Distributed Real-time Systems – Deterministic Protocols for Wireless Networks and Model-Driven Development with SDL

DENNIS CHRISTMANN

Thesis approved
by the Department of Computer Science
of the University of Kaiserslautern (TU Kaiserslautern)
for the award of the Doctoral Degree
Doctor of Engineering (Dr.-Ing.)

Submission date	:	13.01.2015
Date of the viva	:	08.05.2015
Dean	:	Prof. Dr. Klaus Schneider
PhD committee		
Chairperson	:	Prof. Dr. rer. nat. Karsten Berns
Reviewers	:	Prof. Dr-Ing. Reinhard Gotzhein
	:	Prof. Dr-Ing. Jens B. Schmitt

D 386

Acknowledgment

There are many people deserving a big "thank you" for supporting me during my time as student and postgraduate. First of all, I would like to thank all members of the PhD committee – Prof. Reinhard Gotzhein, Prof. Jens Schmitt, and Prof. Karsten Berns – for their role in the PhD process and the effort they had to spend.

A special thanks goes to my doctoral advisor Prof. Reinhard Gotzhein for providing me the opportunity to work in the Networked Systems Group, for the motivation in times of doubts, and for the numerous research-oriented and personal advices. Without his suggestion to apply for the admission to the PhD program, I probably would never have dared to start working in the field of research. The numerous constructive discussions with him made a great contribution to the success of this work. In this regard, I especially also would like to thank my second reviewer Prof. Jens Schmitt and his research group, whose interesting lectures and projects influenced me a lot.

Furthermore, I would like to thank my current and former colleagues Tobias Braun, Markus Engel, Christopher Kramer, Anuschka Igel, Philipp Becker, Thomas Kuhn, Marc Krämer, Christian Webel, Ingmar Fliege, and Alexander Geraldy for many creative discussions and for proof-reading parts of this thesis. In particular, many thanks go to Tobias and Markus for the joint publications and for their far-reaching technical knowledge, which saved me many hours in finding bugs and in interpreting gcc error messages. I would also like to express my gratitude to all former and current students of the Networked Systems Group I have had the privilege of working with. A special thanks goes to Barbara Erlewein, who always supports us in the fight against travel expense reports and the administrative department.

Furthermore, I would like to thank the Carl Zeiss Foundation very much for funding my research topic for more than two and a half years.

My particular thanks also goes to my wife Sandra, my sister Janina, my parents, my grand-parents, and my parents-in-law, who supported me during my entire study period. I am grateful for their patience and appreciation in times of critical deadlines, for the confidence they place in me, and for the time I can spend with them. Finally, I would like to thank all my friends for many joyful hours and for providing a balance to work.

June 2015 Dennis Christmann

Abstract

In a networked system, the communication system is indispensable but often the weakest link w.r.t. performance and reliability. This, particularly, holds for wireless communication systems, where the error- and interference-prone medium and the character of network topologies implicate special challenges. However, there are many scenarios of wireless networks, in which a certain quality-of-service has to be provided despite these conditions. In this regard, distributed real-time systems, whose realization by wireless multi-hop networks becomes increasingly popular, are a particular challenge. For such systems, it is of crucial importance that communication protocols are deterministic and come with the required amount of efficiency and predictability, while additionally considering scarce hardware resources that are a major limiting factor of wireless sensor nodes. This, in turn, does not only place demands on the behavior of a protocol but also on its implementation, which has to comply with timing and resource constraints.

The first part of this thesis presents a deterministic protocol for wireless multi-hop networks with time-critical behavior. The protocol is referred to as Arbitrating and Cooperative Transfer Protocol (ACTP), and is an instance of a binary countdown protocol. It enables the reliable transfer of bit sequences of adjustable length and deterministically resolves contest among nodes based on a flexible priority assignment, with constant delays, and within configurable arbitration radii. The protocol's key requirement is the collision-resistant encoding of bits, which is achieved by the incorporation of black bursts. Besides revisiting black bursts and proposing measures to optimize their detection, robustness, and implementation on wireless sensor nodes, the first part of this thesis presents the mode of operation and time behavior of ACTP. In addition, possible applications of ACTP are illustrated, presenting solutions to well-known problems of distributed systems like leader election and data dissemination. Furthermore, results of experimental evaluations with customary wireless transceivers are outlined to provide evidence of the protocol's implementability and benefits.

In the second part of this thesis, the focus is shifted from concrete deterministic protocols to their model-driven development with the Specification and Description Language (SDL). Though SDL is well-established in the domain of telecommunication and distributed systems, the predictability of its implementations is often insufficient as previous projects have shown. To increase this predictability and to improve SDL's applicability to time-critical systems, real-time tasks, an approved concept in the design of real-time systems, are transferred to SDL and extended to cover node-spanning system tasks. In this regard, a priority-based execution and suspension model is introduced in SDL, which enables task-specific priority assignments in the SDL specification that are orthogonal to the static structure of SDL systems and control transition execution orders on design as well as on implementation level. Both the formal incorporation of real-time tasks into SDL and their implementation in a novel scheduling strategy are discussed in this context. By means of evaluations on wireless sensor nodes, evidence is provided that these extensions reduce worst-case execution times substantially, and improve the predictability of SDL implementations and the language's applicability to real-time systems.

Contents

1. Chapter

Introduction

In a networked system, the communication system is an integral and indispensable part and has often to provide a required degree of Quality-of-Service (QoS). In this regard, distributed real-time systems represent a particularly challenging application domain due to their stringent demands on the communication system in terms of deterministic protocols and implementations with predictable execution times. In wired networks, real-time-capable communication has a long tradition and many successful examples can be found, in particular, in the automotive domain, where communication technologies like the Controller Area Network (CAN) [Int04] and FlexRay [Fle10] have become established standards.

However, in wireless networks in general and Wireless Sensor Networks (WSNs) in particular, there are far less examples and, although there exist several communication standards with QoS support by now – like WirelessHART [Int10] and ISA 100.11a [Int12a] -, their capabilities and diversity are still considerably smaller compared to wired solutions. While these standards are primarily based on combinations of Time Division Multiple Access (TDMA), Frequency Division Multiple Access (FDMA), and Carrier Sense Multiple Access with Collision Avoidance (CSMA/CA), very little attention is paid to binary countdown protocols, which build a separate class of deterministic MAC (Medium Access Control) protocols and provide predictable value-based medium arbitration with bounded delays. In wired networks, the most famous representative of this protocol class is CAN [Int04]. In wireless networks, there is, however, no established standard, yet some variants can be found in research papers (e.g., in [PAT07]). The first part of this thesis proposes a novel binary countdown protocol for wireless multi-hop networks with fewer limitations than existing solutions. It is denoted by ***Arbitrating and Cooperative Transfer Protocol (ACTP)*** and incorporates black bursts as communication primitive.

To run deterministic protocols like ACTP, their implementations have to comply with strict timing constraints. However, this demand is often contrary to the trend towards more complex protocol stacks and software systems, in which several system tasks compete for the scarce hardware resources. A common approach to deal with the increasing complexity of software systems is the application of model-driven development processes, which enable abstraction and improve reusability and productivity. For distributed systems and protocols, a well-known language with support of model-driven development processes is the Specification and Description Language (SDL) [Int12c]. However, as it turned out, SDL implementations suffer from low efficiency and predictability, thereby impeding the utilization of the language for protocols with time-critical behavior and real-time systems, where timeliness is a basic requirement for correctness. To improve SDL's applicability to such systems and to increase the predictability of SDL implementations, the second part of this thesis presents several language extensions for SDL. They are summarized under the name ***SDL real-time tasks*** and transfer an approved concept from the design of real-time systems to SDL [Kop97].

Altogether, this thesis consists of ten main chapters, which are structured into the two parts as follows:

- **Part I** introduces ACTP as a new representative of a deterministic binary countdown protocol for wireless multi-hop networks. This part consists of five chapters and presents the protocol's basics, mode of operation, applications, and evaluation.
- **Part II** proposes SDL extensions to improve the expressiveness of the language and the predictability of its implementations. It also consists of five chapters, describing the extensions and their implementation and evaluation, and is supplemented by Appendices B and C, where the extensions are formally defined.

Appendix A is related to both parts and outlines the Imote 2 platform, which has been used as representative of a wireless sensor node to evaluate both ACTP and the SDL extensions. A general conclusion and a discussion of open topics and future work is provided in Chapter 12. In all chapters in which results of own work are presented, references where these results or parts of them have been published are provided at the beginning of the chapter. These references are numeric and refer to the list of publications on pages 245–247, whereas other references are alphanumeric and can be found in the bibliography starting on page 225.

PART I

ACTP: A DETERMINISTIC BINARY COUNTDOWN PROTOCOL FOR WIRELESS NETWORKS

Though WSNs are no "brand-new" topic and several related standards have evolved over the last two decades, their dissemination is not yet complete. Due to their advantages w.r.t. flexibility, wiring cost, and scalability, they, particularly, more and more replace wired networks. However, there are also scenarios, in which wireless networks cannot provide a suitable solution but wired technologies are still preferred. Reasons for this are diverse: On the one hand, they have their origin in the wireless medium itself, which is hard to control and makes wireless networks in the first instance less predictable and reliable than wired solutions. Thereby, wireless networks cannot be deployed in scenarios, where unreliable communication may lead to personal injuries or material damage. On the other hand, the applicability of wireless networks is restricted by technical limitations, since wireless technologies usually provide lower performance; e.g., in terms of transmission rates. Medium arbitration falls into the same category and is significantly different in wireless networks than in wired networks. This difference mainly stems from the missing capabilities of wireless transceivers to detect collisions on the medium while sending, which is, on the contrary, available with wired mediums and utilized by several popular communication standards like IEEE 802.3 Ethernet [Ins12b] and CAN [Int04].

Compared to wired networks, a frequently mentioned drawback of wireless networks and, in particular, WSNs are limited energy resources that is aggravated by battery-powered nodes. Motivated by this issue, many duty cycling protocols have been devised for WSNs, which identify wireless transceivers as one of the major sources of energy waste and limit their energy demands by coordinated active/inactive cycles. Referring to this, S-MAC [YHE02, YHE04] is probably the most often cited representative, yet there are also enhancements that are more efficient w.r.t. energy consumption, QoS, and multi-hop support (e.g., T-MAC [vDL03] or RMAC [DSJ07]). Though the range of duty cycling protocols is very wide and protocols differ in synchronization, type of coordination, and medium access, they all have in common that they emphasize the importance of energy reduction, which must not be neglected in the design of any WSN protocol.

Because wireless nodes are becoming cheaper and smaller, topologies of wireless networks – and, in particular, WSNs – tend to become more and more complex regarding density, diameter, and node mobility. Thus, they define a new type of ad-hoc networks with (mostly) short links, many nodes, high partial dynamic, and large spatial expansion. However, these properties are contradictory to increasing communication requirements, which are demanded by many new application scenarios from the domain of distributed real-time systems and networked control systems. Thus, protocols for wireless networks and WSNs must not only consider energy as a major concern, but have also to offer QoS. In this regard, deterministic guarantees are a special challenge due to the demanding characteristics of wireless topologies, the hidden station problem [Kar90], and hardware limitations of wireless (sensor) nodes.

Most protocols of WSNs and wireless networks in general cannot provide such a high degree of guarantees and are therefore inapplicable if upper bounds on delays or a particular transfer rate must be met. This, particularly, holds for CSMA/CA-based protocols, which are very common for general purpose applications and, for instance, available in IEEE 802.11 DCF (Distributed Coordination Function) [Ins12a] and contention access periods of IEEE 802.15.4 [Ins11]. Since they suffer from collisions, their end-to-end delays are hardly predictable and without upper bounds. Furthermore, they are not optimal w.r.t. energy efficiency because of costly arbitrations and energy waste due to collisions [YHE02].

Due to these shortcomings, many existing standards additionally provide reservation-based medium access with time slots. Such TDMA-based protocols[1] can, for instance, be found in IEEE 802.11 PCF[2] (Point Coordination Function) [Ins12a], contention free periods of IEEE 802.15.4 [Ins11], and standards from the domain of process automation like WirelessHART [Int10] and ISA 100.11a [Int12a]. As big advantage over CSMA/CA, collisions are avoided entirely with TDMA and upper bounds on end-to-end delays are possible. A downside is that coordination overhead is much higher and that transmission schedules are hard to determine if communication demands are unknown or sporadic. This, in general, leads to overbooking of transmission slots and a waste of network resources.

In this part of the thesis, another class of medium access schemes is discussed that is based on the deterministic elimination of contest and collision resistance, and closes the gap between CSMA/CA's "flexible but unreliable" and TDMA's "reliable but inflexible" mode of operation. Though protocols of this class are neither all-purpose solutions nor replacements of established medium access schemes – TDMA is, for instance, still best suited to process strictly periodic traffic –, they provide attractive solutions to recurring problems of distributed systems like leader election. In the course of this discussion, a new instance of this protocol class is introduced, which is called ACTP and enables deterministic multi-hop arbitrations with bounded delays. ACTP is located at MAC layer and a so-called binary countdown protocol, whose most popular representative is the wired CAN bus [Int04]. Though it is not the first wireless binary countdown protocol, ACTP outperforms previous solutions w.r.t. flexibility and multi-hop support. Thereby, it opens the door to new application scenarios like deterministic and network-wide leader election and value propagation with bounded delays. One of the key foundations of ACTP to guarantee its deterministic mode of operation is the collision-resistant encoding of bits. For this purpose, so-called black bursts [Kuh09, KdI07] are utilized that are incorporated into ACTP together with newly devised optimizations.

Though ACTP is not limited to WSNs but addresses wireless multi-hop networks in general, we consider WSNs as one of the main application domains of ACTP and examine the protocol's implementability w.r.t. performance and energy limitations of wireless sensor nodes.

[1]Some TDMA-based protocols additionally incorporate FDMA, whose pros and cons regarding flexibility and QoS are similar to TDMA.

[2]Though PCF is specified in the IEEE 802.11 standard, its implementation is optional and hardly widespread.

Contributions

Before devising a binary countdown protocol, a certain fundament has to be created, which has to provide collision-resistant bit encoding and an answer to synchronization demands. In this part of the thesis, this fundament is outlined before ACTP, possible application scenarios, and experimental evaluations are presented. In detail, the following contributions are made:

- Though black bursts are no novel concept, they are revisited in this part of the thesis. Based on their original definition, a generalization is presented, prerequisites are discussed, and details on their realization are outlined. In this regard, a special focus is on IEEE 802.15.4-based [Ins11] transceivers and their impact on the detection of black bursts, which is formally considered.

- Optimization measures are proposed to improve the collision resistance of black bursts and the accuracy of black burst detection. For this purpose, the focus is on implementations with TI's CC 2420 transceiver [Tex07], which is a common transceiver in WSNs and also used in the second part of this thesis.

- A new deterministic binary countdown protocol for wireless networks is proposed. The protocol is called ACTP, is decentralized, and provides value-based medium arbitration with constant delays. Different from existing binary countdown protocols, it provides configuration options w.r.t. size of priorities and range of application.

- Timings of ACTP are derived formally and as function of configuration parameters and transceiver-depending properties. Thereby, the protocol's constant runtime can be calculated for each scenario in advance.

- Different operation modes of ACTP are introduced, which differ in application range and purpose. When running ACTP network-wide, it is shown that several beneficial information like the priorities of winners become public knowledge of the network. By presenting ACTP's cooperative mode, it is furthermore shown that binary countdown protocols are no pure arbitration protocols but can also be applied to the reliable and collision-protected propagation of data. Finally, advantages and limitations of running ACTP with restricted application ranges are examined.

- Threats of black bursts as well as ACTP are analyzed in terms of false positives and false negatives. Furthermore, possible countermeasures are presented.

- A set of sample applications is presented, showing the heterogeneity of possible scenarios with ACTP-based solutions. While there is no claim for completeness, examples include both general problems of wireless networks as well as applications in control systems and in projects with industrial partners.

- An excerpt of an SDL specification of ACTP is presented, which defines the protocol's behavior in a formal way and enables simulative evaluations of ACTP in large networks.

- Two implementation variants of ACTP are outlined, which are both developed for Imote 2 sensor platforms [MEMara]. They reflect different stages of development and differ in possible application contexts: One targets stand-alone evaluations; the other supports the protocol's utilization in a comprehensive protocol framework.

- In series of experimental evaluations, the feasibility of black bursts is demonstrated both in indoor environments with external sources of interference and in an outdoor environment. Furthermore, a proof-of-concept is provided for ACTP, thereby validating the protocol's implementation and formally derived timings.
- A survey of related work points out the differences between existing value-based arbitration protocols and ACTP. In this regard, so-called busy tone protocols and previous binary countdown protocols are considered.

Outline

This first part of the thesis is structured as follows:

- **Chapter 2** presents black bursts as foundation of ACTP. It is a mix of a survey of black bursts in previous works and of a deeper discussion on influencing factors and optimization possibilities. In this regard, a definition of black bursts, prerequisites, realization alternatives, and improvements are considered.
- **Chapter 3** proposes ACTP. For this purpose, the general principle of binary countdown protocols is examined. Afterwards, ACTP's terminology and mode of operation is formally introduced by means of a state machine. This chapter furthermore derives timings of the protocol formally and as function of configuration parameters and transceiver-depending properties. Thereafter, ACTP's network-wide application and its restricted and cooperative operation modes are discussed. A further topic is the impact of communication errors and possible countermeasures. Finally, an SDL specification and hand-written implementations of ACTP are sketched.
- **Chapter 4** presents applications of ACTP, including solutions to popular problems of distributed systems like leader election as well as specialized problems of control systems like Try-Once-Discard. In the course of these example applications, the integration of a synchronization protocol and compliance between ACTP and duty cycling is discussed.
- **Chapter 5** shows results of experimental evaluations demonstrating ACTP's feasibility. In this regard, the focus is in most instances on black bursts, whose reliability is the most crucial aspect of ACTP's deterministic behavior. The experiments have been conducted partially indoor in office environments and partially outdoor to assess the reliability of black bursts and ACTP with and without external sources of interference.
- **Chapter 6** surveys related work and relates them to ACTP. Besides other wireless binary countdown protocol, the presentation of related work includes busy tone protocols, whose mode of operation is different from binary countdown protocols but can also provide deterministic and value-based arbitration under certain circumstances.

2. Chapter

Black Bursts: A Communication Primitive for Deterministic Protocols

In wireless systems, the aim is usually to totally prevent simultaneous frame transmissions in the same collision domain, since overlapping transmissions can lead to destructive collisions. With regular frames, this fear is well-founded. A general approach to achieve this aim and to guarantee correct frame transmissions is the application of TDMA-based medium access protocols with reserved transmission slots, thereby serializing transmissions of all nodes in the same collision domain. However, TDMA is not always the best or most efficient solution, if, for instance, a value-based transmission order is desired or if transmissions happen rarely and sporadically.

In this chapter, a collision-resistant communication primitive called *black burst* is introduced, which builds the basis for various protocols with deterministic Quality-of-Service (QoS). Black bursts are no novel approach but have been proposed in several previous works (e.g., in [SK96, SK99, Kuh09]; see also Chapter 6 for a survey of related work). Their first mentioning goes back to 1996 [SK96], yet with a slightly different definition than in this thesis. In this chapter, black bursts are recapitulated and novel optimizations w.r.t. their realization are presented.

Though black bursts achieve significantly slower transmission rates than regular frame transmissions, they distinguish themselves by enabling the simultaneous and collision-protected transmission of small pieces of information. Thereby, they pave the way for various deterministic protocols like the Arbitrating and Cooperative Transfer Protocol (ACTP), which is introduced in Chapter 3 and applied to sample applications in Chapter 4. To begin with, this chapter provides a summary of black bursts (Sect. 2.2), their realization (Sect. 2.3), and measures to improve their robustness and detection accuracy (Sect. 2.4). It is completed by a motivation of black bursts in Sect. 2.1 and a discussion in Sect. 2.5.

The contents presented in this chapter have been published in [8], [10], [11], [12], [16].

2.1 Motivation

Simultaneous transmissions and destructive collisions state a crucial problem in wireless systems, since they can degrade the system's performance and violate its correct behavior. This section illustrates some problems of wireless networks, which occur with regular frame transmissions, in order to motivate the advantages of a collision-protected communication primitive. For this purpose, the section first outlines a common wireless network model and discusses limitations of protocols and transceivers afterwards.

2.1.1 A Common Network Model for Wireless Systems

A common network model of wireless systems [SK99] classifies links either as *communication*, *interference*, or *sensing links* (see Fig. 2.1). While a communication link (V_a to V_b) enables regular data transfer, interference and sensing links are too weak to deliver data frames correctly. Frames sent via an interference link (V_a to V_c), which overlap with another frame at a receiver, are, however, strong enough to collide in a destructive way and to avoid the correct reception of the second frame. Sensing links (V_a to V_d), on the other hand, cannot cause destructive collisions but enable nodes to observe a medium occupancy. Usually, it is assumed that a communication link is always an interference link but not vice versa. Similarly, an interference link is always a sensing link.

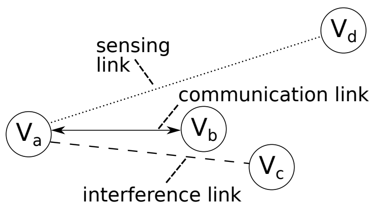


Figure 2.1: Network model with communication, interference, and sensing links [SK99].

Many protocols assume that links are symmetric.[1] This assumption is, in particular, made for communication links, e.g., to enable handshakes or acknowledgments. Yet in reality, symmetry is not necessarily given as shown by empirical studies [KNE03, KNG$^+$04, GKW$^+$02]. The type of link between two nodes can usually not be determined on the basis of the nodes' spatial distance, since signal strengths depend on the environment and objects in the propagation path, which attenuate, reflect, diffract, and scatter the signal. On the one hand, this may prevent communication despite a short distance. On the other hand, situations can exist, where communication is possible over very large distances [GKW$^+$02]. Such situations can, for instance, be found in corridors, which provoke constructive scattering and reflection [VVL$^+$11].

The network model states an explicit simplification and, in particular, interference is much more complicated to capture. In reality, it does not only depend on two nodes whether a transmission by one node causes a destructive collision at the other node. Instead, signal strength difference, temporal offset, and carrier phase offset between all colliding frames decide on the kind of collision, and whether and which frames can still be received correctly [KSM08, WLS14].

2.1.2 Limitations of Wireless Transceivers and State-of-the-Practice Protocols

Different from wired systems, transceivers of wireless systems are usually half-duplex and require switchings between transmission and reception mode. Thus, collision detection, which is, for instance, applied by IEEE 802.3 Ethernet [Ins12b], is not possible in wireless systems. Instead, many communication protocols for wireless systems are based on CSMA/CA, i.e., they apply medium access methods with preventive avoidance of collision. With such methods, each station with data to send listens to the medium before transmitting and starts sending only

[1] In the model used by Sobrinho and Krishnakmuar [SK99] and shown in Fig. 2.1, communication links are directed but always bidirectional, whereas interference and sensing links are undirected.

if the medium is idle. Otherwise, the transmission is deferred until the medium is observed idle again and a random backoff time has been waited. CSMA/CA is, for instance, adopted in IEEE 802.11 [Ins12a], in IEEE 802.15.4's non-beacon mode, and in the contention access period of IEEE 802.15.4's beacon-enabled mode [Ins11]. Since CSMA/CA is a probabilistic approach, it cannot guarantee the absence of destructive collisions and frame loss.

One reason of collisions is the so-called *hidden station problem*, which exists if two nodes share a common neighbor w.r.t. communication links but are not connected by a sensing link. In this situation, CSMA/CA does not work, since a node cannot detect transmissions of the hidden station. To solve this problem, a frequently referenced solution are Request-To-Send / Clear-To-Send (RTS/CTS) handshakes before transmissions [Kar90]. An RTS/CTS handshake informs all neighbors of the sender (via RTS frame) and the receiver (via CTS frame) about the upcoming data transmission and is, for instance, adopted in IEEE 802.11 [Ins12a]. However, in a hidden station situation, RTS/CTS handshakes require a communication link between the receiver and the hidden station to receive the CTS frame, which is not given if they are connected by an interference link only. Many protocols entirely ignore the hidden station problem. Instead, they assume that there are no hidden stations in the network, which is true if the sum of communication and interference range is always smaller than the sensing range [BPC+07]. Another problem that is often mentioned in association with hidden stations is the problem of *exposed stations*. This problem occurs if two stations are in sensing range of each other and one of these stations defers its transmission due to an ongoing transmission of the other node. If transmissions are destined to different receivers that are not in interference range of the respective other sender, the transmission would not cause a destructive collision and is detained unnecessarily. Because the exposed station problem degrades performance but does not provoke collisions, it is less critical than the hidden station problem.

A further reason of collisions are the limitations of wireless transceivers. Among others, these include switching delays, which imply a blind period when changing between transmit and receive mode. Another limitation, which is particularly critical w.r.t. black bursts as discussed in Sect. 2.3.2, is the delay to perform Clear Channel Assessment (CCA) – i.e., the time that is required to detect the current medium state. CCA is a prerequisite for all CSMA/CA-based protocols but is sometimes also performed by TDMA-based protocols – like the standards ISA 100.11a [Int12a] and WirelessHART [Int10] – to detect interfering networks [PC11]. Switching and CCA delays are usually very high with current WSN transceivers and increase collision probabilities in such networks even without hidden stations [KSM08].

Compared to medium access with reserved time slots, which is realized by Guaranteed Time Slots (GTS) in IEEE 802.15.4 [Ins11], WirelessHART [Int10], and ISA 100.11a [Int12a], CSMA/CA-based protocols cannot provide guarantees regarding delays or reliability of communication. Nevertheless, CSMA/CA has other advantages like less coordination overhead and no necessity for synchronization. Furthermore, TDMA-based protocols have, in general, higher (but bounded) average delays and tend to under-utilize the medium if the concrete traffic volume is not known a priori. In addition, medium schedules with TDMA are usually fixed or costly to change and do not support value-based arbitration. With CSMA/CA-based protocols, value-dependent arbitrations can be achieved by varying backoff intervals, yet such a solution cannot properly deal with multi-hop networks and hidden stations, and suffers from the lack of synchronization.

2.2 An Introduction to Black Bursts

In this section, *black bursts* are introduced as a communication primitive that builds the foundation for a deterministic value-based arbitration protocol (see Chapter 3). The introduction of black bursts is similar to their definition in [KdI07, Kuh09]. The first mentioning of the term was earlier but with a different definition [SK96]. In addition, this section discusses prerequisites of black bursts, which must hold for their reliable utilization.

2.2.1 Definition

Different from a regular MAC frame, a black burst does not contain modulated payload but is only a period of energy of pre-defined length. Though a black burst has no payload in the usual sense, it carries two important pieces of information: The *duration* of the caused medium occupancy and the *point in time* when the medium occupancy starts. Since both information remain available if several black bursts of the same length are transmitted (almost) simultaneously, black bursts enable a collision-resistant communication primitive for wireless networks. Usually, one black burst encodes exactly one bit, where a black burst with a duration of $d_{bb} > 0$ represents a binary 1 (dominant bit; called "long 1" in [Kuh09]). Contrariwise, "no transmission" realizes a binary 0 (recessive bit; called "long 0" in [Kuh09]). Since a station sending a recessive bit does not switch its transceiver to transmission mode, it can detect dominant bits of neighbored stations by observing the state of the channel.

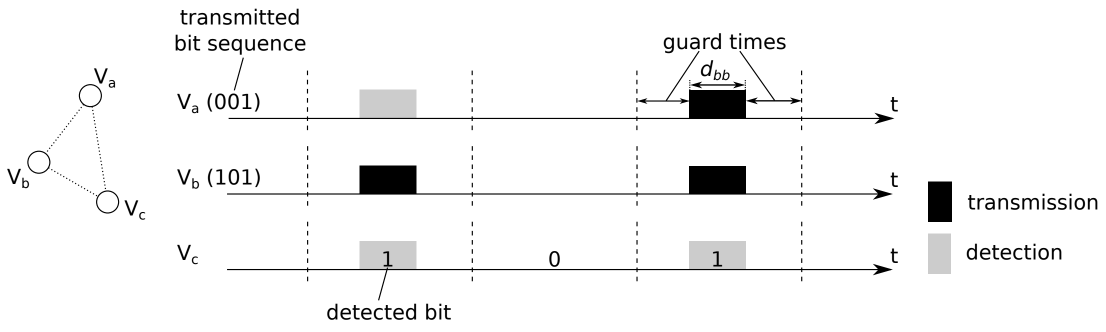


Figure 2.2: Overlapping black burst sequences with two senders (V_a and V_b).

Figure 2.2 illustrates the application of black bursts in a single-hop network with three stations. Note that the nodes are connected by sensing links only, since communication or interference links are not necessary for the transport of black bursts. In the example, bit sequences of length 3 are transmitted, where each bit duration consists of the time to transmit a dominant bit d_{bb} and additional guard times, which are necessary to clearly separate the bit times and to take transceiver characteristics like switching delays into account. Bit sequences are transmitted by two nodes (V_a and V_b) synchronously and encoded by black bursts. A third node (V_c) acts as receiver. Since V_a's bit sequence is 001_2, it actually performs only one black burst transmission during the last bit time. Station V_b sends two dominant bits and stays in receive mode for one bit time. Station V_c does not send any black burst but observes the medium the entire time. Thereby, it is able to decode medium occupancies as dominant bits and to detect recessive bits by the absence of medium occupancy. The bit sequence that is observed by V_c is the bit-wise

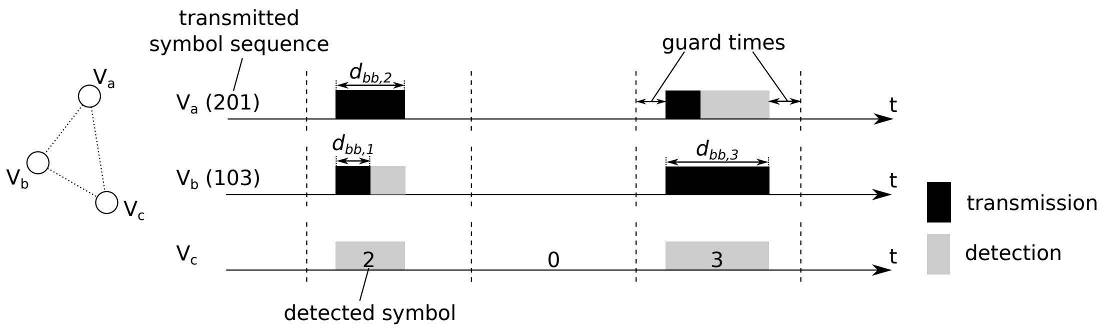


Figure 2.3: Realization of 4-ary symbols with black bursts of several length.

OR operation on both sent sequences. Since stations sending recessive bits are in receive mode, also senders are aware of this bit sequence. Similar to calculating the bit-wise *OR*, a bit-wise *AND* can be achieved by realizing a binary 0 dominantly by a black burst transmission and a 1 recessively by no transmission.

By introducing black bursts with several lengths, black burst encoding can be extended to allow symbols with more than 1 bit.[2] An example with a 4-ary encoding – i.e., with two bits per symbol – is presented in Fig. 2.3. Though the transmission of symbols starts again simultaneously, they may now differ in their length and are encoded such that longer black bursts represent higher numbers. Consequently, a 0 is again realized by "no transmission". Different from the binary encoding, the symbol sequence on the medium is now the result of a symbol-wise *MAX* operation. W.r.t. the duration of black bursts, two constraints have to be considered to enable the utilization of n-ary encodings: First, the difference of durations of different types of black bursts must be large enough to enable receiving stations to clearly distinguish different symbols and to decode their value correctly. For this, inaccuracy of synchronization (see Sect. 2.2.3) must be taken into account additionally. Second, the durations have also to be chosen such that sending stations are able to switch to receive mode after the end of their transmission and to detect transmissions of larger black bursts. Otherwise, sending stations could not determine the symbol sequence on the medium correctly. Depending on the used transceiver, the additional times implied by these constraints can be very large. Thus, it is not possible to declare that n-ary encodings ($n > 2$) are more efficient than a binary encoding without further knowledge about synchronization and hardware.

If not specified differently in the following sections, black bursts are applied with binary encoding, where 1s are encoded dominantly.

2.2.2 Applying Black Bursts in Multi-hop Networks

In multi-hop networks, nodes are only partially in sensing range of each other and further steps become necessary to propagate black bursts in the whole network. One solution is the subdivision of symbol times into several slots, where nodes transmit a symbol of their sequence in the first slot and use subsequent slots to forward received symbols. This mode of operation is illustrated in Fig. 2.4 with a topology with two hop diameter. While the sent bit sequences

[2]Black bursts with several lengths conform more to their original definition in [SK96, SK99]. However, they are originally not used to encode symbols or to build sequences of symbols but for medium arbitration only.

