 206(7):869–907, 2008. URL: <https://doi.org/10.1016/j.ic.2008.04.002>. (Cited on p. 167.)
- [14] Massimo Baleani, Alberto Ferrari, Leonardo Mangeruca, Alberto L. Sangiovanni-Vincentelli, Ulrich Freund, Erhard Schlenker, and Hans-Jörg Wolff. Correct-by-Construction Transformations across Design Environments for Model-Based Embedded Software Development. In *Proceedings of Design, Automation and Test in Europe (DATE 2005)*, volume 2, pages 1044–1049, NY, USA, 2005. IEEE. URL: <https://doi.org/10.1109/DATE.2005.105>. (Cited on p. 163.)
- [15] Ananda Basu, Bensalem Bensalem, Marius Bozga, Jacques Combaz, Mohamad Jaber, Thanh-Hung Nguyen, and Joseph Sifakis. Rigorous Component-Based System Design Using the BIP Framework. *IEEE Software*, 28(3):41–48, 2011. URL: <https://doi.org/10.1109/MS.2011.27>. (Cited on p. 163.)

- [16] Basil Becker, Dirk Beyer, Holger Giese, Florian Klein, and Daniela Schilling. Symbolic Invariant Verification for Systems with Dynamic Structural Adaptation. In *International Conference on Software Engineering (ICSE)*, pages 72–81, New York, NY, USA, 2006. ACM. URL: <https://doi.org/10.1145/1134285.1134297>. (Cited on p. 165.)
- [17] Basil Becker and Holger Giese. Modeling of Correct Self-Adaptive Systems: A Transformation System Based Approach. In *International Conference on Soft Computing As Transdisciplinary Science and Technology*, pages 508–516, New York, NY, USA, 2008. ACM. URL: <https://doi.org/10.1145/1456223.1456326>. (Cited on p. 88.)
- [18] Nelly Bencomo, Pete Sawyer, Gordon Blair, and Paul Grace. Dynamically Adaptive Systems are Product Lines too: Using Model-Driven Techniques to Capture Dynamic Variability of Adaptive Systems. In *International Software Product Line Conference (SPLC)*, NY, USA, 2008. IEEE. URL: <https://doi.org/10.1109/SPLC.2008.69>. (Cited on pp. 273 and 274.)
- [19] Amel Bennaceur, Robert France, Giordano Tamburrelli, Thomas Vogel, Pieter J. Mosterman, Walter Cazzola, Fabio M. Costa, Alfonso Pierantonio, Matthias Tichy, Mehmet Akşit, Pär Emmanuelsen, Huang Gang, Nikolaos Georgantas, and David Redlich. *Mechanisms for Leveraging Models at Runtime in Self-Adaptive Software*, pages 19–46. Springer International, Cham, 2014. URL: https://doi.org/10.1007/978-3-319-08915-7_2. (Cited on p. 86.)
- [20] Luca Berardinelli, Antinisca Di Marco, Stefano Pace, Luigi Pomante, and Walter Tiberti. Energy Consumption Analysis and Design of Energy-Aware WSN Agents in fUML. In *European Conference on Modelling Foundations and Applications (ECMFA)*, volume 9153 of LNCS, pages 1–17. Springer International, Cham, 2015. URL: https://doi.org/10.1007/978-3-319-21151-0_1. (Cited on pp. 11 and 236.)
- [21] Gábor Bergmann, István Dávid, Ábel Hegedüs, Ákos Horváth, István Ráth, Zoltán Ujhelyi, and Dániel Varró. Viatra 3: A Reactive Model Transformation Platform. In *International Conference on Model Transformation (ICMT)*, pages 101–110, Cham, 2015. Springer International. URL: https://doi.org/10.1007/978-3-319-21155-8_8. (Cited on pp. 172, 173, 203, and 221.)
- [22] Sami Beydeda, Matthias Book, and Volker Gruhn. *Model-Driven Software Development*. Springer, 15th edition, 2005. ISBN: 978-3-540-28554-0 . (Cited on p. 11.)
- [23] D. Beyer, A. Noack, and C. Lewerentz. Efficient Relational Calculation for Software Analysis. *IEEE Transactions on Software Engineering (TSE)*, 31(2):137–149, 2005. (Cited on p. 165.)

- [24] Douglas M. Blough, Mauro Leoncini, Giovanni Resta, and Paolo Santi. The K-Neigh Protocol for Symmetric Topology Control in Ad Hoc Networks. In *ACM International Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc)*, pages 141–152, New York, NY, USA, 2003. ACM. URL: <https://doi.org/10.1145/778415.778433>. (Cited on p. 245.)
- [25] Prosenjit Bose, Joachim Gudmundsson, and Michiel Smid. Constructing Plane Spanners of Bounded Degree and LowWeight. *Algorithmica*, 42(3):249–264, 2005. URL: <https://doi.org/10.1007/s00453-005-1168-8>. (Cited on p. 245.)
- [26] Paolo Bottoni, Esther Guerra, and Juan de Lara. Enforced Generative Patterns for the Specification of the Syntax and Semantics of Visual Languages. *Visual Languages & Computing (JVLC)*, 19(4):429–455, 2008. URL: <https://doi.org/10.1016/j.jvlc.2008.04.004>. (Cited on p. 162.)
- [27] Paolo Bottoni, Francesco Parisi-Presicce, and Gabriele Taentzer. *Specifying Coherent Refactoring of Software Artefacts with Distributed Graph Transformations*, pages 95–126. IGI Global, Hershey, PA, USA, 2005. URL: <https://doi.org/10.4018/978-1-59140-527-6.ch005>. (Cited on p. 162.)
- [28] Antonio Bucchiarone, Hartmut Ehrig, Claudia Ermel, Patrizio Pelliccione, and Olga Runge. Rule-Based Modeling and Static Analysis of Self-Adaptive Systems by Graph Transformation. In *Software, Services, and Systems*, volume 8950 of *LNCS*, pages 582–601. Springer International, Cham, 2015. URL: https://doi.org/10.1007/978-3-319-15545-6_33. (Cited on p. 86.)
- [29] Márton Búr, Gábor Szilágyi, András Vörös, and Dániel Varró. Distributed Graph Queries for Runtime Monitoring of Cyber-Physical Systems. In *Fundamental Approaches to Software Engineering (FASE)*, pages 111–128, Cham, 2018. Springer International. URL: https://doi.org/10.1007/978-3-319-89363-1_7. (Cited on pp. 37, 86, and 237.)
- [30] Johannes Bürdek, Sascha Lity, Malte Lochau, Markus Berens, Ursula Goltz, and Andy Schürr. Staged Configuration of Dynamic Software Product Lines with Complex Binding Time Constraints. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 16:1–16:8, New York, NY, USA, 2013. ACM. URL: <https://doi.org/10.1145/2556624.2556627>. (Cited on p. 260.)
- [31] Tracy Camp, Jeff Boleng, and Vanessa Davies. A Survey of Mobility Models for ad hoc Network Research. *Wireless Communications and Mobile Computing*, 2(5):483–502, 2002. URL: <https://doi.org/10.1002/wcm.72>. (Cited on pp. 191 and 192.)
- [32] Mauricio Castillo-Effer, Daniel H. Quintela, Ramiro Jordan, Wayne Westhoff, and Wilfrido Moreno. Wireless Sensor Networks for Flash-Flood Alerting. In *International Caracas Conference on Devices, Circuits and Systems*, volume 1, pages 142–146,

2004. URL: <https://doi.org/10.1109/ICDCS.2004.1393370>. (Cited on pp. 2 and 21.)
- [33] Yunxia Chen and Qing Zhao. On the Lifetime of Wireless Sensor Networks. *IEEE Communications Letters (LCOMM)*, 9(11):976–978, 2005. URL: <https://doi.org/10.1109/LCOMM.2005.11010>. (Cited on p. 254.)
- [34] Xiaoyu Chu and Harish Sethu. A New Power-Aware Distributed Topology Control Algorithm for Wireless Ad Hoc Networks. In *IEEE Global Telecommunications Conference (GLOBECOM)*, pages 1–5, NY, USA, 2011. IEEE. URL: <https://doi.org/10.1109/GLOCOM.2011.6133917>. (Cited on pp. 191 and 243.)
- [35] Xiaoyu Chu and Harish Sethu. Cooperative Topology Control with Adaptation for Improved Lifetime in Wireless Ad-Hoc Networks. In *IEEE International Conference on Computer Communications (INFOCOM)*, pages 262–270, NY, USA, 2012. IEEE. URL: <https://doi.org/10.1109/INFCOM.2012.6195667>. (Cited on pp. 9, 13, 89, 244, 253, and 256.)
- [36] Xiaoyu Chu and Harish Sethu. Cooperative Topology Control with Adaptation for improved lifetime in wireless sensor networks. *Ad Hoc Networks (AdHoc)*, 30:99–114, 2015. URL: <https://doi.org/10.1016/j.adhoc.2015.03.007>. (Cited on p. 89.)
- [37] Lawrence Chung, Brian A. Nixon, and Eric Yu. Using Non-functional Requirements to Systematically Support Change. In *International Symposium on Requirements Engineering*, pages 132–139, NY, USA, 1995. IEEE. URL: <https://doi.org/10.1109/ISRE.1995.512554>. (Cited on p. 26.)
- [38] Robert Clarisó, Jordi Cabot, Esther Guerra, and Juan de Lara. Backwards Reasoning for Model Transformations: Method and Applications. *Systems and Software (JSS)*, 116:113–132, 2016. URL: <https://doi.org/10.1016/j.jss.2015.08.017>. (Cited on p. 165.)
- [39] Bruno Courcelle. The Monadic Second-Order Logic of Graphs – (I) Recognizable Sets of Finite Graphs. *Information and Computation*, 85(1):12 – 75, 1990. URL: [https://doi.org/10.1016/0890-5401\(90\)90043-H](https://doi.org/10.1016/0890-5401(90)90043-H). (Cited on p. 161.)
- [40] John D. Day and Hubert Zimmermann. The OSI Reference Model. *Proceedings IEEE*, 71(12):1334–1340, 1983. URL: <https://doi.org/10.1109/PROC.1983.12775>. (Cited on pp. 28 and 29.)
- [41] Juan de Lara and Esther Guerra. Pattern-Based Model-to-Model Transformation. In *International Conference on Graph Transformation (ICGT)*, pages 426–441, Berlin Heidelberg, 2008. Springer. URL: https://doi.org/10.1007/978-3-540-87405-8_29. (Cited on p. 164.)

- [42] Rogério de Lemos, Holger Giese, Hausi A. Müller, Mary Shaw, Jesper Andersson, Marin Litoiu, Bradley Schmerl, Gabriel Tamura, Norha M. Villegas, Thomas Vogel, Danny Weyns, Luciano Baresi, Basil Becker, Nelly Bencomo, Yuriy Brun, Bojan Cukic, Ron Desmarais, Schahram Dustdar, Gregor Engels, Kurt Geihs, Karl M. Göschka, Alessandra Gorla, Vincenzo Grassi, Paola Inverardi, Gabor Karsai, Jeff Kramer, Antónia Lopes, Jeff Magee, Sam Malek, Serge Mankovskii, Raffaella Mirandola, John Mylopoulos, Oscar Nierstrasz, Mauro Pezzè, Christian Prehofer, Wilhelm Schäfer, Rick Schlichting, Dennis B. Smith, João Pedro Sousa, Ladan Tahvildari, Kenny Wong, and Jochen Wuttke. *Software Engineering for Self-Adaptive Systems: A Second Research Roadmap*, pages 1–32. Springer, Berlin Heidelberg, 2013. URL: https://doi.org/10.1007/978-3-642-35813-5_1. (Cited on pp. 86 and 88.)
- [43] Frederik Deckwerth. *Static Verification Techniques for Attributed Graph Transformations*. PhD thesis, Technische Universität Darmstadt, Darmstadt, 2017. URL: <http://tuprints.ulb.tu-darmstadt.de/6150/>. (Cited on pp. 160, 161, and 172.)
- [44] Frederik Deckwerth and Gergely Varró. Attribute Handling for Generating Preconditions from Graph Constraints. In *International Conference on Graph Transformation (ICGT)*, pages 81–96, Cham, 2014. Springer International. URL: https://doi.org/10.1007/978-3-319-09108-2_6. (Cited on pp. 12, 14, 93, 102, 104, 160, 161, 172, 198, 276, and 278.)
- [45] Flávia C. Delicato, Lidia Fuentes, Nadia Gámez, and Paulo F. Pires. A Middleware Family for VANETs. In *International Conference on Ad-Hoc, Mobile and Wireless Networks (ADHOC-NOW)*, pages 379–384, Berlin Heidelberg, 2009. Springer. URL: https://doi.org/10.1007/978-3-642-04383-3_31. (Cited on p. 273.)
- [46] Flávia C. Delicato, Lidia Fuentes, Nadia Gámez, and Paulo F. Pires. Variabilities of Wireless and Actuators Sensor Network Middleware for Ambient Assisted Living. In *Distributed Computing, Artificial Intelligence, Bioinformatics, Soft Computing, and Ambient Assisted Living (IWANN Workshops)*, volume 5518 of LNCS, pages 851–858, Berlin Heidelberg, 2009. Springer. URL: https://doi.org/10.1007/978-3-642-02481-8_129. (Cited on pp. 273 and 274.)
- [47] Edsger Wybe Dijkstra. *A Discipline of Programming*, volume 1. Prentice Hall, 1976. ISBN: 978-0132158718 . (Cited on pp. 47 and 162.)
- [48] David P. Dobkin, Steven J. Friedman, and Kenneth J. Supowit. Delaunay Graphs are almost as Good as Complete Graphs. *Discrete & Computational Geometry*, 5(4):399–407, 1990. URL: <https://doi.org/10.1007/BF02187801>. (Cited on pp. 205 and 244.)
- [49] M. Dohler, D. Barthel, F. Maraninchi, L. Mounier, S. Aubert, C. Dugas, A. Buhrig, F. Pagnat, M. Renaudin, A. Duda, M. Heusse, and F. Valois. The ARESA Project:

- Facilitating Research, Development and Commercialization of WSNs. In *IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON 2007)*, pages 590–599, 2007. URL: <https://doi.org/10.1109/SAHCN.2007.4292871>. (Cited on p. 9.)
- [50] Falko Dressler, Margit Mutschlechner, Bijun Li, Rüdiger Kapitza, Simon Ripperger, Christopher Eibel, Benedict Herzog, Timo Hönig, and Wolfgang Schröder-Preikschat. Monitoring Bats in the Wild: On Using Erasure Codes for Energy-Efficient Wireless Sensor Networks. *ACM Transactions Sen. Netw.*, 12(1):7:1–7:29, 2016. URL: <https://doi.org/10.1145/2875426>. (Cited on pp. 2 and 21.)
- [51] Adam Dunkels, Björn Gronvall, and Thiemo Voigt. Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors. In *International Conference on Local Computer Networks (LCN)*, pages 455–462, NY, USA, 2004. IEEE. URL: <https://doi.org/10.1109/LCN.2004.38>. (Cited on pp. 13, 170, 171, 204, 205, and 277.)
- [52] Adam Dunkels, Fredrik Österlind, and Zhitao He. An Adaptive Communication Architecture for Wireless Sensor Networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 335–349, New York, NY, USA, 2007. ACM. URL: <https://doi.org/10.1145/1322263.1322295>. (Cited on p. 207.)
- [53] Johannes Dyck and Holger Giese. Inductive Invariant Checking with Partial Negative Application Conditions. In *International Conference on Graph Transformation (ICGT)*, pages 237–253, Cham, 2015. Springer International. URL: https://doi.org/10.1007/978-3-319-21145-9_15. (Cited on pp. 88, 94, and 163.)
- [54] Addison-Wesley Pearson Education. *BUGS in Writing, Revised Edition: A Guide to Debugging Your Prose*. Pearson Education, New York City, NY, US, 1998. ISBN: 978-0201379211 . (Cited on p. 16.)
- [55] Hartmut Ehrig, Karsten Ehrig, Annegret Habel, and Karl-Heinz Pennemann. Theory of Constraints and Application Conditions: From Graphs to High-Level Structures. *Fundamenta Informaticae*, 74:135–166, 2006. URL: <http://content.iospress.com/articles/fundamenta-informaticae/fi74-1-07>. (Cited on p. 85.)
- [56] Hartmut Ehrig, Karsten Ehrig, Ulrike Prange, and Gabriele Taentzer. *Fundamentals of Algebraic Graph Transformation*. Springer, Berlin Heidelberg, 2006. ISBN: 978-3-540-31188-1 . URL: <https://doi.org/10.1007/3-540-31188-2>. (Cited on pp. 57, 70, 160, and 276.)
- [57] EMF4CPP on GitHub. <https://github.com/catedrasaes-umu/emf4cpp>. (last accessed: 2017-02-14). (Cited on p. 237.)

- [58] Fatima Essaadi, Yann Ben Maissa, and Mohammed Dahchour. MDE-Based Languages for Wireless Sensor Networks Modeling: A Systematic Mapping Study. In *International Symposium on Ubiquitous Computing (UNet)*, pages 331–346, Singapore, 2017. Springer Singapore. URL: https://doi.org/10.1007/978-981-10-1627-1_26. (Cited on p. 236.)
- [59] R. Fagin, L.J. Stockmeyer, and M.Y. Vardi. On Monadic NP vs Monadic co-NP. *Information and Computation (INCO)*, 120(1):78 – 92, 1995. URL: <https://doi.org/10.1006/inco.1995.1100>. (Cited on pp. 46, 67, and 211.)
- [60] Thorsten Fischer, Jörg Niere, Lars Torunski, and Albert Zündorf. Story Diagrams: A New Graph Rewrite Language based on the Unified Modeling Language. In *International Workshop on Theory and Application of Graph Transformation (TAGT)*, pages 296–309, Berlin Heidelberg, 1998. Springer. URL: https://doi.org/10.1007/978-3-540-46464-8_21. (Cited on pp. 12, 46, 76, 77, 88, and 276.)
- [61] Chien-Liang Fok, Gruia-Catalin Roman, and Chenyang Lu. Agilla: A Mobile Agent Middleware for Self-Adaptive Wireless Sensor Networks. *ACM Transactions of Autonomous Adaptive Systems (TAAS)*, 4(3):16:1–16:26, 2009. URL: <https://doi.org/10.1145/1552297.1552299>. (Cited on pp. 40, 87, and 90.)
- [62] Martin Fowler, Kent Beck, John Brant, William Opdyke, and Don Roberts. *Refactoring: Improving the Design of Existing Code*. Addison-Wesley Professional, Boston, MA, USA, 1999. ISBN: 9780201485677 . (Cited on p. 162.)
- [63] Harald T. Friis. A Note on a Simple Transmission Formula. *Proceedings IRE*, 34(5):254–256, 1946. URL: <https://doi.org/10.1109/JRProceedings1946.234568>. (Cited on pp. 26 and 255.)
- [64] Lars Fritsche, Jens Kosiol, Andy Schürr, and Gabriele Taentzer. Short-Cut Rules: Sequential Composition of Rules Avoiding Unnecessary Deletions. In *International Workshop on Graph Computation Models (GCM)*, pages 1–28, Bremen, 2018. Uni Bremen. URL: <https://www.gcm2018.uni-bremen.de/>. (Cited on p. 165.)
- [65] Alexander Frömmgen, Mohamed Hassan, Roland Kluge, Mahdi Mousavi, Max Mühlhäuser, Sabrina Müller, Mathias Schnee, Michael Stein, and Markus Weckesser. Mechanism Transitions: A New Paradigm for a Highly Adaptive Internet, 2016. URL: <http://tubiblio.ulb.tu-darmstadt.de/79757/>. (Cited on p. 14.)
- [66] Fabian Fuchs, Markus Völker, and Dorothea Wagner. Simulation-Based Analysis of Topology Control Algorithms for Wireless Ad Hoc Networks. In *Mediterranean Conference on Algorithms (MedAlg)*, pages 188–202, Berlin Heidelberg, 2012. Springer. URL: https://doi.org/10.1007/978-3-642-34862-4_14. (Cited on pp. 9, 19, 23, 29, 239, 241, 243, 244, 245, 246, 247, 249, and 258.)

- [67] Lidia Fuentes, Nadia Gamez, and Pablo Sanchez. Variability in Ambient Intelligence: A Family of Middleware Solution. *Ubiquitous Developments in Ambient Computing and Intelligence: Human-Centered Applications*, pages 71–83, 2011. URL: <https://doi.org/10.4018/978-1-60960-549-0.ch006>. (Cited on pp. 273 and 274.)
- [68] Lidia Fuentes and Nadia Gámez. Configuration Process of a Software Product Line for AmI Middleware. *Universal Computer Science (JUCS)*, 16(12):1592–1611, 2010. URL: <https://doi.org/10.3217/jucs-016-12-1592>. (Cited on p. 273.)
- [69] Sebastian Gabmeyer, Petra Kaufmann, Martina Seidl, Martin Gogolla, and Gerti Kappel. A Feature-Based Classification of Formal Verification Techniques for Software Models. *Software & Systems Modeling (SoSyM)*, pages 1–26, 2017. URL: <https://doi.org/10.1007/s10270-017-0591-z>. (Cited on p. 163.)
- [70] K. Ruben Gabriel and Robert R. Sokal. A New Statistical Approach to Geographic Variation Analysis. *Systematic Biology*, 18(3):259–278, 1969. URL: <https://doi.org/10.2307/2412323>. (Cited on pp. 31 and 250.)
- [71] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995. ISBN: 0-201-63361-2 . (Cited on pp. 43 and 53.)
- [72] Jie Gao, L. J. Guibas, J. Hershberger, Li Zhang, and An Zhu. Geometric Spanners for Routing in Mobile networks. *IEEE J. on Selected Areas in Communications (JSAC)*, 23(1):174–185, 2005. URL: <https://doi.org/10.1109/JSAC.2004.837364>. (Cited on pp. 191 and 244.)
- [73] David Gay, Philip Levis, Robert von Behren, Matt Welsh, Eric Brewer, and David Culler. The nesC Language: A Holistic Approach to Networked Embedded Systems. *SIGPLAN Not.*, 49(4):41–51, 2014. URL: <https://doi.org/10.1145/2641638.2641652>. (Cited on p. 173.)
- [74] Rubino Geiß, Veit Bätz, Daniel Grund, Sebastian Hack, and Adam M. Szalkowski. GrGen: A Fast SPO-Based Graph Rewriting Tool. In *International Conference on Graph Transformation (ICGT)*, volume 4178 of LNCS, pages 383–397, Berlin Heidelberg, 2006. Springer. URL: https://doi.org/10.1007/11841883_27. (Cited on pp. 172, 173, and 237.)
- [75] Oliviu C. Ghica, Goce Trajcevski, Peter Scheuermann, Zachary Bischof, and Nikolay Valtchanov. SIDnet-SWANS – A simulator and integrated development platform for sensor networks applications. In *SenSys’08 - Proceedings of the 6th ACM Conference on Embedded Networked Sensor Systems*, pages 385–386, 2008. (Cited on pp. 173 and 178.)

- [76] Holger Giese and Leen Lambers. Towards Automatic Verification of Behavior Preservation for Model Transformation via Invariant Checking. In *International Conference on Graph Transformation (ICGT)*, pages 249–263, Berlin Heidelberg, 2012. Springer. URL: https://doi.org/10.1007/978-3-642-33654-6_17. (Cited on p. 163.)
- [77] David Giessing. From Programmed Graph Transformation to Embedded C Code: A Case Study. Bachelor’s thesis, TU Darmstadt, 2016. (Cited on pp. 15 and 205.)
- [78] Object Management Group. *OMG Unified Modeling Language Specification 1.4*. OMG, 2001. URL: <http://www.omg.org/spec/UML/>. (Cited on pp. 12, 46, 48, 76, 77, and 90.)
- [79] Object Management Group. *Object Constraint Language 2.0*. OMG, 2003. URL: <https://www.omg.org/spec/OCL/2.0/>. (Cited on p. 165.)
- [80] Object Management Group. *UML Specification, Version 2.0*. OMG, 2005. URL: <http://www.omg.org/spec/UML/>. (Cited on p. 48.)
- [81] Esther Guerra and Juan de Lara. Model View Management with Triple Graph Transformation Systems. In *International Conference on Graph Transformation (ICGT)*, pages 351–366, Berlin Heidelberg, 2006. Springer. URL: https://doi.org/10.1007/11841883_25. (Cited on pp. 162 and 164.)
- [82] Annegret Habel, Reiko Heckel, and Gabriele Taentzer. Graph grammars with Negative Application Conditions. *Fundamenta Informaticae*, 26(3), 1996. URL: <https://doi.org/10.3233/FI-1996-263404>. (Cited on p. 70.)
- [83] Annegret Habel and Karl-Heinz Pennemann. Correctness of High-Level Transformation Systems Relative to Nested Conditions. *Mathematical Structures in Computer Science*, 19(2):245–296, 2009. URL: <https://doi.org/10.1017/S0960129508007202>. (Cited on pp. 56, 67, 85, 160, 166, 270, and 279.)
- [84] Annegret Habel and Hendrik Radke. Expressiveness of Graph Conditions with Variables. In *International Colloquium on Graph and Model Transformation (GraMoT 2010)*, volume 30, Dortmund, 2010. ECEASST. URL: <https://doi.org/10.14279/tuj.eceasst.30.404>. (Cited on pp. 86 and 160.)
- [85] Anthony Hall and Roderick Chapman. Correctness by Construction: Developing a Commercial Secure System. *IEEE Software*, 19(1):18–25, 2002. URL: <https://doi.org/10.1109/52.976937>. (Cited on p. 163.)
- [86] Svein Hallsteinsen, Mike Hinchey, Sooyong Park, and Klaus Schmid. Dynamic Software Product Lines. *Computer*, 41(4):93–95, 2008. URL: <https://doi.org/10.1109/MC.2008.123>. (Cited on p. 13.)

- [87] Hans Hansson and Bengt Jonsson. A Logic for Reasoning about Time and Reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994. URL: <https://doi.org/10.1007/BF01211866>. (Cited on p. 87.)
- [88] Herman Hartmann and Tim Trew. Using Feature Diagrams with Context Variability to Model Multiple Product Lines for Software Supply Chains. In *International Conference on Software Product Lines (SPLC)*, pages 12–21, NY, USA, 2008. IEEE. URL: <https://doi.org/10.1109/SPLC.2008.15>. (Cited on p. 82.)
- [89] Jan Hendrik Hausmann, Reiko Heckel, and Stefan Sauer. Extended Model Relations with Graphical Consistency Conditions. In *Workshop on Consistency Problems in UML-based Software Development (UML 2002)*, Blekinge Institute of Technology, Research Report 2002:06, pages 61–74. Department of Software Engineering and Computer Science, Blekinge Institute of Technology, 2002. URL: <http://www.db.informatik.uni-bremen.de/umlbib/conf/WRKUML2002CP.html>. (Cited on p. 97.)
- [90] Reiko Heckel, Georgios Lajios, and Sebastian Menge. Stochastic Graph Transformation Systems. In *International Conference on Graph Transformation (ICGT)*, pages 210–225, Berlin Heidelberg, 2004. Springer. URL: https://doi.org/10.1007/978-3-540-30203-2_16. (Cited on pp. 84, 87, 88, 167, and 235.)
- [91] Reiko Heckel and Annika Wagner. Ensuring Consistency of Conditional Graph Rewriting – A Constructive Approach. In *Joint COMPUGRAPH/SEMAGRAPH Workshop*, volume 2 of *Electronic Notes in Theoretical Computer Science (ENTCS)*, pages 118–126, Amsterdam, 1995. Elsevier. URL: [https://doi.org/10.1016/S1571-0661\(05\)80188-4](https://doi.org/10.1016/S1571-0661(05)80188-4). (Cited on pp. 12, 14, 47, 56, 57, 62, 84, 93, 94, 97, 102, 103, 115, 160, 161, 162, 165, 198, 271, 276, 278, and 279.)
- [92] Christian Henke, Carsten Rustemeier, Tobias Schneider, Joachim Böcker, and Ansgar Trächtler. RailCab – Ein Schienenverkehrssystem mit autonomen, Linear-motor getriebenen Einzelfahrzeugen. In *Internationaler ETG-Kongress 2007*, pages 1–9, Berlin, Germany, 2007. VDE Verlag. URL: <https://www.vde-verlag.de/proceedings-de/433063053.html>. (Cited on p. 87.)
- [93] Maximilian Herbst. Graph Constraints meet Topology Control: Designing Correct Graph-Based Topology Control Algorithms. Bachelor’s thesis, TU Darmstadt, 2016. (Cited on pp. 15 and 240.)
- [94] Frank Hermann, Susann Gottmann, Nico Nachtigall, Benjamin Braatz, Gianluigi Morelli, Alain Pierre, and Thomas Engel. On an Automated Translation of Satellite Procedures Using Triple Graph Grammars. In *International Conference on Model Transformation (ICMT)*, pages 50–51, Berlin Heidelberg, 2013. Springer. URL: https://doi.org/10.1007/978-3-642-38883-5_4. (Cited on p. 11.)