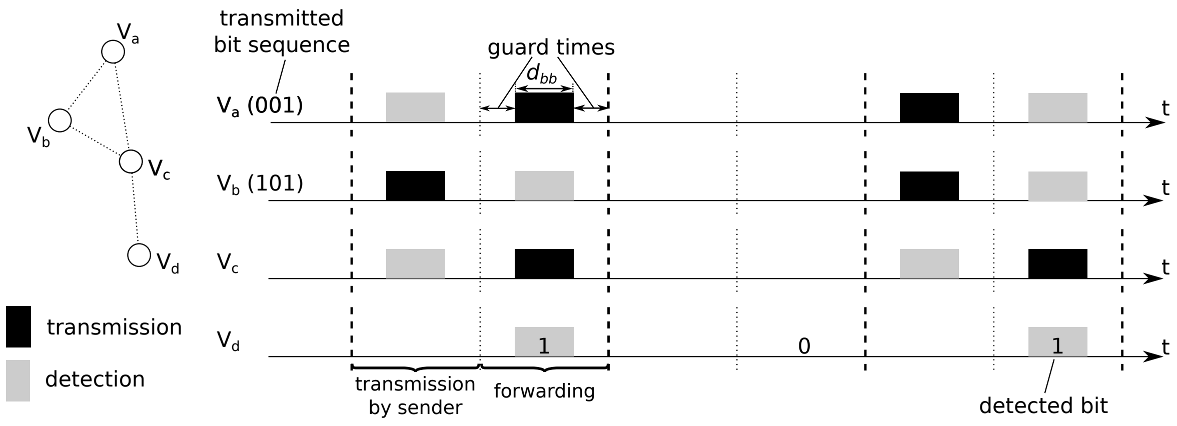


Figure 2.4: Network-wide propagation of black bursts in a multi-hop network.

are the same as in the single-hop example of Fig. 2.2, one bit time consists of two transmission slots now. In the first slot of the first bit time, station V_b transmits its dominant bit, which is forwarded by all receiving stations in the second slot. Thereby, also V_d observes a dominant bit in the first bit time. Since there is no station sending a dominant bit in the second bit time, the medium remains idle. In the third bit time, V_a and V_b both send a dominant bit, which is forwarded by node V_c to inform V_d. By forwarding black bursts, all nodes receive the same bit sequence, which is again the result of an *OR* operation.

In [Kuh09], a different solution is presented to support the multi-hop propagation of dominant bits. Instead of forwarding black bursts in rounds, a node, which has not sent a dominant bit in the current bit time and observes the beginning of a medium occupancy, immediately switches into transmission mode to forward the bit. While this solution allows smaller bit times and is therefore more efficient, it has several drawbacks: First, black bursts encoding n-ary symbols cannot be applied, since nodes do not know the duration of an observed medium occupancy. For the same reason, nodes cannot check whether the duration of a medium occupancy corresponds to a valid black burst. Thereby, the duration cannot be consulted to filter interferences, which may become necessary in a noisy environment. Furthermore, the point in time, when a medium occupancy starts, is less expressive and does not allow any conclusion w.r.t. the distance in hops to the node, which sent the dominant bit first.

2.2.3 Prerequisites

To apply black bursts successfully, several demands on network and hardware must be met. A first requirement is regarding network stability and interferences, which have to be prohibited in order to avoid the reception of incomplete or adulterated symbol sequences. This requirement is summarized as *single-network property* [CGR12] and consists of two parts: First, there must be no active nodes in sensing range running a different MAC protocol on the same communication channel.[3] Similarly, other external sources of interference (e.g., strong engines), which could be detected as medium occupancy, have to be avoided. Second, all pairs of nodes must be connected via some path of communication links[4], which have to be reliable and stable

[3]Originally, the property concerns interference range only, which is, however, not sufficient for black bursts.
[4]Actually, this requirement is stronger than required, since black bursts do not contain modulated payload.

during the transmission of symbol sequences. At first, we go one step further and require all links to be symmetrical, but weaken this property in Sect. 3.6 in the context of ACTP.

Though the single-network property seems to be a strong limitation, it is stipulated by many wireless protocols implicitly. Network performance with CSMA/CA-based protocols, for instance, suffers enormously if external interference occupies the medium permanently [BGS07]. This has, for instance, been observed in [PRML06], where IEEE 802.11 [Ins12a] traffic had a serious impact on an IEEE 802.15.4 [IEE03] network.[5] To satisfy the single-network property, measures like topological control, spatial division, and well-planned channel allocations can be applied.

A second requirement for the successful application of black bursts concerns transceivers, which must provide an upper bound on CCA and switching delays. Ideally, these delays are constant, which usually holds for switching delays but is rarely given for CCA delays. CCA delays, however, often come along with an upper bound – even with customary hardware. From an efficiency perspective, all delays should be very small, because the smaller they are the smaller are a black burst's guard times and overall duration.

A further requirement is tick synchronization, which is needed to have a common starting point of black burst transmissions. To ensure the correct assignment of dominant bits to their position in the bit sequence, synchronization offset must be bounded deterministically. In a single-hop network and with bounded CCA delays, synchronization can be achieved on demand by prefixing each black burst-encoded bit sequence with a dominant Start-of-Frame bit (SoF) and by letting all nodes synchronize to the fastest node. In multi-hop networks, an advanced internal or external synchronization method is required. A more detailed discussion on this topic can be found in Sect. 4.1. In the following, we initially do not refer to a particular synchronization protocol but assume all nodes to be synchronized with a maximal offset of $d_{maxOffset}$. The fact that the actual offset is usually much smaller than $d_{maxOffset}$ is illustrated in Fig. 2.5, where three nodes (V_a, V_b, and V_c) are synchronized to a reference tick t_0. The offset's impact on the detection of black bursts is investigated in Sect. 2.3.2 in more detail.

A last requirement for the utilization of black burst-based protocols is the knowledge about an upper network diameter bound. In multi-hop networks, this knowledge is required to guarantee the network-wide propagation of black bursts. While there is no need to know the exact diameter, it is in general desirable to have a very tight bound, since otherwise the efficiency suffers from unused forwarding slots.

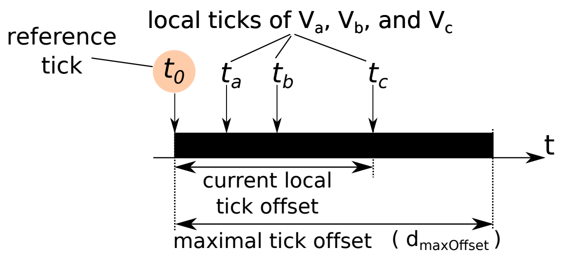

local ticks of V_a, V_b, and V_c
reference tick
t_0
t_a
t_b
t_c
t
current local tick offset
maximal tick offset ($d_{maxOffset}$)

Figure 2.5: Relation between $d_{maxOffset}$ and nodes' actual offset.

[5]Vice versa, IEEE 802.11 networks are only slightly affected (IEEE 802.11b) or not at all (IEEE 802.11g) from interfering IEEE 802.15.4 traffic.

2.3 Realization of Black Bursts

After the conceptual definition of black bursts in the last section, this section outlines aspects of their realization. In this regard, it regularly refers to Texas Instruments' (TI) CC 2420 transceiver [Tex07], which is very prevalent in WSNs and is also used for evaluations in Chapter 5. Details on its hardware features can be found in Appendix A.2. First, this section investigates the creation and transmission of black bursts. Afterwards, their reception is discussed.

2.3.1 Transmission of Black Bursts

The transmission of a black burst is always without preceding arbitration or clear channel assessment. If there is an offset due to inaccurate synchronization, it can, in particular, happen that a node starts a transmission, though the medium is already busy due to a black burst of another node. Before a node occupies the medium, it has to switch its transceiver to transmission mode. From that moment, the node is no longer able to monitor the medium's state. After entering transmission mode, the transceiver places a signal on the medium, thereby increasing the power level in the particular channel. The concrete character of the signal depends on the transceiver's options and implementation. With the CC 2420, alternatives are modulated IEEE 802.15.4 MAC frames and unmodulated signals, which are available in the test mode. Independent of the type of signal, the duration of the caused medium occupancy has to be long enough to enable a reliable detection (see Sect. 2.3.2). After the transmission is complete, the transceiver has to switch back to receive mode, which must be finished before the medium can be monitored again. Since switching between modes usually takes some time, the resulting delays have to be considered carefully. The IEEE 802.15.4 [Ins11] standard, for instance, prescribes switching delays of at most 12 symbol periods (192 μs in the 2.4 GHz band).

In this thesis, all black burst-based protocols are realized with TI's CC 2420 transceiver and modulated MAC frames. Further MAC layer features – like automatic acknowledgments – are not use, but configuration options are exhausted to decrease the duration of black bursts. The applied frame format is no longer IEEE 802.15.4-compliant and shown in Fig. 2.6. Compared to a standard-compliant frame, the preamble is shortened by 2 bytes and the checksum is omitted. The length of the payload is set to the minimal possible value (1 byte[6]). The resulting frames have a length of 5 bytes (10 symbols), which corresponds to a transmission duration of $d_{bb} = 160\,\mu s$. W.r.t. switching delays, the delay to switch from transmission to reception mode is given with 192 μs in the data sheet [Tex07], but has been empirically determined with 128 μs [Eng13, ECG14]. The opposite direction can be configured with either 192 μs or 128 μs.

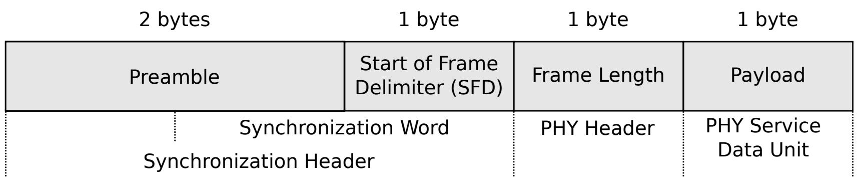


Figure 2.6: Irregular frame format to realize black bursts with TI's CC 2420 transceiver [Tex07].

[6]The CC 2420 can also be configured to send frames with 0 bytes payload. Since we discovered that the transceiver behaves faulty with this configuration (see [Eng13] and Chapter 5), we stay with 1 byte payload.

2.3.2 Detection of Black Bursts

Since black bursts do not carry modulated payload, their reception is only via the transceiver's CCA mechanism. Because CCA is prescribed by IEEE 802.15.4 [Ins11], it is included in all standard-compliant transceivers. With TI's CC 2420 transceiver, the current CCA status is available at an output pin. On the Imote 2 platform, this pin is connected to a GPIO pin of the microcontroller, which enables notifications by hardware interrupts when the CCA status changes. In the following, we first identify classes of errors that can occur w.r.t. the detection of black bursts. Afterwards, CCA variants of IEEE 802.15.4-compliant transceivers are investigated in detail. Thereafter, the impact of CCA delays is discussed.

2.3.2.1 Types of Errors

In general, two types of errors can occur with black burst-based communication:

- A *false positive* occurs, if a dominant bit is detected but no station in sensing range is sending a black burst. This leads to a wrong binary 1 in the received bit sequence.
- A *false negative* represents the opposite error, i.e., if a sent black burst is lost. As result, the detected bit sequence has a wrong 0.

Both error classes are independent of the concrete CCA implementation. In multi-hop networks, these errors do not only have an impact on the monitored bit sequence of a single node but are possibly propagated into further parts of the network.

If all prerequisites like the single-network property are met, neither false positives nor false negatives happen. In noisy environments like in production plants and without an adequate setup and node placement, the single-network property does potentially not hold. To detect and correct errors in such environments, a possible countermeasure is *Forward Error Correction* (FEC). FEC is based on increasing redundancy and is, for instance, used by IEEE 802.15.1 (Bluetooth) [IEE05], where bits in frame headers are repeated three times. This topic is further discussed in Sect. 3.6.2 in the context of ACTP.

2.3.2.2 Types of CCA

CCA is a measure to monitor the state of the medium and is located in the physical layer of the protocol stack. Since it is crucial for all CSMA-based MAC protocols, it is supported by almost all wireless transceivers and, in particular, stipulated by prevalent standards like IEEE 802.11 [Ins12a] and IEEE 802.15.4 [Ins11]. For the realization of black bursts, it is essential to understand how CCA is implemented in the deployed transceiver and which configuration options exist. The following paragraphs provide a survey of its implementation in TI's CC 2420 transceiver [Tex07], which is in most instances prescribed by the IEEE 802.15.4 standard [Ins11].

The maximal time to perform CCA and to detect the current medium state is predetermined by the standard with eight symbol periods, which corresponds to 128 µs in the 2.4 GHz PHY layer of IEEE 802.15.4 that is realized by the CC 2420 transceiver. For this PHY layer, the standard describes three CCA modes: Energy detection, carrier sense, and a combination of both.

With energy detection, the transceiver reports on a busy medium if the energy level on the medium exceeds a configurable threshold, which has to be set to a larger value than the noise floor. Energy detection is the simplest measure to detect a busy medium. Since it is non-coherent with signal characteristics like modulation and spreading technique, it is error-prone in environments with much interference and noise. Due to this demand on large Signal-to-Noise Ratios (SNRs), energy detection is more applicable in narrow band systems than in wide-band systems [RR06].

The second CCA mode (carrier sense) requires the detection of valid IEEE 802.15.4 carriers and does not consider the medium's energy level. With this mode, the CCA mechanism reports on a busy medium only if a signal with valid modulation and spreading characteristics is detected. The third CCA mode is a combination of the first two modes, i.e., the medium is reported busy if the signal characteristics coincide with a valid IEEE 802.15.4 signal AND/OR[7] the energy level exceeds the threshold. While the detection of valid IEEE 802.15.4 carriers works well if only one station is sending, it suffers from collisions and invalid chip sequences if there are multiple senders [Kuh09]. This is also confirmed in [KdI07], where the number of correctly received black bursts decreases with increasing number of senders. Thus, using energy detection to detect black bursts is the better choice. Therefore, it is also assumed in the rest of this thesis and described with greater detail below.

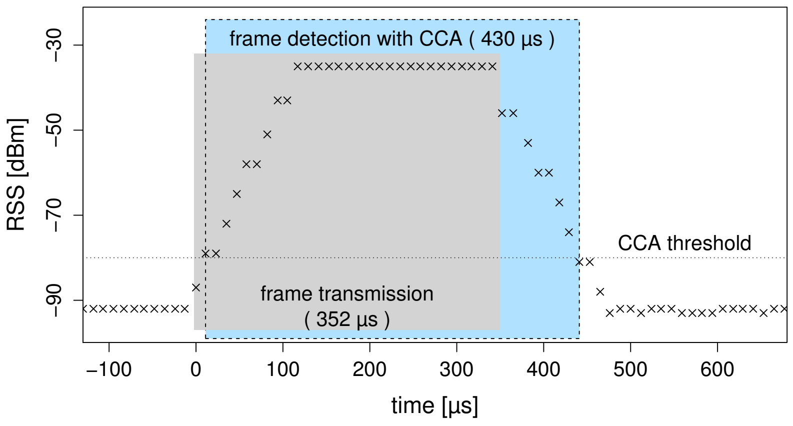
In TI's CC 2420 transceiver, energy detection works on Received Signal Strength Indicators (RSSIs), which can be converted into Received Signal Strengths (RSSs in [dBm]) by adding a constant offset. The RSSI is averaged over eight symbol periods (128 µs) and updated for each symbol (every 16 µs). If the RSSI exceeds a configurable threshold, the channel is indicated as busy. The transceiver furthermore supports hysteresis to avoid reporting on flapping medium states. While RSSIs were very imprecise and deficient with older hardware, experiments in [SDTL06] demonstrate that CC 2420's RSSI implementation provides a very good estimator for the Packet Reception Ratio (PRR). They also show that RSSI values are very similar on both sides of a link and that they have very small variations over time. Similar results regarding link reciprocity and stability could be obtained in our experiments [CM10, CMS10]; even for links without line-of-sight. All these properties are beneficial for a reliable detection of the current medium state and, thus, for the implementation of black bursts.

Figure 2.7 illustrates CC 2420's mode of operation w.r.t. RSSI calculation and CCA. The figure shows the course of the RSSs of two MAC frame transmissions with 11 bytes length (352 µs), which took place in the same environment and with the same node placement. During both transmissions, the noise level was about -93 dBm. The transmission power was reduced from 0 dBm in Fig. 2.7(a) to -25 dBm in Fig. 2.7(b), thereby reducing the SNR from 58 dB to 20 dB. Though both figures suffer from measuring inaccuracy, since RSSIs could only be monitored with intervals of about 11 µs[8], they provide some insights into CC 2420's RSSI calculation: First, the actual RSS of a frame is not discovered immediately after detecting the first symbol of the transmission but seven symbol periods (112 µs) later.[9] This delay occurs due to the averaging of the RSSI over eight symbols. Similar considerations hold for the end of the transmission. Second, the RSS does neither jump directly from the noise floor level to the actual RSS of the frame (or vice versa) nor does it increase (decrease) continuously between both extrema. Instead, the increase (decrease) is step-wise, where the step width corresponds to the RSSI update inter-

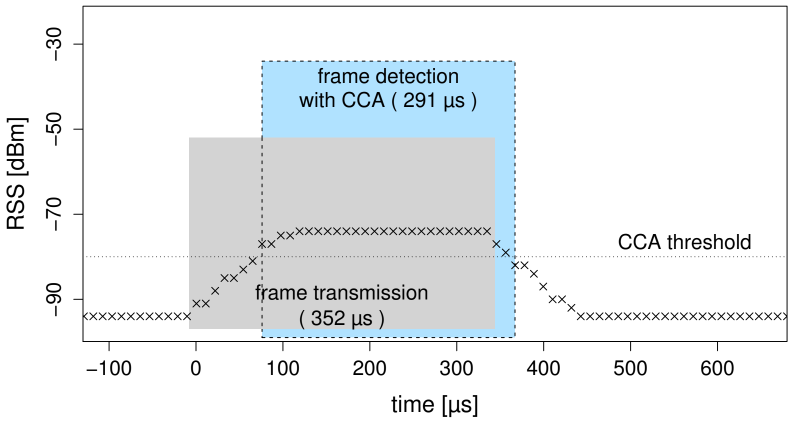
[7] The standard allows both AND and OR combinations of both conditions.

[8] The bottleneck in determining the current RSSI value is not the microcontroller but the interconnection to the transceiver via SPI bus.

[9] Due to the measuring inaccuracy, the delay can be up to 123 µs in the plot.



(a) Transmission power: 0 dBm.



(b) Transmission power: -25 dBm.

Figure 2.7: Illustration of CC 2420's CCA mechanism for two transmission power levels. For illustrative purpose, the CCA threshold is given in dBm.

val. The plots furthermore illustrate that the signal strength has a large impact on the CCA delay. With -25 dBm transmission power (Fig. 2.7(b)), CCA indicates the medium occupancy very late, because the RSSI exceeds the CCA threshold not until the 6th symbol of the transmitted frame is received. In the opposite way, the end of transmission is detected with a large delay when using 0 dBm transmission power (Fig. 2.7(a)). These delays have to be considered in implementations of black bursts and are investigated in the following sections in more detail.

2.3.2.3 Impact of the CCA Delay on the Detected Duration of Single Black Bursts

As shown in the previous section, the delay to perform CCA can add up to $d_{maxCCA} = 128\,\mu s$ with TI's CC 2420 transceiver. In this section, the impact of this delay on the observed duration of valid black burst transmissions is discussed.

In Fig. 2.8, the transmission of a dominant bit is shown, where the duration of its detection is increased maximally. This case occurs if the signal is very strong, since this provokes that the start of the transmission is detected immediately – i.e., the first received symbol is sufficient such that the RSSI value exceeds the CCA threshold – and that the end of the transmission is

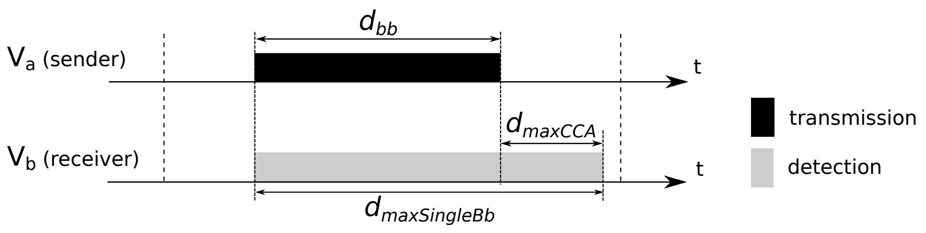


Figure 2.8: Maximal detected duration of a single black burst.

perceived with maximal delay – i.e., if eight symbols are required after the end of the transmission to let the RSSI fall below the CCA threshold. Formally, the maximal detected duration of a single black burst calculates to:

$$d_{maxSingleBb} = d_{bb} + d_{maxCCA} \tag{2.1}$$

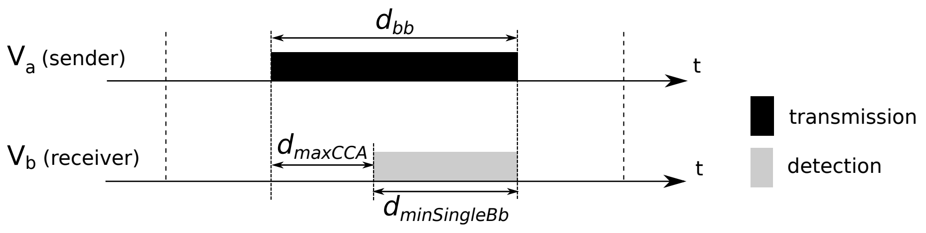


Figure 2.9: Minimal detected duration of a single black burst.

The impact of weak signals on black burst detection is illustrated in Fig. 2.9. Here, the RSSI exceeds the threshold not until the eighth symbol of the transmission but the end of the transmission is already detected with the first symbol thereafter. In summary, the minimal possible duration of a detected single black burst yields:

$$d_{minSingleBb} = d_{bb} - d_{maxCCA} \tag{2.2}$$

This equation shows that the duration of a black burst d_{bb} must not be smaller than the maximal CCA delay d_{maxCCA}, since otherwise the receiver may possibly not see any medium occupancy and miss the black burst.

2.3.2.4 Impact of Synchronization Inaccuracy on the Expected Detection Time of Black Bursts

To communicate with black bursts reliably, it is crucial that a node associates a detected black burst with the same transmission slot as the sender of the black burst. But due to synchronization inaccuracy, there is, in general, a discrepancy between the start time of the transmission at the sender and the time, at which the receiver expects the sender to start. As a consequence, the detection of a black burst is shifted from the local point of view of the receiver. In order to still enable the correct assignment of a detected black burst to the sender's transmission time, this discrepancy must be quantified. The following investigates this issue in more detail.

Since black bursts require (tick) synchronization with bounded offset (see Sect. 2.2.3), the maximal synchronization inaccuracy is known at all network nodes. However, the order, in

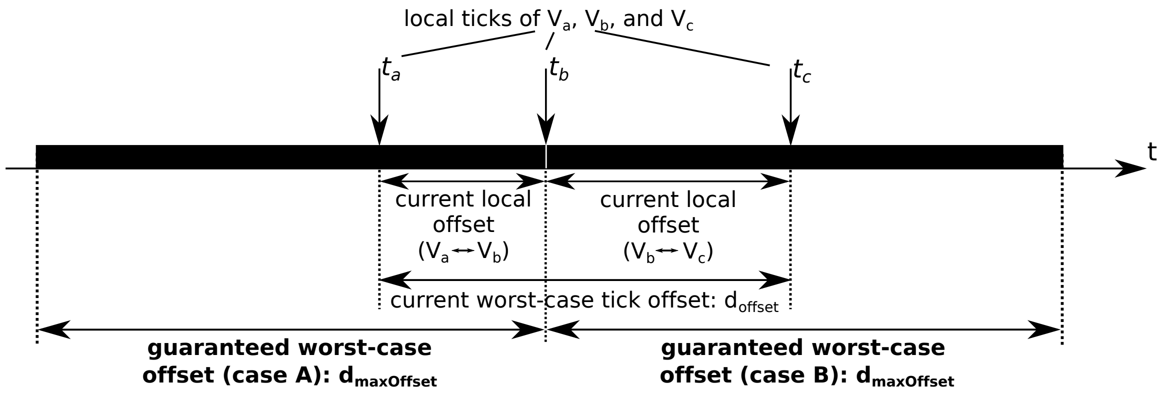

local ticks of V_a, V_b, and V_c
t_a
t_b
t_c
t
current local offset (V_a↔V_b)
current local offset (V_b↔V_c)
current worst-case tick offset: d_{offset}
guaranteed worst-case offset (case A): $d_{maxOffset}$
guaranteed worst-case offset (case B): $d_{maxOffset}$

Figure 2.10: Synchronization inaccuracy from the local point of view of node V_b.

which nodes perceive their local ticks, is unknown, which has to be considered when implementing black bursts. Figure 2.10 illustrates this issue with three nodes. Due to inaccurate synchronization, all nodes perceive their local ticks at different global points in time and have therefore local offsets to each other. These local offsets are in general variable and shown from node V_b's point of view. Due to the lack of global knowledge, node V_b neither knows its current local offsets to V_a and V_c nor the network's current worst-case offset d_{offset}. Instead, it must rely on the offset bound $d_{maxOffset}$, which is provided by the synchronization protocol. In this regard, two cases must be distinguished, which are depicted in the figure with case A and case B: In case A, V_b would perceive its tick after all other nodes, whereas in case B, V_b's tick would be located before all other local ticks. Though both cases cannot occur simultaneously, they must be considered together, since the current situation is unknown without global knowledge and usually somewhere in between – as in the example.

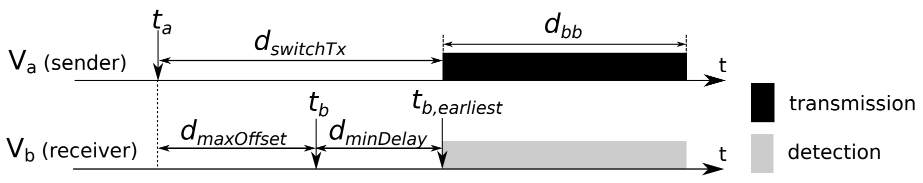


Figure 2.11: Premature detection of a black burst due to synchronization offset.

In Fig. 2.11, it is assumed that V_b perceives its tick $d_{maxOffset}$ after V_a, which initiates the transmission of a black burst at t_a. At $t_a + d_{switchTx}$, where $d_{switchTx}$ is the switching delay from reception to transmission mode, V_a starts to occupy the medium. Due to its offset, V_b does not detect the black burst at $t_b + d_{switchTx}$, when it would expect the start of a possible black burst, but at $t_{b,earliest} = t_b + d_{minDelay}$, where $d_{minDelay}$ is shorter than $d_{switchTx}$ and calculated as follows:

$$d_{minDelay} = d_{switchTx} - d_{maxOffset} \tag{2.3}$$

Note that $d_{minDelay}$ becomes negative if the maximal offset is larger than the switching delay. In this case, V_b could observe the black burst before it would expect V_a to initiate the transmission. Nevertheless, assigning the black burst to the correct transmission time must also be possible and non-ambiguous in this case.

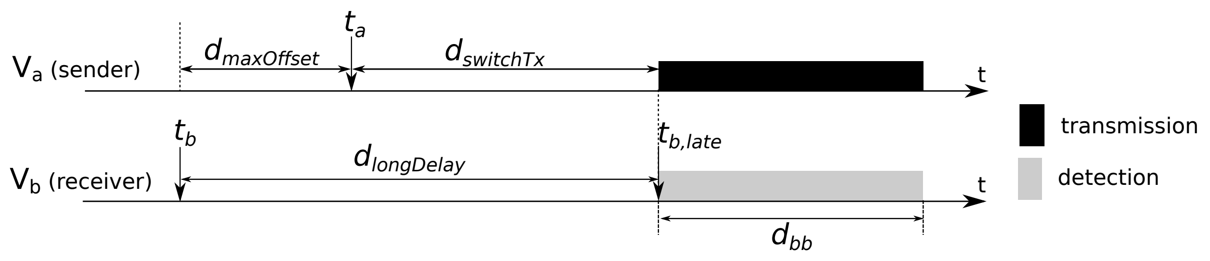


Figure 2.12: Late detection of a black burst due to synchronization offset.

Figure 2.12 illustrates the opposite case, in which V_b perceives its tick $d_{maxOffset}$ before V_a. Here, the black burst is detected by V_b later than expected at $t_{b,late} = t_b + d_{longDelay}$, with:

$$d_{longDelay} = d_{switchTx} + d_{maxOffset} \tag{2.4}$$

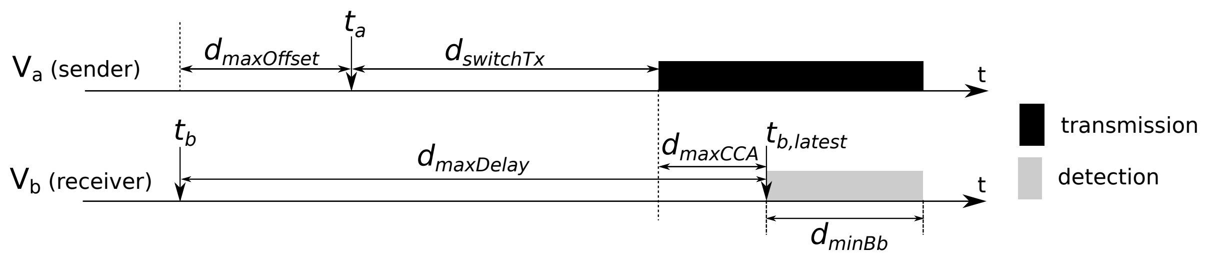


Figure 2.13: Latest detection of a black burst due to synchronization offset and CCA delay.

From node V_b's perspective, $d_{longDelay}$ is, however, not the largest possible detection delay of a valid black burst transmission, but the delay can additionally be increased by the CCA delay as shown in Fig. 2.13. The resulting delay $d_{maxDelay}$ is calculated as follows:

$$d_{maxDelay} = d_{switchTx} + d_{maxOffset} + d_{maxCCA} \tag{2.5}$$

Since in all cases, the black burst is sent at a valid point in time, V_b must record the reception of a dominant bit. Formally, it has to associate the detected black burst with $t_b + d_{switchTx}$, which is the time at which V_b would expect the start of a black burst detection, if the detection by CCA starts within the following interval:

$$[t_{b,earliest} \quad , \quad t_{b,latest}] \tag{2.6}$$

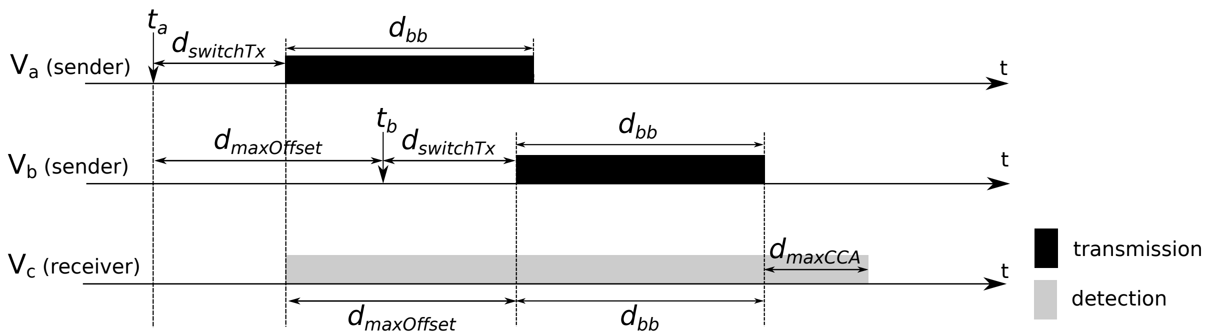
$$[t_b + d_{switchTx} - d_{maxOffset} \quad , \quad t_b + d_{switchTx} + d_{maxOffset} + d_{maxCCA}] \tag{2.7}$$

This shows again the importance of bounded values for $d_{switchTx}$, $d_{maxOffset}$, and d_{maxCCA}.

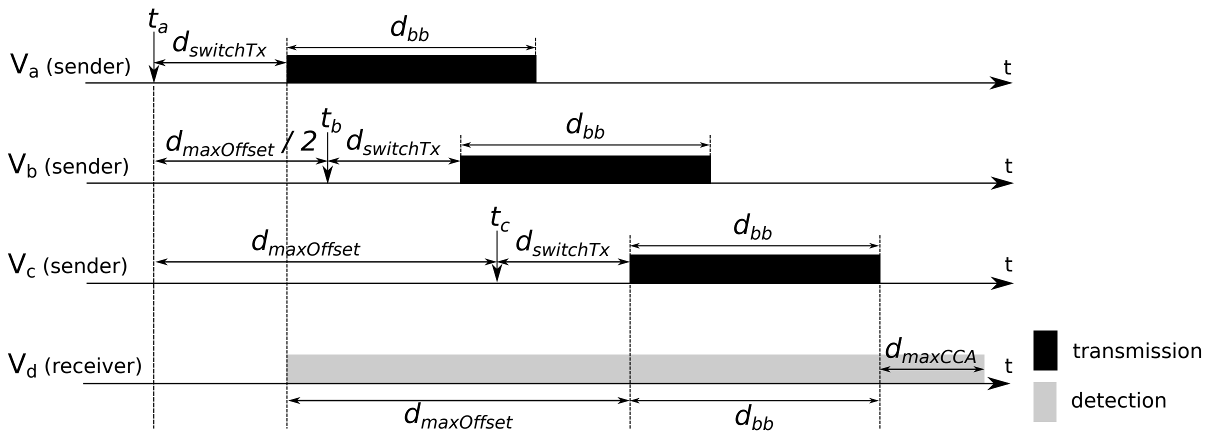
2.3.2.5 Impact of Synchronization Inaccuracy on the Duration of Overlapping Black Bursts

If several nodes transmit overlapping black bursts, synchronization offset can also cause the detection of an elongated medium occupancy. Resulting consequences are now investigated.

Figure 2.14 illustrates the maximal elongation of a black burst-caused medium occupancy by two examples: In Fig. 2.14(a), senders V_a and V_b are synchronized with maximal offset $d_{maxOffset}$. Since $d_{maxOffset}$ is smaller than the duration of a black burst transmission d_{bb}, the two black



(a) Example with $d_{maxOffset} < d_{bb}$ and two senders with synchronization offset of $d_{maxOffset}$.



(b) Example with $d_{maxOffset} > d_{bb}$ and three senders.

Figure 2.14: Elongation of medium occupancy due to overlapping black burst transmissions.

bursts overlap partially and prolong the medium occupancy by $d_{maxOffset}$. A similar situation is shown in Fig. 2.14(b), where three nodes send black bursts simultaneously but with local offsets. Though this example assumes that $d_{maxOffset}$ is larger than d_{bb}, the three transmissions are shifted such that there is one large medium occupancy. Consequently, the medium occupancy is prolonged by $d_{maxOffset}$ as well. If furthermore CCA delay is taken into account, the duration of the detected medium occupancy can additionally be increased by d_{maxCCA} as shown for both situations. Thus, the maximal possible detection duration calculates as follows:

$$d_{maxBb} = d_{maxOffset} + d_{bb} + d_{maxCCA} \tag{2.8}$$

In addition to prolonging the medium occupancy, a large offset can also cause the detection of separate black bursts. This case is shown in Fig. 2.15 and requires special attention, since the receiver V_c must store the reception of one dominant bit only, yet it actually detects two black bursts.

The maximal possible detection duration can be used to filter invalid medium occupancies and to avoid false positives. Similarly, the minimal possible duration can be consulted to filter too short medium occupancies. Because synchronization offset does not shorten a medium occupancy, this duration has already been given in Eq. 2.1:

$$d_{minBb} = d_{minSingleBb} = d_{bb} - d_{maxCCA} \tag{2.9}$$

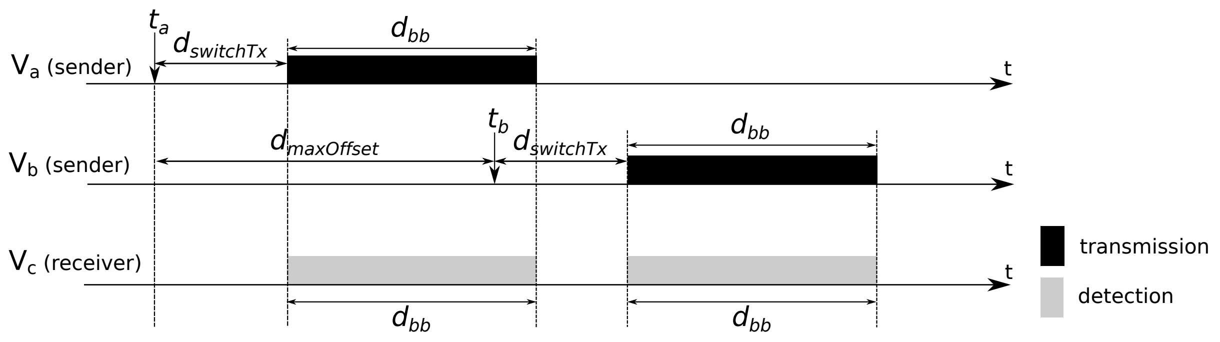


Figure 2.15: Detection of two consecutive black bursts due to very large synchronization offset.

In summary, a node accepts a medium occupancy as dominant bit if its duration is within following interval:

$$[d_{minBb} \quad , \quad d_{maxBb}] \tag{2.10}$$

$$[d_{bb} - d_{maxCCA} \quad , \quad d_{maxOffset} + d_{bb} + d_{maxCCA}] \tag{2.11}$$

2.4 Improvement of Black Burst Implementations

Since communication with black bursts is different from communication with regular frames, wireless transceivers are not optimized for their application. However, transceivers often provide configuration options, which can improve black burst implementations w.r.t. efficiency and reliability. In the following, such options are discussed for TI's CC 2420 transceiver [Tex07].

2.4.1 Unique Synchronization Words

In our implementation, a black burst is realized as regular MAC frame (see Sect. 2.3.1). Thus, it is possible that a black burst is demodulated as regular MAC frame if the receiving node can detect the correct chip sequence of the frame's preamble and SFD byte.[10] But since black bursts contain no regular payload, there is actually no need to receive a black burst as MAC frame and, thus, it is conceivable to prevent their reception as MAC frame.

Looking into the IEEE 802.15.4 standard [Ins11] reveals that the reception of a black burst as MAC frame is not only unnecessary but has to be avoided at any price, because according to the standard, a reception is considered to be in progress until the number of octets specified in a frame's length field has been received. During this "reception time", all CCA modes must report a busy medium; even if the transmission is no longer in progress and/or the medium's energy level is below the CCA threshold. This behavior is indeed correct if the length field of the received frame is valid, but if the length field is corrupted, which is commonly the case for overlapping transmissions, it will cause the adulteration of the perceived medium occupancy. This, in turn, can lead to false negatives and threatens the reliability of black burst detections. Corresponding implications have actually been observed in previous experiments

[10]Because the preamble of the frame that is used as black burst is shortened and shorter preambles reduce the probability for a successful synchronization between sender and receiver, the probability to receive a black burst as MAC frame is already reduced but still existent without further measures.

[Eng13], where valid black bursts were sent but the CCA mechanism of the CC 2420 transceiver indicated medium occupancies with far too long durations.

To prevent the reception of black bursts as MAC frames, we configure each CC 2420 transceiver with a unique synchronization word (see frame format in Fig. 2.6). The synchronization word has a length of four symbols [Tex07] and defines the last part of the preamble and the content of the SFD byte. It is used by the transceiver to search for valid frames on the medium and to distinguish them from noise and interference. By setting unique synchronization words on each node, nodes will no longer receive black bursts as MAC frames. Thus, the CCA mechanism is no longer confused by corrupted length fields but based on the medium's energy level only. To re-enable communication with regular frames after the application of black bursts, the default synchronization word has to be restored. As further measure to avoid the reception of black bursts as MAC frames, the demodulator correlation threshold, which a chip sequence must exceed before the transceiver searches for the SFD, can be set to maximum [Eng13]. Thereby, the probability of undesired frame receptions is additionally reduced.

2.4.2 Calibration of the CCA Threshold

The calibration of the transceiver's CCA threshold is a crucial step w.r.t. the reliability of black burst-based communication. If the threshold is set too high, the delay to detect the beginning of a black burst and the probability of false negatives increase. If, on the contrary, the threshold is too low, noise can be misinterpreted as black burst and false positives can arise. Consequently, the CCA threshold should be configured as a function of noise floor and expected signal strengths. In this regard, it has to be considered that noise and the RSSs of transmissions are usually environment- and channel-specific [CM10, CMS10].

The range of black bursts is in general independent of the range of regular MAC frames, since black bursts are detected by their RSS and the CCA threshold, whereas regular MAC frames require the correct detection of symbols. Thus, it is possible that black bursts can be sent over larger, equal, or smaller distances than MAC frames. However, a usual assumption is that sensing ranges are larger than communication ranges, which is, for instance, confirmed in [KNE03] by disproving the mistaken axiom *"If I can hear you at all, I can hear you perfectly"*. Hence, it is reasonable to assume that the range of black bursts (sensing range) is larger than the range of MAC frames (communication range), yet inspecting the CCA threshold is required to confirm this statement.

The option to calibrate the CCA threshold renders possible to configure the CCA mechanism such that communication and sensing ranges become almost equal. For this purpose, the threshold must be set to a signal strength, at which regular MAC frames are just received correctly. This demands a high correlation between RSSs and frame reception rates, which seems to be true for modern wireless transceivers like TI's CC 2420 as supported by experiments in [SL06, RCM$^+$06]. W.r.t. the approximation of both ranges, it must, however, be kept in mind that overlapping black burst transmissions interfere constructively[11]. Thus, two overlapping black bursts may reach a node, which is not reached by a single transmitter.

[11]Destructive interference is prevented by design due to different synchronization words. For a proof by experiments, see Chapter 5.

2.4.3 Back Calculation of the Transmission Start of Black Bursts

To assign dominant bits to their correct position in the bit sequence and to accurately determine the duration of a medium occupancy, it is crucial to identify the start of transmissions with high precision. This time is, however, distorted by synchronization offset and CCA delay. While we assume that the offset must be condoned, the CCA mechanism's mode of operation offers opportunities for optimization, which are investigated in the following.

According to the data sheet of the CC 2420 transceiver [Tex07] and as illustrated in Fig. 2.7, the CCA status is calculated by comparing the RSSI against a threshold. The RSSI is averaged over eight symbols and is formally calculated for some symbol time i as follows:

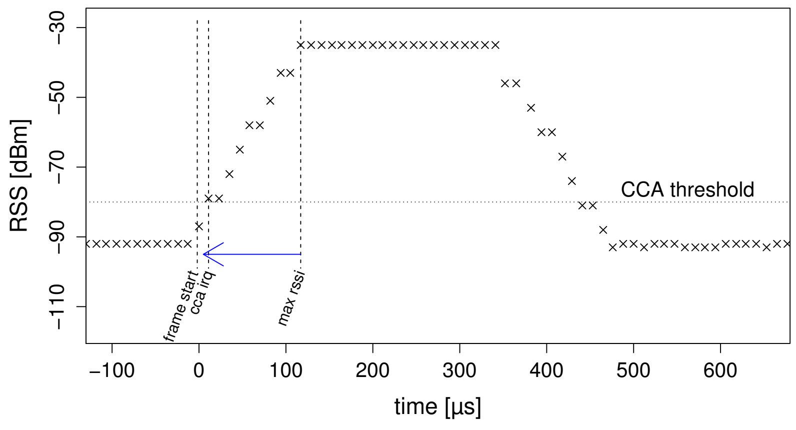
$$RSSI(i) = \Big(\sum_{j=i-7}^{i} symbol(j)\Big)/8, \tag{2.12}$$

where $symbol(j)$ is the signal strength indicator of the j-th symbol.

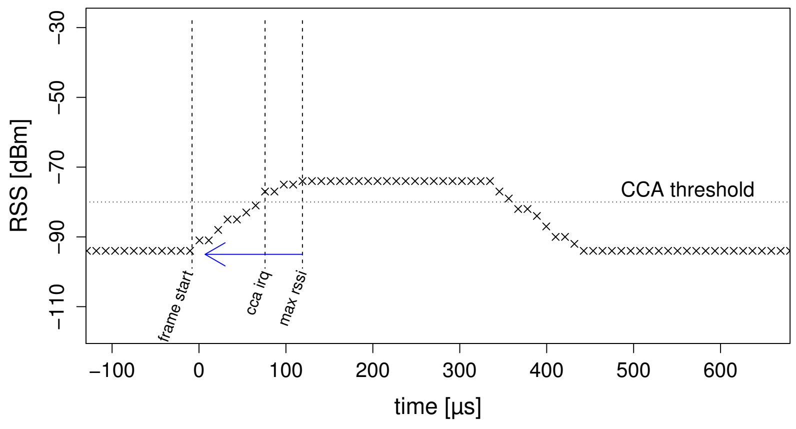
After a transmission starts and under the assumption that its signal strength is sufficiently larger than the noise floor, the RSSI value in the transceiver of a receiver increases with each new symbol. After eight symbols, the RSSI value reaches the maximum that can be converted to the transmission's actual RSS, which is assumed to be almost constant. Hence, a receiver can back calculate the start time of a transmission by monitoring the gradient of the RSSI value, by storing the timestamp when the value reaches its maximum first, and by subtracting seven symbol durations from this timestamp. By this approach, the start of a transmission can be derived more accurately, yet some factors – like the symbol duration's granularity of 16 µs – still remain as inaccuracy. As a further advantage, the detection accuracy becomes independent of the CCA threshold, which has a large impact on delays that occur until the CCA mechanism indicates a busy medium.

Figure 2.16 illustrates the back calculation of the start time of a transmission with the examples from Sect. 2.3.2. The time when the CCA mechanism reports on a busy medium is marked with *cca irq*; the time when the maximal RSSI value is observed with *max rssi*. The arrow starting from this mark indicates the subtraction step with seven symbol durations (-112 µs), where the arrowhead represents the back calculated start time. Consequently, the improvement compared to the CCA interrupt's timestamp corresponds to the distance between the *cca irq* line and arrowhead. In the example with 0 dBm transmission power, the back calculation improves the accuracy only slightly (6 µs more accurate detection), because the delay between transmission start and CCA interrupt is already very small due to the large signal strength. In the second example with -25 dBm transmission power, the improvement is more significant (69 µs), since the CCA interrupt occurs very late.

If there are more senders with overlapping but slightly shifted transmissions, the RSSI will probably not converge within eight symbols after the CCA interrupt. In this case, the back calculated timestamp is probably larger than the timestamp of the CCA interrupt and therefore more inaccurate. To guarantee that the optimization does not worsen the accuracy, the timestamp of the CCA interrupt has to be used in this case as fallback. In sum, the presented back calculation provides a measure to improve the average accuracy of black burst detection but cannot enhance the worst-case accuracy.



(a) Transmission power: 0 dBm.



(b) Transmission power: -25 dBm.

Figure 2.16: Back calculation of the start time of a transmission.

2.5 Discussion

This chapter has introduced a communication primitive that is based on the fact that not everything is lost in case of collisions. If, in particular, transmissions start simultaneously, useful information can be encoded in the start and duration of the caused medium occupancy. This fact is the foundation for black bursts, which are transmissions without modulated payload but with pre-defined starting times and duration. By encoding a dominant bit by the transmission of a black burst and a recessive bit by the absence of medium occupancy, simultaneous transmissions of black burst-encoded bit sequences yield a sequence on the medium that is the result of a bit-wise OR operation on all sent sequences. This principle can also be extended to n-ary symbols by encoding symbol values with black bursts of different length. Thereby, the results of symbol-wise MAX operations are observable on the medium. Since black bursts have no payload in the usual sense, their detection is via the CCA mechanism of the transceiver only. In this regard, energy detectors, which are included in all IEEE 802.15.4-compliant transceivers [Ins11], state an applicable CCA variant.

One key requirement of black bursts is the availability of synchronization with bounded offset. On the one hand, this requirement must be met to separate time periods, in which black burst transmissions are enabled, from time periods with transfer of regular data frames. On the other hand, black burst transmissions would not transport any information if receiving nodes could not associate the time of detection with a bit/symbol time. A further requirement for the application of black bursts is the validity of the single-network property, which is stated here explicitly but stipulated by almost all wireless protocols. It is advisably verified by experiments in the final target environment, since simulations and analyses are often not adequate to exactly reproduce environment-specific signal propagations. In this regard, we could, for instance, show that combinations of channels and node positions have a significant influence on the RSS of data frames [CM10, CMS10]. Thus, it is in general advisable to follow a parameterized protocol design, in which configuration options like transmission power and CCA threshold are not fixed but determined in accordance with the environment and its noise floor. Though the requirements appear to be very demanding, they pave the way for black bursts, collision-resistant encoding of bits and symbols, and deterministic protocols like ACTP, which is introduced in the next chapter. Depending on the concrete scenario, these advantages predominate and state a price that is worth paying.

Besides introducing the concept of black bursts, this chapter also provides a derivation of their duration, which is mainly dictated by characteristics of the wireless transceivers. In this regard, we refer to TI's popular IEEE 802.15.4-compliant CC 2420 transceiver, which also serves as evaluation platform of our implementation (see Chapters 3 and 5). So far, processing delays that are additionally required by the microcontroller are omitted, yet they must be considered in final implementations. Referring to this, [Eng13, ECG14] provides numerous information on relevant run-time costs.

Because black burst-based communication deviates from communication with regular data frames, current transceivers like TI's CC 2420 are consequently not optimized for it. Yet, they are sufficient to demonstrate the feasibility and advantages of black bursts. With the CC 2420 transceiver, the drawbacks of an off-the-shelf platform are observable by very large switching and CCA delays. However, it also offers configuration options, which can be exploited to optimize its application to black bursts. Some of them – like shortening the preamble of MAC frames – are surveyed in this chapter. By monitoring the gradient of the RSSI value, the CC 2420 transceiver furthermore provides a way to back calculate the start time of a transmission more accurately. Thereby, receivers can determine the transmission start of black bursts more accurately, which also helps to improve the average accuracy of other black burst-based protocols like the synchronization protocol Black Burst Synchronization (BBS, [GK11a, GK11b]).

3. Chapter

The Arbitrating and Cooperative Transfer Protocol (ACTP)

In the previous chapter, black bursts have been introduced as communication primitive for wireless networks. This chapter now proposes a deterministic communication protocol with predictable runtime, which adopts this communication primitive. The protocol is called *Arbitrating and Cooperative Transfer Protocol* (ACTP) and enables value-based arbitration and cooperative data transfer in dense wireless networks. It is fully distributed and an instance of a so-called binary countdown protocol, whose famous representative in the wired domain is CAN [Int04]. By presenting a binary countdown protocol for wireless networks with multi-hop and network-wide application range, this chapter refutes previous works [WMW05, MLK06], which negate the implementability of such protocols in the wireless domain. Compared to state-of-the-practice medium access schemes that are based on TDMA or FDMA, ACTP grants exclusive access right to the medium on demand by solving contest in a deterministic and value-based way. Thereby, ACTP closes a gap of existing wireless medium access schemes and represents a suitable solution w.r.t. reliable communication when no fixed message schedules are available.

The rest of this chapter is structured as follows: Section 3.1 motivates the benefits of a deterministic, value-based arbitration protocol. With ACTP, Sect. 3.2 introduces such a protocol, explains its mode of operation, and derives the protocol's timings. In Sect. 3.3, ACTP's network-wide application is presented. Sections 3.4 and 3.5 then explain ACTP's cooperative application mode and the protocol's application with radii that are smaller than the network diameter. Errors caused by situations, in which the single-network property is violated, and countermeasures are discussed in Sect. 3.6. Section 3.7 surveys an SDL specification of ACTP and two manual implementation versions. Finally, Sect. 3.8 provides a discussion and compares ACTP to state-of-the-practice medium access schemes.

Results of this chapter have been published in [8], [9], [12], [14], [16], [20], and [23].

3.1 Motivation

To achieve a desired quality-of-service in wireless networks, many protocols have been proposed avoiding simultaneous transmissions in the same communication channel and collision domain. For this purpose, they apply medium access schemes like TDMA, FDMA, SDMA (Spatial Division Multiple Access), or combinations of them.

With TDMA, transmissions are temporally serialized, requiring synchronization between communicating nodes and coordinated transmission schedules, which, in turn, demands exact knowledge about nodes' communication volumes and patterns. If these are unknown or if nodes create sporadic or even aperiodic network traffic, overbooking is required, resulting in a waste of network resources or disqualifying TDMA at all. FDMA, in turn, does not subdivide time into transmission slots but frequencies into communication channels. Similar to TDMA, FDMA requires coordination and synchronization among nodes to enable rendezvous on a particular communication channel, since wireless nodes – and, in particular, wireless sensor nodes – are usually equipped with a single transceiver only. Like TDMA and FDMA, SDMA's main effort incurs either offline by configuration or at runtime and includes adequate node placement, the installation of directional antennae, and the calibration of the transmission power of nodes. Unlike TDMA, FDMA and SDMA increase network throughput by allowing simultaneous transmissions, yet they make higher demands on hardware or manual effort.

In static networks with known communication demands, combinations of TDMA, FDMA, and SDMA can often provide adequate solutions with high quality-of-service. In this regard, it has, however, to be noted that concrete realizations often suffer from self-made limitations. IEEE 802.15.4 [Ins11], for instance, indeed applies TDMA in so-called Contention Free Periods (CFPs), but only allows seven communication slots (called *Guaranteed Time Slots*) per PAN (Personal Area Network) coordinator and superframe. Another example with TDMA, but limitations, is the industrial standard WirelessHART [Int10], which supports significantly more time slots but only with a fixed duration of 10 ms.

Since communication solutions with TDMA, FDMA, or SDMA are optimized for long-lasting message schedules, they are usually too inflexible if communication demands are unknown and variable or if the network is very dynamic. In such situations, access to the medium must often be granted dynamically and priority-based, which is not supported by any of them.

In wired systems, a well-known solution providing deterministic value-based arbitration is CAN [Int04], a wired single-hop communication bus, which is prevalent in the automotive domain. In CAN, contest among stations is solved by applying a binary countdown protocol during an arbitration phase that precedes the actual transmission of data. During the arbitration phase, stations start to transmit unique message identifiers simultaneously and bit-wise, where a 0 is realized by a dominant bus level and a 1 by a recessive one. The principle of CAN's arbitration is illustrated in Fig. 3.1 by an example with four bit identifiers and two competing stations. During the third bit time, station 1 sends a recessive 1, detects the dominant 0 of station 2, and stops competing. After the fourth bit, station 2 is aware of winning the contest, since it has sent its identifier completely, and all other nodes including station 1 know the winner's identifier (0100_2), since it has been observed on the medium.

While CAN looks back to a history of several decades and is installed in millions of cars, the idea of its binary countdown arbitration can hardly be found in wireless protocols. There are

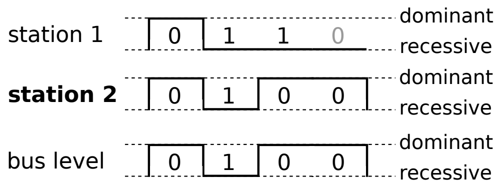


Figure 3.1: Medium arbitration by binary countdown in CAN [Int04].

even papers, which state that this kind of arbitration cannot be transferred to wireless systems. [WMW05, MLK06], for instance, conclude that CAN cannot be adopted in the wireless domain due to the lack of full-duplex transceivers and the hidden station problem. Similar statements ("no such arbitration is possible for wireless channels") can be found in [TNT07, NL09], where CAN is presented as wired solution for a control theoretical problem called *Try-Once-Discard* (see also Sect. 4.2.2). In the following, these statements are disproved by the presentation of ACTP, which implements CAN's arbitration approach for wireless multi-hop networks. Though ACTP is not the first binary countdown protocol for wireless networks, it outperforms previous protocols in terms of multi-hop support, overhead, and energy consumption. Detailed comparisons and a survey of related work can be found in Chapter 6.

3.2 ACTP – Protocol Description

The *Arbitrating and Cooperative Transfer Protocol* (ACTP) is a binary countdown protocol for wireless multi-hop networks. Since it incorporates black bursts to encode bits in a collision-protected way, it requires all nodes to be synchronized. In the following, it is assumed that all nodes are synchronized with a maximal offset of $d_{maxOffset}$. The installation of ACTP in a complete setup with integrated synchronization protocol will be treated in Chapter 4.

3.2.1 Mode of Operation

Transferring CAN to the wireless domain opens the question of how to deal with the hidden station problem. Yet, almost all wireless binary countdown protocols ignore this problem completely and forbid hidden stations by requiring the sensing range to be larger than communication plus interference range [BPC$^+$07]. In the following, we do not ignore hidden stations but present ACTP with a configurable application range of n_{hops} hops. This range will be called *arbitration radius* and depends on the protocol's application context. To provide deterministic medium arbitration despite hidden stations, an adequate value is, for instance, $n_{hops} = 2$.

During its execution, ACTP subdivides time into different time slots (see Fig. 3.2). An entire ACTP run with an arbitration radius of n_{hops} (= 4 in Fig. 3.2) and bit sequences, which are also called *arbitration sequences* in the context of ACTP, of length n_{bits} (= 3 in Fig. 3.2) is called *ACTP phase*. An ACTP phase, in turn, is subdivided into *bit phases*, where one bit phase includes the time to process a single bit and to propagate this bit over n_{hops} hops. Bit phases are again divided into bit rounds, comprising the time to process one bit along a single hop. For each ACTP run, all participating nodes must use identical values for n_{bits} and n_{hops}.

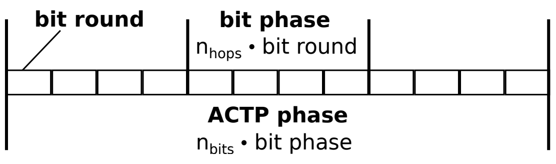
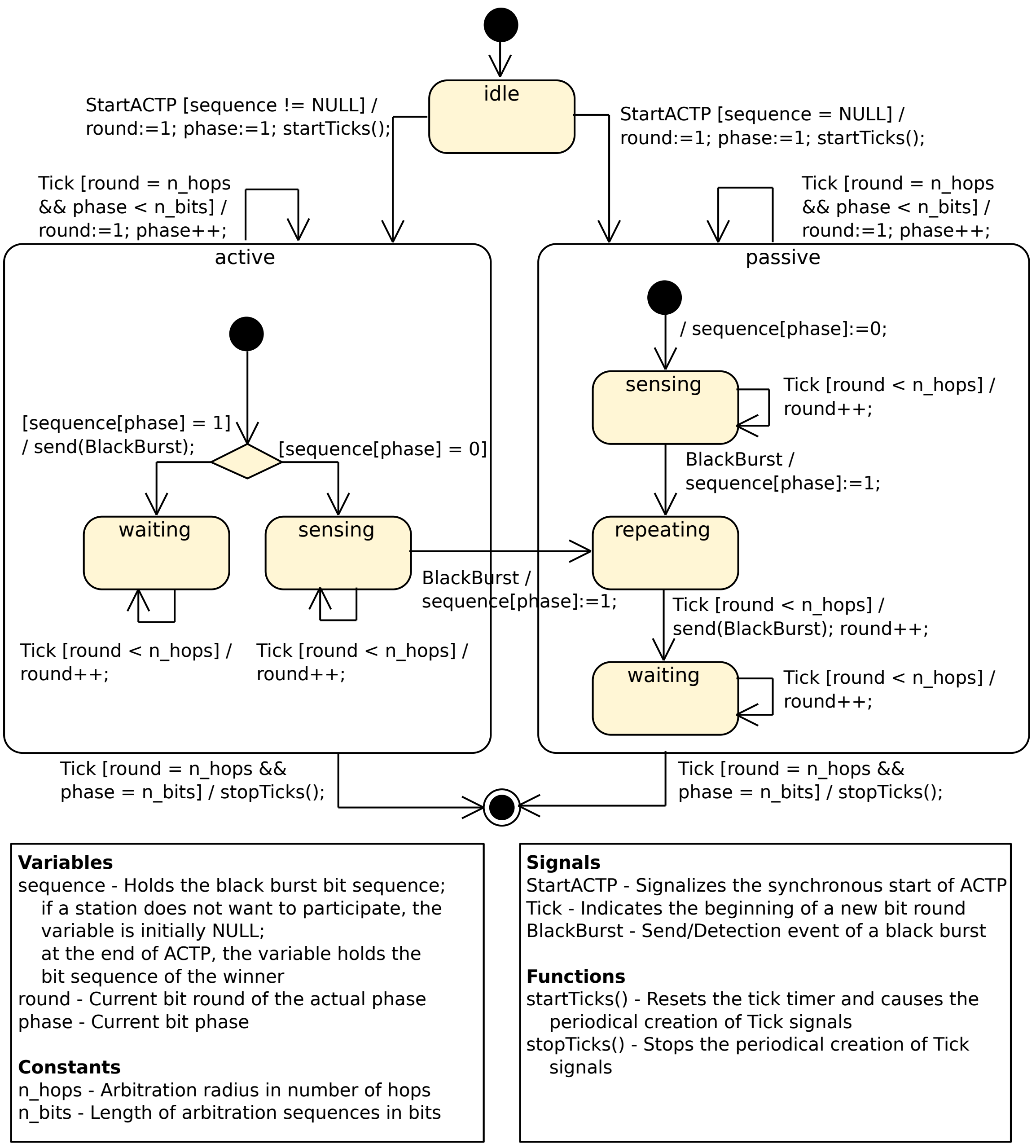


Figure 3.2: Terminology of ACTP.


idle
StartACTP [sequence != NULL] / round:=1; phase:=1; startTicks();
StartACTP [sequence = NULL] / round:=1; phase:=1; startTicks();
Tick [round = n_hops && phase < n_bits] / round:=1; phase++;
active
Tick [round = n_hops && phase < n_bits] / round:=1; phase++;
passive
[sequence[phase] = 1] / send(BlackBurst);
[sequence[phase] = 0]
waiting
sensing
Tick [round < n_hops] / round++;
Tick [round < n_hops] / round++;
BlackBurst / sequence[phase]:=1;
/ sequence[phase]:=0;
sensing
Tick [round < n_hops] / round++;
BlackBurst / sequence[phase]:=1;
repeating
Tick [round < n_hops] / send(BlackBurst); round++;
waiting
Tick [round < n_hops] / round++;
Tick [round = n_hops && phase = n_bits] / stopTicks();
Tick [round = n_hops && phase = n_bits] / stopTicks();

Variables
sequence - Holds the black burst bit sequence;
 if a station does not want to participate, the
 variable is initially NULL;
 at the end of ACTP, the variable holds the
 bit sequence of the winner
round - Current bit round of the actual phase
phase - Current bit phase

Constants
n_hops - Arbitration radius in number of hops
n_bits - Length of arbitration sequences in bits

Signals
StartACTP - Signalizes the synchronous start of ACTP
Tick - Indicates the beginning of a new bit round
BlackBurst - Send/Detection event of a black burst

Functions
startTicks() - Resets the tick timer and causes the
 periodical creation of Tick signals
stopTicks() - Stops the periodical creation of Tick
 signals

Figure 3.3: UML [Obj22] state machine of a single ACTP run.

ACTP's protocol behavior is illustrated in Fig. 3.3 by a statechart. The statechart assumes system ticks for each new bit round and is simplified, since it does not go into detail w.r.t. synchronization offset and transceiver-depending delays. These factors are discussed in Sect. 3.2.2.

The start of an ACTP phase occurs on each node simultaneously, where the point in time is either configured offline or by dynamic agreement. Nodes holding a bit sequence switch to state *active*, in which they process their sequence bit-wise, starting with the most significant bit. If a node's currently processed bit is a dominant 1, it sends a black burst in the first bit round of the corresponding bit phase. Afterwards, it waits in substate *waiting* until the start of the next bit phase, when it continues with the next bit of its sequence. If a node's current bit is a recessive 0, it switches to substate *sensing*, where it tries to detect black bursts of other nodes. If this actually happens in any bit round, the node stops processing its bit sequence and switches to state *passive*.

A node not holding a bit sequence monitors detected bits and acts as repeater. Therefore, it directly switches to state *passive* and listens to the medium. If the node does not detect a black burst during an entire bit phase, it stores a recessive 0 for this bit phase. If a black burst is detected, the node stores a 1 and switches to substate *repeating*, where it waits until the beginning of the next bit round. If this bit round is still in the same bit phase, the node sends a black burst to propagate the dominant bit one hop along in order to ensure its n_{hops} propagation. If, on the other hand, the next bit round already lies in the subsequent bit phase or after the end of the ACTP phase, no black burst is sent and the node either re-enters state *passive* to detect new black bursts or stops the protocol.