- [95] Jason Hill, Robert Szewczyk, Alec Woo, Seth Hollar, David Culler, and Kristofer S. J. Pister. System Architecture Directions for Networked Sensors. *SIGOPS Oper. Syst. Rev.*, 34(5):93–104, 2000. URL: <https://doi.org/10.1145/384264.379006>. (Cited on p. 90.)
- [96] Gerard J. Holzmann. *The SPIN Model Checker: Primer and Reference Manual*, volume 1003. Addison-Wesley Reading, 2004. ISBN: 0-321-22862-6. (Cited on p. 166.)
- [97] Danny Hughes, Phil Greenwood, Geoff Coulson, and Gordon Blair. GridStix: Supporting Flood Prediction Using Embedded Hardware and Next Generation Grid Middleware. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 1–6, 2006. URL: <https://doi.org/10.1109/WOWMOM.2006.49>. (Cited on pp. 2 and 21.)
- [98] Marieke Huisman and Julia Rubin, editors. *Fundamental Approaches to Software Engineering (FASE)*. Springer International, Cham, 2017. URL: <https://doi.org/10.1007/978-3-662-54494-5>. (Cited on p. 50.)
- [99] Sajid Hussain and Obidul Islam. An Energy Efficient Spanning Tree Based Multi-Hop Routing in Wireless Sensor Networks. In *IEEE Wireless Communications and Networking Conference (WCNC)*, pages 4383–4388, NY, USA, 2007. IEEE. URL: <https://doi.org/10.1109/WCNC.2007.799>. (Cited on p. 84.)
- [100] Daniel Jackson. Alloy: A Lightweight Object Modelling Notation. *ACM Transactions Softw. Eng. Methodol. (TOSEM)*, 11(2):256–290, 2002. URL: <https://doi.org/10.1145/505145.505149>. (Cited on p. 167.)
- [101] hs Jaxenter. Eclipse Modeling Framework – Interview with Ed Merks. <https://jaxenter.com/eclipse-modeling-framework-interview-with-ed-merks-100007.html>, 2010. Last accessed: 2018-09-24. (Cited on p. 50.)
- [102] Márk Jelasity. Gossip. In *Self-organising Software: From Natural to Artificial Adaptation*, pages 139–162, Berlin Heidelberg, 2011. Springer. URL: https://doi.org/10.1007/978-3-642-17348-6_7. (Cited on p. 87.)
- [103] Lujun Jia, Rajmohan Rajaraman, and Christian Scheideler. On Local Algorithms for Topology Control and Routing in Ad Hoc Networks. In *Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA)*, pages 220–229, New York, NY, USA, 2003. ACM. URL: <https://doi.org/10.1145/777412.777447>. (Cited on p. 29.)
- [104] Frédéric Jouault, Freddy Allilaire, Jean Bézivin, Ivan Kurtev, and Patrick Valduriez. ATL: A QVT-like Transformation Language. In *Companion to the 21st ACM SIGPLAN Symposium on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA)*, pages 719–720, New York, NY, USA, 2006. ACM. URL: <https://doi.org/10.1145/1176617.1176691>. (Cited on p. 172.)

- [105] Nafiseh Kahani, Mojtaba Bagherzadeh, James R. Cordy, Juergen Dingel, and Daniel Varró. Survey and Classification of Model Transformation Tools. *Software & Systems Modeling (SoSyM)*, 2018. URL: <https://doi.org/10.1007/s10270-018-0665-6>. (Cited on p. 172.)
- [106] Joseph M. Kahn, Randy H. Katz, and Kristofer S. J. Pister. Next Century Challenges: Mobile Networking for “Smart Dust”. In *International Conference on Mobile Computing and Networking (MobiCom)*, pages 271–278, New York, NY, USA, 1999. ACM. URL: <https://doi.org/10.1145/313451.313558>. (Cited on p. 2.)
- [107] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical report, Software Engineering Institute, Carnegie-Mellon University, 1990. URL: <https://resources.sei.cmu.edu/library/asset-view.cfm?assetid=11231>. CMU/SEI-90-TR-21, ESD-90-TR-222. (Cited on p. 81.)
- [108] Ahmet Serkan Karatas, Halit Oguztüzün, and Ali H. Dogru. Mapping Extended Feature Models to Constraint Logic Programming over Finite Domains. In *International Conference on Software Product Lines (SPLC)*, pages 286–299, Berlin Heidelberg, 2010. Springer. URL: https://doi.org/10.1007/978-3-642-15579-6_20. (Cited on p. 81.)
- [109] Brad Karp and Hsiang Te Kung. GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 243–254, New York, NY, USA, 2000. ACM. URL: <https://doi.org/10.1145/345910.345953>. (Cited on pp. 28, 31, 191, 205, 243, and 250.)
- [110] Michael Katelman, José Meseguer, and Jennifer Hou. Redesign of the LMST Wireless Sensor Protocol through Formal Modeling and Statistical Model Checking. In *IFIP Joint International Conference on Formal Techniques for Distributed Systems (FMOODS)*, volume 5051 of LNCS, pages 150–169. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-68863-1_10. (Cited on p. 167.)
- [111] Ajab Khan. *Stochastic Simulation of P2P VoIP Network Reconfiguration Using Graph Transformation*. PhD thesis, University of Leicester, 2013. URL: <http://hdl.handle.net/2381/10428>. (Cited on p. 235.)
- [112] Ajab Khan and Reiko Heckel. Model-based stochastic simulation of super peer promotion in P2P VoIP using graph transformation. In *International Conference on Data Communication Networking (DCNET)*, pages 1–11, NY, USA, 2011. IEEE. URL: <https://ieeexplore.ieee.org/document/6835773>. (Cited on p. 235.)
- [113] Ajab Khan, Reiko Heckel, Paolo Torrini, and István Ráth. Model-Based Stochastic Simulation of P2P VoIP Using Graph Transformation System. In *Analytical and*

- Stochastic Modeling Techniques and Applications*, pages 204–217, Berlin Heidelberg, 2010. Springer. URL: https://doi.org/10.1007/978-3-642-13568-2_15. (Cited on p. 235.)
- [114] Ittipong Khemapech, Alan Miller, and Ishbel Duncan. A survey of Transmission Power Control in Wireless Sensor Networks. In *Annual Postgraduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting (PGNet9s)*, pages 15–20, 2007. URL: <http://www.cms.livjm.ac.uk/pgnet2007/Proceedings/>. (Cited on p. 254.)
- [115] Sukun Kim. *Wireless Sensor Networks for High Fidelity Sampling*. PhD thesis, EECS Department, University of California, Berkeley, 2007. URL: <http://www2.eecs.berkeley.edu/Pubs/TechRpts/2007/EECS-2007-91.html>. (Cited on pp. 2 and 21.)
- [116] Sukun Kim, Shamim Pakzad, David Culler, James Demmel, Gregory Fennes, Steven Glaser, and Martin Turon. Health Monitoring of Civil Infrastructures Using Wireless Sensor Networks. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, pages 254–263, New York, NY, USA, 2007. ACM. URL: <https://doi.org/10.1145/1236360.1236395>. (Cited on pp. 2 and 21.)
- [117] Roland Kluge, Michael Stein, David Giessing, Andy Schürr, and Max Mühlhäuser. cMoflon: Model-Driven Generation of Embedded C Code for Wireless Sensor Networks. In *European Conference on Modelling Foundations and Applications (ECMFA)*, pages 109–125, Cham, 2017. Springer International. URL: https://doi.org/10.1007/978-3-319-61482-3_7. (Cited on pp. 15 and 224.)
- [118] Roland Kluge, Michael Stein, Gergely Varró, Andy Schürr, Matthias Hollick, and Max Mühlhäuser. A Systematic Approach to Constructing Incremental Topology Control Algorithms using Graph Transformation. *Visual Languages & Computing (JVLC)*, 38:47–83, 2016. URL: <https://doi.org/10.1016/j.jvlc.2016.10.003>. (Cited on pp. 14, 94, and 150.)
- [119] Roland Kluge, Michael Stein, Gergely Varró, Andy Schürr, Matthias Hollick, and Max Mühlhäuser. A Systematic Approach to Constructing Families of Incremental Topology Control Algorithms Using Graph Transformation. *Software & Systems Modeling (SoSyM)*, 18(1):279–319, 2017. URL: <https://doi.org/10.1007/s10270-017-0587-8>. (Cited on pp. 14, 15, 31, 94, 150, 178, 239, 240, 241, 244, 258, 269, and 273.)
- [120] Roland Kluge, Michael Stein, Gergely Varró, Andy Schürr, Matthias Hollick, and Max Mühlhäuser. A Systematic Approach to Constructing Families of Incremental Topology Control Algorithms Using Graph Transformation. In *Software Engineering and Software Management (SE)*, LNI, pages 109–110, Bonn, 2018. Gesellschaft für Informatik. URL: <https://dl.gi.de/20.500.12116/16340>. (Cited on p. 14.)

- [121] Roland Kluge, Michael Stein, Gergely Varró, Andy Schürr, Matthias Hollick, and Max Mühlhäuser. A systematic approach to constructing families of incremental topology control algorithms using graph transformation. In *International Conference on Graph Transformation (ICGT)*, LNCS, Berlin Heidelberg, 2018. Springer. URL: <https://www.hpi.uni-potsdam.de/giese/events/icgt2018/slides/Kluge+18.pdf>. (Cited on pp. 14 and 205.)
- [122] Roland Kluge, Gergely Varró, and Andy Schürr. A Methodology for Designing Dynamic Topology Control Algorithms via Graph Transformation. In *International Conference on Model Transformation (ICMT)*, volume 9152 of LNCS, pages 199–213. Springer International Publishing, 2015. URL: https://doi.org/10.1007/978-3-319-21155-8_15. (Cited on pp. 14, 94, and 150.)
- [123] Jens Kosiol, Lars Fritsche, Nebras Nassar, Andy Schürr, and Gabriele Taentzer. Towards Establishing Consistency between Graph Transformation Rules and Atomic Graph Constraints Using Multi-Amalgamation. In *Workshop on Algebraic Development Techniques (WADT)*, London, UK, 2018. University of London. URL: <http://wadt18.cs.rhul.ac.uk/submissions/WADT18A52.pdf>. (Cited on p. 165.)
- [124] Christian Krause and Holger Giese. Probabilistic Graph Transformation Systems. In *International Conference on Graph Transformation (ICGT)*, pages 311–325, Berlin Heidelberg, 2012. Springer. URL: https://doi.org/10.1007/978-3-642-33654-6_21. (Cited on pp. 84, 87, 88, 167, and 279.)
- [125] Christian Krause, Matthias Tichy, and Holger Giese. Implementing Graph Transformations in the Bulk Synchronous Parallel Model. In *Fundamental Approaches to Software Engineering*, pages 325–339, Berlin Heidelberg, 2014. Springer. URL: https://doi.org/10.1007/978-3-642-54804-8_23. (Cited on p. 173.)
- [126] Lakshman Krishnamurthy, Robert Adler, Phil Buonadonna, Jasmeet Chhabra, Mick Flanigan, Nandakishore Kushalnagar, Lama Nachman, and Mark Yarvis. Design and Deployment of Industrial Sensor Networks: Experiences from a Semiconductor Plant and the North Sea. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 64–75, New York, NY, USA, 2005. ACM. URL: <https://doi.org/10.1145/1098918.1098926>. (Cited on pp. 1 and 21.)
- [127] Joseph B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings Amer. Math. Soc.*, pages 48–50, 1956. URL: <https://doi.org/10.1090/S0002-9939-1956-0078686-7>. (Cited on pp. 84 and 161.)
- [128] Fabian Kuhn, Roger Wattenhofer, and Aaron Zollinger. Ad hoc networks beyond Unit Disk Graphs. *Wireless Networks*, 14(5):715–729, 2008. URL: <https://doi.org/10.1007/s11276-007-0045-6>. (Cited on p. 22.)

- [129] Thomas Kühne. What is a Model? In *Language Engineering for Model-Driven Software Development*, number 04101 in Dagstuhl Seminar Proceedings, Dagstuhl, Germany, 2005. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany. URL: <http://drops.dagstuhl.de/opus/volltexte/2005/23>. (Cited on pp. 11 and 48.)
- [130] Géza Kulcsár, Michael Stein, Immanuel Schweizer, Gergely Varró, Max Mühlhäuser, and Andy Schürr. Rapid Prototyping of Topology Control Algorithms by Graph Transformation. In *International Workshop on Graph-Based Tools (GraBaTs)*, volume 68, pages 1–15, ECEASST, 2014. ECEASST. URL: <https://doi.org/10.14279/tuj.eceasst.68.957>. (Cited on pp. 31, 191, 206, 235, 239, 241, and 244.)
- [131] Marta Kwiatkowska, Gethin Norman, and David Parker. PRISM 4.0: Verification of Probabilistic Real-Time Systems. In *Computer Aided Verification (CAV)*, pages 585–591, Berlin Heidelberg, 2011. Springer. URL: https://doi.org/10.1007/978-3-642-22110-1_47. (Cited on p. 87.)
- [132] Marta Kwiatkowska, Gethin Norman, Roberto Segala, and Jeremy Sproston. Automatic Verification of Real-Time Systems with Discrete Probability Distributions. *Theoretical Computer Science*, 282(1):101–150, 2002. URL: [https://doi.org/10.1016/S0304-3975\(01\)00046-9](https://doi.org/10.1016/S0304-3975(01)00046-9). Real-Time and Probabilistic Systems. (Cited on p. 87.)
- [133] Richard Lai. A survey of Communication Protocol Testing. *Systems and Software (JSS)*, 2002. URL: [https://doi.org/10.1016/S0164-1212\(01\)00132-7](https://doi.org/10.1016/S0164-1212(01)00132-7). (Cited on p. 167.)
- [134] Heiner Lasi, Peter Fettke, Hans-Georg Kemper, Thomas Feld, and Michael Hoffmann. Industry 4.0. *Business & Information Systems Engineering*, 6(4):239–242, 2014. URL: <https://doi.org/10.1007/s12599-014-0334-4>. (Cited on p. 1.)
- [135] Erhan Leblebici, Anthony Anjorin, and Andy Schürr. Developing eMoflon with eMoflon. In *International Conference on Model Transformation (ICMT)*, volume 8568 of LNCS, pages 138–145, Cham, 2014. Springer International. URL: https://doi.org/10.1007/978-3-319-08789-4_10. (Cited on pp. 12, 77, 169, 171, 172, 173, 198, 221, and 277.)
- [136] Ben Leong, Sayan Mitra, and Barbara Liskov. Path Vector Face Routing: Geographic Routing with Local Face Information. In *IEEE International Conference on Network Protocols (ICNP)*, pages 1–12, NY, USA, 2005. IEEE. URL: <https://doi.org/10.1109/ICNP.2005.32>. (Cited on p. 28.)