Since both n_{bits} and n_{hops} are configurable but fixed, all nodes terminate the protocol simultaneously. Thus, the duration of an ACTP phase is constant and pre-calculable. All nodes still in state *active* when stopping the protocol have sent the numerically greatest arbitration sequence and arise as winners. If the network diameter is large compared to the arbitration radius, several winners with different bit sequences are possible. However, if the arbitration radius is larger than or equal to the network diameter and if bit sequences are unique, there is only one network-wide winner.

3.2.2 Derivation of Bit Round Durations

In the previous section, system ticks were assumed to indicate the beginning of bit rounds, whereas their interval was not discussed further. A crucial prerequisite to guarantee ACTP's termination with fixed delays is that these system ticks occur periodically, thereby requiring a constant bit round duration. This section now looks at this duration and provides its derivation under consideration of synchronization inaccuracy and hardware limitations. We then show that all influencing factors are either constant or bounded, thereby proving the foundation of ACTP's constant runtime and deterministic protocol behavior.

In a perfect world, the only relevant factor in a bit round's duration $d_{bitRound}$ would be the pure transmission duration of a black burst d_{bb}. Yet in the real world, the following transceiver-dependent factors have additionally to be taken into account:

- d_{maxCCA}: The maximal delay to detect the medium's current state of occupation. It depends on the CCA mode of the used wireless technology (see Sect. 2.3.2) and has to be bounded and smaller than d_{bb} to avoid false negatives. With TI's CC 2420 transceiver, this delay comprises eight symbol durations.

- $d_{switchRx}$: Delay to switch from transmission to reception mode. This transition is often done automatically by the transceiver after a transmission is complete and usually has constant delay, which also holds for TI's CC 2420 transceiver.

- $d_{accessRx}$: Duration to switch from transmission to reception mode plus delay to monitor the medium's current state. For TI's CC 2420 transceiver, $d_{accessRx}$ is the sum of $d_{switchRx}$ and d_{maxCCA}.

- $d_{switchTx}$: Delay to switch from reception to transmission mode, which is required before each transmission. This delay is usually constant, which is true for TI's CC 2420 transceiver.
- d_{pause}: Configurable additional duration to observe an idle medium in between subsequent black burst transmissions. This time also depends on the CCA mode. With energy detectors, the duration has to be set equal or larger than the symbol duration. For reasons of efficiency, d_{pause} should be selected as small as possible.

From a performance perspective, transceivers with corresponding small delays are desirable. This is, however, no prerequisite of ACTP, which only requires that these factors are either constant or bounded. Example values for TI's CC 2420 transceiver can be found in Appendix A.2. If a scenario comprises different transceivers, the maxima of each factor have to be chosen for the derivation of $d_{bitRound}$. In addition to the transceiver-specific factors above, the following delays influence the timing of ACTP as well:

- $d_{maxOffset}$: Synchronization inaccuracy, which distorts the synchronous start of bit rounds. For the success of ACTP, it is important that the synchronization mechanism guarantees a bounded offset.[1]
- d_{prop}: Physical propagation delay of black bursts. Because this delay is very small for small distances that are usually present in WSNs, it is neglected in the following.
- $d_{execution}$: Execution delay due to limitations of the processing unit. This delay can, for instance, additionally defer the detection of a black burst or delay its transmission start. Though this delay has to be taken into account in an implementation, it is ignored in the following, since it is not ACTP-inherent.

After the introduction of relevant factors, the following analysis derives constraints that must hold for the successful application of ACTP. These constraints are illustrated in Fig. 3.4 and define the duration of a bit round $d_{bitRound}$. Some of them are only required if the arbitration radius is one hop; others only for larger arbitration radii. Though we mention the affected radii for each constraint, we do not treat the cases separately. Instead, for simplicity, we demand the validity of all constraints and derive a single definition of $d_{bitRound}$.

Constraint 3.2.1. Necessary for all n_{hops}.

The detection of black bursts sent by nodes in bit round $1 \le j \le n_{hops}$ of bit phase $1 \le i \le n_{bits}$ must be associated to the same bit phase and bit round by all receiving stations.

By this constraint, it is assured that detected dominant bits are assigned to the correct position in the bit sequence, which is necessary to guarantee that the node with the numerically greatest sequence wins the contest. With arbitration radii larger than one, the constraint is additionally mandatory to guarantee the propagation of dominant bits across the full n_{hops} distance. To fulfill this constraint, two cases have to be distinguished. They are illustrated in Fig. 3.4 for multi-hop arbitration radii ($n_{hops} > 1$). For $n_{hops} = 1$, cases and implications are similar but without distinction between bit phases and bit rounds.

[1] In full applications of ACTP in Chapter 4, the synchronization protocol BBS is adopted, whose synchronization offset is again transceiver-dependent. However, BBS is not prescribed by ACTP and, thus, $d_{maxOffset}$ is in the first instance transceiver-independent.

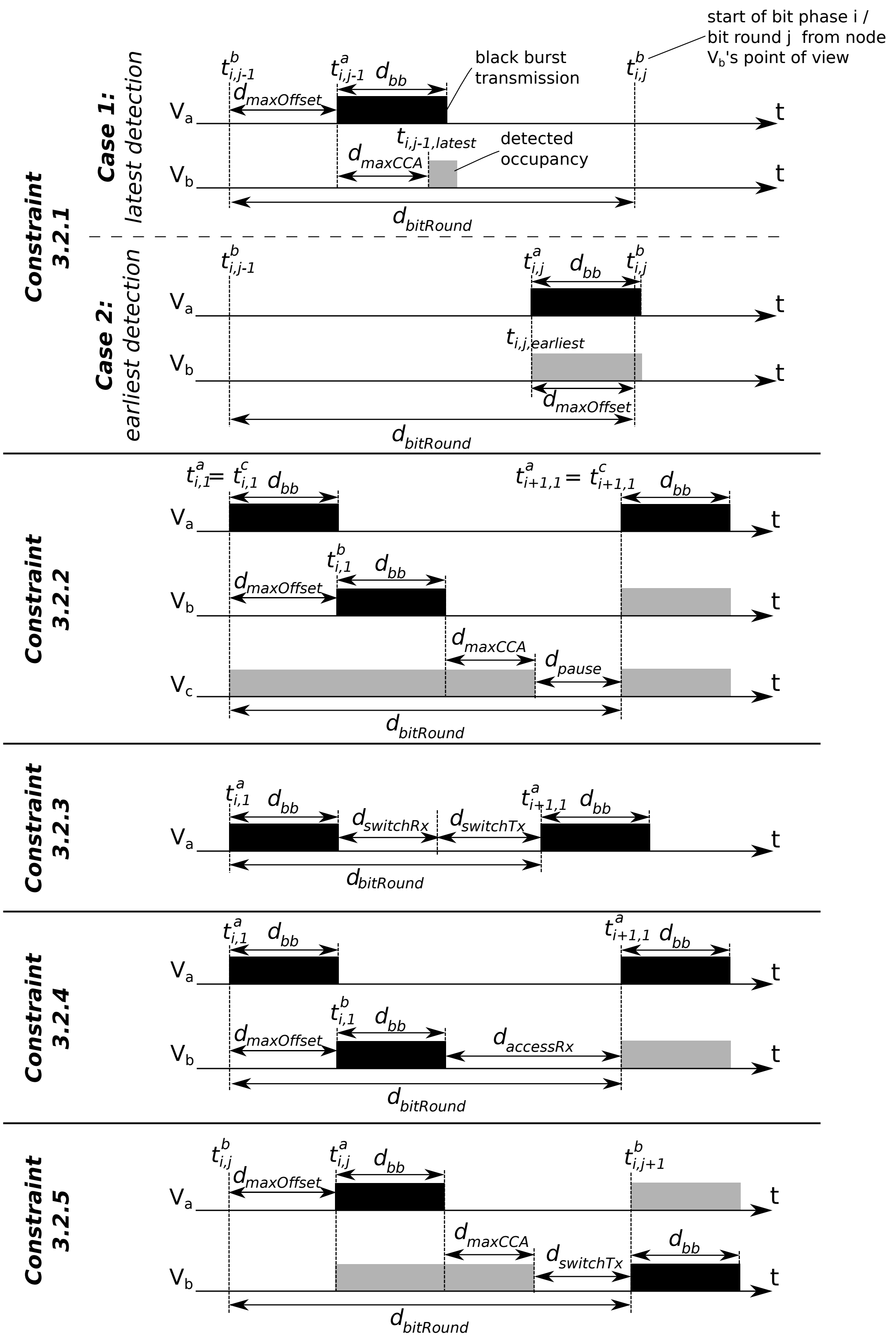


Figure 3.4: Constraints on the duration of a bit round.

- *Case 1 – Latest detection of black bursts:* Due to synchronization inaccuracy, node V_a starts the black burst transmission in bit round $j-1$ with a worst-case synchronization offset of $d_{maxOffset}$. Node V_b detects this black burst with additional CCA delay of d_{maxCCA}. Though V_b expects black bursts at $t^b_{i,j-1}$, which is the locally perceived beginning of bit round $j-1$, it actually detects the black burst at $t_{i,j-1,latest} = t^b_{i,j-1} + d_{maxOffset} + d_{maxCCA}$, yet it must still assign the black burst to bit round $j-1$ non-ambiguously.

- *Case 2 – Earliest detection of black bursts:* Node V_a starts bit round j $d_{maxOffset}$ earlier than V_b and V_b detects the sent black burst immediately without CCA delay. Though this detection event is at $t_{i,j,earliest} = t^b_{i,j} - d_{maxOffset}$ and thereby earlier than expected, V_b must assign the dominant bit to bit round j non-ambiguously.

Note that both cases cannot occur simultaneously, since $d_{maxOffset}$ is the total upper offset bound. Thus, V_b cannot be synchronized with an offset of $+d_{maxOffset}$ to some node V_a and with $-d_{maxOffset}$ offset to some other node V'_a. However, both cases must be taken into account, because a node does not know its current synchronization offsets to its neighbors. Thus, $(t_{i,j,earliest} - t_{i,j-1,latest}) \geq d_{pause}$ must hold to clearly assign the beginning of a black burst detection to the correct bit round and bit phase. With $t^b_{i,j} = t^b_{i,j-1} + d_{bitRound}$, this results in the following formal constraint for $d_{bitRound}$:

$$d_{bitRound} \geq 2 \cdot d_{maxOffset} + d_{maxCCA} + d_{pause} \tag{3.1}$$

Constraint 3.2.2. Necessary for all n_{hops}.

Black bursts that are sent in two adjacent bit rounds must be detected as independent black bursts by all receiving nodes.

By this constraint, we forbid that black bursts cause a single bit round-spanning busy tone on the medium. This is required to enable the correct assignment of black burst detections to bit phases/rounds and to filter false positives by means of the duration of medium occupancies. The worst-case scenario, which threatens this constraint, is presented in Fig. 3.4 for one-hop arbitration radius. For $n_{hops} > 1$, a similar example can be devised with identical implications. With $n_{hops} = 1$, the worst-case occurs if station V_a sends two dominant bits in adjacent bit phases, if a second station V_b sends a dominant bit with a temporal offset of $d_{maxOffset}$, and if the detection of the end of V_b's transmission is additionally deferred by the maximal CCA delay. Formally, these conditions yield the following constraint:

$$d_{bitRound} \geq d_{maxOffset} + d_{bb} + d_{maxCCA} + d_{pause} \tag{3.2}$$

Constraint 3.2.3. Necessary for $n_{hops} = 1$.

An active node must be able to transmit consecutive dominant bits in subsequent bit phases.

This constraint is illustrated in Fig. 3.4 under the assumption that the transceiver's transition from transmission to reception mode cannot be interrupted and has to be finished before a new transmission can start. In this case, the constraint yields the following inequality:

$$d_{bitRound} \geq d_{bb} + d_{switchRx} + d_{switchTx} \tag{3.3}$$

However, measurements with TI's CC 2420 transceiver revealed that – in contradiction to the data sheet [Tex07] – the transition from transmission to reception mode is interruptible [Eng13]. In this case, $d_{switchRx}$ can be omitted in the formula. Vice versa, switching from reception to transmission mode is always necessary, thereby prohibiting optimizations to omit $d_{switchTx}$.

Constraint 3.2.4. Necessary for $n_{hops} \leq 2$.

A node transmitting a black burst must be able to detect a black burst in the next bit round.

This constraint covers two scenarios: First, in case of $n_{hops} = 1$, the constraint must hold for active nodes, which must always detect black bursts when sending recessively; even if they sent a black burst in the previous bit phase. This case is illustrated in Fig. 3.4, where V_b is the node sending recessively in bit phase $i + 1$. Second, this constraint must be fulfilled with $n_{hops} = 2$ for passive stations forwarding a dominant bit in the last bit round of a bit phase, since they may have active neighbors sending dominantly in the first bit round of the subsequent bit phase. For both scenarios, the resulting formal constraint is as follows:

$$d_{bitRound} \geq d_{maxOffset} + d_{bb} + d_{accessRx} \tag{3.4}$$

Constraint 3.2.5. Necessary for $n_{hops} > 1$.

A passive station detecting a black burst in bit round $j < n_{hops}$ must be able to forward the black burst in bit round $j + 1$.

The worst-case for this constraint is illustrated in Fig. 3.4 and occurs if the repeating node (V_b) perceives the start of bit round j $d_{maxOffset}$ earlier than the sender of the first black burst (V_a) and if the detection of the black burst's end is delayed by the maximal CCA delay. Taking additionally the switching delay into account, this constraint is formally expressed as follows:

$$d_{bitRound} \geq d_{maxOffset} + d_{bb} + d_{maxCCA} + d_{switchTx} \tag{3.5}$$

To get the aggregated and smallest possible duration of bit rounds, the maximum of all constraints has to be determined:

$$\begin{aligned} d_{bitRound} \;=\; & max\{2 \cdot d_{maxOffset} + d_{maxCCA} + d_{pause}, \\ & d_{maxOffset} + d_{bb} + d_{maxCCA} + d_{pause}, \\ & d_{bb} + d_{switchRx} + d_{switchTx}, \\ & d_{maxOffset} + d_{bb} + d_{accessRx}, \\ & d_{maxOffset} + d_{bb} + d_{maxCCA} + d_{switchTx}\} \end{aligned} \tag{3.6}$$

Starting from $d_{bitRound}$, the derivation of bit phase duration and ACTP phase duration is straightforward:

$$d_{bitPhase} = n_{hops} \cdot d_{bitRound} \tag{3.7}$$

$$d_{actp} = n_{bits} \cdot d_{bitPhase} = n_{bits} \cdot n_{hops} \cdot d_{bitRound} \tag{3.8}$$

To obtain concrete values of $d_{bitRound}$, $d_{bitPhase}$, and d_{actp}, scenario- and transceiver-specific properties like arbitration radius, bit sequence length, and switching delays have to be inserted

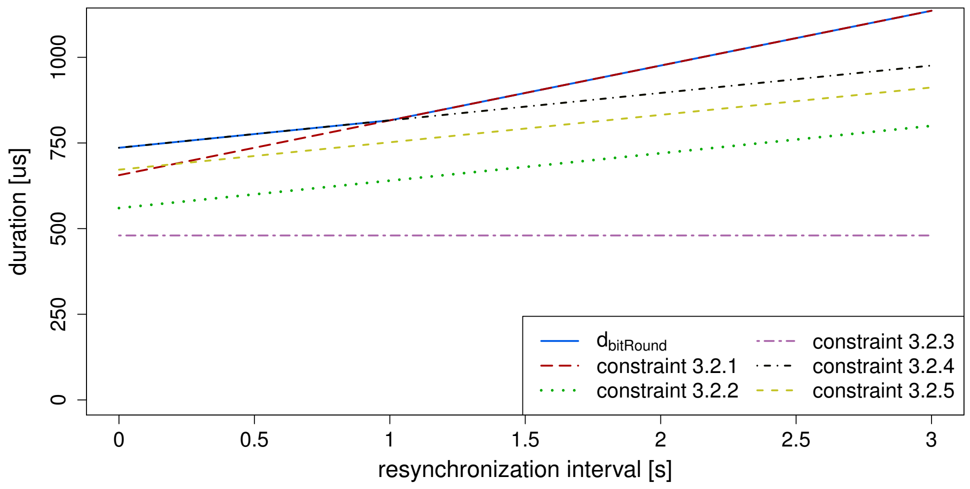


Figure 3.5: Exemplary values of $d_{bitRound}$ and its defining constraints.

into Equations 3.6, 3.7, and 3.8. Example values of $d_{bitRound}$ and its defining constraints are plotted in Fig. 3.5 as function of the resynchronization interval, which controls the nodes' maximal offset $d_{maxOffset}$. The values are calculated with TI's CC 2420 transceiver and under the assumption of synchronization with the master-based variant of BBS. Because all parameters of $d_{bitRound}$, $d_{bitPhase}$, and d_{actp} are constant (like switching delays) or pre-configured (like arbitration radius), the runtime of ACTP can be calculated in advance.[2] Together with its predictable mode of operation, this turns ACTP into a deterministic protocol.

3.3 Network-wide Application of ACTP

Due to its configurable application range, ACTP can also be applied network-wide. In this case, it is sufficient to estimate the network's diameter by an upper bound that is assigned to n_{hops}. For reasons of efficiency, this bound should be as tight as possible. In the following, we assume that $n_{maxHops}$ is such an upper bound and set $n_{hops} = n_{maxHops}$. In the next subsection, we first show ACTP's network-wide application by means of an example. Afterwards, positive side effects of such an application are discussed.

3.3.1 Example

Figure 3.6 illustrates the protocol's mode of operation in a multi-hop network with $n_{hops} = n_{maxHops} = 4$ and with arbitration sequences of length $n_{bits} = 3$. At the beginning of the ACTP phase at t_0, stations V_b, V_d, V_e, and V_f hold bit sequences and are therefore active, while V_a and V_c act as repeaters. Because their first bit is dominant, all active nodes transmit a black burst in the first bit round of the first bit phase. These black bursts are received by V_a and V_c, and forwarded in the second bit round. Since all stations are already aware of the dominant bit after the first bit round, the forwarding by nodes V_a and V_c would actually not be necessary. This information is, however, not available without global knowledge, thereby making forwarding

[2]Note that the runtime is, in particular, independent of the number of participating nodes.

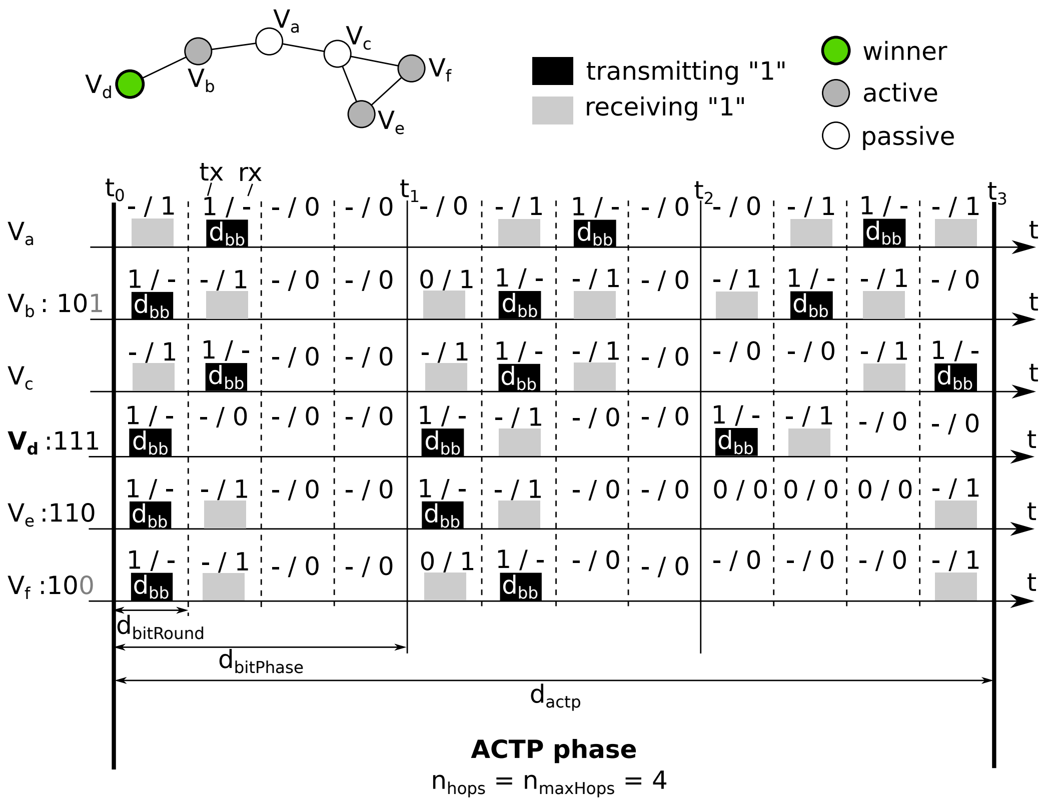


Figure 3.6: Network-wide application of ACTP in a network with 4 hops diameter.

mandatory. For similar reasons, all stations must wait the third and fourth bit round of this bit phase, though there is no further transmission.

At t_1, the second bit phase starts, in which only V_d and V_e send dominant bits in the first bit round. After detecting this bit, V_b and V_f become passive. Together with V_c, they act as repeaters and forward black bursts in the second bit round. V_a, in turn, forwards a dominant bit in the third bit round, though it would again not be necessary in the given scenario.

When the third bit phase starts at t_2, only V_d and V_e are active and transmit the third bit of their arbitration sequence. Because this bit is recessive for V_e, V_e remains silent and listens for dominant bits. V_d's dominant bit is, on the other hand, repeated by V_b, V_a, and V_c in the second, third, and fourth bit round, respectively, when it is finally detected by V_e and V_f. Thus, all four bit rounds are required in this bit phase to propagate V_d's dominant bit network-wide.

3.3.2 Implicitly Available Information

Running ACTP with network-wide arbitration radius and unique bit sequences achieves a network-wide arbitration with a single winner node. As further result, all nodes are aware of the winner's arbitration sequence, since it was present on the medium. For some applications of ACTP, retrieving the winner's sequence is a useful extra and enables the collision-protected transfer of small payload (see also further applications of ACTP in Chapter 4).

Because Constraint 3.2.1 demands the unique assignment of detected black bursts to bit rounds, all nodes can furthermore deduce their distance to the winning node in "sensing hops".

This distance corresponds to the bit round number of the last detected dominant bit, since this bit has its single origin at the winner. In the example of the previous section, for instance, node V_f detects the last dominant bit in the fourth bit round. Thus, it knows that the winner node is four hops away, though it does neither know the full path to the winner nor its id.

Applying ACTP with non-unique priorities, i.e., with assigning the same arbitration sequences to multiple nodes, does not guarantee a single winner, yet the network-wide awareness of the winner(s) bit sequence is preserved. Furthermore, nodes can also still deduce the distance to the nearest winner(s) from the detection of the last dominant bit. The number of winners or distances to other winners can, however, not be determined.

3.4 Cooperative Transfer with ACTP

The term *Cooperative* in ACTP suggests that the protocol is not only applicable to arbitrations but also to (cooperative) data transfer. This operation mode of ACTP does not differ from its application to arbitration, but the usage is different: Instead of allowing stations with different bit sequences, cooperative transfer requires only one active node at the beginning of an ACTP phase.[3] The bit sequence of this node is then propagated across the arbitration radius and becomes common knowledge of the network if the radius satisfies the network diameter.

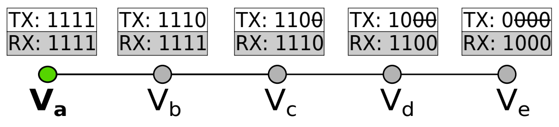
Compared to the dissemination of information with regular data transfers, the black burst-based propagation of information guarantees a constant transfer delay that is independent of the number of network nodes. Thus, ACTP provides an efficient and low-overhead method to distribute small pieces of information in dense networks. Useful applications are, for instance, the signaling of network mode changes and the network-wide distribution of time values. More details on example applications are given in Chapter 4.

3.5 Restricted Application of ACTP

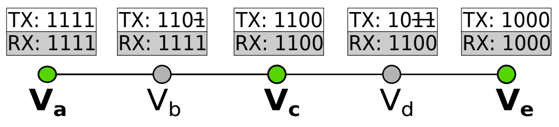
Running ACTP with an arbitration radius $n_{hops} < n_{maxHops}$ is called *restricted application of ACTP*. With this configuration, ACTP still guarantees that a winning node is the only winner within its n_{hops} neighborhood as long as bit sequences are unique. Advantages of restricted applications w.r.t. arbitration are the increase of the number of winning nodes without violating the protocol's deterministic behavior and better utilization of network and energy resources by improving spatial reuse. With $n_{hops} = 2$, for instance, and under the assumption of equal transmission ranges of black bursts and regular data frames, restricted applications of ACTP handle the hidden station problem in a deterministic way.

Though the restricted application of ACTP enables multiple winners, there is no guarantee that the number of winners is maximal. There can, in particular, be situations, in which a node loses during an ACTP phase but has no winner in its n_{hops}-hop neighborhood when the protocol terminates. There can also be cases, in which a node wins, though other nodes in its n_{hops}-hop neighborhood had numerically greater arbitration sequences. These phenomenons have also been observed in previous works [YYH03b, PATR07a], where they are summarized as *multi-hop competing problem*. They are illustrated in Fig. 3.7: In Fig. 3.7(a), bit sequences are assigned

[3] Alternatively, there can be multiple active nodes as long as they send identical bit sequences.



(a) Less winners than possible [YYH03b].



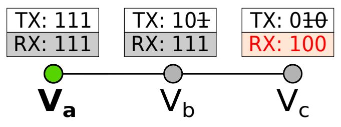
(b) Winners despite neighbors with greater bit sequences.

Figure 3.7: Multi-hop competing anomalies in a network with $n_{maxHops} = 4$ and $n_{hops} = 1$.

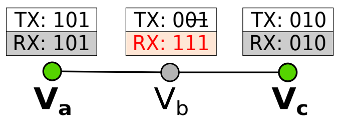
such that there is only one winner, though ACTP's parameters ($n_{maxHops} = 4$ and $n_{hops} = 1$) allow up to three winners. Figure 3.7(b) presents the second anomaly, where in addition to V_a, two nodes (V_c and V_e) win, though their direct neighbors have numerically greater sequences. In general, the number of winners depends on the arbitration radius, the network diameter, and the length of bit sequences and their assignment to network nodes. Independent of these parameters, ACTP guarantees that at least one winner arises if at least one node is initially active.

A drawback of restricted applications of ACTP is regarding the validity of detected bit sequences. Since nodes could have monitored so-called *phantom frames* [BBCG11], which are bit sequences that are actually not sent by any node, they can no longer retrieve reliable information from detected bit sequences. Figure 3.8 illustrates this anomaly with two examples. Both examples show the same network with two hops diameter but different assignment of arbitration sequences. In both examples, ACTP runs with $n_{hops} = 1$. In Fig. 3.8(a), arbitration sequences are assigned such that V_c only detects the prefix of V_b's bit sequence, since V_b loses against V_a in the second bit phase. Bit sequences in Fig. 3.8(b) produce two winners (V_a and V_c). Because both nodes send their bit sequence completely, V_b observes the OR combination of both. In larger scenarios, nodes may – in addition to detecting prefixes only or results of OR operations – also detect a combination of both types of distortions.

While a detected bit sequence does no longer provide reliable information, it is sometimes sufficient to know whether there is at least one winning node within a node's n_{hops}-hop neighborhood. This can be achieved by suffixing bit sequences with a dominant End-of-Frame (EOF) bit, which is evaluated by passive nodes to determine two pieces of information: First, the node can conclude that there is no winner in n_{hops} range if the detected EOF bit is recessive (like for node V_c in Fig. 3.8(a)). Otherwise, there is at least one winner within the arbitration radius. Second, if the EOF bit is actually dominant, it can derive the distance in hops to (one of) the nearest winners, which is equal to the bit round number in which the EOF bit is detected.



(a) Reception of a bit sequence prefix.



(b) Reception of the OR of two bit sequences.

Figure 3.8: Distortion of detected bit sequences with $n_{maxHops} = 2$ and $n_{hops} = 1$.

Though the detection of bit sequences can be distorted when nodes compete with restricted arbitration radius and different bit sequences, cooperative data transfer is still applicable. By ensuring that no node participates with a different bit sequence, ACTP still provides the reliable propagation of information over n_{hops} hops and with bounded delay.

3.6 Errors and Countermeasures

Up to now, ACTP has been introduced under the assumption of symmetrical links and a valid single-network property. In this section, we discuss implications if this assumption does not apply (Sect. 3.6.1). Furthermore, measures w.r.t. error control are presented (Sect. 3.6.2).

3.6.1 Violation of the Single-Network Property

The single-network property is introduced in Sect. 2.2.3 and consists of two parts: The first part prohibits all sources of interference that may cause the detection of false positives and threaten the correctness of ACTP. In general, this part of the property must fully apply. In some situations, however, tuning the CCA threshold can fade out interference if it is weaker than the signal strength of valid black bursts.

The second part of the property requires a stable path of symmetrical (communication) links between all pairs of nodes. If ACTP is applied network-wide, link symmetry is actually not necessary. Instead, it is sufficient that all node pairs are connected via some path of links with at most $n_{hops} = n_{maxHops}$ hops length. Yet, some implicitly available information like the distance to the winning node in hops becomes inaccurate with asymmetrical links. Running ACTP with $n_{hops} < n_{maxHops}$ requires a more careful consideration of results if links can be asymmetrical, since nodes can eliminate other nodes with numerically larger bit sequences due to asymmetrical distances. This phenomenon is illustrated in Fig. 3.9, where V_a's second bit defeats V_c.

Because results are less predictable with asymmetrical links, link symmetry is – in particular, with restricted applications of ACTP – desirable. Regarding the probability of asymmetrical (communication) links, contradictory results can be found in the literature: [GKW$^+$02], for instance, reports that 5-15% of all links are asymmetrical, whereas [CM10, CMS10] present experiments with a high correlation between the RSS at both sides of a link and conclude that link symmetry is a valid assumption in homogeneous WSNs. A possible explanation for the deviating results are the different environments and transceivers (RFM's TR1000 [RFM12] vs. TI's CC 2420 [Tex07]), which differ in transfer rate, modulation, and frequency band. As a conclusion, link symmetry should not be presumed implicitly but checked a priori and specifically for the used hardware platform and environment.

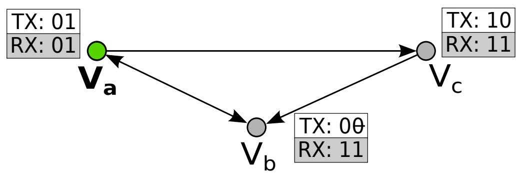


Figure 3.9: V_a defeats V_c despite lower priority due to link asymmetry ($n_{hops} = 1$, $n_{maxHops} = 2$).

However, not link asymmetry but link instability is a bigger threat to ACTP, since it can lead to undesired and confusing results w.r.t. winnings nodes and detected bit sequences. It is particularly possible that two nodes with different priorities win the contest, though they are temporarily within arbitration radius of each other. Hence, unstable links must be avoided entirely, which can be achieved by a controlled node placement.

3.6.2 Error Control and Redundancy

If false positives or negatives cannot be excluded, countermeasures become necessary to detect – and ideally correct – them. In communication systems, two general approaches of error control are distinguished [ASSC02]: *Automatic Repeat reQuest (ARQ)*, where the receiver sends feedback to the sender after the end of the transmission to inform about success or to trigger retransmissions [WMW05], and *Forward Error Control (FEC)*, which adds redundancy to transmissions to enable error correction by receivers [WMW05]. In the following, the incorporation of both schemes into ACTP is discussed.

3.6.2.1 ARQ – Automatic Repeat reQuest

Though incorporating ARQ into ACTP is in general possible, it can only be applied with network-wide arbitration radius, since otherwise, different bit sequences may overlap at passive nodes and cause the detection of “valid” dominant and recessive bits in the same bit round. Because dominant bits overrule recessive ones, passive nodes could accordingly not acknowledge the detection of recessive bits. If ACTP is executed with network-wide arbitration radius, error control with ARQ is more compatible. In this case, checksums can be appended to bit sequences, which enable passive nodes to check the correctness of a received bit sequence. Since several nodes may receive a bit sequence incorrectly and the notification about a transmission error must be sent network-wide as well, notifications have to be encoded in a collision-protected way. This can be achieved by running ACTP again with $n_{hops} = n_{maxHops}$ and a single bit that is set dominantly if the checksum does not match (NAK (Negative Acknowledgment)) and recessively otherwise (ACK (Acknowledgment)).[4] Thus, retransmissions are required if at least one node sends a dominant bit during the transmission of ACK/NAK.

3.6.2.2 FEC – Forward Error Control

A jeopardy of ARQ is that a single false positive/negative during the transmission of the (negative) acknowledgment can lead to an inconsistent network state, where some nodes start retransmissions and others not. Therefore, FEC, which is, for instance, applied by IEEE 802.15.1 Bluetooth [IEE05], is a better solution to improve ACTP’s robustness against detection faults. In this regard, an obvious way to add redundancy is the repetition of bits – either bit-wise or sequence-wise –, where the number of repetitions serves as regulation screw: If bits are repeated once, only error detection but no correction is possible. If otherwise the number of repetitions is higher, a majority decision can be applied to correct errors.

A higher robustness is achieved if repetitions are bit-wise, where bit rounds are subdivided into several *repetition rounds*. The reason why bit level repetition should be preferred to se-

[4]See also Sect. 4.3.4 about the application of ACTP to acknowledgments.

quence level repetition is the competitive character of ACTP: If repetition is sequence-wise and, for instance, with two repetitions (i.e., the sequence is sent three times), two false negatives/positives in different repetitions can lead to three different winner sequences, where the actually correct winner completes its bit sequence only once. If, however, repetition is on bit level, majority decisions are bit-wise and the result of the ACTP run stays correct except if two errors occur in two repetition rounds of the same bit round.

3.7 Realization of ACTP

There are in total three realizations of ACTP: An SDL [Int12c] specification and two manual implementations for the Imote 2 platform [MEMara], which are the result of student projects. ACTP's SDL specification (Sect. 3.7.1) was applied in simulative evaluations with the simulator PartsSim [BGK08]. The first manual implementation is stand-alone (Sect. 3.7.2.1); the second one a protocol of the *Black burst-integrated Protocol Stack* (BiPS) [Eng13] (Sect. 3.7.2.2). They serve as proof-of-concept of the practical feasibility of black bursts and ACTP.

3.7.1 SDL Specification of ACTP

ACTP's SDL specification is part of MacZ [BGK07], a MAC layer for wireless ad-hoc networks with QoS support. Parts of MacZ and an excerpt of the specification of ACTP are shown in Fig. 3.10. Though tool chains exist to automatically transform SDL specifications into implementations for the Imote 2 platform (see Part II of this thesis), the main objective of the specification is ACTP's usage in performance simulations with PartsSim [BGK08]. As trials have shown, the specification can actually not be transformed into implementations automatically due to efficiency reasons and a too large overhead that is caused by the specification of MacZ.

The SDL specification shows ACTP in the context of an application. The required synchronization is accomplished by block `Synch` containing a specification of BBS [GK11a]. To send/detect black bursts, block `BlackBurstCoDec` is utilized, which monitors medium occupancies by the CCA information of the transceiver.

The behavior of ACTP is specified in the corresponding process, which is shown in the center and with two transitions exemplarily. To decouple applications, the ACTP process is supported by processes `Ctrl` and `Queue`, where `Ctrl` is responsible for the timely and synchronous start of ACTP and `Queue` stores bit sequences of upcoming ACTP runs. Shortly before a run of ACTP starts, `Ctrl` informs `ACTP` by signal `announce` about the ID of the upcoming ACTP run. Thereupon, `ACTP` requests the corresponding bit sequence from `Queue`. The actual start of ACTP is triggered by process `Ctrl` with the SDL signal `enable`, whose consuming transition is given in the figure: If the node is active – i.e., if it holds a bit sequence to transmit – and the first bit dominant, a black burst transmission is triggered. Otherwise, the protocol waits for a black burst detection. The reaction to such a detection is illustrated in the second example transition, where an active node is turned into a passive node (if not already occurred before) and the dominant bit is stored in the received bit sequence. Afterwards, the node waits for the end of the bit round, either to forward the dominant bit (if there is still a bit round left in this bit phase) or to await the end of the bit phase.

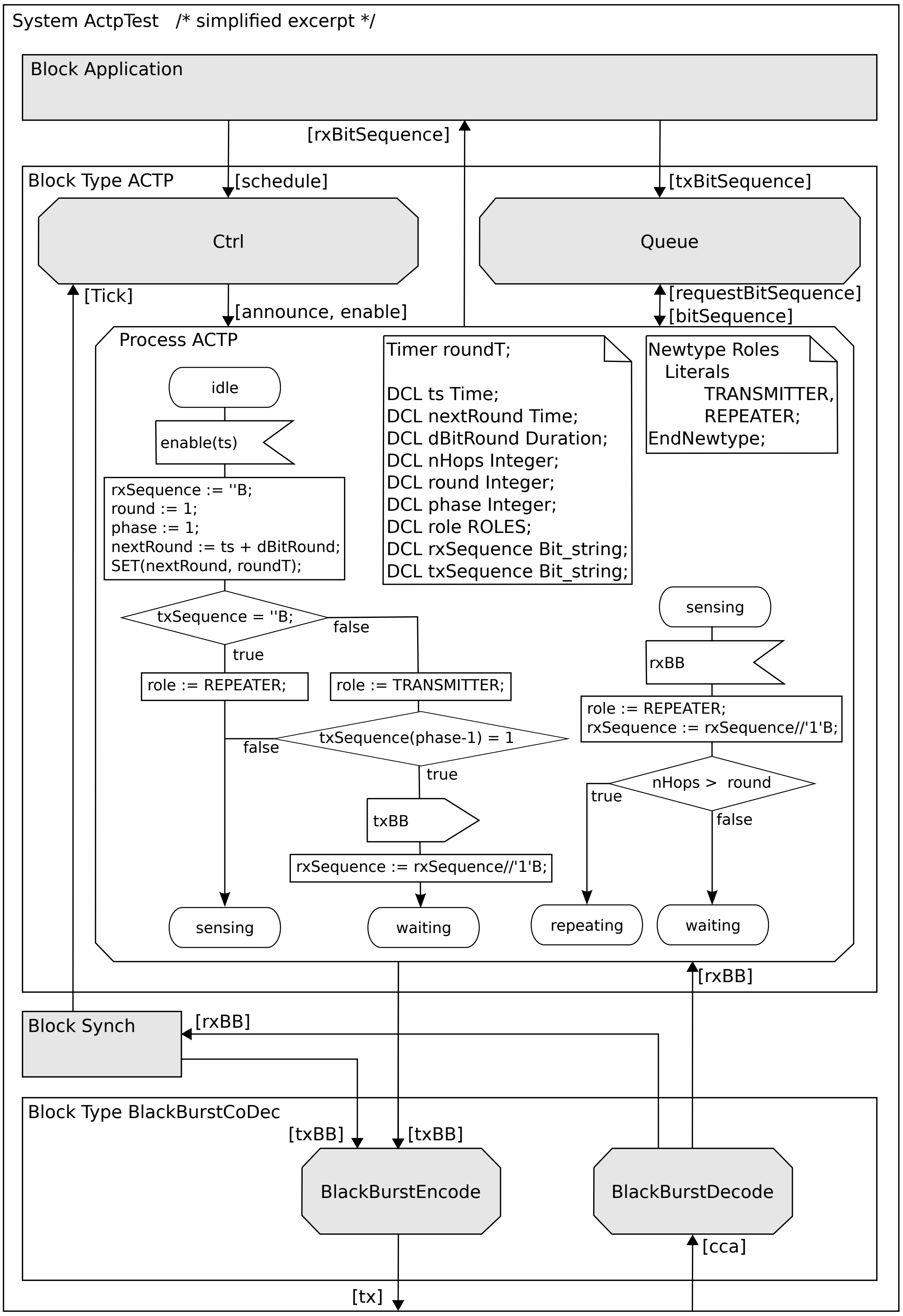


Figure 3.10: Excerpt of the SDL specification of ACTP in MacZ.

3.7.2 Hand-written Implementations for the Imote 2 Platform

Both manual ACTP implementations are bare, i.e., they do not rely on an operating system. Thus, they have full control over hardware and interrupts, which helps to minimize delays and enables the deactivation of undesired hardware interrupts during time-critical intervals. The underlying hardware platform is an Imote 2 [MEMara], which is a platform for WSNs and equipped with TI's CC 2420 transceiver [Tex07]. More details on the Imote 2 and the CC 2420 transceiver can be found in Appendix A. As sketched during the introduction of black bursts in Sect. 2.3.1, we realize black bursts by irregular MAC frames of minimal length. To detect them on the medium, hardware interrupts are utilized, that are triggered by changes of the transceiver's CCA output pin, which is interconnected to a GPIO pin of the microcontroller. In both implementation variants, the required synchronization is accomplished by BBS.

3.7.2.1 Stand-alone Implementation of ACTP

ACTP's first implementation variant is part of a test framework, whose objective is the evaluation of black burst-based protocols. The framework's use in comprehensive scenarios is not in the focus. An excerpt of the implementation's interface and the interplay with applications and BBS is given by Fig. 3.11.[5] An `Application`, which is going to run ACTP, registers the start time of the ACTP phase, which is relative to synchronization ticks, and its priority (i.e., the arbitration sequence) at the `ACTP` component. It furthermore provides a callback, which is invoked after the end of the ACTP phase, to receive the result of the protocol's run. The `ACTP` component, in turn, registers a callback at `BBS` to get notified about synchronization ticks that are afterwards utilized to schedule starts of ACTP phases. Thereupon, bit rounds and bit phases are scheduled relative to the start of ACTP phases. To detect dominant bits, `ACTP` registers a further callback at the `BlackBurst` component, which senses the medium for black bursts and provides an interface for black burst-encoded bit transmissions.

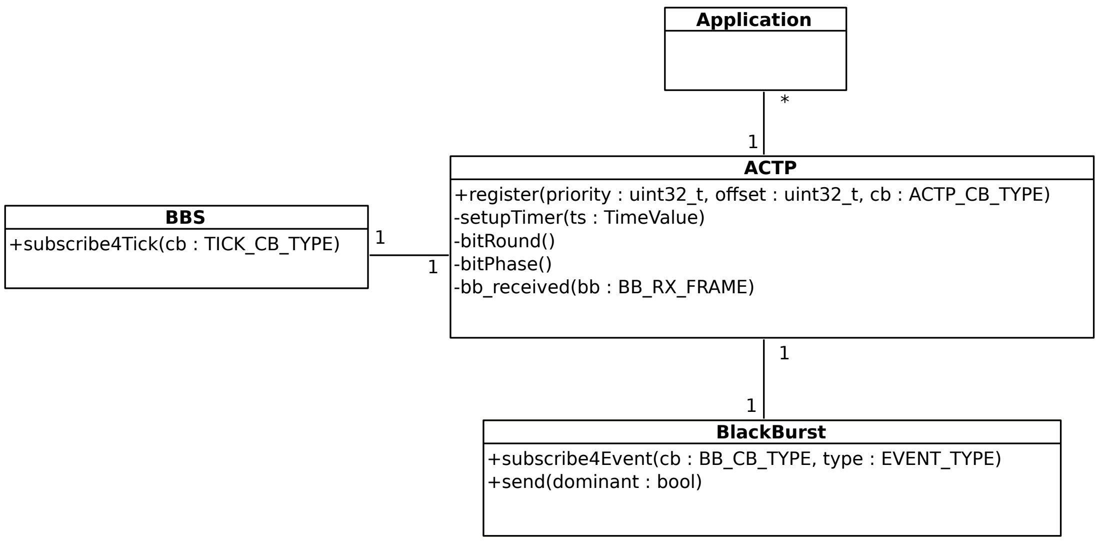

Application
*
1
ACTP
+register(priority : uint32_t, offset : uint32_t, cb : ACTP_CB_TYPE)
-setupTimer(ts : TimeValue)
-bitRound()
-bitPhase()
-bb_received(bb : BB_RX_FRAME)
BBS
+subscribe4Tick(cb : TICK_CB_TYPE)
1
1
1
1
BlackBurst
+subscribe4Event(cb : BB_CB_TYPE, type : EVENT_TYPE)
+send(dominant : bool)

Figure 3.11: Architecture of the first implementation variant of ACTP.

[5]Technically, the implementation is in C. The illustration as class diagram is for visualization only.

3.7.2.2 Integration of ACTP into BiPS

The second implementation variant is part of BiPS, a protocol framework for Imote 2 nodes. The framework itself and the incorporation of BBS are results of a student's thesis [Eng13]. A derivation and validation of BBS timing constraints in the context of BiPS can be found in [ECG14]. Over the last years, BiPS has been extended continuously with several MAC protocols and basic Operation System (OS) functionalities in student projects[6] and projects with industrial partners[7]. Currently, it includes four MAC protocols: *Contention-Based MAC* (short: CB; CSMA/CA-based), *Reservation-Based MAC* (short RB; TDMA-based), *ACTP*, and a MAC protocol for *Mode-Based* communication [BGK14] (short MB). The OS functionalities of BiPS comprise drivers for peripheral devices, timers, a scheduling hierarchy, and an application interface to decouple time-critical protocols like BBS and ACTP from applications and higher-layer protocols with less stringent time constraints. Different from its stand-alone implementation, ACTP's integration into BiPS enables applications in larger scenarios and in combination with other MAC protocols. It also reduces the "burden" of the implementation, since many functionalities like the scheduling of the start of an ACTP phase and the queueing of upcoming transmissions are provided by the BiPS framework.

The architecture of BiPS is shown in Fig. 3.12 in interrelation with higher-layer protocols and applications. It follows a layered approach to enable abstractions from technical details of lower layers. Thereby, higher layers can mostly be implemented in a platform-independent way. The lowest layer in the architecture provides system software and interfaces to access hardware subsystems like hardware timers and DMA (Direct Memory Access). It is used by layer 1, which contains hardware drivers for peripheral devices like the CC 2420 transceiver. Layer 2 comprises all MAC protocols, the synchronization protocol BBS, and a multiplexer to access them. Black burst-related functionalities, which are required by both BBS and ACTP, are encapsulated in a separate component. Higher layers – e.g., scenario-specific applications

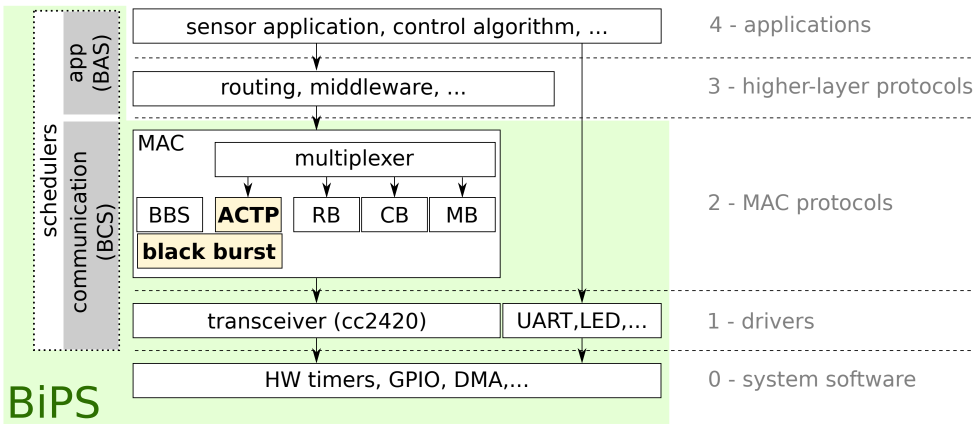


Figure 3.12: Architecture of BiPS [Eng13].

[6]Besides smaller projects, the following bachelor and master theses cover extensions of BiPS or the framework's interface to higher-layer protocols: [Eng13, Kra13a, Kra14, Mat14].

[7]BiPS is used in the joint project "Softwareinnovationen für das digitale Unternehmen" (SINNODIUM) of the Software-Cluster (`http://www.software-cluster.com/`) and is running in the demonstrator "Vertikale Integration von Produktionsprozessen" (VI-P), which is located at the SmartFactory^{KL} (`http://www.smartfactory-kl.de/`).

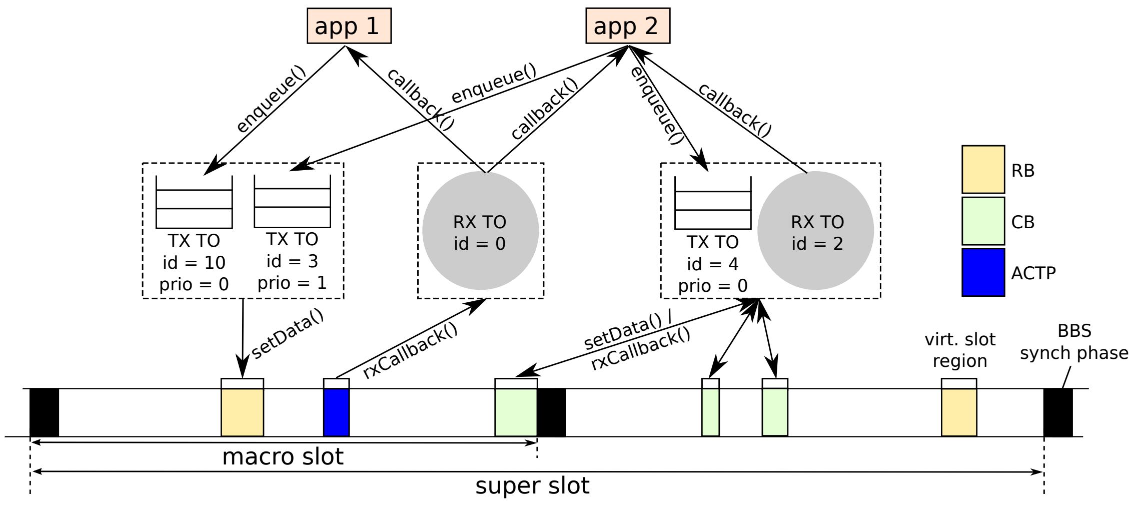


Figure 3.13: Transmission opportunities: Homogeneous protocol interface of BiPS.

or routing protocols – are not part of BiPS but use it as a service. They can be implemented manually or model-driven with SDL [BCGM14].

The execution of BiPS is controlled by two schedulers: The BiPS Application Scheduler (BAS) is responsible for executing non-time-critical parts, which are usually components in layer 3 and 4. Because BAS is irrelevant w.r.t. ACTP, it is not further described here. Time-critical protocols of BiPS like ACTP are scheduled and dispatched by the BiPS Communication Scheduler (BCS), which runs in interrupt context and has priority over BAS. The main tasks of BCS are the slotting of time and the timely (de-)activation of MAC protocols, which must occur synchronously among all communicating nodes. To build time slots, BCS subdivides time into so-called *macro slots*, where each macro slot starts with the application of BBS. MAC protocols run in so-called *virtual slot regions*, which are placed relatively to synchronization ticks of BBS within a macro slot and are associated with a single MAC protocol (e.g., ACTP). In parts of a macro slot without virtual slot region, the transceiver is put into an energy saving state. During a virtual slot region, BCS is responsible for the timely activation of the associated MAC protocol and for the protocol's deactivation at the region's end. The concrete types and placement of virtual slot regions are scenario-specific and defined by configuration. To improve the flexibility of placing virtual slot regions, macro slots can be grouped to build so-called *super slots*.

The data interface of BiPS is realized by the multiplexer, which provides a homogeneous interface to all MAC protocols and consists of so-called *Transmission Opportunities* (TOs). TOs are identified by unique IDs and are either designed for transmissions (TX TOs) or receptions (RX TOs). Their main objective is to temporally decouple higher-layer protocols from time-critical functionalities of BiPS. An example of the interplay between TX TOs, RX TOs, applications, and virtual slot regions is given in Fig. 3.13 and has the following properties: A TX TO is associated with one or more virtual slot regions. It holds a queue – either ordered by FIFO or deadlines – to store upcoming transmissions and transfers them to the responsible MAC protocol at the start of an associated slot region. After being informed about success or failure of a transmission by the MAC protocol, the TX TO passes this information to the initiator of the transmission and deletes the transmission request from its queue. If a transmission failed, it can optionally keep the transmission request in its queue and trigger retransmissions. Because

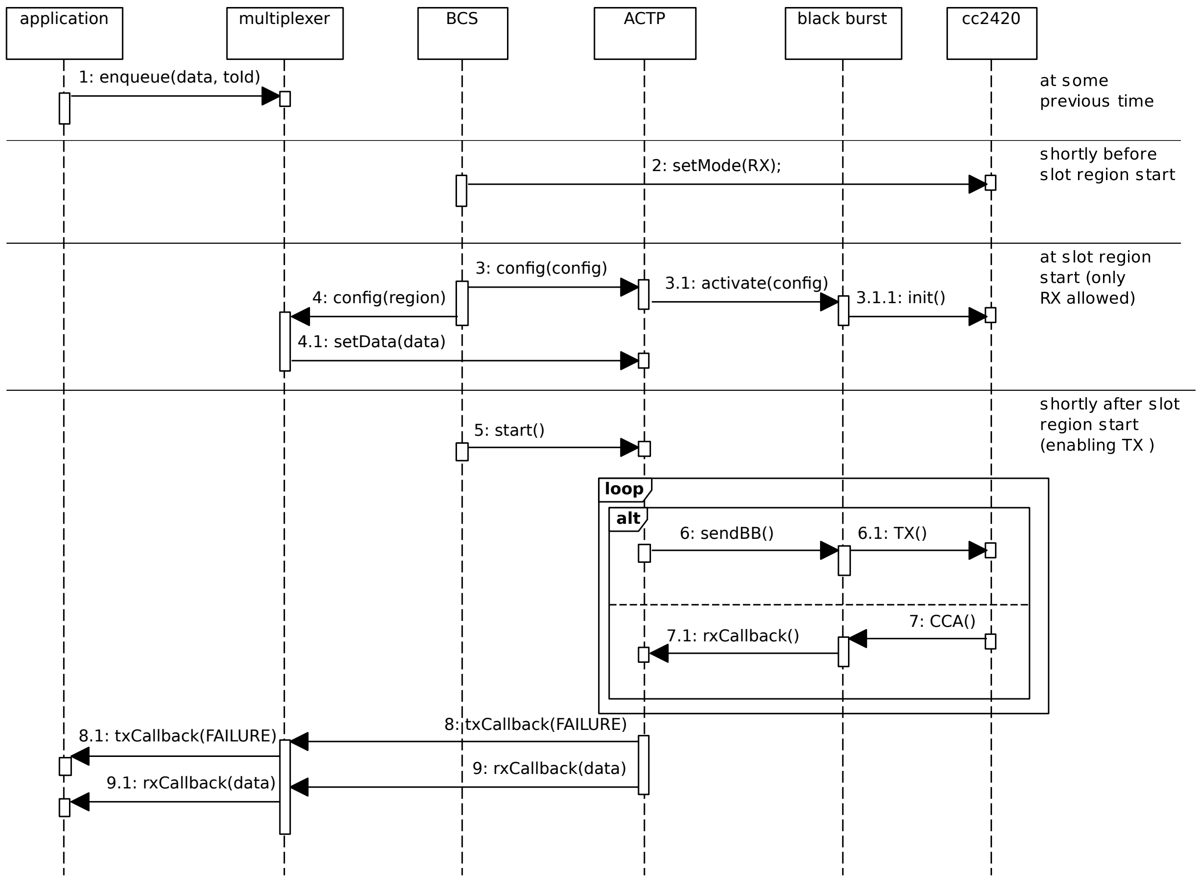


Figure 3.14: Example of an ACTP run in a virtual slot region of BiPS.

a virtual slot region can be associated with more than one TX TO, priorities are introduced to guarantee unambiguous transmission orders. Similar to TX TOs, RX TOs are also associated with one or more virtual slot regions. Frames received during a slot region are delivered by the active MAC protocol to the associated RX TO and forwarded to applications, which have to register at the RX TO in advance.

By means of a sequence diagram, Fig. 3.14 presents an example of an ACTP run in BiPS. At any time, an application enqueues a transmission request containing the arbitration sequence to transmit into a TX TO of the multiplexer. To send the sequence with ACTP, this TX TO must be associated with a virtual slot region of type ACTP. Shortly before the beginning of the slot region, BCS triggers the activation of the CC 2420 transceiver to guarantee its operational readiness. When the slot region actually starts, BCS activates ACTP, which, in turn, prepares for black burst transmissions and detections, and orders the multiplexer to provide the data of the next transmission. Shortly after the start of the virtual slot region[8], BCS triggers ACTP to start its execution, i.e., to process its bit sequence. After the protocol terminates, ACTP invokes two callbacks: The fist callback informs about the result of the ACTP run that is assumed to be `FAILURE` in the example. The second callback delivers the detected bit sequence, which has outperformed the own sequence, to the multiplexer, where it is forwarded to the application.

[8]This delay pays tribute to synchronization offset and is derived from $d_{maxOffset}$. Only after this delay, a MAC protocol is allowed to occupy the medium.

3.8 Discussion

State-of-the-practice medium access schemes for wireless systems can roughly be divided into two groups: Either contention-based with probabilistic chances of success, or reservation-based with bounded medium access delays but significant lead time and/or offline configuration. With ACTP, this chapter introduces a binary countdown protocol for wireless multi-hop networks that complements this list of access schemes. The protocol is fully distributed and implementable with customary transceivers. Due to deterministic and value-based elimination of contest, ACTP is well-suited in situations, in which a single node has to be elected dynamically but within fixed time bounds. Different from existing deterministic protocols based on TDMA or FDMA, nodes running ACTP do not require transmission schedules or knowledge about their neighborhood. Thereby, ACTP comes with very low storage requirements and is highly robust against topology changes. The advantages of ACTP's value-based and deterministic arbitration are, in particular, visible if nodes have seldom and sporadic communication demands. Here, TDMA-based solutions usually waste network resources due to overbooking. With ACTP, on the other hand, medium access can be granted when needed.

Though these examples show that ACTP has its strengths, it is no replacement of existing schemes. If, for instance, no guarantees are required, CSMA/CA-based protocols state a suitable solution with very low overhead and less requirements w.r.t. synchronization. If, on the other hand, guarantees are required but message schedules are fixed, periodical, and long-lasting, TDMA-based protocols are more adequate than ACTP w.r.t. access delay, overhead, and duty cycling[9].

In addition to presenting ACTP's mode of operation, this chapter also derives the duration of bit rounds, which represent the time that is required to propagate a single bit along one hop. In this regard, it is shown that influencing delays are constant or bounded and only depend on transceiver-specific properties, the maximal synchronization offset, and configurable protocol parameters like the length of bit sequences. Since all influencing factors are independent of the number of competing nodes, ACTP is well-suited for networks with high node density.

Besides its application with network-wide arbitration radius, the chapter furthermore presents restricted applications of ACTP that provide, for instance, a deterministic solution to the hidden station problem. Though restricted applications still ensure deterministic elimination of contest, they cannot maximize the number of winners due to the multi-hop competing problem. An additional side effect of ACTP results from its collision-protected encoding of bits, which enables cooperative data transfers in order to propagate small pieces of information with fixed transfer delays. Thereby, ACTP becomes open to further application scenarios like (network-wide) signaling.

Because ACTP utilizes black bursts, which encode recessive bits by the absence of transmissions, it is not distinguishable whether "there is no active node within arbitration radius" or "all active nodes are sending recessively". To avoid such situations, one solution is the prefixing of each bit sequence with a dominant Start-of-Frame (SoF) bit. Thus, after the first bit phase, passive nodes are aware whether there are active nodes and can switch off their transceivers if this is not the case.

[9]As shown in [CGKW10, CGKW13], reservation-based protocols with exclusive medium access achieve a very high energy efficiency compared to any contention-based protocol.

Due to the interpretation of medium occupancies, the biggest threat of ACTP – and black bursts in general – are external interferences, which can be caused by engines incidentally or by neighbored networks negligently. In this regard, ACTP is notably more susceptible than state-of-the-practice transmission techniques, where measures like spreading codes can be applied to improve their robustness. To ensure ACTP's correct mode of operation, interference sources have to be prohibited. The same holds for unstable links, which must not occur during an ACTP run but are acceptable between runs. Asymmetric links are tolerable in principle but have undesired side effects and should, therefore, also be avoided with restricted applications of ACTP. In general, these preconditions can be established by a controlled node deployment and predetermined frequency allocations.

4. Chapter

Applications of ACTP

Due to its capabilities for deterministic medium arbitration and data propagation, ACTP provides solutions to a wide range of problems of distributed systems. In the following, solutions to some general as well as control system-related problems are outlined exemplarily. Among others, these solutions comprise complete communication systems with integrated synchronization protocol and regular data transmissions. Since one of ACTP's main application domain are WSNs, energy consumption is a further topic that is addressed in the course of this chapter. In this regard, ACTP's compliance with duty cycling, which is not self-evident for wireless binary countdown protocols as substantiated by the presentation of related work in Chapter 6, is discussed.

The remainder of this chapter is subdivided into five sections: Section 4.1 discusses the combination of ACTP with synchronization protocols and duty cycling. In Sect. 4.2, applications of ACTP are presented, in which the protocol is used for arbitration. Afterwards, Sect. 4.3 outlines ACTP's application to cooperative data propagation and distributed agreements on common values. Thereafter, the utilization of ACTP in the context of a QoS routing protocol is sketched in Sect. 4.4. Finally, Sect. 4.5 concludes this chapter.

Contents of this chapter have been published in [3], [9], [14], [16], [20], and [22].

4.1 Motivation

One of the first tasks when running some binary countdown protocol is the establishment of synchronization. In this section, corresponding alternatives are presented together with their pros and cons. Furthermore, the impact of resynchronization intervals is discussed.

4.1.1 Synchronization Alternatives

In wired single-hop topologies like bus topologies for CAN [Int04], synchronization is available implicitly, since start and end of transmissions are observed directly by all nodes that are connected to the medium. Thereby, dominant SoF bits can be introduced to integrate synchronization into a binary countdown protocol inherently. In wireless single-hop networks, this mode of operation can in principal be adopted, and there are indeed instances of wireless binary countdown protocols like in [Kuh09, PAT07], which perform this kind of *on demand synchronization* by prefixing bit sequences with SoF bits and by synchronizing to the earliest detected SoF bit. However, such ad-hoc synchronizations have several drawbacks: First, additional overhead is introduced by SoF bits in each run of the protocol and nodes possibly get synchronized more often than needed, which particularly holds if the protocol runs with high frequency. Second, spontaneous synchronization is not compliant with duty cycling, because SoF bits can in principle be sent at any time and nodes can therefore not put their transceivers into an energy saving state.

In the literature (e.g., [PATR07a]), there are also binary countdown protocols for multi-hop networks with SoF-based synchronization. They are based on the flooding of dominant bits through the network and reveal further drawbacks of this kind of synchronization. One of them is caused by concurrent but time-shifted starts of SoF bit transmissions, requiring additional measures and guard times to merge SoF bits. Another drawback is w.r.t. termination, because a node receiving an SoF bit does not know the previous itinerary of the bit and consequently must always propagate the bit one hop further. Thereby, the propagation of SoF bits terminates not until the entire network is synchronized, though the actual application range of the binary countdown protocol may be much smaller and is, for instance, only two hops in [PAT07].

Instead of inherently integrating synchronization into the behavior of the binary countdown protocol, it is more efficient to separate the synchronization. In this regard, external synchronization, where nodes receive synchronization messages from an external reference clock, as well as internal synchronization, where synchronization messages are exchanged among network nodes, are possible. A famous and often adopted external synchronization solution for wireless networks is via the Global Positioning System (GPS; [US 08]), which provides synchronization with bounded offset. A representative of an internal synchronization protocol with bounded offset is Black Burst Synchronization (BBS; [GK11b, GK11a]). While synchronization offset is very small with GPS, BBS has the advantage that it comes without additional hardware and less power requirements, and can also be applied indoor.

Though every synchronization method with bounded maximal offset can be used to accomplish the required synchronization of ACTP, we argue for an internal solution with BBS. This solution is also implemented in the protocol framework BiPS (see Sect. 3.7.2.2). Originally, BBS establishes tick synchronization that is less strong than time synchronization but sufficient for ACTP. It comes in master-based, decentralized, and hybrid variants. Though all variants are adequate, the master-based or hybrid variant should be preferred due to lower synchronization offsets.[1] Because hardware clocks deviate in clock rate and drift, periodical re-synchronization is required. In BiPS, this is achieved by subdividing time into macro slots and by executing BBS at the beginning of each macro slot. Runs of ACTP are scheduled relative to synchronization ticks in ACTP-based virtual slot regions. Because these slot regions can be placed in macro slots arbitrarily, weakly periodic applications of ACTP are possible. Since with this approach, ACTP starts at pre-defined points in time only, it is, on the one hand, compliant with other MAC protocols, which can be executed in disjoint virtual slot regions, and supports, on the other hand, duty cycling by switching transceivers to an idle state in unallocated parts of a macro slot.

[1]The synchronization offset influences the duration of bit rounds in terms of $d_{maxOffset}$ and should therefore be as small as possible.