- [137] Robert V. Levine and Ara Norenzayan. The Pace of Life in 31 Countries. *Journal of Cross-Cultural Psychology*, 30(2):178–205, 1999. URL: <https://doi.org/10.1177/0022022199030002003>. (Cited on p. 192.)
- [138] By Mo Li and Zhenjiang Li and Athanasios V. Vasilakos. A Survey on Topology Control in Wireless Sensor Networks: Taxonomy, Comparative Study, and Open Issues. *Proceedings of the IEEE*, 101(12):2538–2557, 2013. URL: <https://doi.org/10.1109/JProceedings2013.2257631>. (Cited on pp. 21 and 28.)
- [139] Li Li, Joseph Y. Halpern, Paramvir Bahl, Yi-Min Wang, and Roger Wattenhofer. Analysis of a Cone-Based Distributed Topology Control Algorithm for Wireless Multi-Hop Networks. In *ACM Symposium on Principles of Distributed Computing (PODC)*, pages 264–273, New York, NY, USA, 2001. ACM. URL: <http://doi.acm.org/10.1145/383962.384043>. (Cited on p. 247.)
- [140] Ning Li, Jennifer C. Hou, and Lui Sha. Design and Analysis of an MST-Based Topology Control Algorithm. *IEEE Transactions on Wireless Communications*, 4(3):1195–1206, 2005. URL: <https://doi.org/10.1109/TWC.2005.846971>. (Cited on pp. 84, 86, 160, 167, 206, 208, 211, 247, and 277.)
- [141] Xiang-Yang Li. Approximate MST for UDG Locally. In *Computing and Combinatorics*, pages 364–373, Berlin Heidelberg, 2003. Springer. URL: https://doi.org/10.1007/3-540-45071-8_37. (Cited on p. 243.)
- [142] Xiang-Yang Li, Gruia Calinescu, and Peng-Jun Wan. Distributed Construction of a Planar Spanner and Routing for Ad Hoc Wireless Networks. In *IEEE International Conference on Computer Communications (INFOCOM)*, volume 3, pages 1268–1277, NY, USA, 2002. IEEE. URL: <https://doi.org/10.1109/INFCOM.2002.1019377>. (Cited on pp. 191 and 244.)
- [143] Xiang-Yang Li, Wen-Zhan Song, and Weizhao Wang. A Unified Energy-Efficient Topology for Unicast and Broadcast. In *International Conference on Mobile Computing and Networking (MobiCom)*, pages 1–15, New York, NY, USA, 2005. ACM. URL: <http://doi.org/10.1145/1080829.1080831>. (Cited on pp. 28, 191, and 246.)
- [144] Xiang-Yang Li, Ivan Stojmenovic, and Yu Wang. Partial Delaunay Triangulation and Degree Limited Localized Bluetooth Scatternet Formation. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 15(4):350–361, 2004. URL: <https://doi.org/10.1109/TPDS.2004.1271184>. (Cited on p. 246.)
- [145] Xiang-Yang Li, Peng-Jun Wan, and Yu Wang. Power Efficient and Sparse Spanner for Wireless Ad Hoc Networks. In *International Conference on Computer Communication and Networks (ICCCN)*, pages 564–567, NY, USA, 2001. IEEE. URL: <https://doi.org/10.1109/ICCCN.2001.956322>. (Cited on p. 246.)

- [146] Xiang-Yang Li, Peng-Jun Wan, Yu Wang, and Ophir Frieder. Sparse Power Efficient Topology for Wireless Networks. In *Annual Hawaii International Conference on System Sciences (HICSS)*, pages 3839–3848, NY, USA, 2002. IEEE. URL: <https://doi.org/10.1109/HICSS.2002.994518>. (Cited on pp. 245 and 246.)
- [147] Xiang-Yang Li, Yu Weng, Peng-Jun Wan, Wen-Zhan Song, and Ophir Frieder. Localized Low-Weight Graph and its Applications in Wireless Ad Hoc Networks. In *IEEE International Conference on Computer Communications (INFOCOM)*, volume 1, pages 431–442, 2004. URL: <https://doi.org/10.1109/INFCOM.2004.1354515>. (Cited on p. 247.)
- [148] Yanjun Li, Zhi Wang, and Yeqiong Song. Wireless Sensor Network Design for Wildfire Monitoring. In *World Congress on Intelligent Control and Automation (WCICA)*, volume 1, pages 109–113. IEEE, 2006. URL: <https://doi.org/10.1109/WCICA.2006.1712372>. (Cited on pp. 2 and 21.)
- [149] Roman Lim, Federico Ferrari, Marco Zimmerling, Christoph Walser, Philipp Sommer, and Jan Beutel. FlockLab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, pages 153–165, New York, NY, USA, 2013. ACM. URL: <https://doi.org/10.1145/2461381.2461402>. (Cited on pp. 9, 170, 171, 225, and 252.)
- [150] Igor Litovsky, Yves Métivier, and Wiesław Zielonka. The Power and the Limitations of Local Computations on Graphs. In *Graph-Theoretic Concepts in Computer Science*, pages 333–345, Berlin Heidelberg, 1993. Springer. URL: https://doi.org/10.1007/3-540-56402-0_58. (Cited on pp. 157 and 272.)
- [151] Jilei Liu and Baochun Li. MobileGrid: Capacity-Aware Topology Control in Mobile Ad Hoc Networks. In *International Conference on Computer Communication and Networks (ICCCN)*, pages 570–574, NY, USA, 2002. IEEE. URL: <https://doi.org/10.1109/ICCCN.2002.1043127>. (Cited on p. 245.)
- [152] Peter Csaba Ölveczky and Stian Thorvaldsen. Formal Modeling, Performance Estimation, and Model Checking of Wireless Sensor Network Algorithms in Real-Time Maude. *Theoretical Computer Science (TCS)*, 410(2):254–280, 2009. URL: <https://doi.org/10.1016/j.tcs.2008.09.022>. (Cited on p. 236.)
- [153] Renita Machado and Sirin Tekinay. A Survey of Game-Theoretic Approaches in Wireless Sensor Networks. *Computer Networks (GACETA)*, 52(16):3047–3061, 2008. URL: <https://doi.org/10.1016/j.gaceta.2008.07.003>. (Cited on p. 89.)
- [154] Alan Mainwaring, David Culler, Joseph Polastre, Robert Szewczyk, and John Anderson. Wireless Sensor Networks for Habitat Monitoring. In *International Workshop on Wireless Sensor Networks and Applications (WSNA)*, pages 88–97, New York,

- NY, USA, 2002. ACM. URL: <https://doi.org/10.1145/570738.570751>. (Cited on p. 2.)
- [155] Tubaishat Malik, Zhuang Peng, Qi Qi, and Shang Yi. Wireless sensor networks in intelligent transportation systems. *Wireless Communications and Mobile Computing (WCM)*, 9(3):287–302, 2009. URL: <https://doi.org/10.1002/wcm.616>. (Cited on p. 1.)
- [156] Francisco Martins, Luís Lopes, and João Barros. Towards the Safe Programming of Wireless Sensor Networks. In *International Workshop on Programming Language Approaches to Concurrency and Communication-centric Software*, volume 17 of *Electronic Proceedings in Theoretical Computer Science (EPTCS)*, pages 49–62. Open Publishing Association, 2010. URL: <https://doi.org/10.4204/EPTCS.17.5>. (Cited on p. 9.)
- [157] Maria Maximova, Holger Giese, and Christian Krause. Probabilistic Timed Graph Transformation Systems. In *International Conference on Graph Transformation (ICGT)*, pages 159–175, Cham, 2017. Springer International. URL: https://doi.org/10.1007/978-3-319-61470-0_10. (Cited on pp. 84, 87, 167, and 235.)
- [158] Tanja Mayerhofer, Philip Langer, and Gerti Kappel. A Runtime Model for fUML. In *Workshop on Models@Run.Time (MRT)*, pages 53–58, New York, NY, USA, 2012. ACM. URL: <https://doi.org/10.1145/2422518.2422527>. (Cited on p. 90.)
- [159] Tom Mens, Serge Demeyer, and Dirk Janssens. Formalising Behaviour Preserving Program Transformations. In *International Conference on Graph Transformation (ICGT)*, pages 286–301, Berlin Heidelberg, 2002. Springer. URL: https://doi.org/10.1007/3-540-45832-8_22. (Cited on p. 162.)
- [160] Dario Mirizzi. Design and Implementation of a Demonstrator for Topology Adaptation Algorithms. Bachelor’s thesis, TU Darmstadt, 2015. (Cited on pp. 15 and 178.)
- [161] Dov Monderer and Lloyd S. Shapley. Potential Games. *Games and Economic Behavior (Game)*, 14(1):124–143, 1996. URL: <https://doi.org/10.1006/game.1996.0044>. (Cited on pp. 9 and 89.)
- [162] Shunsuke Mori, Takaaki Umedu, Akihito Hiromori, Hirozumi Yamaguchi, and Teruo Higashino. Data-centric programming environment for cooperative applications in WSN. In *IFIP/IEEE International Symposium on Integrated Network Management (IM 2013)*, pages 856–859, 2013. URL: <https://ieeexplore.ieee.org/abstract/document/6573096/>. (Cited on p. 9.)
- [163] Glenford J. Myers, Corey Sandler, and Tom Badgett. *The Art of Software Testing*. John Wiley & Sons, 3 edition, 2011. ISBN: 978-1-118-03196-4. URL: <https://www.wiley.com/en-us/The+Art+of+Software+Testing%2C+3rd+Edition-p-9781118031964>. (Cited on p. 167.)

- [164] Nebras Nassar, Jens Kosiol, Thorsten Arendt, and Gabriele Taentzer. OCL2AC: Automatic Translation of OCL Constraints to Graph Constraints and Application Conditions for Transformation Rules. In *International Conference on Graph Transformation (ICGT)*, pages 171–177, Cham, 2018. Springer International. URL: https://doi.org/10.1007/978-3-319-92991-0_11. (Cited on pp. 161, 172, 270, and 279.)
- [165] Nebras Nassar, Jens Kosiol, and Hendrik Radke. Rule-Based Repair of EMF Models: Formalization and Correctness Proof. In *International Workshop on Graph Computation Models (GCM)*, pages 1–16, Pisa, Italy, 2017. Università di Pisa. URL: <http://pages.di.unipi.it/corradini/Workshops/GCM2017/papers/Nassar-Kosiol-Radke-GCM2017.pdf>. (Cited on p. 166.)
- [166] Nebras Nassar, Hendrik Radke, and Thorsten Arendt. Rule-Based Repair of EMF Models: An Automated Interactive Approach. In *International Conference on Graph Transformation (ICGT)*, pages 171–181, Cham, 2017. Springer International. URL: https://doi.org/10.1007/978-3-319-61473-1_12. (Cited on p. 166.)
- [167] Marisa Navarro Gomez, Fernando Orejas Valdés, Elvira Pino Blanco, and Leen Lambers. A Logic of Graph Conditions Extended with Paths. In *International Workshop on Graph Computation Models (GCM)*, pages 1–15, 2016. URL: <http://gcm2016.inf.uni-due.de/papers/navarro-orejas-pino-lambers.pdf>. (Cited on pp. 85 and 166.)
- [168] Lukas Neumann. Integration of the Graph Pattern Matcher Democles into a Network Simulator. Bachelor’s thesis, TU Darmstadt, 2016. (Cited on pp. 15 and 178.)
- [169] National Highway Traffic Safety Association (NHTSA). Vehicle-to-Vehicle Communication. <https://www.nhtsa.gov/technology-innovation/vehicle-vehicle-communication>, 2018. Last accessed: 2018-05-11. (Cited on p. 1.)
- [170] Amy Nordrum. Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated. <https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated>, 2016. Last accessed on 2018-02-16. (Cited on p. 1.)
- [171] Peter Csaba Ölveczky. Real-Time Maude and its Applications. In *Rewriting Logic and its Applications (WRLA)*, pages 42–79, Cham, 2014. Springer International. URL: https://doi.org/10.1007/978-3-319-12904-4_3. (Cited on p. 236.)
- [172] Óscar Ortiz, Ana Belén García, Rafael Capilla, Jan Bosch, and Mike Hinchey. Runtime Variability for Dynamic Reconfiguration in Wireless Sensor Network Product Lines. In *International Conference on Software Product Lines (SPLC)*, pages 143–150, New York, NY, USA, 2012. ACM. URL: <https://doi.org/10.1145/2364412.2364436>. (Cited on pp. 273 and 274.)