4.1.2 On the Impact of Resynchronization Intervals

A question arising when synchronizing periodically is regarding the optimal resynchronization interval. In the presented communication solutions, this interval is given by the duration of a macro slot d_{macro}. In general, if choosing a too small interval, a large amount of network resources are dedicated to synchronization. But if, on the contrary, d_{macro} is very large, the maximal synchronization offset $d_{maxOffset}$ becomes significant due to diverging hardware clocks. This, in turn, has a negative impact on the efficiency of synchronized protocols like TDMA or ACTP, which is directly influenced by $d_{maxOffset}$ as derived in Equation 3.6. On the one hand, larger values of $d_{maxOffset}$ increase the runtime of the protocol. On the other hand, they also influence the protocol's energy efficiency, which decreases due to larger guard times that must be waited by receivers to detect a potential transmission.

In [CGKW13], we derive optimal resynchronization intervals under the assumption that the considered network traffic consists of reservation-based frame transmissions in exclusive slot regions. For this purpose, the energy consumption of receivers, which enable their transceivers only to receive scheduled transmissions and for synchronization purpose, is considered. In this regard, a variable $d_{rxInterval}$ is introduced to evaluate different intervals of frame receptions. Since receivers must ensure the timely activation of their transceiver, their wake-up time before an expected frame reception directly depends on $d_{maxOffset}$. Thus, synchronization-related overhead at receivers occurs both due to the *synchronization* itself and due to *guard times* that have to be introduced to guarantee the timely operational readiness of transceivers. Their proportions of a macro slot are calculated as follows:

$$o_{guard} = \underbrace{\frac{d_{macro}}{d_{rxInterval}} \cdot d_{avgGuard}}_{\text{total guard time per macro slot}} \cdot \frac{1}{d_{macro}} = \frac{d_{avgGuard}}{d_{rxInterval}}, \tag{4.1}$$

$$o_{sync} = \frac{d_{sync}}{d_{macro}}, \tag{4.2}$$

$$o_{node} = o_{sync} + o_{guard}, \tag{4.3}$$

where d_{sync} is the convergence delay of the synchronization protocol and $d_{avgGuard}$ the average guard time per frame reception. Ignoring switching delays of transceivers, $d_{avgGuard}$ is identical to $d_{maxOffset}$, which is the case if the average synchronization offset of nodes is 0 µs.

In Fig. 4.1, a plot illustrates a node's total proportional overhead o_{node} for three reception intervals $d_{rxInterval}$ and as function of the resynchronization interval d_{macro}. Crosses on each line mark the optimal interval, i.e., the minimum of the curve. To determine o_{sync}, synchronization by BBS is assumed with TI's CC 2420 transceiver and a network of two hop diameter. The plot demonstrates that the optimal resynchronization interval highly depends on the traffic volume: If the volume is low, infrequent synchronizations achieve a better efficiency, since o_{guard} is less affected by increasing synchronization offsets. Vice versa, the portion of synchronization offset is small in relation to guard times if the communication volume is high.

In sum, there is no universal optimal resynchronization interval but the interval depends on the synchronization protocol's convergence delay, offset, and the nodes' traffic profile. Fur-

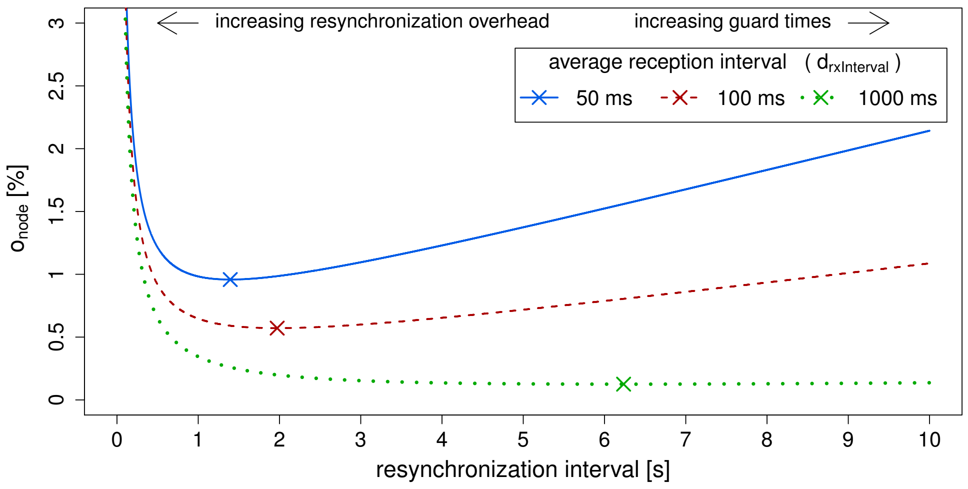


Figure 4.1: Overhead as function of resynchronization intervals and three traffic patterns.

thermore, there is usually no single traffic profile but each node has its own communication demands. Here, one possible approach is to minimize the overall overhead:

$$o_{network} = \sum_{i=1}^{n} o_{node,i}, \tag{4.4}$$

where $o_{node,i}$ is the overhead of node i. However, this approach may lead to unfair intervals, for which some nodes "pay" significantly more than others. Thus, a possibly better approach is the determination of a more balanced interval, which satisfies node-specific overheads regarding a (realizable) threshold $o_{threshold}$:

$$\forall_{i=1}^{n} o_{node,i} \leq o_{threshold}. \tag{4.5}$$

4.2 Deterministic Medium Arbitration

In this section, two applications are presented, in which ACTP is utilized as typical arbitration protocol. In this regard, Sect. 4.2.1 illustrates an ACTP-based solution to the hidden station problem. Section 4.2.2 presents the application of ACTP to a problem class of control systems, requiring network-wide and value-based arbitration to realize Maximum-Error-First (MEF) scheduling of network resources.

4.2.1 ACTP-based Elimination of Hidden Stations

By applying ACTP with an arbitration radius of two hops, ACTP also provides a solution to the hidden station problem. A corresponding configuration of medium slotting and ACTP parameters is presented in Fig. 4.2. In the shown macro slots, data slots are placed periodically and relative to the start of the macro slot. They are subdivided into an arbitration phase, where ACTP is applied with the aforementioned two hops radius and arbitration sequences of six bits length, and a data transfer phase, where the winner transmits a regular data frame. If the transmission of regular data frames is unicast, data transfer phases can optionally be dimensioned

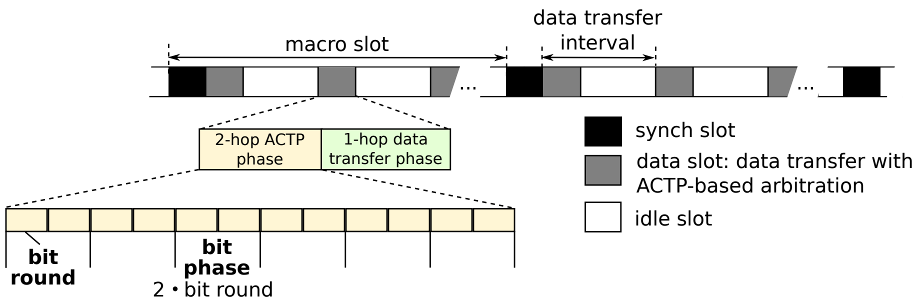


Figure 4.2: ACTP as deterministic solution to the hidden station problem.

to enable receivers the transmission of acknowledgments. W.r.t. the arbitration radius, it has to be noted that arbitration is among sensing range, whereas regular data transfers reach communication range. Usually, it is assumed that communication ranges are not larger than sensing ranges, which is also assumed here and must hold for the correct operation of the solution.

To guarantee collision-free transmissions of regular data frames, it must be ensured that arbitration sequences are nonambiguous. A simple way to achieve uniqueness is the usage of node identifiers. To improve fairness, a better solution are two-part arbitration sequences, which consist of the (unique) node id and a prefixed (non-unique) waiting indicator that is initially zero and incremented after a node loses arbitration.

Compared to RTS/CTS handshakes [Kar90], which are referenced as default solution to the hidden station problem, 2-hop arbitration with ACTP differs in several aspects: First, ACTP only allows transmissions at pre-defined points in time, whereas RTS/CTS handshakes are usually applied in unsynchronized networks and together with CSMA/CA, and can start whenever the medium is observed idle. Since data transfer phases have fixed length in the configuration of the macro slot, the presented solution furthermore causes a waste of network resources if the lengths of data frames differ. An additional drawback of the presented ACTP-based arbitration is the deterioration of the exposed station problem, because reservation encompasses two "sensing hops" around the sender and not only one "communication hop" around sender and receiver, and may furthermore suffer from the multi-hop competing problem. Besides these drawbacks, ACTP has significant advantages as solution to the hidden station problem, which are the support of duty cycling and the deterministic elimination of contest. Thus, it can guarantee that one station arises as winner within fixed time bounds, which cannot be guaranteed by RTS/CTS handshakes due to their susceptibility to collisions.

4.2.2 Try-Once-Discard – Maximum Error First Scheduling with ACTP

Its predictable behavior makes ACTP to an attractive candidate for several problems of the control systems' domain. One of these problems is the Try-Once-Discard (TOD) protocol [WYB02], whose properties can be described in a formal way and allow stability proofs of (networked) control systems. Though the theoretical background of TOD has been studied intensively, a practical realization for wireless networks has been missing. After outlining TOD, this subsection presents such a realization and shows its applicability by means of a case study.

4.2.2.1 Background of TOD

TOD has been introduced in [WYB02] and describes the dynamical scheduling of network resources according to a Maximum-Error-First (MEF) strategy. The protocol is called "Try-Once-Discard", because a node never tries to send the same sample twice but always uses fresh values. By applying MEF scheduling, TOD ensures that the sensor node reporting the greatest weighted error among all participating nodes gains access to the medium and is allowed to send its value to a controller. A significant advantage of TOD is the characterization of its properties by Lyapunov functions [NT04, CTN07]. Thereby, stability of a control system can be proven and estimates w.r.t. communication requirements can be derived.

Regarding communication requirements, two values are of particular interest for the stability of a system [HTvdWN09]: The Maximum Allowable Transmission Interval (MATI), which describes the (worst-case) sampling and communication interval, and the Maximum Allowable Delay (MAD), which limits the tolerable transfer delay between reading of samples and their delivery to a controller. A usual assumption is that MAD is not larger than MATI, which implies that one transfer is finished before the next one starts.

4.2.2.2 ACTP-based Realization of TOD

When devising a realization of TOD, protocol designers can ignore the control theoretical background and the form of the Lyapunov function. Instead, they can concentrate on the length of weighted errors and samples, and on communication requirements given by MATI and MAD, which must hold to guarantee the system's stability. In a concrete solution, these requirements must be defined by the control application and make the following demands on the communication system: First, the system must support the periodical and synchronous reading of new samples, which calls for a synchronization protocol. Second, nodes have to run an arbitration protocol, in which the node with the greatest weighted error, which is calculated as a function of the read sample, must arise as network-wide winner and is afterwards allowed to transfer its full sample value to a controller. Since contest must start simultaneously and the winner has to be determined at runtime, this demands – in addition to a synchronization protocol – a dynamical and value-based medium arbitration. Because all these steps must be performed in compliance with MATI and MAD, arbitration and data transfer must be finished within fixed time bounds, thereby rejecting all probabilistic and collision-prone protocols and turning ACTP into an attractive solution, which is additionally well-suited for multi-hop topologies.

An ACTP-based solution is presented in Fig. 4.3 [CGSW14]. Macro slots are subdivided into time slots of fixed length, which, in turn, consist of a TOD round, where a single sample is read and potentially sent to a controller if its weighted error is maximal, and an idle phase, where

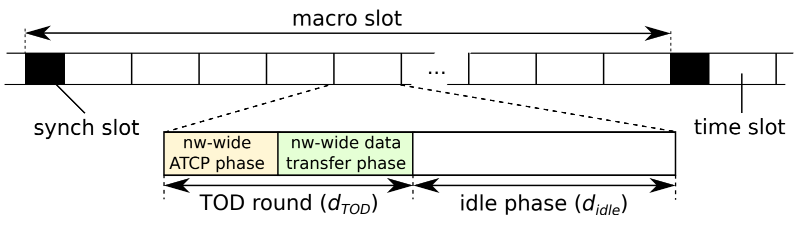


Figure 4.3: Realization of TOD with BBS and ACTP.

nodes can switch-off their transceivers to save energy. A TOD round starts with an arbitration phase, where ACTP runs with network-wide arbitration radius and with arbitration sequences consisting of weighted errors and node identifiers to break ties. The ACTP phase is followed by a data transfer phase, in which the winner transmits its sample in regular data frames with high precision. The data transfer phase is dimensioned so that the sample can be forwarded to a controller across the entire network along a pre-established route.

To fulfill a system's stability conditions, the combined duration of ACTP phase and data transfer phase must be smaller than MAD, i.e., $d_{TOD} \leq MAD$ must hold. Because this duration depends on fixed parameters, it can be calculated in advance in order to check whether MAD is met. As further condition, the sum of durations of TOD round and idle phase must not exceed MATI (i.e., $d_{TOD} + d_{idle} \leq MATI$), which can also be checked offline. To this end, the idle phase represents a regulation screw w.r.t. duty cycling and must be sized small when communication demands are high and vice versa. Since node identifiers are part of arbitration sequences and thus received by all nodes during the ACTP phase, all nodes can check early after the end of the ACTP phase whether they are part of the route between winner and controller.[2] If this is not the case, they can already switch off their transceiver during the data transfer phase to decrease their duty cycle supplementarily.

4.2.2.3 Case Study – An Unstable Batch Reactor

The unstable batch reactor in [WYB02] is a standard example for stability proofs of control systems. As shown in [JLS+10], its stability is given if $MATI = 10.65\,\text{ms}$ is satisfied. In the following, an ACTP-based communication solution to this scenario is sketched.

The solution is illustrated in Fig. 4.4 by one example TOD round and applies arbitration sequences of length $n_{bits} = 7$, comprising a weighted error of length $n_{error} = 4$ and a node identifier of length $n_{id} = 3$. The network diameter is assumed to be $n_{maxHops} = 3$, which is also used as arbitration radius n_{hops}. In the shown TOD round, both V_d and V_g hold the same maximal weighted error (14). Thus, they are both still in contest after ACTP's first four bit phases. Since V_d's identifier is larger, V_g gets defeated during the last three bit phases when ACTP processes node ids. Therefore, V_d is the only winner at the end of the ACTP phase and is – under consideration of synchronization inaccuracy and guard times – allowed to transmit its sample in the subsequent data transfer phase. Due to network-wide arbitration, this sample is forwarded by V_e and V_g immediately and without further arbitration until it finally arrives at controller V_h.

To check whether the solution meets the scenario's stability requirements, the durations of ACTP phase (d_{actp}) and data transfer phase (d_{data}) have to be calculated and compared with MATI. In [CGSW14], we computed these delays for Atmel's AT86RF230 transceiver [Atm09], which is similar to TI's CC 2420 transceiver [Tex07] but has some beneficial properties w.r.t. switching times and CCA delay. With this transceiver, Equation 3.8 yields $d_{actp} = 6.405\,\text{ms}$.

The duration of data transfer phase is calculated according to

$$d_{data} = d_{frame} \cdot n_{hops} + (n_{hops} - 1) \cdot d_{IFS} + 2 \cdot d_{maxOffset}$$

and results in $d_{data} = 2.184\,\text{ms}$ under the assumption of 10 bytes regular payload. In total, this yields a TOD round duration of $d_{TOD} = d_{actp} + d_{data} = 8.589\,\text{ms}$, which satisfies MATI and leaves

[2]This additionally assumes that nodes are aware of all routes, in which they participate.

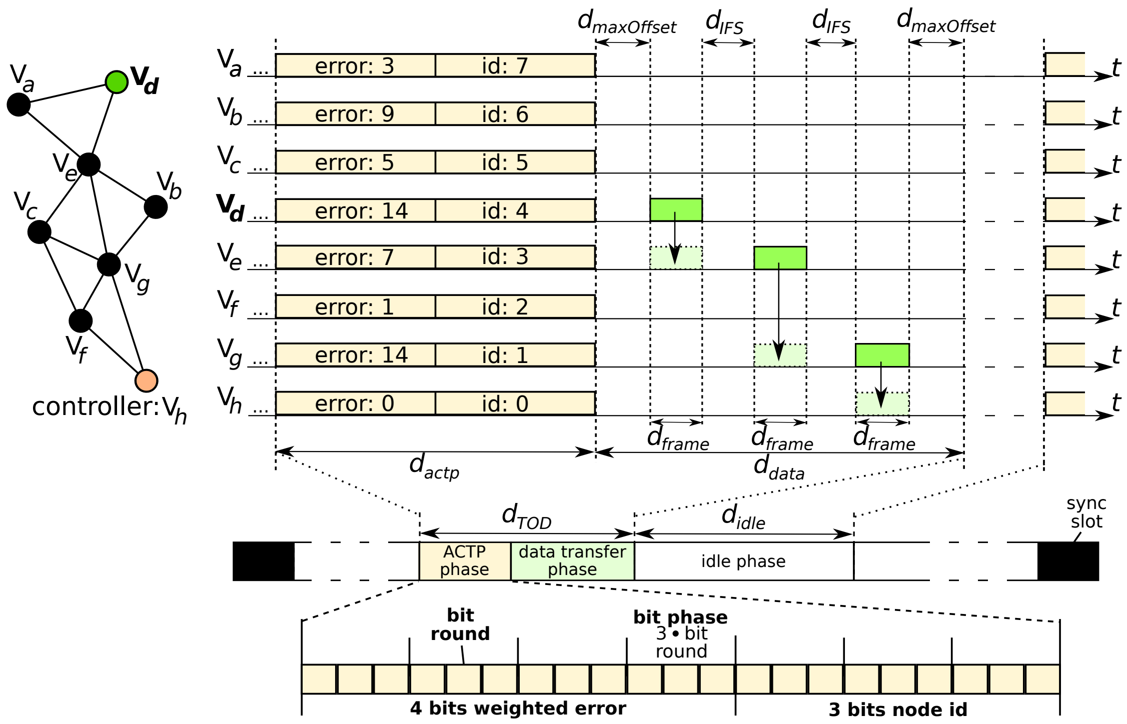


Figure 4.4: Example TOD round in the case study ($n_{hops} = 3, n_{bits} = 7$).

$MATI - d_{TOD} = 2.061$ ms space in between TOD rounds for synchronization and energy saving. These gaps are actually very small, thereby offering only little opportunities to save energy. They are also too small to host a full synchronization phase of BBS, which consequently has to be subdivided into n_{hops} synchronization rounds and distributed over the macro slot.[3]

4.3 Distributed Agreement on a Common Value

In this section, ACTP's support for distributed voting and value transfer is illustrated by several example scenarios. These scenarios include cases, where several nodes run ACTP with different bit sequences and have to agree on a common value, as well as cases, in which ACTP's cooperative mode is applied to distribute values within guaranteed time bounds.

4.3.1 Network-wide Leader Election

In communication systems, leader election is a recurring task, e.g., to select coordinating nodes, synchronization masters, or cluster heads. Due to deterministic and multi-hop-spanning elimination of contest, ACTP provides an attractive solution to network-wide leader election. Applying ACTP to the election of local leaders requires special attention due to multi-hop competing problems as outlined in Sect. 3.5 and is not further discussed in this section.

An example configuration for ACTP-based network-wide leader election with networks up to a diameter of $n_{maxHops} = 4$ hops is illustrated in Fig. 4.5. In the presented slot configuration,

[3]Since the scenario is extremely demanding, it is hard to realize with hardware of WSNs. In this regard, it should also be noted that the requirements cannot be satisfied with TI's CC 2420 transceiver.

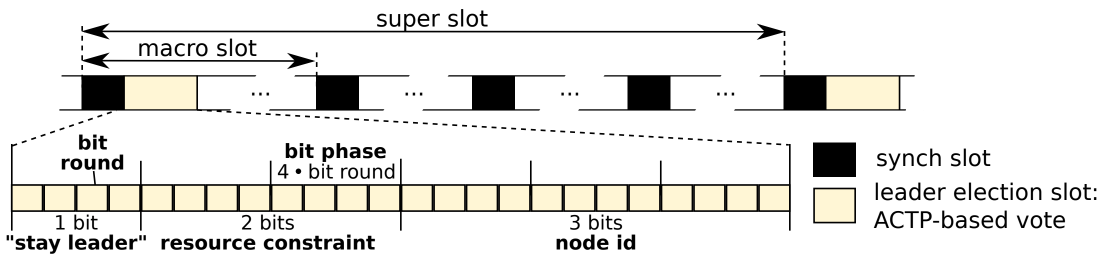


Figure 4.5: Network-wide leader election with ACTP.

super slots consisting of four macro slots are introduced and leader election takes place in the first macro slot only to reduce overhead and to run leader (re-)election with desired intervals. The sent arbitration sequences have a length of $n_{bits} = 6$ bits and are subdivided into three parts: The first bit is called *stay leader* and can only be set dominantly by the current leader to enforce its reelection. If the current leader does no longer candidate, this bit is recessive, thereby giving all other participating nodes a chance to win. As positive side effect, leader node failures are detected implicitly if this bit is recessive. The next two bits are referred to as *resource constraint* and enable prioritization of nodes with more (remaining) hardware resources. This can, for instance, also capture energy resources of battery-powered nodes, where a node with fresh batteries starts with 11_2 and decreases the value when energy resources diminish. The last part contains node identifiers to break ties and to guarantee uniqueness of bit sequences.

Since the arbitration radius corresponds to the network diameter, all nodes are aware of the winner's identifier at the end of the ACTP phase. Furthermore, they know the distance to the new leader (see Sect. 3.3.2 about implicitly available information) as long as every node votes for itself only. This limitation is yet not necessary and nodes can also vote for other nodes, though it has to be ensured that the voted node is actually available and able to run as leader. In this case, the voted node learns about its election by means of the monitored bit sequence.

4.3.2 Time Synchronization on Top of Tick Synchronization

Though tick synchronization is weaker than time synchronization, it is sufficient for ACTP and many other functionalities of distributed systems like synchronous duty cycling and medium slotting. However, for many application-level functionalities – like timestamping of events –, time synchronization is required, which can be built on top of tick synchronization by associating synchronization ticks with time values. Thereby, time synchronization is established with the same accuracy as tick synchronization. Corresponding extensions of BBS have been proposed in [CGK09, GK11a].[4]

Since BBS and its extensions are internal synchronization protocols, it is reasonable to introduce *virtual* clocks, which are independent of external time sources in the first instance but allow conversion into external time if required. Their values can be subdivided into a macro tick value and a micro tick value. Macro tick values are coarse-grained and incremented with each tick, i.e., at the beginning of each new macro slot. Thus, their physical equivalence depends on the duration of macro slots. Micro tick values, on the other hand, are fine-grained, increase

[4]For the cooperative exchange of time values, [CGK09, GK11a] incorporate a different protocol, whereas the solution below applies ACTP.

relative to the last synchronization tick, and are reset to 0 when macro ticks increase. Their physical unit is in principal unrestricted but is limited by the granularity of hardware clocks in practice. With the Imote 2 platform (see Appendix A), micro ticks of 1 µs can be achieved, whereas a reasonable (but not prescribed) granularity for macro ticks is 1 s.

While micro tick values are calculated by each node independently, macro tick values must be communicated in the network. In a stable network, it is in general sufficient to communicate them once during the network's initialization. However, in real-world scenarios, nodes may also join the network later, thereby asking for the periodical exchange of macro tick values. In this regard, a trade-off between overhead and a node's entry delay has to be found. To ensure that deterministic tick synchronization turns into deterministic time synchronization, exchange of macro tick values must be reliable and with bounded transfer delay, thereby making ACTP with network-wide propagation an attractive candidate.

A corresponding example configuration of super slots and ACTP parameters is presented in Fig. 4.6 for a network diameter of $n_{hops} = n_{maxHops} = 3$ hops. In the example, a super slot consists of 128 macro slots and macro tick values are exchanged in the first macro slot only. ACTP's arbitration sequences have a length of 28 bits and are divided into three parts: The *master's vote* bit can only be set dominantly by a master node to enforce the propagation of its macro tick value and reflects the master-based variant of BBS. The second part contains the macro tick value that is associated with the last synchronization tick, i.e., with the tick of macro slot 0. The last part completes the bit sequences with a checksum of two bits to avoid the acceptance of corrupted macro tick values. Retransmissions in case of transmission errors are not scheduled but nodes have to wait until the next regular exchange of the macro tick value.

If a master is available in the network, the first bit of detected arbitration sequences is dominant. Thereby, all nodes but the master refrain from the transmission of their macro tick value and the ACTP run turns into a quasi-cooperative transfer with a single data source. If, however, the master node fails, the first bit is recessive and all nodes continue sending their macro tick value. As a consequence, they determine the maximum of all sent values, thereby potentially skipping values but never setting the virtual clock to the past. Because macro tick values are sent every 128th macro slot only, new nodes have – assuming a macro slot duration of 1 s – to wait up to $\sim$ 2 minutes before being time-synchronized. However, this drawback is put into perspective by two advantages w.r.t. overhead: First, network resources are saved, because the number of ACTP runs is reduced. Second, the length of arbitration sequences can be decreased, because the last 7 bits of the macro tick value do not need to be transferred. Thus, macro tick values can have a length of 32 bits, though only 25 bits are propagated with ACTP.

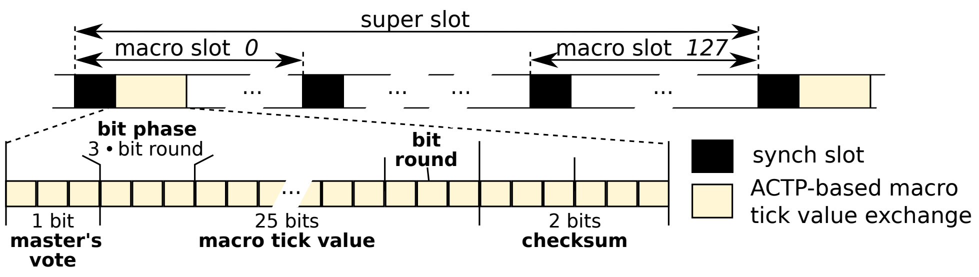


Figure 4.6: Exchange of macro tick values with ACTP.

4.3.3 Signaling of Network Mode Changes

A further possible application of ACTP is the signaling of changes of the operation mode of a network. Such an application is deployed in the VI-P demonstrator of the joint project SINNODIUM, where two operation modes are supported by the network: A *calibration mode*, in which the topology of the network is determined [Kra13a], and an *on-duty mode*, in which sensor and actuator services are provided via a middleware [Kra14]. Since both modes differ in the communication demands of nodes, each mode runs with a particular slot configuration: In calibration mode, transmission slots are assigned such that all nodes, whose id is in a preconfigured range, own an equal number of slots. In the on-duty mode, slotting is different and according to the service-dependent communication demands.

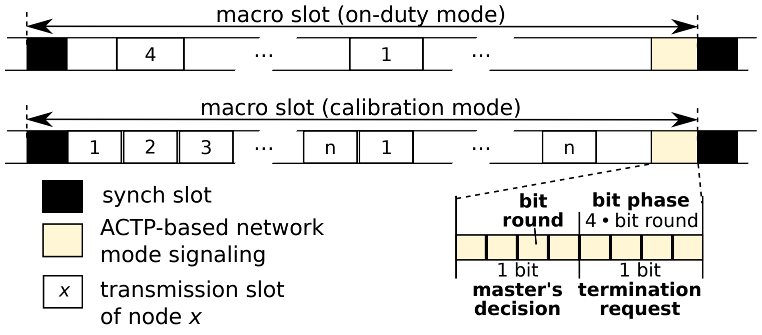

macro slot (on-duty mode)
4
...
1
...
macro slot (calibration mode)
1
2
3
...
n
1
...
n
synch slot
ACTP-based network mode signaling
x
transmission slot of node x
bit round
bit phase
4 • bit round
1 bit
master's decision
1 bit
termination request

Figure 4.7: Signaling of network mode changes with ACTP.

The calibration mode is only required during the initialization phase until all mandatory nodes are available and the network is stable. To terminate this phase and to switch between calibration and on-duty mode in an atomic way, virtual slot regions with ACTP are introduced in both slot configurations. Figure 4.7 presents this setup for a network with $n_{maxHops} = 4$ hops diameter. In both macro slot configurations, ACTP is located at the end of the macro slots and runs with arbitration sequences of 2 bits length and a radius of $n_{nops} = n_{maxHops}$. The first bit of the sent sequences is reserved for the master of the network, which triggers the termination of the initialization phase and the change to the on-duty mode by sending this bit dominantly. The second bit is used by all non-master nodes to request the termination of the calibration mode. For this purpose, a non-master node sends a dominant bit if the network is not yet seen stable and recessive otherwise. Thus, the second bit is detected dominantly on the medium as long as some node monitors unstable network conditions. Though nodes can detect whether all other nodes see a stable network by the second bit, this bit is not evaluated to trigger changes to the on-duty mode. Instead, nodes switch to on-duty mode only if the master sends a dominant first bit. Hence, the master can enforce a mode change if the network does not seem to become stable or – the other way round – delay the termination of the initialization phase even under stable network conditions if mandatory nodes have not yet joined the network.

4.3.4 Network-wide Acknowledgments with ACTP

In wireless systems, existing ARQ schemes for uni-cast transmissions can usually not be applied to confirm broad- or multi-cast transmissions, since regular ACK or NAK frames would collide destructively. By encoding acknowledgments in arbitration sequences, ACTP enables a way to remove this limitation and to make overlapping confirmations possible. This approach is not only applicable to confirm single-hop data transfers, but can also be applied to multi-hop or network-wide data transfers, which are, for instance, performed by ACTP. In this regard, see also Sect. 3.6.2 about ACTP and error control.

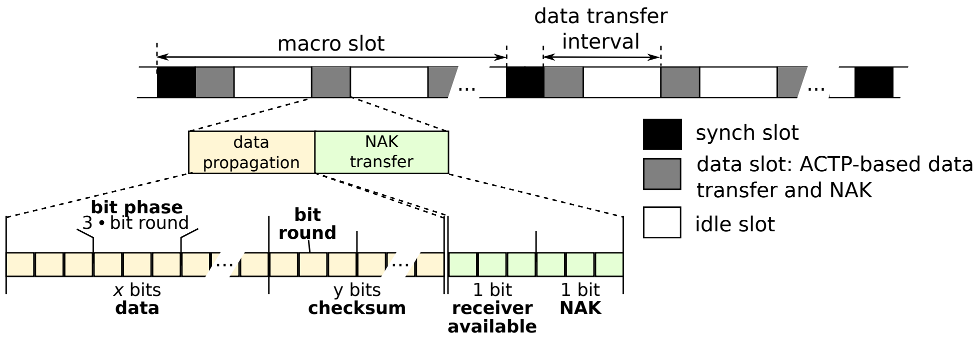


Figure 4.8: Realization of network-wide negative acknowledgments with ACTP.

A reasonable example configuration of a corresponding macro slot is illustrated in Fig. 4.8. Here, data is transferred in periodical data slots with ACTP and 3 hops radius. Immediately after the ACTP-based data propagation, a second ACTP run is scheduled with bit sequences of 2 bits length and the same arbitration radius. Due to a checksum that is appended to the data propagated in the first ACTP run, receivers can check whether the received data is corrupted and can send an accordant confirmation in the second ACTP run. In more detail, the bit sequences of the second run are as follows: The first bit is set dominantly by all receivers, thereby informing the senders of the data that there is actually a node listening. Though this information is not necessary w.r.t. error control, it is in some cases a useful information for senders that is also not available with conventional multi-/broadcast transmissions. The second bit of the sequence represents the actual error flag with a negative acknowledgment semantics, i.e., it is dominant if an error has been detected. Thus, senders of the first ACTP run are informed about a transmission error as soon as (at least) one receiver sends a dominant second bit in the second ACTP run.

As long as radii are network-wide, the propagation of data in the first ACTP run can occur both with ACTP's cooperative as well as arbitrating mode. However, implications of errors are more far-reaching if ACTP is applied in arbitration mode and if nodes participate with different bit sequences, because detected errors do not only allude to data errors but can also imply arbitration errors with wrong or even multiple winners. In this case, retransmissions can reveal significantly different winners and bit sequences.

4.4 ACTP as Communication Primitive of a QoS Routing Protocol

Several applications of ACTP can be found in the routing protocol *Black Burst-based Quality-of-Service Routing* (BBQR) [BBCG11]. The protocol's objective is to discover a shortest path between a source and a destination satisfying a desired QoS specification. By serializing concurrent route discoveries and avoiding destructive frame collisions, BBQR guarantees route discoveries with constant delay. Since the full BBQR protocol is outside the scope of this thesis, the following paragraphs provide only a brief overview of its behavior, which is described by four phases (see also Fig. 4.9):

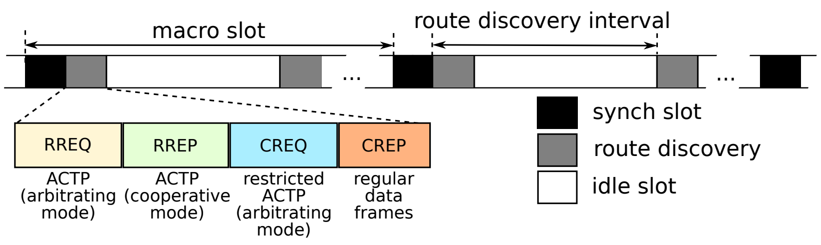


Figure 4.9: Application of ACTP in BBQR [BBCG11].

The first phase is called *Route REQuest selection* (RREQ) and determines the source node of the upcoming route discovery. For this purpose, one ACTP run is performed by all nodes with open route requests and with network-wide arbitration radius. Thereby, all nodes participating actively during this ACTP run become potential sources of the route discovery. To guarantee a unique winner, the identifiers of the nodes are included in the arbitration sequences. Furthermore, QoS requirements and the destination's identifier are included in the sequence to inform the destination about its role in the route discovery.

In the second phase (*Route REPly* (RREP)), ACTP's cooperative mode[5] is used to discover all routes fulfilling the QoS requirements. This phase is initiated by the destination node and consists of several ACTP runs traveling hop-by-hop towards the source. The number of ACTP runs depends on a pre-configured parameter $n_{maxRouteLength}$, which defines an upper hop bound on tolerable route lengths. Since only nodes that can fulfill the QoS requirements start an ACTP run during this second phase, only feasible routes are found.

The third phase is called *Construction REQuest* (CREQ) and selects one of the feasible routes. For this purpose, ACTP is applied – starting at the source node – at most $n_{maxRouteLength}$ times with restricted arbitration radii of 2 hops. After the destination receives the CREQ bit sequence, a feasible route has been selected. Phase four (Construction REPly (CREP)) informs finally all nodes along the route about node identifiers of their successors. Since there is no contest anymore, regular data frames are used.

[5]Actually, [BBCG11] incorporates a different cooperative protocol but ACTP would work as well.

4.5 Discussion

In the literature, there are many wireless protocols requiring synchronization but do not state how to achieve it or incorporate external solutions like GPS [US 08]. This statement also holds for many existing binary countdown protocols (see Chapter 6.2 for a comprehensive comparison). In this chapter, we revisit ACTP by presenting a complete communication solution with macro slots, integrated synchronization, and ACTP-based virtual slot regions. Thereby, we show that ACTP can be combined with an internal yet independent synchronization protocol, which satisfies demands on network-wide synchronization with bounded offset and comes without additional hardware requirements. Furthermore, the presented solution enables the flexible scheduling of ACTP runs at pre-defined but configurable positions within macro slots.

Since synchronization happens independently from runs of ACTP, compliance with duty cycling is achieved, which is a precondition for every protocol of WSNs. Though suggested periodical (re-)synchronizations seem to increase overhead if the the protocol runs very rarely, they often cause no extra costs due to other distributed functionalities requiring synchronization like TDMA. Furthermore, the overhead is even lower if the protocol runs frequently. In this regard, it can be observed that existing binary countdown protocols with on demand synchronization (see e.g., WiDom [PAT07] in Sect. 6.2.5) suffer from much higher overhead due to too frequent and inefficient synchronizations.

Regarding duty cycling in wireless networks, Ye et al. identified four sources of energy waste [YHE02, YHE04]: *Overhearing* (receiving data that is addressed to a different node), *collisions* (losing data due to destructive overlap of transmissions), *idle listening* (listening to the medium while there is no transmission), and *control (packet) overhead* (transmission/reception of management traffic). In relation to these sources, ACTP-based arbitration first of all "wastes" energy due to control overhead caused by costly black burst transmissions. However, this is the price to pay in order to reduce the other sources of energy waste: Overhearing by giving nodes the opportunity to switch off their transceiver after receiving the winners' bit sequence, idle listening by running ACTP in weakly periodic virtual slot regions only, and, in particular, collisions by avoiding destructive overlap entirely.

While the deterministic solution of contest is one of ACTP's biggest advantages, its high dependence on synchronization reveals the protocol's major drawback. Thus, ACTP runs must only start at pre-determined points in time and with pre-configured parameters. A further drawback of ACTP is disclosed by the case study in Sect. 4.2.2 and concerns the transmitted data volume, which is significantly lower than data rates of regular transmission techniques.

Though these drawbacks prevent ACTP's usage as general purpose protocol, the seven applications in this chapter illustrate the actual target domain of ACTP: Deterministic arbitration and value propagation. In applications, in which ACTP is used for arbitration, it is shown that ACTP resolves contest value-based, deterministically, with bounded delay, and across multiple hops. Further applications illustrate ACTP's capabilities to encode payload by black bursts, which is utilized for collision-protected data transfer and distributed value agreements. Thus, ACTP is not only an arbitration protocol but provides an interface for reliable multi-hop data propagation, which outperforms collision-prone communication schemes in dense networks. Due to its configurability w.r.t. bit sequence length and arbitration radius, the protocol is not specific for a single scenario but can be tailored to particular needs and network properties.

5. Chapter

Experimental Evaluation of ACTP

To provide evidence of their benefits and correctness, communication protocols – and software systems in general – require thorough testing and assessments. This, particularly, holds for protocols adopting relatively unusual communication primitives like black bursts. In this chapter, results of evaluations are presented to provide this evidence for ACTP.

While the overall objective of this chapter is the provision of a proof-of-concept for ACTP, its main interest is in the evaluation of black bursts. In this regard, their collision resistance receives particular attention, since it is the key foundation of ACTP's deterministic behavior. However, evaluations of ACTP are presented as well with the focus on the protocol's correctness w.r.t. arbitration and value transfer. Though simulative evaluations would have the advantage of being fully reproducible, this chapter goes for experimental evaluations with Imote 2 platforms (see Appendix A). The reasons for this decision are twofold: First, properties of simulation models may possibly deviate from the real world in some important points, thereby reducing the expressiveness of the results. Furthermore, evaluating ACTP and black bursts by experiments enables to investigate how they perform in dynamic and partially uncontrollable environments, in which, for instance, sources of external interference exist.

The rest of this chapter is structured as follows: Section 5.1 motivates the need of experimental evaluations. In Sect. 5.2, the reliability of black burst transmissions is assessed and compared to regular MAC frame transmissions. Section 5.3 presents results of a series of outdoor experiments with the focus on overlapping black burst transmissions. Section 5.4 proves the optimization from Sect. 2.4.3 regarding the accuracy of black burst detection. In Sect. 5.5, an indoor experiment is presented to validate ACTP's implementation and to provide evidence of the protocol's benefits. Finally, Sect. 5.6 draws conclusions.

Some of the presented results have been published in [8], [16], and [22].

5.1 Motivation

A common way to evaluate communication protocols is by the usage of network simulators like ns-2[1], ns-3[2], OMNeT++[3], or TOSSIM[4]. During their first days, these simulators supported only very simple wireless propagation models like models, whose communication success is influenced by node distances only. But as, e.g., shown in [CM10], the correlation between distance and RSS is sometimes not very high – in particular, in indoor environments with small

[1] http://www.isi.edu/nsnam/ns/

[2] http://www.nsnam.org/

[3] http://www.omnetpp.org/

[4] http://www.tinyos.net/

distances –, thereby disqualifying such simple models for realistic evaluations. However, current simulator versions also support more sophisticated models that consider noise and signal variability by means of statistical approaches. In this regard, a famous example is the shadowing model, which can, for instance, be found in ns-2 and respects signal variations by Gaussian random variables. To improve simulation accuracy further, a current trend is the derivation of statistical noise and interference models from empirical data. The TinyOS SIMulator (TOSSIM), for instance, supports the Closest-fit Pattern Matching (CPM) algorithm [LCL07], which uses signal strength traces from real-world scenarios to derive such models.

Yet, not only the fidelity of medium and propagation models has been improved over the last years, but also the simulation of hardware could be enhanced. This includes wireless transceivers, where much attention has been paid to TI's CC 2420 transceiver [Tex07] due to its popularity. As a consequence, there are now a couple of CC 2420 models – e.g., for ns-3 [IG13, Gro12] and TOSSIM [SJK07] – providing a detailed model of the transceiver and IEEE 802.15.4-based communication. They support, for instance, the configuration of synchronization words, CCA thresholds and hysteresis, as well as power and channel settings.

In this chapter, we do not apply simulations but prefer real-world experiments in partially noisy and dynamic environments with the Imote 2 platform [MEMara]. Though experiments on the wireless channel are usually not completely reproducible due to measurement inaccuracy and low control over noise and interference, they should be favored, since simulators do still not fully coincide with reality and black bursts depend on some physical properties that are only insufficiently captured by simulators. Examples of such simulator shortcomings are the neglect of spatial correlation of noise and interference in CPM [LCL07] and the missing accumulation of overlapping interferences in [IG13], which make these medium and transceiver models unsuitable for the evaluation of the collision resistance of black bursts.

Another aspect that argues for experiments is the missing or inaccurate consideration of some important hardware characteristics in simulation models. In [Eng13, ECG14], it is, for instance, shown that the switching times of the CC 2420 transceiver are slightly larger than the corresponding data sheet values and that delays of hardware interrupt handling must not be neglected. These delays are, however, often not respected in models, since they are simplified and/or derived from data sheets. Thus, experimental evaluations are indispensable to validate the realization of the timing constraints of ACTP.

5.2 Indoor Experiments with Black Bursts

In this section, results of two small-scale experiment series are presented to provide a proof-of-concept for black burst-based communication (Sect. 5.2.1) and to relate black bursts to regular data frames (Sect. 5.2.2). All experiments were conducted in an office environment without validity of the single-network property.

5.2.1 Reliability of Overlapping Black Burst Transmissions

For black burst-based communication protocols, it is crucial that black burst transmissions are actually collision-resistant. In the following, a lightweight evidence of this property is provided by provoking overlapping transmissions of black bursts.

5.2.1.1 Experiment Setup

The assembled topology is illustrated in Fig. 5.1 and consisted of one master and two slaves. The master synchronized both slaves by periodically sending beacons, which were realized by regular IEEE 802.15.4-compliant MAC frames [Ins11], in 1 s intervals. To enable filtering of failed synchronization rounds in a post-processing step, each beacon carried a round number that was recorded by slaves. BBS was not incorporated into the scenario, because beacon-based synchronization is sufficient in single-hop networks and achieves better accuracy.[5]

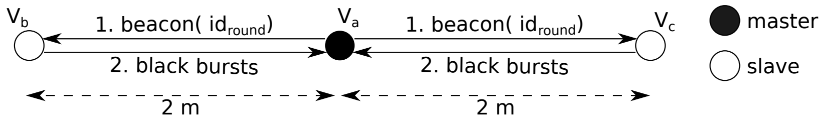


Figure 5.1: Topology to validate the collision resistance of black bursts.

In each synchronization interval, both slaves sent 40 black bursts in intervals of 20 ms simultaneously and with maximal transmission power of 0 dBm. Apart from beacons, the master did not send any data but listened for black bursts on the medium and reported them to a connected PC. In total, the experiment ran for 5,000 synchronization rounds, but due to missed beacons by one or both slaves, three rounds had to be invalidated during post-processing.

5.2.1.2 Experiment Results

The number of detected black bursts is presented in Fig. 5.2 for the first 300 synchronization rounds. Though the plot illustrates a high success rate, it also shows synchronization rounds with false negatives, i.e., where sent black bursts are lost. In the first hundred rounds, there are five of them, which are distributed over five synchronization rounds, yet also rounds with more missed black bursts can be observed at the end of the excerpt. Over the entire runtime of the experiment, synchronization rounds with up to six false negatives were monitored, and the overall detection ratio adds up to 99.1%.

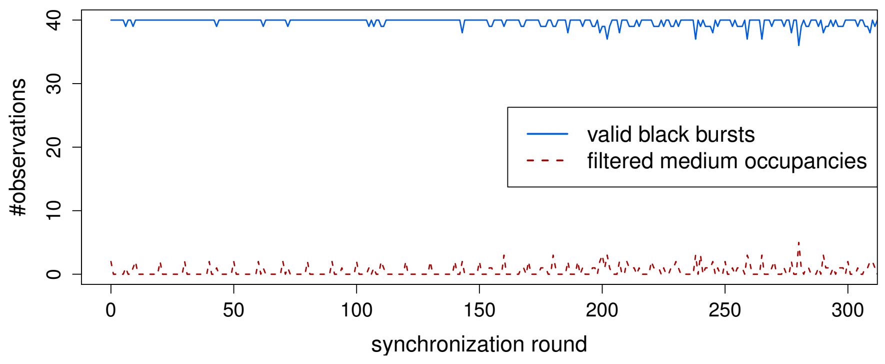


Figure 5.2: Detection ratio of black bursts and filtered invalid medium occupancies.

[5]According to the data sheet of the CC 2420 transceiver, accuracy of SFD-based synchronization is about 3 µs, whereas the BBS version, which was available at the experiments' time, achieved 128 µs base offset. In the meantime, BBS has been optimized and now achieves similar accuracy for single-hop synchronization [Eng13, ECG14].

Analyzing the origin of these faults reveals that for many false negatives the master could indeed detect medium occupancies but had to filter them out, because their duration did not coincide with the duration of valid black bursts (see Sect. 2.3.2 and the number of “invalid” medium occupancies in Fig. 5.2). Since we observed significantly less false negatives in experiments without external interference (see Sect. 5.3), the results manifest that external interference does not only increase the obvious risk of false positives but also of false negatives due to distortion of medium occupancies. In particular, the influence of external interference was in the presented experiment higher on false negatives than on false positives, since only two unexpected black bursts were observed during the experiment's runtime.[6] As a lesson learned from the relatively high number of false negatives, the results substantiate the necessity of a valid single-network property, which was not given in this experiment.

5.2.2 Black Burst-based Communication vs. Regular Data Frames

This section investigates differences between transmission ranges of black bursts and regular MAC frames. The results lay the foundation for the adjustment of both ranges by calibrating the CCA threshold of transceivers (see also Sect. 2.4.2).

5.2.2.1 Experiment Setup

The used topology is shown in Fig. 5.3 and consisted of two nodes that were positioned with a distance of 0.5 m. Different from their usual realization with shortened frame length and random synchronization word, black bursts were implemented as regular and IEEE 802.15.4-compliant frames to enable their additional reception as MAC frame, which was possible due to the absence of overlapping transmissions. The receiver's role was taken by node V_a, which recorded the reception of a black burst if a corresponding medium occupancy was detected by the CCA mechanism and/or the reception of a MAC frame if the SFD was detected and the frame's checksum was correct. The transceiver's CCA threshold was set to -97 dBm, which is very low and was just above the noise floor level.

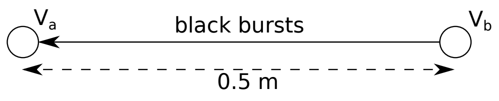


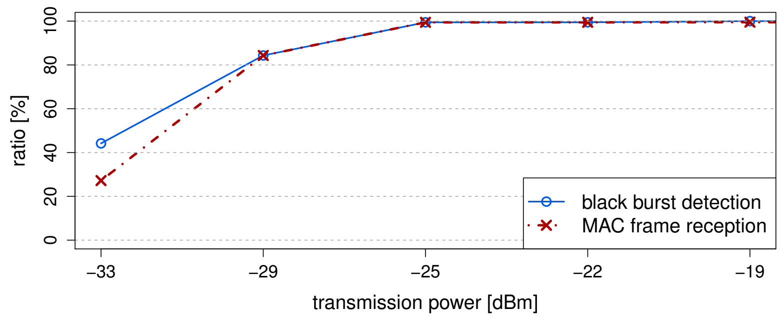
Figure 5.3: Topology to compare ranges of black bursts and regular MAC frames.

The distance between both nodes was kept constant but the output power for transmissions was varied to emulate RSS changes. Due to the small distance between sender and receiver, experiments ran only with power levels $\{1, \ldots, 9\}$, ranging from about -33 dBm to -12 dBm. For each power level, 30,000 transmissions were performed with intervals of 20 ms.

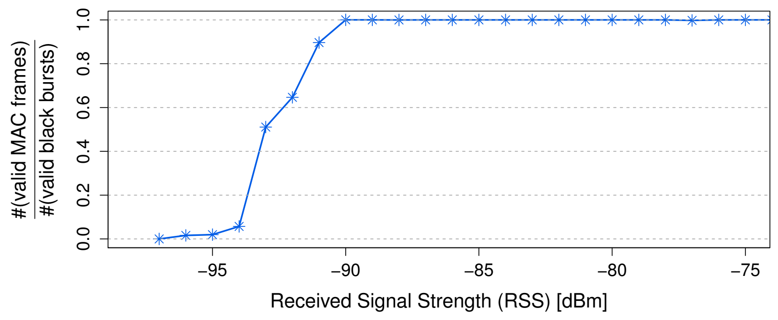
5.2.2.2 Experiment Results

The first plot in Fig. 5.4(a) presents the reception ratios of black bursts and MAC frames for five transmission power levels. With an output power of -33 dBm, 44% of all transmissions are detected as black bursts, whereas only 27% of them are received as MAC frames. Starting

[6] I.e., the master interpreted two medium occupancies as black bursts without any black burst transmission.



(a) Detection/Reception ratios of black bursts and MAC frames as function of transmission power.



(b) Ratio between number of detected black bursts and received MAC frames as function of RSS.

Figure 5.4: Comparisons between black burst and MAC frame transmissions.

from an output power of -25 dBm, both ratios are almost 100%. Though the gap is smaller for larger power levels, the detection ratio of black bursts is never below the reception ratio of regular MAC frames. Interestingly, detection ratios of black bursts even become 100% for an output power of -22 dBm (and higher), whereas reception ratios of regular MAC frames are only about 99.5%. Thus, black burst transmissions are more reliable in this experiment, yet with smaller entropy (1 bit in black bursts vs. several bytes in MAC frames). In summary, the results confirm the usual assumption that sensing ranges are larger than communication ranges. Since the sensing range can actively be adjusted by changing the transceivers' CCA threshold, this statement is, however, not universal, but should be understood as "configurations are feasible (and usual), in which the sensing range is larger than the communication range".

As implied by Fig. 5.4(a), black bursts can be detected for RSS values, at which regular MAC frames are received corrupted or not at all. To investigate this issue in more detail, Fig. 5.4(b) plots the ratio between correctly received MAC frames and correctly detected black bursts as function of RSSs. For low RSSs, the plot demonstrates that the number of detected black bursts is indeed higher, since the ratio is clearly smaller than 1. Only with RSSs of -90 dBm and higher, the ratio becomes roughly 1. Thus, a CCA threshold of -90 dBm seems to be adequate to equalize the transmission ranges of black bursts and MAC frames. For some applications like BBQR (see Sect. 4.4), this is a desirable property, yet other applications (e.g., BBS) benefit from larger black burst ranges. In this regard, it has, however, to be considered that simultaneously sent black bursts can overlap constructively, thereby increasing their power level and transmission range. This fact is investigated in the next section.

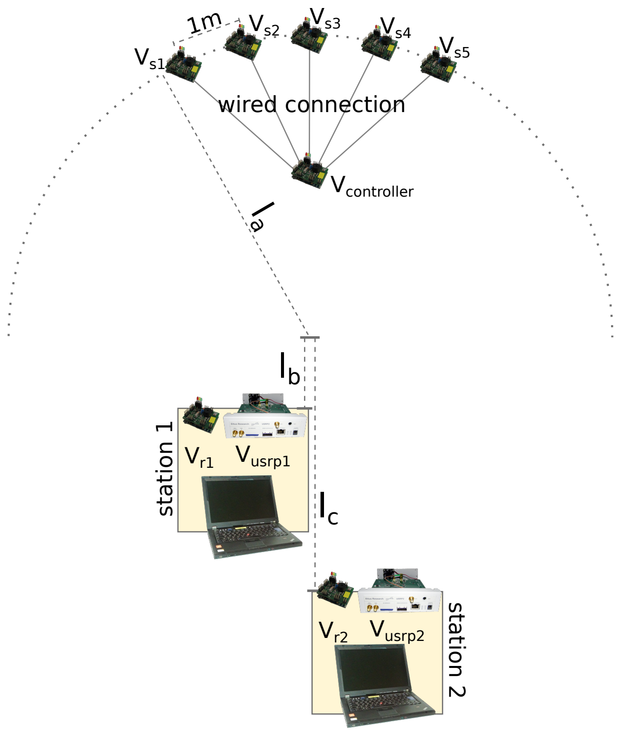
5.3 Outdoor Experiments with Black Bursts

This section presents evaluation results of a larger series of outdoor experiments. The focus of these experiments was on detection ratios of dominant and recessive bits, false positives/negatives, and accuracy of black burst detections. The experiments were repeated with different numbers of simultaneous transmitters to validate the collision resistance of black bursts.

5.3.1 Experiment Setup

The experiments took place in a semi-controlled environment and with line-of-sight between all nodes. The validity of the single-network property could be achieved by running the experiments on a forest parking lot outside the range of interfering networks. In total, eight Imote 2 nodes were deployed and placed as shown in Fig. 5.5. Up to five of them ($V_{s1},\dots,V_{s5}$) acted as transmitters of black bursts with maximal output power; two others (V_{r1}, V_{r2}) were used as receivers and connected to laptops. The last node served as controller and triggered transmissions of the five sending nodes by wire. Thereby, the start of black burst transmissions was highly synchronized with very low offset. Besides Imote 2 nodes, two software-defined radios (Universal Software Radio Peripheral (USRP 2), [Ett13]) were deployed next to the receiving Imote 2 nodes to gather more fine-grained information on the medium state.

Experiments ran with five different distance setups. While the placement of the five transmitters was never changed, distances to the receivers were varied from 5 m up to 30 m (see Table 5.1). For each distance configuration, experiments were conducted with three numbers of transmitters (1, 2, and 5). In every distance/transmitter configuration, 60,000 black bursts



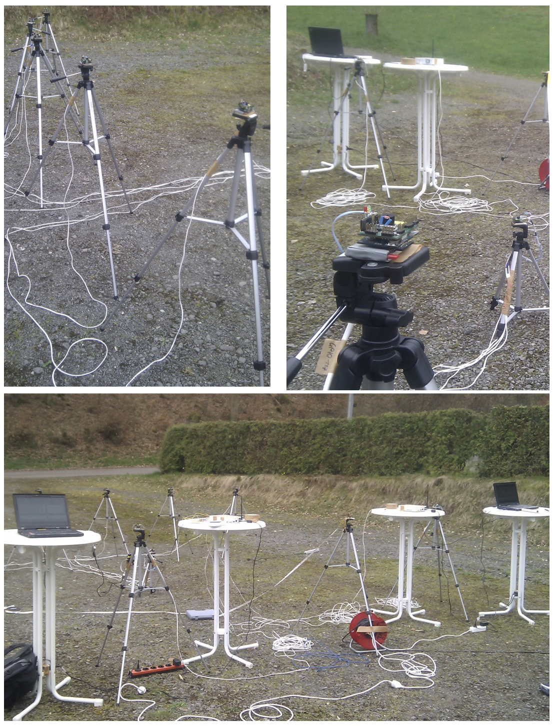


Figure 5.5: Topology and pictures of the experiments' setup.

experiment series	l_a	l_b	l_c	approx. distance V_{r1} $(= l_a + l_b)$	to senders V_{r2} $(= l_a + l_c)$
1	5 m	0 m	0 m	5 m	5 m
2	5 m	0 m	5 m	5 m	10 m
3	10 m	0 m	10 m	10 m	20 m
4	10 m	10 m	10 m	20 m	20 m
5	20 m	10 m	10 m	30 m	30 m

Table 5.1: Variations of l_a, l_b, and l_c in the five experiment series.

were sent in intervals of 10 ms. To enable the evaluation of false positives, recessive bits are assumed in between black burst transmissions. According to the configuration, the run of a single experiment took 10 minutes. Consequently, the entire experiment series lasted 150 minutes.

5.3.2 Experiment Results

In a first step, the success rates of black bursts (true positives) are evaluated for each configuration of distances and transmitters. For receiver V_{r2}, corresponding results are plotted in Fig. 5.6. Up to 20 m, the plot shows perfect success rates of 100%, which are independent of the number of senders. At 30 m, the distance approximates the sensitivity range of the transceiver and the success rate falls to 97.9% with a single sender. Yet, with two and five senders, success rates increase again, thereby indicating additive interference of overlapping black burst transmissions. Success rates of receiver V_{r1} are not shown in the plot, since they are similar – yet, a bit lower – to V_{r2}'s results and improve with increasing number of transmitters as well. To evaluate success rates of recessive bit transmissions, the number of false positives is counted. However, there was actually no wrong black burst detection during any experiment, which is due to the validity of the single-network property.

To take a closer look at the type of black burst interference, the energy level on the medium is evaluated with help of the USRP 2 at station 1 (V_{usrp1}). The resulting SNRs during black burst

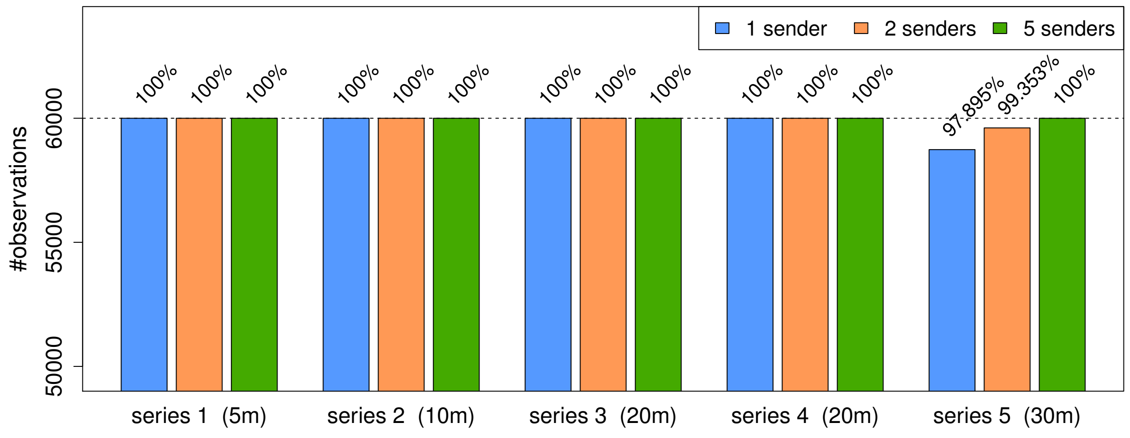


Figure 5.6: Success rate of black burst transmissions (true positives) at node V_{r2}.

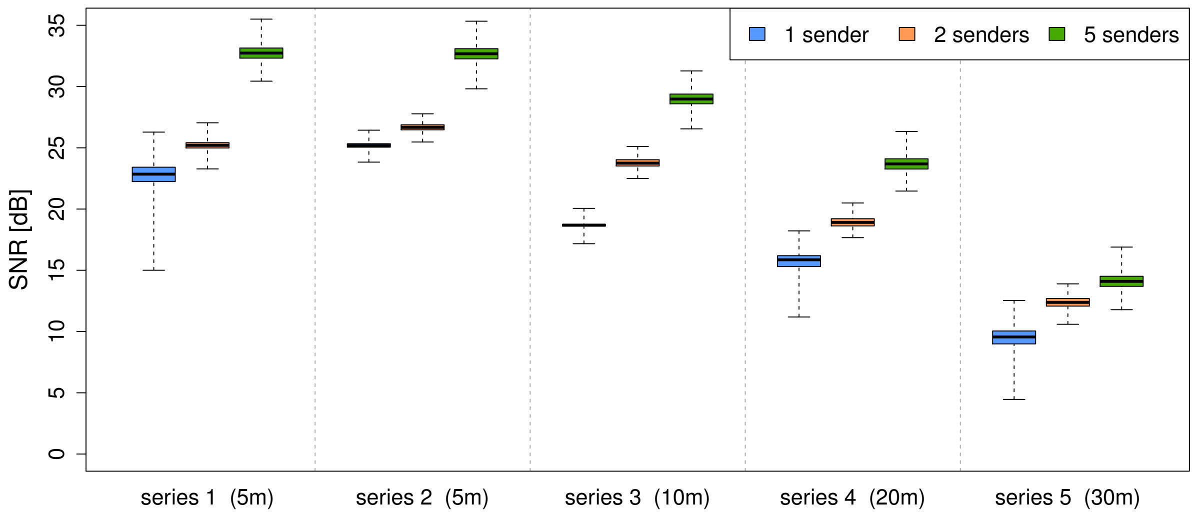


Figure 5.7: SNRs of black burst transmissions.

transmissions are plotted in Fig. 5.7, where black lines mark the median, boxes the first/third quartile, and whiskers the min/max SNRs. The results confirm the assumption of additive overlap, because SNRs never decrease with increasing number of transmitters. The additive interference can be explained both by chaotic signal propagation, which makes signal cancellation very unlikely, and by preventative countermeasures in the implementation (unique synchronization words) that prevent nodes from sending identical signals.

Further evidence of additive interference is provided by Fig. 5.8, where durations of detected black bursts are plotted. The histograms exemplarily show results of the third experiment series and nodes V_{r2} and V_{usrp2}. Results of the Imote 2 illustrate the mode of operation of the CCA mechanism, since they correspond to a multiple of the symbol duration of 16 µs.[7] Different from the Imote 2, which sees medium occupancies mostly longer than the transmissions actually took, the USRP 2 detects medium occupancies much more accurately and with lower spreading. In this regard, it has to be noted that sent black bursts were implemented by non-regular MAC frames with four bytes length (128 µs). But as suggested by the fact that the durations detected by the USRP 2 are smaller than 128 µs and as confirmed by later experiments in [Eng13], this realization is faulty, since the CC 2420 transceiver stops transmissions, in which the length field is set to 0, prematurely. As a result, later implementations realize black bursts by MAC frames of 160 µs length to guarantee that the duration of black bursts is larger than the maximal CCA delay ($d_{maxCCA} = 128$ µs).

For both node types, Fig. 5.8 shows an increase of detected durations with increasing number of senders. However, the increase is much higher for the Imote 2 (154.9 µs on average with one sender to 177.6 µs on average with five senders) than for the USRP 2 (119.9 µs to 125.2 µs). One reason of the increase is the time difference in the start of transmissions, because the controller cannot trigger all senders in full synchronicity. For the Imote 2, the main reason is, however, different and lies in the mode of operation of the CCA mechanism: Since CCA is based on RSSIs that are averaged over eight transmission symbols, an increasing number of senders, which cause a raise of the energy level on the medium, provokes an earlier exceedance of the

[7]The small deviations of ± 1 µs are due to the clock's granularity and measurement inaccuracies.