- [173] Jianping Pan, Thomas Hou, Lin Cai, Yi Shi, and Sherman X. Shen. Topology Control for Wireless Sensor Networks. In *International Conference on Mobile Computing and Networking (MobiCom)*, pages 286–299, New York, NY, USA, 2003. ACM. URL: <http://doi.org/10.1145/938985.939015>. (Cited on p. 29.)
- [174] Francesco Parisi Presicce Paolo Bottoni, Andrew Fish. Incremental Update of Constraint-Compliant Policy Rules. In *International Workshop on Graph Computation Models (GCM)*, Dortmund, 2010. ECEASST. URL: <http://dx.doi.org/10.14279/tuj.eceasst.39.651>. (Cited on p. 163.)
- [175] Karl-Heinz Pennemann. *Development of Correct Graph Transformation Systems*. PhD thesis, Universität Oldenburg, 2009. URL: <http://oops.uni-oldenburg.de/884/>. (Cited on pp. 160 and 279.)
- [176] Martin Pfannemüller, Janick Edinger, Markus Weckesser, Roland Kluge, Manisha Luthra, Robin Klose, Christian Becker, and Andy Schürr. Demo: Visualizing Adaptation Decisions in Pervasive Communication Systems. In *Demonstration on Pervasive Computing and Communications (PerCom Demo)*, pages 1–3, NY, USA, 2019. IEEE. (Cited on pp. 14 and 184.)
- [177] Martin Pfannemüller, Markus Weckesser, Roland Kluge, Janick Edinger, Manisha Luthra, Robin Klose, Christian Becker, and Andy Schürr. CoalaViz: Supporting Traceability of Adaptation Decisions in Pervasive Communication Systems. In *International Workshop on Mobile Ubiquitous Systems, Infrastructures, Communications and Applications (MUSICAL)*, pages 1–6, NY, USA, 2019. IEEE. (Cited on pp. 14 and 184.)
- [178] Dennis E. Phillips, Mohammad-Mahdi Moazzami, Guoling Xing, and Jonathan M. Lees. A Sensor Network for Real-Time Volcano Tomography: System Design and Deployment. In *International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, 2017. URL: <https://doi.org/10.1109/ICCCN.2017.8038445>. (Cited on pp. 2 and 21.)
- [179] Detlef Plump. The Graph Programming Language GP. In *Algebraic Informatics*, pages 99–122, Berlin Heidelberg, 2009. Springer. URL: https://doi.org/10.1007/978-3-642-03564-7_6. (Cited on p. 161.)
- [180] Klaus Pohl, Günter Böckle, and Frank van der Linden. *Software Product Line Engineering*. Springer, 1st edition, 2005. ISBN: 978-3-540-24372-4 . URL: <https://doi.org/10.1007/3-540-28901-1>. (Cited on p. 81.)
- [181] Joseph Polastre, Robert Szewczyk, and David Culler. Telos: Enabling Ultra-Low Power Wireless Research. In *ACM/IEEE Conference on Information Processing in Sensor Networks (IPSN)*, pages 364–369, NY, USA, 2005. IEEE. URL: <https://doi.org/10.1109/IPSN.2005.1440950>. (Cited on pp. 3, 9, 170, 171, 222, and 225.)

- [182] Jesús M. T. Portocarrero, Flavia C. Delicato, Paulo F. Pires, Bruno Costa, Wei Li, Weisheng Si, and Albert Y. Zomaya. RAMSES: A New Reference Architecture for Self-Adaptive Middleware in Wireless Sensor Networks. *Ad Hoc Networks (AdHoc)*, 55:3 – 27, 2017. URL: <https://doi.org/10.1016/j.adhoc.2016.11.004>. (Cited on p. 40.)
- [183] Jesús M. T. Portocarrero, Flavia C. Delicato, Paulo F. Pires, and Thais V. Batista. Reference Architecture for Self-Adaptive Management in Wireless Sensor Networks. In *International Conference on Adaptive and Intelligent Systems (ICAIS)*, pages 110–120, Cham, 2014. Springer International. URL: https://doi.org/10.1007/978-3-319-11298-5_12. (Cited on pp. 40, 81, 273, and 274.)
- [184] Jesús M. T. Portocarrero, Flávia C. Delicato, Paulo F. Pires, Taniro C. Rodrigues, and Thais V. Batista. SAMSON: Self-Adaptive Middleware for Wireless Sensor Networks. In *ACM Symposium on Applied Computing (SAC)*, pages 1315–1322, New York, NY, USA, 2016. ACM. URL: <https://doi.org/10.1145/2851613.2851766>. (Cited on pp. 40 and 236.)
- [185] Christopher M. Poskitt. *Verification of Graph Programs*. PhD thesis, University of York, 2013. URL: <http://etheses.whiterose.ac.uk/4700/>. (Cited on pp. 161, 172, and 173.)
- [186] Christopher M. Poskitt and Detlef Plump. A Hoare Calculus for Graph Programs. In *International Conference on Graph Transformation (ICGT)*, pages 139–154, Berlin Heidelberg, 2010. Springer. URL: https://doi.org/10.1007/978-3-642-15928-2_10. (Cited on p. 161.)
- [187] Christopher M. Poskitt and Detlef Plump. Hoare-Style Verification of Graph Programs. *Fundamenta Informaticae*, 118(1-2):135–175, 2012. URL: <https://doi.org/10.3233/FI-2012-708>. (Cited on pp. 161 and 163.)
- [188] Christopher M. Poskitt and Detlef Plump. Verifying Monadic Second-Order Properties of Graph Programs. In *International Conference on Graph Transformation (ICGT)*, pages 33–48, Cham, 2014. Springer International. URL: https://doi.org/10.1007/978-3-319-09108-2_3. (Cited on p. 161.)
- [189] John Postel. UDP: User Datagram Protocol. *IETF RFC 768*, 1980. URL: <https://tools.ietf.org/html/rfc768>. (Cited on p. 192.)
- [190] Dumitru Potop-Butucaru and Benoît Caillaud. Correct-by-Construction Asynchronous Implementation of Modular Synchronous Specifications. *Proceedings of the International Conference on Application of Concurrency to System Design (ACSD)*, pages 48–57, 2005. URL: <https://doi.org/10.1109/ACSD.2005.10>. (Cited on p. 163.)

- [191] Robert C. Prim. Shortest Connection Networks And Some Generalizations. *Bell System Technical Journal*, 36(6):1389–1401, 1957. URL: <https://doi.org/10.1002/j.1538-7305.1957.tb01515.x>. (Cited on p. 214.)
- [192] Bose Prosenjit, Joachim Gudmundsson, and Pat Morin. Ordered Theta Graphs. *Computational Geometry (ComGeo)*, 28(1):11 – 18, 2004. URL: <https://doi.org/10.1016/j.comgeo.2004.01.003>. Canadian Conference on Computational Geometry (CCCG). (Cited on pp. 246 and 253.)
- [193] Junaid Qadir and Osman Hasan. Applying Formal Methods to Networking: Theory, Techniques, and Applications. *IEEE Communications Surveys Tutorials (COMST)*, 17(1):256–291, 2015. URL: <https://doi.org/10.1109/COMST.2014.2345792>. (Cited on p. 9.)
- [194] Diego V. Queiroz, Marcelo S. Alencar, Ruan D. Gomes, Iguatemi E. Fonseca, and Cesar Benavente-Peces. Survey and systematic mapping of industrial wireless sensor networks. *Journal of Network and Computer Applications (JNCA)*, 97:96–125, 2017. URL: <https://doi.org/10.1016/j.jnca.2017.08.019>. (Cited on p. 234.)
- [195] Clément Quinton, Daniel Romero, and Laurence Duchien. Cardinality-based Feature Models with Constraints: A Pragmatic Approach. In *International Conference on Software Product Lines (SPLC)*, pages 162–166, New York, NY, USA, 2013. ACM. URL: <https://doi.org/10.1145/2491627.2491638>. (Cited on p. 82.)
- [196] Hendrik Radke. Weakest Liberal Preconditions Relative to HR^* Graph Conditions. In *International Workshop on Graph Computation Models (GCM)*, pages 165–178, 2010. URL: <http://formale-sprachen.informatik.uni-oldenburg.de/~skript/fs-pub/Radk10b.pdf>. (Cited on pp. 86 and 160.)
- [197] Hendrik Radke. *A Theory of HR^* Graph Conditions and their Application to Meta-Modeling*. PhD thesis, Carl von Ossietzky-Universität Oldenburg, Germany, 2016. URL: <http://oops.uni-oldenburg.de/2803>. (Cited on pp. 160 and 161.)
- [198] Hendrik Radke, Thorsten Arendt, Jan Steffen Becker, Annegret Habel, and Gabriele Taentzer. Translating Essential OCL Invariants to Nested Graph Constraints Focusing on Set Operations. In *International Conference on Graph Transformation (ICGT)*, volume 9151 of LNCS, pages 155–170. Springer International, Cham, 2015. URL: https://doi.org/10.1007/978-3-319-21145-9_10. (Cited on p. 161.)
- [199] Hendrik Radke, Thorsten Arendt, Jan Steffen Becker, Annegret Habel, and Gabriele Taentzer. Translating Essential OCL Invariants to Nested Graph Constraints For Generating Instances of Meta-Models. *Science of Computer Programming (SciCo)*, 152:38–62, 2018. URL: <https://doi.org/10.1016/j.scico.2017.08.006>. (Cited on pp. 161 and 278.)

- [200] Ashikur Rahman and Pawel Gburzynski. Hidden Problems with the Hidden Node Problem. In *Biennial Symposium on Communications (BSC)*, pages 270–273, NY, USA, 2006. IEEE. URL: <https://10.1109/BSC.2006.1644620>. (Cited on p. 3.)
- [201] Ram Ramanathan and Regina Rosales-Hain. Topology Control of Multihop Wireless Networks Using Transmit Power Adjustment. In *IEEE International Conference on Computer Communications (INFOCOM)*, volume 2, pages 404–413, 2000. URL: <https://doi.org/10.1109/INFCOM.2000.832213>. (Cited on p. 245.)
- [202] Maneesha V. Ramesh. Real-Time Wireless Sensor Network for Landslide Detection. In *International Conference on Sensor Technologies and Applications (SENSORCOMM)*, pages 405–/409, 2009. URL: <https://doi.org/10.1109/SENSORCOMM.2009.67>. (Cited on pp. 2 and 21.)
- [203] Tifenn Rault, Abdelmadjid Bouabdallah, and Yacine Challal. Energy Efficiency in Wireless Sensor Networks: A Top-Down Survey. *Computer Networks (Com-Net)*, 67:104–122, 2014. URL: <https://doi.org/10.1016/j.comnet.2014.03.027>. (Cited on p. 1.)
- [204] Arend Rensink. The GROOVE Simulator: A Tool for State Space Generation. In *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 3062 of LNCS, pages 479–485, Berlin Heidelberg, 2004. Springer. URL: https://doi.org/10.1007/978-3-540-25959-6_40. (Cited on pp. 88 and 172.)
- [205] Arend Rensink, Ákos Schmidt, and Dániel Varró. Model Checking Graph Transformations: A Comparison of Two Approaches. In *International Conference on Graph Transformation (ICGT)*, volume 3256 of LNCS, pages 226–241. Springer, Berlin Heidelberg, 2004. URL: https://doi.org/10.1007/978-3-540-30203-2_17. (Cited on p. 166.)
- [206] Stefan Rührup, Christian Schindelbauer, Klaus Volbert, and Matthias Grünewald. Performance of distributed algorithms for topology control in wireless networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, page 8, NY, USA, 2003. IEEE. URL: <https://10.1109/IPDPS.2003.1213107>. (Cited on p. 246.)
- [207] Björn Richerzhagen. *Mechanism Transitions in Publish/Subscribe Systems – Adaptive Event Brokering for Location-Based Mobile Social Applications*. PhD thesis, Technische Universität, Darmstadt, 2017. URL: <https://tuprints.ulb.tu-darmstadt.de/6669/>. (Cited on p. 2.)
- [208] Björn Richerzhagen, Dominik Stingl, Julius Rückert, and Ralf Steinmetz. Simonstrator: Simulation and Prototyping Platform for Distributed Mobile Applications. In *International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools)*, pages 99–108, ICST, Brussels, Belgium,

2015. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST). URL: <https://doi.org/10.4108/eai.24-8-2015.2261064>. (Cited on pp. 8, 13, 15, 23, 169, 171, 173, 179, and 277.)
- [209] Nils Richerzhagen, Roland Kluge, Björn Richerzhagen, Patrick Lieser, Boris Koldhofe, Ioannis Stavrakakis, and Ralf Steinmetz. Better Together: Collaborative Monitoring for Location-based Services. In *International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, pages 14–22, NY, USA, 2018. IEEE. URL: <https://doi.org/10.1109/WoWMoM.2018.8449798>. (Cited on pp. 14 and 252.)
- [210] Nils Richerzhagen, Björn Richerzhagen, Michael Lipinski, Markus Weckesser, Roland Kluge, Rhaban Hark, and Ralf Steinmetz. Exploring Transitions in Mobile Network Monitoring in Highly Dynamic Environments. In *Demonstrations of the IEEE Conference on Local Computer Networks (LCN Demonstrations)*, pages 1–3, 2016. URL: <https://www.ieeelcn.org/prior/LCN41/lcn41demos/RicherzhagenN.pdf>. (Cited on p. 14.)
- [211] Volkan Rodoplu and Teresa H. Meng. Minimum Energy Mobile Wireless Networks. *IEEE Journal on Selected Areas in Communications (JSAC)*, 17(8):1333–1344, 1999. URL: <https://doi.org/10.1109/49.779917>. (Cited on pp. 31, 191, 205, 244, and 250.)
- [212] Taniro Rodrigues, Priscilla Dantas, Flávia C. Delicato, Paulo F. Pires, Luci Pirmez, Thais Batista, Claudio Miceli, and Albert Zomaya. Model-Driven Development of Wireless Sensor Network Applications. In *International Conference on Embedded and Ubiquitous Computing (EUC)*, pages 11–18, NY, USA, 2011. IEEE. URL: <https://doi.org/10.1109/EUC.2011.50>. (Cited on p. 90.)
- [213] Taniro Rodrigues, Flávia Delicato, Thais Batista, Paulo Pires, and Luci Pirmez. An Approach Based on the Domain Perspective to Develop WSN Applications. *Software & Systems Modeling (SoSyM)*, pages 1–29, 2015. URL: <https://doi.org/10.1007/s10270-015-0498-5>. (Cited on p. 90.)
- [214] Grzegorz Rozenberg, editor. *Handbook of Graph Grammars and Computing by Graph Transformation*, volume 1: Foundations. World Scientific, Singapore, 1997. URL: <https://doi.org/10.1142/3303>. (Cited on pp. 57, 70, and 276.)
- [215] Alessandra Russo and Andy Schürr, editors. *Fundamental Approaches to Software Engineering (FASE)*. Springer International, Cham, 2018. URL: <https://doi.org/10.1007/978-3-319-89363-1>. (Cited on p. 50.)
- [216] Karsten Saller, Malte Lochau, and Ingo Reimund. Context-Aware DSPLs: Model-Based Runtime Adaptation for Resource-Constrained Systems. In *International*

- Software Product Line Conference Co-located Workshops (SPLC 2013 Workshops)*, pages 106–113, New York, NY, USA, 2013. ACM. URL: <https://doi.org/10.1145/2499777.2500716>. (Cited on p. 274.)
- [217] Karsten Saller, Sebastian Oster, Andy Schürr, Julia Schroeter, and Malte Lochau. Reducing Feature Models to Improve Runtime Adaptivity on Resource Limited Devices. In *International Conference on Software Product Lines (SPLC)*, pages 135–142, New York, NY, USA, 2012. ACM. URL: <https://doi.org/10.1145/2364412.2364435>. (Cited on p. 274.)
- [218] Paolo Santi. *Topology Control in Wireless Ad Hoc and Sensor Networks*. Wiley, Chichester, England, 1 edition, 2005. ISBN: 978-0-470-09453-2 . (Cited on pp. 2 and 3.)
- [219] Paolo Santi. Topology Control in Wireless Ad Hoc and Sensor Networks. *ACM Computing Surveys (CSUR)*, 37(2):164–194, 2005. URL: <https://doi.org/10.1145/1089733.1089736>. (Cited on pp. 2, 3, 6, 17, 21, 22, 23, 26, 27, 30, 84, 201, 205, 239, 241, 242, 243, 244, 245, 246, 247, and 269.)
- [220] Jochen Schiller, Achim Liers, Hartmut Ritter, Rolf Winter, and Thiemo Voigt. ScatterWeb – Low Power Sensor Nodes and Energy Aware Routing. In *Hawaiian International Conference on System Sciences (HICSS)*, pages 286–294, NY, USA, 2005. IEEE. URL: <https://doi.org/10.1109/HICSS.2005.529>. (Cited on p. 90.)
- [221] Thomas Schnabel, Markus Weckesser, Roland Kluge, Malte Lochau, and Andy Schürr. CardyGAN: Tool Support for Cardinality-based Feature Models. In *International Workshop on Variability Modelling of Software-Intensive Systems (VaMoS)*, pages 33–40, New York, NY, USA, 2016. ACM. URL: <https://doi.org/10.1145/2866614.2866619>. (Cited on pp. 14 and 82.)
- [222] Julia Schroeter, Peter Mucha, Marcel Muth, Kay Jugel, and Malte Lochau. Dynamic Configuration Management of Cloud-based Applications. In *International Conference on Software Product Lines (SPLC)*, pages 171–178, New York, NY, USA, 2012. ACM. URL: <https://doi.org/10.1145/2364412.2364441>. (Cited on p. 81.)
- [223] Andy Schürr. Specification of Graph Translators with Triple Graph Grammars. In *Graph-Theoretic Concepts in Computer Science*, pages 151–163, Berlin Heidelberg, 1995. Springer. (Cited on p. 165.)
- [224] Andy Schürr and Felix Klar. 15 Years of Triple Graph Grammars. In *International Conference on Graph Transformation (ICGT)*, pages 411–425, Berlin Heidelberg, 2008. Springer. URL: https://doi.org/10.1007/978-3-540-87405-8_28. (Cited on pp. 163 and 165.)
- [225] Andy Schürr, Andreas J. Winter, and Albert Zündorf. The PROGRES Approach: Language and Environment. In *Handbook of Graph Grammars and Computing by*

- Graph Transformation*, pages 487–550, Singapore, 1999. World Scientific. URL: https://doi.org/10.1142/9789812815149_0013. (Cited on p. 237.)
- [226] Immanuel Schweizer. *Energy-Efficient Urban Sensing*. PhD thesis, TU Darmstadt, Göttingen, 2013. URL: <http://tubiblio.ulb.tu-darmstadt.de/64551/>. (Cited on pp. 2 and 21.)
- [227] Immanuel Schweizer, Michael Wagner, Dirk Bradler, Max Mühlhäuser, and Thorsten Strufe. kTC – Robust and Adaptive Wireless Ad-Hoc Topology Control. In *International Conference on Computer Communication and Networks (ICCCN)*, pages 1–9, NY, USA, 2012. IEEE. URL: <https://doi.org/10.1109/ICCCN.2012.6289318>. (Cited on pp. 4, 11, 13, 31, 33, 82, 191, 192, 206, 208, 241, 243, 251, and 277.)
- [228] Immanuel Schweizer, Ralf Zimmermann, Michael Stein, and Max Mühlhäuser. a-kTC: Integrating Topology Control into the Stack. In *International Conference on Local Computer Networks (LCN)*, pages 414–417, NY, USA, 2015. IEEE. URL: <https://doi.org/10.1109/LCN.2015.7366341>. (Cited on pp. 21, 28, 29, 31, and 206.)
- [229] Simon Schwichtenberg, Ivan Jovanovikj, Christian Gerth, and Gregor Engels. CrossEcore: An Extendible Framework to Use Ecore and OCL Across Platforms (Poster). In *International Conference on Software Engineering (ICSE)*, pages 292–293, New York, NY, USA, 2018. ACM. URL: <https://doi.org/10.1145/3183440.3194976>. (Cited on p. 237.)
- [230] Joseph A. Shaw. Radiometry and the Friis Transmission Equation. *American Journal of Physics*, 81(1):33–37, 2013. URL: <https://doi.org/10.1119/1.4755780>. (Cited on p. 26.)
- [231] Hai-Yan Shi, Wan-Liang Wang, Ngai-Ming Kwok, and Sheng-Yong Chen. Game Theory for Wireless Sensor Networks: A Survey. *Sensors*, 12(7):9055–9097, 2012. URL: <https://doi.org/10.3390/s120709055>. (Cited on p. 89.)
- [232] Gerry Siegemund. *Self-Stabilizing Algorithms in Wireless Sensor Networks*. PhD thesis, Hamburg University of Technology, Hamburg, Germany, 2017. (Cited on pp. 4 and 38.)
- [233] Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase, and Simon Helsen. *Model-Driven Software Development: Technology, Engineering, Management*. John Wiley & Sons Ltd, The Atrium, Southern Gate, Chichester, West Sussex PO19 8SQ, England, 2006. ISBN: 978-0-470-02570-3. (Cited on pp. 11, 48, and 50.)
- [234] Thomas Stahlbuhk, Brooke Shrader, and Eytan Modiano. Topology Control for Wireless Networks with Highly-Directional Antennas. In *International Symposium on Modeling and Optimization in Mobile, Ad Hoc, and Wireless Networks (WiOpt)*, pages 1–8, NY, USA, 2016. IEEE. URL: <https://doi.org/10.1109/WIOPT.2016.7492931>. (Cited on pp. 3, 21, 26, and 27.)

- [235] Michael Stein, Mathias Fischer, Immanuel Schweizer, and Max Mühlhäuser. A Classification of Locality in Network Research. *ACM Computing Surveys (CSUR)*, 50(4):53:1–53:37, 2017. URL: <https://doi.org/10.1145/3092693>. (Cited on pp. 6 and 24.)
- [236] Michael Stein, Alexander Frömmgen, Roland Kluge, Lin Wang, Augustin Wilberg, Boris Koldehofe, and Max Mühlhäuser. Scaling Topology Pattern Matching: A Distributed Approach. In *ACM Symposium on Applied Computing (SAC)*, pages 1–10, New York, NY, USA, 2018. ACM. URL: <https://doi.org/10.1145/3167132.3167241>. (Cited on p. 14.)
- [237] Michael Stein, Roland Kluge, Dario Mirizzi, Stefan Wilk, Andy Schürr, and Max Mühlhäuser. Transitions on Multiple Layers for Scalable, Energy-Efficient and Robust Wireless Video Streaming. In *IEEE International Conference on Pervasive Computing and Communication Workshops (PerCom Workshops)*, pages 1–3, NY, USA, 2016. IEEE. URL: <https://doi.org/10.1109/PERCOMW.2016.7457079>. (Cited on pp. 14, 15, and 178.)
- [238] Michael Stein, Géza Kulcsár, Immanuel Schweizer, Gergely Varró, Andy Schürr, and Max Mühlhäuser. Topology Control with Application Constraints. In *International Conference on Local Computer Networks (LCN)*, pages 438–441, 2015. URL: <https://doi.org/10.1109/LCN.2015.7366313>. (Cited on pp. 31, 191, and 244.)
- [239] Michael Stein, Tobias Petry, Immanuel Schweizer, Martina Bachmann, and Max Mühlhäuser. Topology Control in Wireless Sensor Networks: What Blocks the Breakthrough? In *International Conference on Local Computer Networks (LCN)*, pages 389–397, NY, USA, 2016. IEEE. URL: <https://doi.org/10.1109/LCN.2016.67>. (Cited on pp. 9, 30, 31, 82, 170, 171, 192, 204, 205, 206, 207, 208, 214, 223, 225, 231, 239, 241, 244, 248, 251, and 277.)
- [240] Michael Stein, Karsten Weihe, Augustin Wilberg, Roland Kluge, Julian M. Klomp, Mathias Schnee, Lin Wang, and Max Mühlhäuser. Distributed Graph-Based Topology Adaptation Using Motif Signatures. In *Workshop on Algorithm Engineering and Experiments (ALENEX)*, pages 1–12, Philadelphia, PA, USA, 2017. SIAM. URL: <https://doi.org/10.1137/1.9781611974768.1>. (Cited on pp. 14, 15, 182, and 183.)
- [241] Dave Steinberg, Frank Budinsky, Marcelo Paternostro, and E Merks. *EMF: Eclipse Modeling Framework*. Addison Wesley Professional, Boston, 2008. ISBN: 978-0321331885. (Cited on pp. 48 and 50.)
- [242] Dominik Stingl, Christian Gross, Julius Rückert, Leonhard Nobach, Aleksandra Kovacevic, and Ralf Steinmetz. PeerfactSim.KOM: A Simulation Framework for Peer-to-Peer Systems. In *International Conference on High Performance Computing and*

- Simulation (HPCS)*, pages 577–584. IEEE, 2011. URL: <https://doi.org/10.1109/HPCSim.2011.5999877>. (Cited on pp. 178, 180, and 235.)
- [243] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications. In *Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications (SIGCOMM)*, pages 149–160, New York, NY, USA, 2001. ACM. URL: <https://doi.org/10.1145/383059.383071>. (Cited on p. 167.)
- [244] Daniel Strüber, Julia Rubin, Thorsten Arendt, Marsha Chechik, Gabriele Taentzer, and Jennifer Plöger. RuleMerger: Automatic Construction of Variability-Based Model Transformation Rules. In *Fundamental Approaches to Software Engineering (FASE)*, pages 122–140, Berlin Heidelberg, 2016. Springer. URL: https://doi.org/10.1007/978-3-662-49665-7_8. (Cited on p. 273.)
- [245] Daniel Strüber, Julia Rubin, Marsha Chechik, and Gabriele Taentzer. A Variability-Based Approach to Reusable and Efficient Model Transformations. In *Fundamental Approaches to Software Engineering (FASE)*, pages 283–298, Berlin Heidelberg, 2015. Springer. URL: https://doi.org/10.1007/978-3-662-46675-9_19. (Cited on p. 273.)
- [246] Daniel Strüber and Stefan Schulz. A Tool Environment for Managing Families of Model Transformation Rules. In *International Conference on Graph Transformation (ICGT)*, pages 89–101, Cham, 2016. Springer International. URL: https://doi.org/10.1007/978-3-319-40530-8_6. (Cited on p. 273.)
- [247] Robert Szewczyk, Alan Mainwaring, Joseph Polastre, John Anderson, and David Culler. An Analysis of a Large Scale Habitat Monitoring Application. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 214–226, New York, NY, USA, 2004. ACM. URL: <https://doi.org/10.1145/1031495.1031521>. (Cited on p. 21.)
- [248] Gabriele Taentzer. AGG: A Tool Environment for Algebraic Graph Transformation. In *Applications of Graph Transformations with Industrial Relevance (AGTIVE)*, volume 1779 of *LNCS*, pages 481–490, Berlin Heidelberg, 2000. Springer. URL: https://doi.org/10.1007/3-540-45104-8_41. (Cited on pp. 86 and 172.)
- [249] Gabriele Taentzer, Manuel Ohrndorf, Yngve Lamo, and Adrian Rutle. Change-Preserving Model Repair. In *Fundamental Approaches to Software Engineering (FASE)*, pages 283–299, Berlin Heidelberg, 2017. Springer. URL: https://doi.org/10.1007/978-3-662-54494-5_16. (Cited on p. 166.)
- [250] Gabriele Taentzer and Arend Rensink. Ensuring Structural Constraints in Graph-Based Models with Type Inheritance. In *Fundamental Approaches to Software Engineering (FASE)*, volume 3442 of *LNCS*, pages 64–79, Berlin Heidelberg, 2005.

- Springer. URL: https://doi.org/10.1007/978-3-540-31984-9_6. (Cited on p. 160.)
- [251] Hideaki Takagi and Leonard Kleinrock. Optimal Transmission Ranges for Randomly Distributed Packet Radio Terminals. *IEEE Transactions on Communications (TCOM)*, 32(3):246–257, 1984. URL: <https://doi.org/10.1109/TCOM.1984.1096061>. (Cited on p. 27.)
- [252] Kenji Tei, Ryo Shimizu, Yoshiaki Fukazawa, and Shinichi Honiden. Model-Driven-Development-Based Stepwise Software Development Process for Wireless Sensor Networks. *IEEE Transactions on Systems, Man, and Cybernetics (TSMC)*, 45(4):675–687, 2015. URL: <https://doi.org/10.1109/TSMC.2014.2360506>. (Cited on p. 90.)
- [253] George F. Riley Thomas R. Henderson, Mathieu Lacage. Network Simulations with the ns-3 Simulator. In *Demonstrations of ACM Annual Conference of the Special Interest Group on Data Communications (SIGCOMM)*, page 527, New York, NY, USA, 2008. ACM. URL: <http://conferences.sigcomm.org/sigcomm/2008/papers/p527-hendersonA.pdf>. (Cited on pp. 7, 173, and 235.)
- [254] Gilman Tolle, Joseph Polastre, Robert Szewczyk, David Culler, Neil Turner, Kevin Tu, Stephen Burgess, Todd Dawson, Phil Buonadonna, David Gay, and Wei Hong. A Macroscope in the Redwoods. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 51–63, New York, NY, USA, 2005. ACM. URL: <https://doi.org/10.1145/1098918.1098925>. (Cited on p. 21.)
- [255] Stefan Tomaszek, Erhan Leblebici, Lin Wang, and Andy Schürr. Virtual Network Embedding: Reducing the Search Space by Model Transformation Techniques. In *Theory and Practice of Model Transformation*, pages 59–75, Cham, 2018. Springer International. URL: https://doi.org/10.1007/978-3-319-93317-7_2. (Cited on p. 279.)
- [256] Paolo Torrini, Reiko Heckel, and István Ráth. Stochastic Simulation of Graph Transformation Systems. In *Fundamental Approaches to Software Engineering*, pages 154–157, Berlin Heidelberg, 2010. Springer. URL: https://doi.org/10.1007/978-3-642-12029-9_11. (Cited on pp. 88 and 235.)
- [257] Godfried T. Toussaint. The Relative Neighbourhood Graph of a Finite Planar Set. *Pattern Recognition*, 12(4):261–268, 1980. URL: [https://doi.org/10.1016/0031-3203\(80\)90066-7](https://doi.org/10.1016/0031-3203(80)90066-7). (Cited on p. 31.)
- [258] Frank van der Linden, Klaus Schmid, and Eelco Rommes. *Software Product Lines in Action*. Springer, 1 edition, 2007. ISBN: 978-3-540-71436-1 . URL: <https://doi.org/10.1007/978-3-540-71437-8>. (Cited on p. 81.)

- [259] András Varga and Rudolf Hornig. An Overview of the OMNeT++ Simulation Environment. In *International Conference on Simulation Tools and Techniques for Communications, Networks and Systems & Workshops (Simutools)*, pages 60:1–60:10, ICST, Brussels, Belgium, 2008. Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering (ICST). URL: <http://dl.acm.org/citation.cfm?id=1416222.1416290>. (Cited on pp. 7, 173, and 180.)
- [260] Gergely Varró, Anthony Anjorin, and Andy Schürr. Unification of Compiled and Interpreter-Based Pattern Matching Techniques. In *European Conference on Modelling Foundations and Applications (ECMFA)*, pages 368–383, Berlin Heidelberg, 2012. Springer. URL: https://doi.org/10.1007/978-3-642-31491-9_28. (Cited on pp. 15, 172, 173, 176, and 237.)
- [261] Alexey Vinel, Lin Lan, and Nikita Lyamin. Vehicle-to-Vehicle Communication in C-ACC/Platooning Scenarios. *IEEE Comm. Magazine*, 53(8):192–197, 2015. URL: <https://doi.org/10.1109/MCOM.2015.7180527>. (Cited on p. 1.)
- [262] András Vörös, Márton Búr, István Ráth, Ákos Horváth, Zoltán Micskei, László Balogh, Bálint Hegyi, Benedek Horváth, Zsolt Mázló, and Dániel Varró. MoDeS3: Model-Based Demonstrator for Smart and Safe Cyber-Physical Systems. In *NASA Formal Methods*, pages 460–467, Cham, 2018. Springer International. URL: https://doi.org/10.1007/978-3-319-77935-5_31. (Cited on p. 237.)
- [263] Hiroshi Wada, Pruet Boonma, Junichi Suzuki, and Katsuya Oba. Modeling and Executing Adaptive Sensor Network Applications with the Matilda UML Virtual Machine. In *International Conference on Software Engineering and Applications (SEA)*, pages 216–225, Anaheim, CA, USA, 2007. ACTA Press. URL: <http://dl.acm.org/citation.cfm?id=1647636.1647674>. (Cited on p. 40.)
- [264] Dorothea Wagner and Roger Wattenhofer. *Algorithms for sensor and ad hoc networks: advanced lectures*. Springer, Berlin Heidelberg, 2007. ISBN: 978-3-540-74990-5. URL: <https://doi.org/10.1007/978-3-540-74991-2>. (Cited on pp. 3, 4, 6, 17, 21, 22, 23, 25, 26, 27, 28, 29, 31, 35, 84, 201, 205, 239, 241, 243, 244, 245, 246, and 247.)
- [265] Shiyong Wang, Jiafu Wan, Di Li, and Chunhua Zhang. Implementing smart factory of industrie 4.0: An outlook. *International Journal of Distributed Sensor Networks*, 12(1):3159805, 2016. URL: <https://doi.org/10.1155/2016/3159805>. (Cited on p. 1.)
- [266] Shiyong Wang, Jiafu Wan, Daqiang Zhang, Di Li, and Chunhua Zhang. Towards Smart Factory for Industry 4.0: A Self-Organized Multi-Agent System with Big Data Based Feedback and Coordination. *Computer Networks (ComNet)*, 101:158 – 168, 2016. (Cited on p. 1.)

- [267] Yu Wang. Topology Control for Wireless Sensor Networks. In *Wireless Sensor Networks and Applications, Signals and Communication Technology*, pages 113–147. Springer Science+Business Media, Boston, MA, USA, 2008. URL: https://doi.org/10.1007/978-0-387-49592-7_5. (Cited on pp. 2, 3, 6, 17, 21, 22, 25, 26, 27, 28, 30, 84, 201, 205, 239, 241, 242, 243, 244, 245, 246, 247, and 250.)
- [268] Yu Wang and Xiang-Yang Li. Localized Construction of Bounded Degree and Planar Spanner for Wireless Ad Hoc Networks. *Mobile Networks and Applications*, 11(2):161–175, 2006. URL: <https://doi.org/10.1007/s11036-006-4469-5>. (Cited on p. 245.)
- [269] Roger Wattenhofer, Li Li, Paramvir Bahl, and Yi-Min Wang. Distributed Topology Control for Power Efficient Operation in Multihop Wireless Ad Hoc Networks. In *IEEE International Conference on Computer Communications (INFOCOM)*, volume 3, pages 1388–1397, NY, USA, 2001. IEEE. URL: <https://doi.org/10.1109/INFCOM.2001.916634>. (Cited on p. 246.)
- [270] Roger Wattenhofer and Aaron Zollinger. XTC: A Practical Topology Control Algorithm for Ad-Hoc Networks. In *International Parallel and Distributed Processing Symposium (IPDPS)*, pages 216–223, NY, USA, 2004. IEEE. URL: <https://doi.org/10.1109/IPDPS.2004.1303248>. (Cited on pp. 31, 191, 205, 243, 250, and 251.)
- [271] Markus Weckesser, Roland Kluge, Martin Pfannemüller, Michael Matthé, Andy Schürr, and Christian Becker. Optimal Reconfiguration of Dynamic Software Product Lines Based on Performance-Influence Models. In *International Conference on Software Product Lines (SPLC)*, volume 1, pages 98–109, New York, NY, USA, 2018. ACM. URL: <https://doi.org/10.1145/3233027.3233030>. (Cited on pp. 14, 182, 269, and 280.)
- [272] Markus Weckesser, Malte Lochau, Thomas Schnabel, Björn Richerzhagen, and Andy Schürr. Mind the Gap! Automated Anomaly Detection for Potentially Unbounded Cardinality-Based Feature Models. In *Fundamental Approaches to Software Engineering (FASE)*, pages 158–175, Berlin Heidelberg, 2016. Springer. URL: https://doi.org/10.1007/978-3-662-49665-7_10. (Cited on p. 82.)
- [273] Andrew Whitmore, Anurag Agarwal, and Li Da Xu. The Internet of Things – A survey of topics and trends. *Information Systems Frontiers*, 17(2):261–274, 2015. URL: <https://doi.org/10.1007/s10796-014-9489-2>. (Cited on p. 1.)
- [274] Matthias Wichtlhuber, Sebastian Bücken, Roland Kluge, Mahdi Mousavi, and David Hausheer. Of Strategies and Structures: Motif-Based Fingerprinting Analysis of Online Reputation Networks. In *International Conference on Local Computer Networks (LCN)*, pages 469–476, NY, USA, 2016. IEEE. URL: <https://doi.org/10.1109/LCN.2016.76>. (Cited on p. 14.)

- [275] Markus Winand. The Three-Valued Logic of SQL. <http://modern-sql.com/concept/three-valued-logic>, 2018. Last accessed: 2018-05-23. (Cited on p. 37.)
- [276] Tim Winter. RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks. *IETF RFC 6550*, 2012. URL: <https://tools.ietf.org/html/rfc6550>. (Cited on pp. 84 and 192.)
- [277] Alec Woo, Terence Tong, and David Culler. Taming the Underlying Challenges of Reliable Multihop Routing in Sensor Networks. In *International Conference on Embedded Networked Sensor Systems (SenSys)*, pages 14–27, New York, NY, USA, 2003. ACM. URL: <https://doi.org/10.1145/958491.958494>. (Cited on pp. 4 and 34.)
- [278] The working group for WLAN standards. IEEE 802.11 Wireless Local Area Networks. <http://grouper.ieee.org/groups/802/11/> (visited: 2018-11-23). (Cited on p. 192.)
- [279] Andrew Chi-Chih Yao. On Constructing Minimum Spanning Trees in k -Dimensional Spaces and Related Problems. *SIAM Journal on Computing*, 11(4):721–736, 1982. URL: <https://doi.org/10.1137/0211059>. (Cited on p. 252.)
- [280] Pamela Zave. Understanding SIP through Model-Checking. In *Principles, Systems and Applications of IP Telecommunications. Services and Security for Next Generation Networks*, volume 5310 of *LNCS*, pages 256–279. Springer, Berlin Heidelberg, 2008. URL: https://doi.org/10.1007/978-3-540-89054-6_13. (Cited on pp. 9, 166, and 167.)
- [281] Pamela Zave. Using Lightweight Modeling to Understand Chord. *SIGCOMM Comput. Commun. Rev.*, 42(2):49–57, 2012. URL: <https://doi.org/10.1145/2185376.2185383>. (Cited on pp. 9 and 167.)
- [282] Pamela Zave. Reasoning about Identifier Spaces: How to Make Chord Correct. *IEEE Transactions on Software Engineering (TSE)*, 43(12):1144–1156, 2017. URL: <https://doi.org/10.1109/TSE.2017.2655056>. (Cited on pp. 9 and 167.)

A

LIST OF FIGURES

Figure 1.1	Example: WSN	4
Figure 1.2	Overview of development phases of a TC algorithm	5
Figure 1.3	Example: Local knowledge	8
Figure 2.1	Location of Chapter 2 in TC algorithm development process	17
Figure 2.2	Example: Multi-graph	20
Figure 2.3	Example: Topology with transmission ranges	24
Figure 2.4	OSI Reference Model with possible roles of a TC mechanism	29
Figure 2.5	Interaction of TC mechanism with other components	30
Figure 2.6	Example: Topology control with kTC ($k = 1.5$)	32
Figure 2.7	Example: Integrated representation of input and virtual topology	38
Figure 2.8	Architecture of a TC mechanism	39
Figure 2.9	Example: Dynamic topology control ($k = 1.5$)	41
Figure 2.10	Architecture of a TC multi-mechanism	42
Figure 3.1	Location of Chapter 3 in TC algorithm development process	45
Figure 3.2	Connection of modeling languages and specification refinement	46
Figure 3.3	Four-layer MDA metamodeling hierarchy	49
Figure 3.4	Example: Topology metamodel	54
Figure 3.5	Example: Topology with pending link-weight modification events	55
Figure 3.6	Example: Loop pattern p_{loop}	58
Figure 3.7	Example: Parallel-links pattern $p_{parallel}$	59
Figure 3.8	Example: Inactive-link pattern p_i	60
Figure 3.9	Example: kTC patterns $p_{kTC,i}$ and $p_{kTC,a}$	60
Figure 3.10	Example: Matches m_1 and m_2 of inactive-link and kTC pattern	61
Figure 3.11	Structure of a graph constraint	62
Figure 3.12	Example: Structural constraints $C_{no-loops}$ and $C_{no-parallel-links}$	63
Figure 3.13	Example: kTC-specific graph constraints $C_{kTC,i}$ and $C_{kTC,a}$	64
Figure 3.14	Example: Equivalent marking constraints C_u and C_{ai}	65
Figure 3.15	Example: Maxpower-specific graph constraints	65
Figure 3.16	Sketches for proof of preserved connectivity for kTC	69

Figure 3.17	Structure of a GT rule	70
Figure 3.18	Illustration of GT rule applicability	71
Figure 3.19	Example: TC rules	72
Figure 3.20	Example: Context event rules	74
Figure 3.21	Example: Auxiliary rules $R_{\text{find-n}}$, $R_{\text{find-e}}$, $R_{\text{find-u}}$, and R_u	75
Figure 3.22	Story diagram of Maxpower algorithm	77
Figure 3.23	Story diagrams of context event handlers	79
Figure 3.24	Story diagrams for Maxpower-specific context event handlers	80
Figure 3.25	Example: Feature diagram with feature model instance (grey)	83
Figure 3.26	LMST-specific graph constraints for limited neighborhood size x	85
Figure 4.1	Location of Chapter 4 in TC algorithm development process	93
Figure 4.2	Story diagram of TC algorithm template	99
Figure 4.3	Example: Violation of consistency	101
Figure 4.4	Schema of application conditions synthesis algorithm	103
Figure 4.5	Detailed view of synthesis algorithm	105
Figure 4.6	Gluing sequence (schematic)	108
Figure 4.7	Example: Gluing $p'_{i,i,2}$ of RHS_i and $p_{kTC,i}$ with extension to $c'_{i,i,2,1}$	109
Figure 4.8	Results of specialization step ① for $(R_i, C_{kTC,i})$	110
Figure 4.9	Results of anticipation step ② for $(R_i, C_{kTC,i})$	111
Figure 4.10	Postcondition and application condition premises for $(R_a, C_{kTC,a})$	117
Figure 4.11	Refined activation rule R'_a	118
Figure 4.12	Refined link addition rule R'_{+e}	118
Figure 4.13	Postcondition and application condition premises for $(R_{+e}, C_{kTC,a})$	119
Figure 4.14	Refined link removal rule R'_{-e}	121
Figure 4.15	Synthesis results for $(R_{-e}, C_{kTC,i})$	122
Figure 4.16	Selected application condition premises for $(R_{\text{mod-w}}, C_{kTC,a})$	123
Figure 4.17	Selected synthesis results for $(R_{\text{mod-w}}, C_{kTC,i})$	124
Figure 4.18	Refined link-weight modification rule $R'_{\text{mod-w}}$	125
Figure 4.19	Effect of synthesized $PAC_{\text{mod-w},3}$ and $PAC_{\text{mod-w},4}$	126
Figure 4.20	Story diagram of kTC after constructive approach	128
Figure 4.21	Violations of application conditions of $R'_{\text{mod-w}}$ in Figure 4.3a	129
Figure 4.22	Example: Resolution of application condition violations	132
Figure 4.23	Insertion of anticipation loop in story diagram (schematic)	136
Figure 4.24	Example: Effect of NACs of anticipation rules for kTC	138
Figure 4.25	kTC-specific link addition and removal handlers	140
Figure 4.26	kTC-specific link-weight modification handler	141
Figure 4.27	Proof sketch for correctness of anticipation loop synthesis algo.	142
Figure 4.28	Sketch of refined context event handler for proof of correctness	145
Figure 4.29	Specification extension for TC algorithm parameter reconfiguration	148
Figure 4.30	Example: Consistency violations due to modified kTC parameter	149
Figure 4.31	Example: Constructive approach results for $R_{\text{mod-kTC-k}}$	151

Figure 4.32	kTC-specific constraints with marking constraints for v_{13} and v_{32}	152
Figure 4.33	Refined TC rules based on C_W^M	153
Figure 4.34	Example: Solution to nontermination of TC algorithm specification	155
Figure 4.35	Structure of synthesized anticipation loops	156
Figure 4.36	Refined unmarking handler and unmarking rule R'_u for kTC	157
Figure 4.37	Sketch of refined TC algorithm specification	159
Figure 5.1	Location of Chapter 5 in TC algorithm development process	169
Figure 5.2	Build process architecture of eMOFLON	174
Figure 5.3	Structure of SIMONSTRATOR with COBOLT	180
Figure 5.4	TopologyControlComponent with important interfaces	181
Figure 5.5	Architecture of EMoflonFacade	184
Figure 5.6	SIMONSTRATOR visualization capabilities	185
Figure 5.7	COALA visualization capabilities	185
Figure 5.8	Sketch of transformation of application conditions into eMOFLON	187
Figure 5.9	Simulation setup with variable parameters/seeds	192
Figure 5.10	Steps in methodology for discussion of correctness	193
Figure 5.11	Execution time comparison of simulation vs. TC	200
Figure 5.12	Execution time comparison batch vs. dynamic	201
Figure 5.13	Link state mod. comparison batch vs. dynamic	202
Figure 5.14	Architecture of the ToCoCo evaluation framework	206
Figure 5.15	ToCoCo configuration space and selected options	207
Figure 5.16	cMOFLON build process with reuse from eMOFLON	209
Figure 5.17	Example: Topology of l*kTC with $k = 1.3, a = 1.5$	211
Figure 5.18	Example: Virtual topology of LMST	212
Figure 5.19	cMOFLON metamodel with kTC, l*kTC, and LMST	213
Figure 5.20	Specification of l*kTC	214
Figure 5.21	Specification of LMST	217
Figure 5.22	Anatomy of a generated TC algorithm in cMOFLON	218
Figure 5.23	FLOCKLAB location map	226
Figure 5.24	Execution time comparison of manual and generated TC algorithms	232
Figure 5.25	Execution time comparison for l*kTC variants	233
Figure 6.1	Location of Chapter 6 in TC algorithm development process	239
Figure 6.2	Illustration of lune, circumcircle, and θ -dominating region	241
Figure 6.3	Visualization of landscape of collected TC algorithms	242
Figure 6.4	Example: Reduced network lifetime with kTC	255
Figure 6.5	Example: Extended network lifetime with e-kTC	258
Figure 6.6	Feature diagram of with two TC algorithm families	260
Figure 6.7	Example: Refinement for triangle-based TC algorithms	263
Figure 6.8	Example: Refinement in the family of cone-based TC algorithms	263
Figure 6.9	Construction of alternative AU-path	265

Figure 6.10	Case distinction in induction step	266
Figure 6.11	Example: Graph constraints for YGG	268
Figure 6.12	Decision diagram for applicability discussion	271
Figure 7.1	Overview of development phases of a TC algorithm (repeated) .	275

B

LIST OF TABLES

Table 2.1	Alternative representations of input and virtual topology	36
Table 3.1	Summary of consistency levels for TC mechanisms	66
Table 3.2	Operationalization of context events	74
Table 4.1	Example: Preservation and violation of weak consistency w.r.t. kTC_{100}	
Table 4.2	Role of indices in synthesis algorithm description	104
Table 4.3	Filter rules during synthesis algorithm	105
Table 4.4	Summary of application condition synthesis results for kTC	106
Table 4.5	Example: Violations of application condition in Figure 4.3a	131
Table 4.6	Reasons for inapplicability of R'_a and R'_i	154
Table 5.1	Comparison of developed tools COBOLT and cMOFLON	171
Table 5.2	Summary of configuration parameters for simulation experiments	191
Table 5.3	Coverage for SDM-related eMOFLON projects	197
Table 5.4	Type mappings in cMOFLON	220
Table 5.5	Size comparison eMOFLON vs. cMOFLON	228
Table 5.6	Binary image size comparison	230
Table 6.1	Topology control landscape	243

C

LIST OF ALGORITHMS

4.1	Constructive approach algorithm	102
4.2	Gluing generation algorithm	112
4.3	Anticipation loop synthesis algorithm	135

D

LIST OF EXAMPLES

1.1	Example (Underlay, virtual, and overlay topology)	3
1.2	Example (Local monitoring and updating responsibilities)	7
1.3	Example (Challenges)	9
2.8	Example (Graphs)	19
2.13	Example (Topology)	23
2.17	Example (Local view)	25
2.26	Example (kTC)	33
2.31	Example (TC mechanism topology)	37
2.32	Example (Dynamic TC)	40
3.2	Example (Metamodeling terminology and notation)	50
3.4	Example (Metamodel of running example)	52
3.5	Example (Topology model and notation)	53
3.6	Example (Local consistency properties)	56
3.9	Example (Structural consistency patterns)	58
3.10	Example (kTC-specific patterns)	59
3.12	Example (Pattern match)	61
3.15	Example (Pattern and match extension)	62
3.19	Example (Structural constraints)	63
3.20	Example (kTC-specific constraints)	63
3.21	Example (Maxpower-specific constraint)	64
3.23	Example (Proof of connectivity preservation for kTC)	67
3.27	Example (TC rules)	72
3.28	Example (Context event rules)	73
3.29	Example (Auxiliary rules)	73
3.32	Example (Story diagram of Maxpower algorithm)	77
3.33	Example (Story diagrams of generic context event handlers)	78
3.34	Example (SDM specification of Maxpower-specific context event handlers)	78
3.35	Example (Feature model and feature model instance)	82

4.8	Example (Correctness of Maxpower specification)	97
4.9	Example (Control flow specification templates)	98
4.10	Example (Consistency violations by template specification)	99
4.13	Example (Gluing)	108
4.14	Example (Gluing substep 1.1 for $(R_i, C_{kTC,i})$)	109
4.15	Example (Extension substep 1.2 in detail)	112
4.16	Example (Extension substep 1.2 for $(R_i, C_{kTC,i})$)	112
4.17	Example (Self-gluing substep 1.3 for $(R_i, C_{kTC,i})$)	113
4.18	Example (Anticipation step 2 for $(R_i, C_{kTC,i})$)	114
4.19	Example (Correctness of kTC specification after synthesis algorithm)	126
4.21	Example (Violation of application condition and resolution)	130
4.23	Example (Anticipability preconditions)	133
4.25	Example (Notation of anticipation loop synthesis algorithm)	134
4.26	Example (Anticipation loop synthesis step 3 for kTC)	136
4.27	Example (Effect of application conditions of anticipation loops)	137
4.28	Example (Correctness of kTC specification after anticipation loop synthesis)	137
4.33	Example (Nontermination of kTC with C_W^M)	152
4.34	Example (Unmarking handler)	156
4.35	Example (Retraction loops for kTC)	157
5.1	Example (l*kTC)	210
5.2	Example (LMST)	211
6.5	Example (Reduced network lifetime with kTC)	254
6.9	Example (Extended network lifetime with e-kTC)	257
6.11	Example (Constraint refinement)	262
6.12	Example (Proof of connectivity for triangle-based TC algorithms)	264
6.13	Example (Constraints for YGG)	267

E

LIST OF THEOREMS

4.30	Theorem (Correctness of synthesized anticipation loops)	142
4.32	Theorem (Correctness of refined context event handler specification) . . .	144

 TABLE OF CONTENTS (DETAILED)

1	THE NEED FOR CORRECT-BY-CONSTRUCTION TOPOLOGY CONTROL	1
1.1	Challenges in wireless communication systems engineering	3
1.2	Goals	10
1.3	Contributions and thesis structure	11
1.4	Publications and supervised theses	14
1.5	Technical remarks	16
2	FUNDAMENTALS OF TOPOLOGY CONTROL	17
2.1	Graph theory	18
2.2	Network topologies	21
2.3	Performance of topologies	26
2.4	Topology control	27
2.4.1	Topology control in the network stack	28
2.4.2	Required information for topology control	30
2.4.3	Locality	30
2.4.4	Running example: kTC	31
2.5	Dynamic topology control	34
2.5.1	Connecting input and virtual topology	35
2.5.2	Context event handling	37
2.5.3	Architecture of a topology control mechanism	39
2.5.4	Adaptive wireless sensor networks	40
3	SELECTION OF SPECIFICATION LANGUAGES	45
3.1	Metamodeling	48
3.2	Local consistency properties	56
3.3	Global consistency properties	67
3.4	Elementary topology modifications	70
3.5	Execution order of topology modifications	76
3.6	Configuration space specification	81
3.7	Related work	84
3.7.1	Snapshot-based approaches	84

3.7.2	Graph-grammar-based approaches	86
3.7.3	Game-theoretic approaches	89
3.7.4	Role-centric approaches	89
4	SYNTHESIS OF CORRECT TOPOLOGY CONTROL MECHANISMS	93
4.1	Consistency preservation and violation	95
4.2	Constructive approach	102
4.3	Application of constructive approach to running example	106
4.3.1	Refinement of topology control rules	106
4.3.2	Refinement of context event rules	116
4.3.3	Interpretation of synthesized application conditions	126
4.4	Synthesis of anticipation loops	130
4.4.1	Violation of application conditions	130
4.4.2	Anticipation loop synthesis algorithm	133
4.4.3	Proof of correctness for anticipation loop synthesis algorithm	139
4.4.3.1	Correctness of synthesized anticipation loops	139
4.4.3.2	Correctness of refined context event handlers	144
4.5	Applicability of anticipation loop synthesis algorithm	147
4.5.1	Discussion of applicability	147
4.5.2	Handling parameter modifications of topology control algorithms	148
4.5.3	Enforcing termination of topology control algorithm specification	150
4.6	Related work	160
4.6.1	Conceptual extensions of the constructive approach	160
4.6.2	Further usage scenarios of constructive approach	162
4.6.3	Approaches to ensuring consistency properties	163
5	TOOL SUPPORT FOR RAPID EVALUATION	169
5.1	Selection of modeling tool	172
5.2	Model-based simulative evaluation with Cobolt	178
5.2.1	Selection of network simulator	178
5.2.2	Architecture	179
5.2.2.1	Structure of simulation setup in Simonstrator	179
5.2.2.2	Design decisions	180
5.2.2.3	Cobolt-specific components	181
5.2.3	Implementation aspects	186
5.2.3.1	Integration of eMoflon and Simonstrator	186
5.2.3.2	Specifying multiple application conditions in eMoflon	186
5.2.3.3	Consistency checking	189
5.2.3.4	Code and tool availability	189
5.2.4	Evaluation of Cobolt	189
5.2.4.1	Research questions	189
5.2.4.2	Evaluation setup	190

5.2.4.3	Results and discussion for RQ ₁ (correctness)	193
5.2.4.4	Results and discussion for RQ ₂ (efficiency)	199
5.2.4.5	Results and discussion for RQ ₃ (dynamic topology control)	201
5.3	Model-based testbed evaluation with cMoflon	204
5.3.1	Architecture	205
5.3.1.1	Contiki	205
5.3.1.2	ToCoCo	206
5.3.1.3	Integration with eMoflon build process	208
5.3.2	Specification	208
5.3.2.1	The l*kTC algorithm	208
5.3.2.2	The LMST algorithm	211
5.3.2.3	Basic metamodel of cMoflon	212
5.3.2.4	Specifying kTC	213
5.3.2.5	Specifying l*kTC	213
5.3.2.6	Specifying LMST	214
5.3.2.7	Supported metamodeling concepts and extension points	216
5.3.3	Code generation	216
5.3.3.1	Mimicking object orientation	216
5.3.3.2	Representing a topology control algorithm	218
5.3.3.3	Representing motes and links	218
5.3.3.4	Representing data types and helper classes	219
5.3.3.5	Representing link states	219
5.3.3.6	Representing object creation and deletion	220
5.3.3.7	Representing matches and collections of matches	221
5.3.3.8	Providing hop-count information	223
5.3.3.9	Binary image size	223
5.3.3.10	Tool and code availability	224
5.3.4	Evaluation of cMoflon	224
5.3.4.1	Research Questions	224
5.3.4.2	Experimental setup	225
5.3.4.3	Results and discussion for RQ ₁ (correctness)	226
5.3.4.4	Results and discussion for RQ ₂ (efficiency)	229
5.4	Related work	235
5.4.1	Network simulation and model-driven engineering	235
5.4.2	Code generation for wireless sensor networks	236
6	DEVELOPMENT OF FAMILIES OF TOPOLOGY CONTROL ALGORITHMS	239
6.1	Landscape of topology control algorithms	241
6.2	Specification of topology control algorithms	248
6.2.1	Element properties and context events	248
6.2.2	Presentation of topology control algorithms	249

6.2.2.1	Maxpower algorithm (revisited)	249
6.2.2.2	Gabriel graph algorithm	250
6.2.2.3	Relative neighborhood algorithm	250
6.2.2.4	XTC algorithm	250
6.2.2.5	kTC algorithm (revisited)	251
6.2.2.6	l*kTC algorithm (revisited)	251
6.2.2.7	Yao Graph Algorithm	252
6.2.2.8	Theta-graph algorithm	253
6.2.2.9	e-kTC algorithm	253
6.2.2.10	The minimum-weight predicate	258
6.3	Specification of families of topology control algorithms	260
6.3.1	Specification using feature models	260
6.3.2	Specification using constraint refinement	261
6.4	Proving global consistency properties	264
6.5	Discussion of applicability	267
6.5.1	Discussion based on examples from the topology control landscape	267
6.5.2	Discussion based on modeling techniques	269
6.6	Related work	273
7	CONCLUSION	275
7.1	Summary	276
7.2	Outlook	278
	BIBLIOGRAPHY	283
Appendix A	LIST OF FIGURES	316
Appendix B	LIST OF TABLES	320
Appendix C	LIST OF ALGORITHMS	321
Appendix D	LIST OF EXAMPLES	322
Appendix E	LIST OF THEOREMS	324
Appendix F	TABLE OF CONTENTS (DETAILED)	325