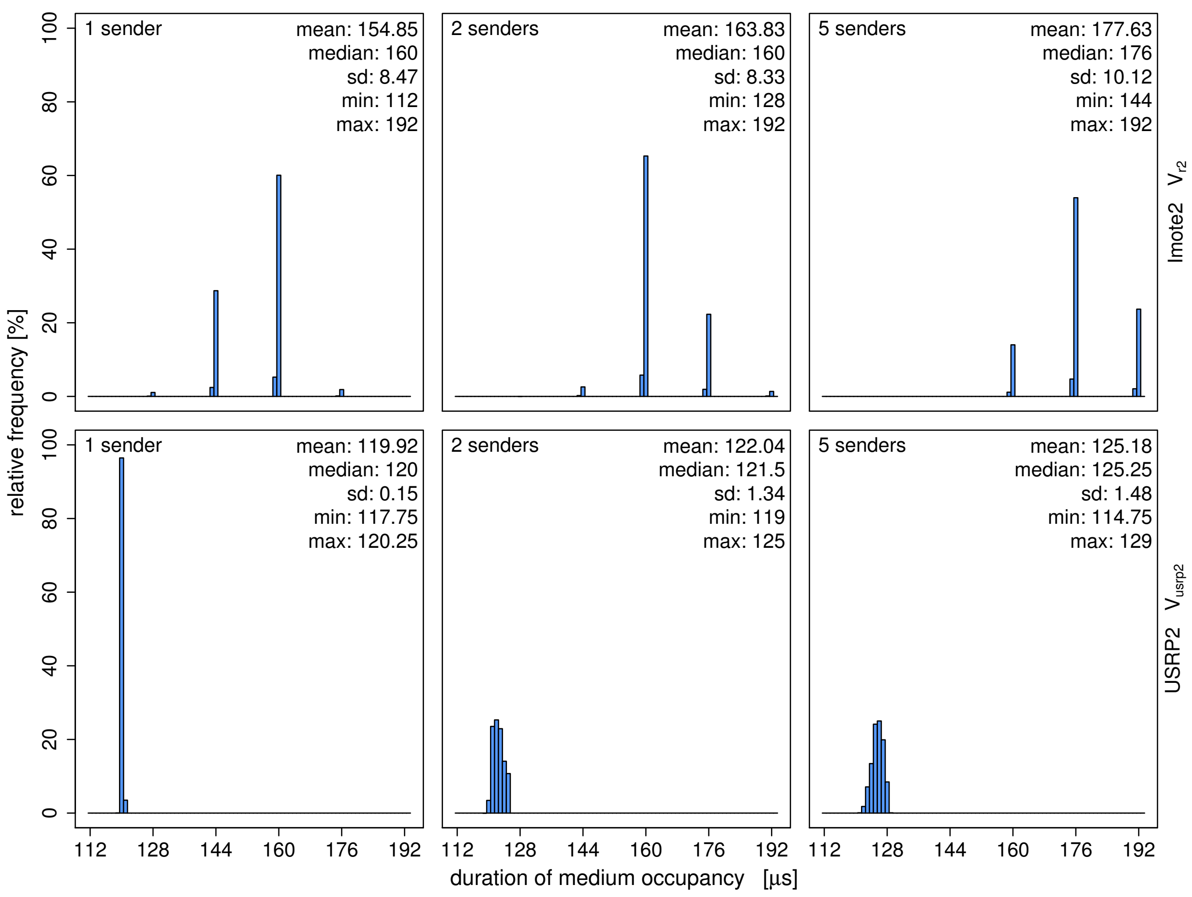


Figure 5.8: Histograms over durations of medium occupancies caused by black bursts.

CCA threshold at the beginning of transmissions and, vice versa, a later lower deviation after the end of transmissions.

In a last step, the accuracy of black burst detection is evaluated w.r.t. temporal offset between V_{r1} and V_{r2}. The corresponding results are plotted by box-and-whisker plots in Fig. 5.9 for the experiment series one to four. On average, node V_{r1} observes the start of black bursts later than V_{r2}. Thus, it is reasonable to assume that the transceiver of V_{r2} worked better than V_{r1}'s one. This statement is also supported by the median of the offsets in the first series, which is about $-25\,\mu s$, though the distance of both nodes to the transmitters was identical. Similarly, the nodes' offset in the second experiment series is almost symmetrical around $0\,\mu s$, though V_{r2}'s distance to the transmitters was twice as long.

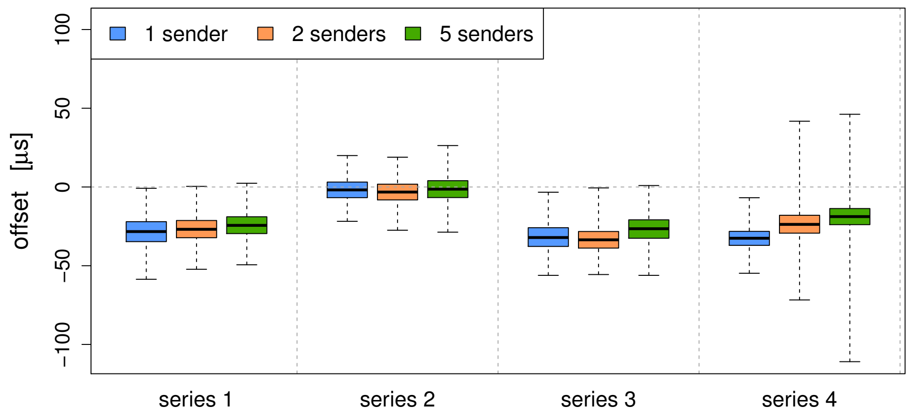


Figure 5.9: Temporal offset regarding detection of black bursts between V_{r1} and V_{r2}.

5.4 Evaluation of Optimized Transmission Detections

In Sect. 2.4.3, an optimization regarding the calculation of the start times of transmission was presented that is based on the back calculation of the transmission start by observing the gradient of the RSSI. By means of an experimental validation, this section now provides evidence of the optimization's correctness and quantifies improvements over transmission detections that utilize timestamps of CCA events only.

5.4.1 Experiment Setup

The deployed topology consisted of two nodes and is presented in Fig. 5.10. The sender V_a periodically sent black bursts in intervals of 10 ms that were detected by receiver V_b's CCA mechanism, which was configured with a threshold of -85 dBm. In addition to monitoring the timestamp of the CCA event, the receiver applied the optimization from Sect. 2.4.3 to get a more accurate timestamp of the start of a transmission. To compare the timestamps of the detected/-calculated transmission start with the actual transmission start, both nodes were connected by wire, whose level was changed by the sender at the beginning of a new transmission. The experiment ran with three different distances between sender and receiver: 2 m, 5 m, and 10 m. Furthermore, the sender varied the output power in eight steps from -25 dBm to 0 dBm. For each distance/power configuration, 10,000 black bursts were sent.

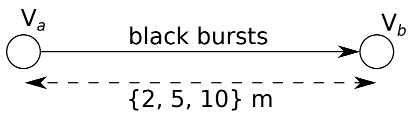


Figure 5.10: Topology to evaluate the back calculation of the start times of transmissions.

5.4.2 Experiment Results

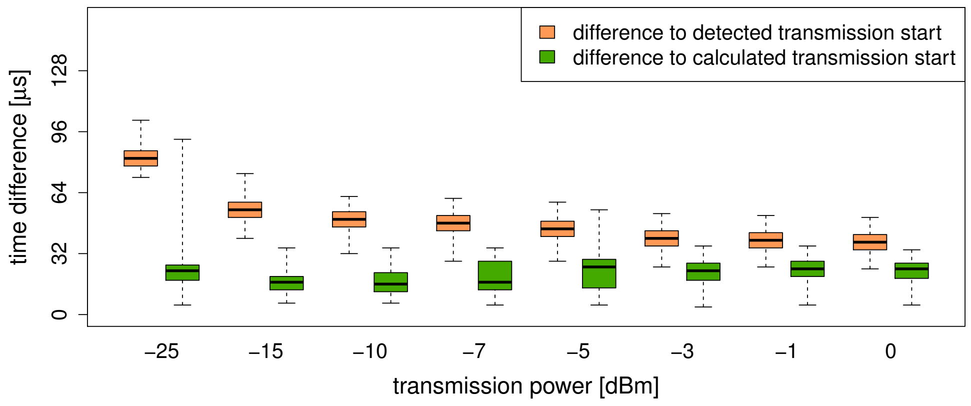
Results of these experiments are presented in Fig. 5.11 for all distance and output power configurations. The box-and-whisker plots show the time difference between the start of transmission at the sender and the detected/calculated start at the receiver.

For all distances and power levels, back calculated timestamps are more accurate than the timestamps of the CCA events, thereby providing evidence of the usefulness of the optimization. The improvements are, in particular, observable when transmission power is low. At a distance of 2 m and with -25 dBm output power, for instance, the median of time differences is 82 µs with detected timestamps and 23 µs with optimized timestamp calculation. Moving sender and receiver apart even increases this gap. An explanation of this effect is the difference between CCA threshold and the RSSs of transmissions. In this regard, see also Table 5.2 that presents the average RSSs for each distance and transmission power. For large distances and/or low output power, RSSs – and thus also differences to the CCA threshold – are very small. Hence, more transmission symbols are required in the averaged RSSI calculation of the transceiver to exceed the CCA threshold. Consequently, delays until the CCA mechanism indicates a busy medium increase. Without back calculation, these increasing delays remain unnoticed.

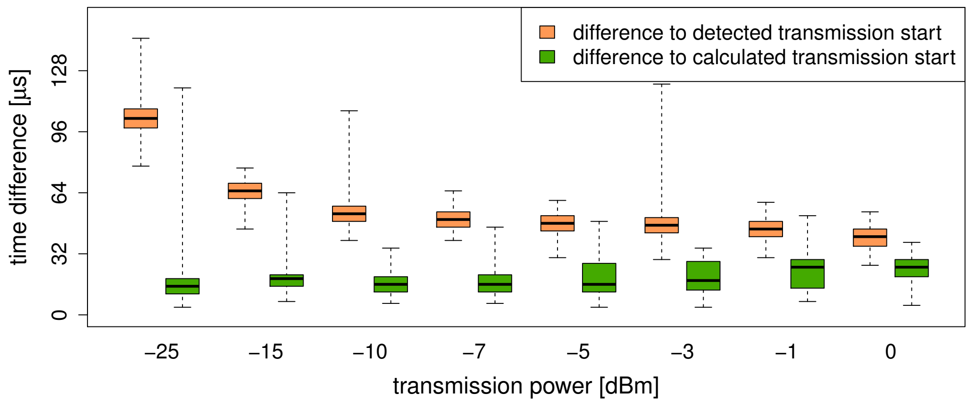
While the median of the time differences is always significantly better with back calculated timestamps, worst-case differences are sometimes not such large. This particularly holds for low transmission powers, where some outliers push worst-case differences of back calculated timestamps up. A possible explanation of this observation are again the low RSSs of the corresponding transmissions and the variability of the signal strengths of symbols. These conditions can together defer the detection of a transmission by more than eight symbol periods, which can only partially be corrected by the back calculation. The fact that there are some cases, in which the timestamp of the CCA event is larger than 128 µs (e.g., at 5 m distance, the worst-case difference is 145 µs), supports this assumption. A further indication confirming this assumption is the very high number of false negatives in configurations with low transmission power. For instance, only 25 out of 10,000 black bursts were perceived with -25 dBm transmission power and 10 m distance.

	-25 dBm	**-15 dBm**	**-10 dBm**	**-7 dBm**	**-5 dBm**	**-3 dBm**	**-1 dBm**	**0 dBm**
2 m	-78.0 dBm	-67.9 dBm	-62.9 dBm	-59.6 dBm	-55.9 dBm	-55.2 dBm	-54.9 dBm	-54.0 dBm
5 m	-81.8 dBm	-71.6 dBm	-65.9 dBm	-63.0 dBm	-60.3 dBm	-58.7 dBm	-55.5 dBm	-53.5 dBm
10 m	-85.0 dBm	-77.9 dBm	-73.2 dBm	-69.0 dBm	-66.1 dBm	-64.4 dBm	-63.0 dBm	-61.7 dBm

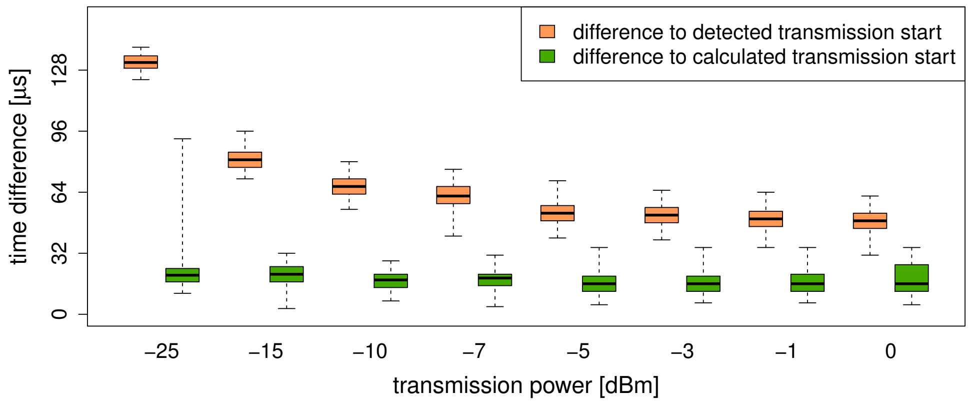
Table 5.2: Average RSSs as function of distance and transmission power.



(a) 2 m distance between sender and receiver.



(b) 5 m distance between sender and receiver.



(c) 10 m distance between sender and receiver.

Figure 5.11: Time difference between transmissions' start and detected/calculated start.

5.5 Evaluation of ACTP: A Proof-of-Concept

In this section, the feasibility of ACTP is demonstrated by means of an experimental evaluation in a small multi-hop scenario. Besides quantifying ACTP's reliability, the evaluation furthermore comprises a check of the protocol's timing constraints (see Sect. 3.2.2) and a validation of their correct realization in the stand-alone implementation (see Sect. 3.7.2.1).[8] Furthermore, it provides a practical proof-of-concept that periodical resynchronization is sufficient and that compliance with duty cycling and other MAC protocols is achievable.

5.5.1 Experiment Setup

In the topology, four nodes were deployed as shown in Fig. 5.12. They were placed such that the network diameter yielded $n_{maxHops} = 2$ hops. Though the scenario is small, it is sufficient to test ACTP's key properties, which are multi-hop arbitration and collision-resistant propagation of bit sequences. The experiment ran indoor in an office building and with violation of the single-network property due to neighbored IEEE 802.11 [Ins12a] and IEEE 802.15.1 networks [IEE05]. The nodes were distributed over three rooms, where people were additionally moving around. Thus, nodes were mostly neither in line-of-sight of each other nor in a static environment.

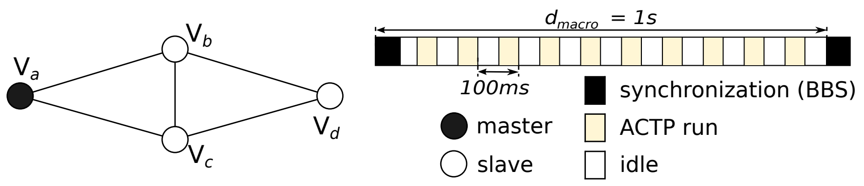


Figure 5.12: Topology and macro slot configuration to evaluate ACTP.

The experiment lasted about five hours and synchronization was performed with the master-based variant of BBS with resynchronization intervals of 1 s. In each macro slot, 10 ACTP runs with network-wide arbitration radius were placed in equidistant intervals of 100 ms. Thus, a total of 180,000 arbitrations are included in the results of the evaluation. The sent arbitration sequences had a length of eight bits and were pre-defined and recurring in each macro slot. They were assigned such that each node wins at least two ACTP runs per macro slot. Their concrete values are given in Table 5.3 in their decimal form.

	run 1	run 2	run 3	run 4	run 5	run 6	run 7	run 8	run 9	run 10
V_a	**213**	156	76	103	**178**	45	91	57	27	30
V_b	22	**189**	99	71	110	**155**	209	59	**219**	73
V_c	187	188	**155**	66	44	79	**210**	23	51	**201**
V_d	89	14	103	**245**	3	100	104	**198**	113	33

Table 5.3: Priority assignment in the evaluation scenario of ACTP.

[8]ACTP's incorporation into BiPS (Sect. 3.7.2.2) was also evaluated in a student's project. Since the results are similar, they are not further discussed.

5.5.2 Experiment Results

The node-specific success rates are plotted in Fig. 5.13. In each plot, rates are shown for all ACTP applications as function of their relative run in a macro slot, i.e., an x-axis value of 1 corresponds to the first ACTP run after synchronization. A run of ACTP is rated as success if either the node wins correctly, i.e., if it has the highest priority and completes the transmission of the corresponding bit sequence (plotted as dots), or if it loses correctly, i.e., if it finishes ACTP as passive node and receives the correct bit sequence of the winner (marked with crosses). The results show that the node-specific success rates are very similar and above 99.9%, and that there is no big difference between nodes on the borderline of the network (V_a and V_d) and nodes in the center (V_b and V_c), which is a strong indication of successful multi-hop propagation.

Figure 5.14 aggregates the node-specific success rates to overall rates. The dots show ACTP runs that are completely successful, i.e., where both the correct node wins and its bit sequence is received by all other nodes without faults. The crosses additionally count all runs, in which

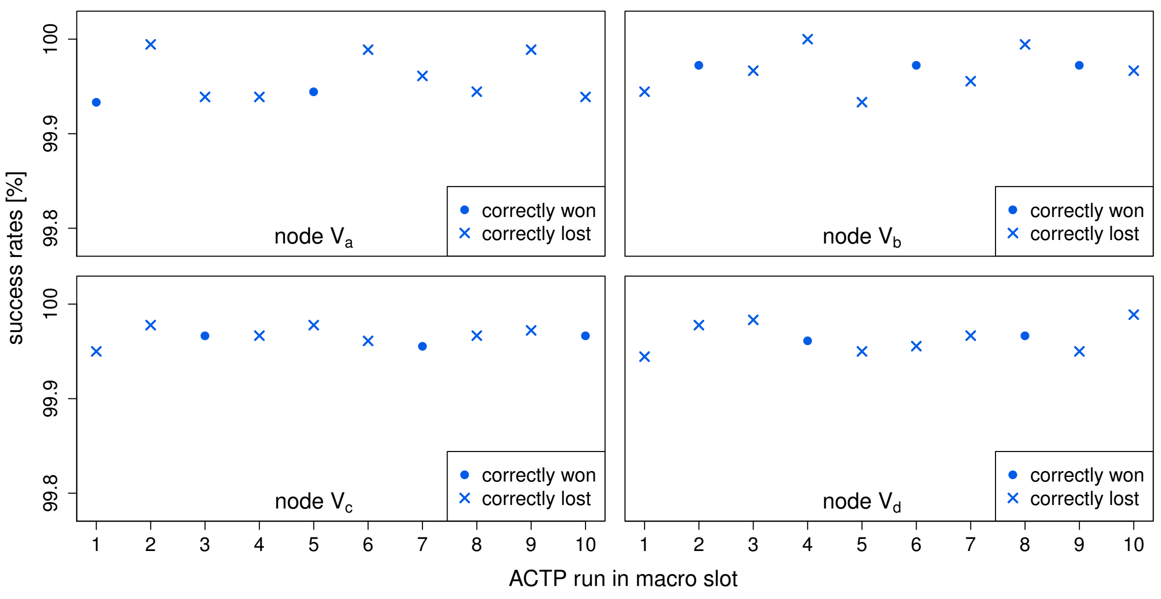


Figure 5.13: Success rates of ACTP for each node and per run in a macro slot.

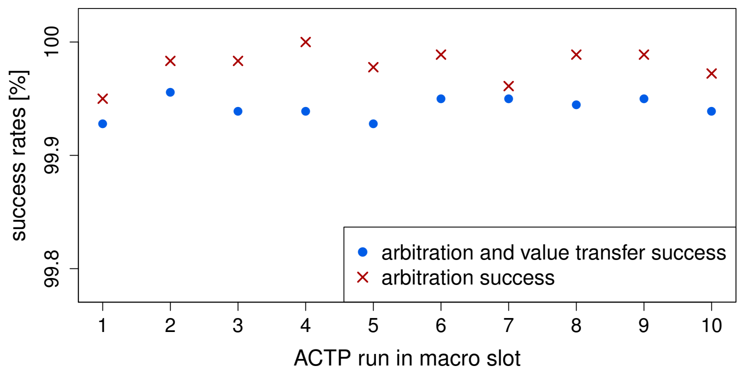


Figure 5.14: Rates of total success and successful arbitrations.

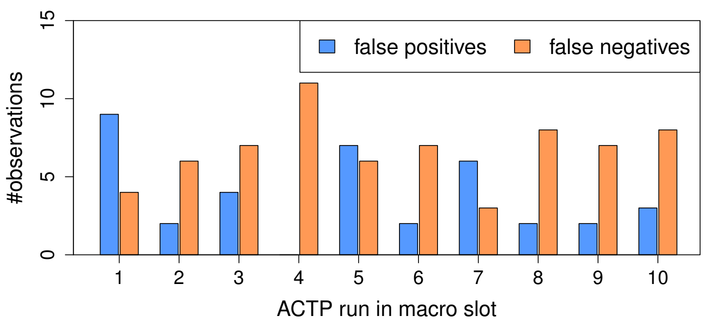


Figure 5.15: Causes of faults in ACTP runs.

the correct node wins but at least one losing node detects a faulty bit sequence. Though these runs are dangerous w.r.t. ACTP's support of value transfer, they are still correct w.r.t. arbitration. In sum, the total success rate is 99.94% and arbitrations are successful in 99.98% of all applications of ACTP. The figure – as well as Fig. 5.13 – additionally demonstrates that the configured resynchronization intervals are sufficient and that there is no deterioration of success rates during macro slots despite increasing synchronization offset.

In total, we observed 37 out of 180,000 ACTP runs with wrong or multiple winners and 67 runs with minor faults. To investigate these faults in more detail, we count the number of false positives and negatives that lead to the faulty outcome. The results are presented in Fig. 5.15 and show that false negatives occur more often than false positives (67 vs. 37). A possible reason for false positives can be found in the violation of the single-network property and the misinterpretation of interference as black burst. The existence of external interference is also a possible explanation of false negatives, since it can deteriorate and prolong medium occupancies. The fact that 137 of such invalid medium occupancies were observed and filtered during ACTP runs of the experiment confirms this assumption.

5.6 Discussion

In this chapter, results of experiments are presented to provide a proof-of-concept of black bursts and ACTP. The chapter, particularly, demonstrates that the implementations of black bursts and ACTP do not show any indication of destructive interference of overlapping black burst transmissions. Thereby, it provides a practical verification of a key property of the collision resistance of black bursts. Results of outdoor experiments in Sect. 5.3 furthermore point out that SNRs of black bursts become even larger with increasing number of transmitters, which argues for additive interference and confirms previous experiments [KdI07].

While experiments with success rates of 99.9% and more demonstrate the feasibility of black bursts and ACTP, they also reveal that the CC 2420 transceiver is indeed adequate to serve as proof-of-concept platform but inappropriate for efficient implementations of black bursts. The reasons for this are the large delays of the transceiver – e.g., regarding switching to transmission/reception mode and detection of medium state changes – and its operation in the 2.4 GHz ISM band. In particular, the ISM band can state a problem and requires special attention, since it is overcrowded by WLAN and Bluetooth traffic and impedes compliance with the single-network property. In this regard, outdoor experiments (Sect. 5.3), where the single-network property was satisfied, substantiate that success rates of 100% are achievable for adequate distances and that false positives can be avoided entirely. Experiments in indoor office environments (Sect. 5.2 and Sect. 5.5) illustrate consequences of external interference in terms of false positives and false negatives. In summary, this chapter stresses the importance of the single-network property and argues for a controlled setup with dedicated communication channels to reliably run black burst-based protocols.

The presented experiments were conducted during different stages of development and reveal a continuous enhancement of the implementation. In this regard, the comparatively bad results from Sect. 5.2 can – besides the violation of the single-network property – also be explained by identical synchronization words, which can lead to an adulterated view of the state of the medium, since the detection of black bursts as regular MAC frames is not prevented. In later experiments, transceivers were configured with random synchronization words to eliminate this problem and to guarantee the detection of black bursts as medium occupancy only.

Though most false positives and false negatives could be traced back to external interference, Sect. 5.4 – and particularly Fig. 5.11 – identify a different problem for black bursts, which is not caused by external interference but occurs due to weak links between sender and receiver and variability of signal strengths. These boundary conditions can lead to situations, in which the CC 2420 transceiver requires more than eight transmission symbols before its RSSI value exceeds the CCA threshold. As consequence, black bursts are detected later than $d_{maxCCA} = 128\,\mu s$, which should actually be the maximal delay to detect any transmission. Since exceeding d_{maxCCA} impedes guarantees regarding the association of a black burst with its actual start time, it states a great threat for ACTP as well as BBS, which could no longer provide any upper bound on the maximal synchronization offset. Thus, such situations must be avoided at any price, which is in most cases possible by filtering transmissions whose maximal RSS is not significantly larger than the CCA threshold.

This chapter is complemented by a proof-of-concept of ACTP, which validates the protocol's implementation and provides a practical proof of its timing constraints that were derived in Sect. 3.2.2. By evaluating ACTP in a full setup with BBS and macro slots, this section also demonstrates that realizations of wireless multi-hop binary countdown protocols that are compliant with duty cycling and other MAC protocols are possible. Though the topology was small-scale and external interference was not excluded, the experiment was sufficient to show ACTP's feasibility and mode of operation w.r.t. deterministic medium arbitration and value transfer. Success rates of $> 99.9\%$ could furthermore demonstrate a high predictability even in environments with invalid single-network property.

6. Chapter

Related Work: Busy Tone and Binary Countdown Protocols

In the literature, there are a plenty of medium arbitration protocols. Some of them are devised for general purpose applications, others tailored for specific application domains like WSNs. Due to the energy constraints of nodes, most of WSN-specific protocols target first of all energy efficiency and duty cycling. In this regard, prominent examples of synchronized duty cycling protocols are S-MAC [YHE02, YHE04] and RMAC [DSJ07]. Since these protocols come with low predictability, i.e., they can neither guarantee delay bounds nor that a particular station wins the contest, they are not in the focus of this chapter. Instead, the following survey is limited to protocols that apply some kind of collision-resistant arbitration scheme and can be divided into two categories: *Busy tone protocols* encoding the priority of a station in the duration of a buzz signal and *binary countdown protocols* resolving contest bit-wise. All of the discussed protocols are distributed protocols, which do not rely on central coordinators organizing medium access. Some of them assume that the network is without hidden stations. However, in the corresponding context, the definition of *no hidden station* is less stringent and is already satisfied if the sum of communication and interference range is smaller than the sensing range [BPC$^+$07].

The rest of this chapter is structured as follows: In Sect. 6.1, representatives of busy tone protocols are discussed. Section 6.2 outlines examples of binary countdown protocols. Finally, Sect. 6.3 compares the presented protocols and shows differences to ACTP.

6.1 Busy Tone Protocols: The Origin of Black Bursts

The term *black burst* is first mentioned in [SK96] and is defined as pulse of energy of specified duration. Consequently, a black burst contains no further payload besides the start and end time of its transmission. Different from the usage of black bursts in the context of ACTP and BBS, black bursts may differ in length in their original definition.

In [SK96], black bursts build the foundation of a distributed medium access scheme that is referred to as *black burst arbitration* in the following. The general mode of operation of black burst arbitration is as follows: Each station intending to send a data frame first waits until the medium is idle. Then, it starts to jam the channel with a black burst, where the duration of the black burst reflects the priority of the transmission. After the end of its black burst transmission, the station listens to the medium to detect black bursts of other stations. If the station does not sense another transmission, it is allowed to send its data frame. Thereby, the node sending the longest black burst wins. If a station detects a busy medium after its black burst transmission, it defers its data transmission and waits until the medium is free again when it contends with a new black burst.

In the literature, there are several MAC protocols adopting black burst arbitration.[1] They differ i.a. in the calculation of black burst durations, in requirements for hardware, and in assumptions on the network. Some of them are presented below in more detail.

6.1.1 The First Mention of Black Bursts by Sobrinho and Krishnakumar

With the objective to guarantee collision-free transmissions of real-time traffic with bounded delays, Sobrinho and Krishnakumar introduced black bursts as part of a medium arbitration scheme in [SK96]. The arbitration scheme was developed as extension of IEEE 802.11 [Ins12a] and has been extended in [SK99]. In their works, the authors distinguish between data stations, which have no real-time constraints and apply IEEE 802.11 DCF regularly, and real-time stations with periodical communication demands with bounded medium access delays. To prioritize real-time stations over data stations, different classes of Inter-Frame-Spacings (IFSs) are introduced. To solve competition between real-time stations, black burst arbitration is applied.

Before black burst arbitration is actually used, a real-time station must first initiate a real-time session, which is implicitly done by sending the first frame of the session with conventional CSMA/CA of IEEE 802.11 [Ins12a]. Afterwards, subsequent frames of the session are sent in fixed intervals with preceding black burst arbitration. In steady state and without data stations, there is no real contention, since all real-time stations share the medium in round robin fashion. However, due to data stations and real-time stations starting a session, the transmission of a real-time frame can be deferred. This may afterwards result in contention between two real-time stations, which is solved by black burst arbitration, where the duration of a black burst is proportional to the time the node has already been waiting. Thus, the node waiting for the longest time sends the longest black burst and wins. As result, real-time transmissions are delayed for the duration of at most one frame transmission.

To improve efficiency, [SK99] presents an extension of the arbitration scheme that is called *chaining* and enables the invitation of a station to send a data frame without further arbitration. The paper additionally sketches the realization of real-time sessions with different transmission intervals. Though this improves flexibility of the protocol, it introduces additional overhead to guarantee uniqueness of black burst durations.

The presented protocol is evaluated both by simulations to compare black burst arbitration with CSMA/CA and by detailed analyses of stability conditions, which assess whether access delays are bounded under a given workload. Though the first frame of a real-time session may still be delayed unboundedly, the presented arbitration process guarantees that – assuming an adequate workload – frames of existing real-time sessions are sent collision-free and with bounded delay. A disadvantage of the approach is that only multi-hop networks without hidden stations are supported.

[1]Not all of them actually use the term black burst but often consider jamming signals as *preambles*. This term is more common in the context of asynchronous duty cycling protocols, where preambles are sent to wake up a node. In this regard, well-known examples with sender-based preamble transmissions are WiseMac [EHD04b, EHD04a] and X-MAC [BYAH06].

6.1.2 Elimination by Sieving MAC and Deadline Bursting MAC

In [PDO02], the authors present two MAC protocols with deterministic collision resolution. They are called *Elimination by Sieving* (ES-DCF) and *Deadline Bursting* (DB-DCF). The objective of both protocols is the provision of upper bounds on transmission delays in single-hop networks. To this effect, nodes are classified as real-time nodes, which execute the proposed protocols, and non-real-time nodes running regular IEEE 802.11 DCF with increased IFSs [Ins12a]. Traffic of real-time nodes is subdivided into soft- and hard-real-time traffic, where hard-real-time traffic is prioritized.

ES-DCF consists of four phases. The first phase is called *elimination phase* and selects all nodes with highest priority. This is achieved by a passive arbitration, in which a backoff is waited that is calculated as function of the priority – thereby, also preferring hard- over soft-real-time traffic – and urgency of the data frame. Since these two parameters are possibly identical for several nodes, random subgrades are introduced to reduce the probability of multiple winners. In the second phase, which is called *channel acquisition phase*, all winners try to perform an RTS/CTS handshake. Since multiple winners can still exist, the handshake can fail due to RTS frame collisions. If this is the case, the conflict is solved in a *collision resolution phase* by means of a black burst arbitration, where the duration of black bursts is proportional to unique node ids. Afterwards, the node sending the longest black burst repeats the *channel acquisition phase*, whereas nodes with shorter black bursts wait until the channel becomes idle again before repeating the *collision resolution phase*. If the handshake of the *channel acquisition phase* is finally successful, data is transferred in a *data transmission phase*.

DB-DCF is very similar to ES-DCF and performs better with soft-real-time traffic but worse with hard-real-time traffic. It, particularly, also consists of four phases, where the last three phases are identical to ES-DCF. In the *elimination phase*, however, contention is solved actively with black burst arbitration and black burst durations that depend on the urgency of the data frame, i.e., on a relative deadline: The closer the deadline, the longer the black burst. Thus, the most urgent nodes become winners of this phase. Since there is potentially more than one winner, conflicts may exist, which are solved in the *collision resolution phase* as in ES-DCF.

Both protocols are evaluated in simulations. Implementations on hardware do not exist. Though both protocols use RTS/CTS handshakes to address the hidden station problem, it is questionable whether they perform well in multi-hop topologies with hidden stations, because conflicts are only solved if conflicting nodes are in sensing range of each other.

6.1.3 A Black Burst-based MAC Protocol with Dynamic ID Assignment

In [SLWT04], a distributed MAC protocol for single-hop networks is proposed. The protocol supports data transfer with several priority levels and adopts black burst arbitration. Its periodical operation cycle is subdivided into three phases: *Priority classification period*, *ID initialization period*, and *data transfer*.

In the first phase, black burst arbitration is executed by each station with pending data, where the duration of black bursts is proportional to the non-unique priority of the sending station. Thus, all highest priority stations emerge as winner of this phase. To avoid collisions of the data frames of the winners, their transmission order is determined in the ID initialization period. Here, a randomized initialization protocol is incorporated to assign each winner a consecutive unique id. The protocol is probabilistic, fair, and based on the construction of a contention tree, where the ids of nodes are derived from their positions in the tree. In the third phase, all winning nodes send all their frames in round robin fashion in the order of their ids.

The protocol has been evaluated by simulations. Implementations do not exist. A drawback of the ID initialization protocol is its inefficiency and the missing upper delay bound on its termination. Thus, the overall protocol does not provide bounds on access delays, too.

6.1.4 Soft Real-time Chains

Soft Real-time Chains is a MAC protocol for wireless multi-hop networks without hidden stations [BPC+07]. The protocol is distributed, unsynchronized, and targets networks with temporary but strong requirements for high transmission rates. It is developed as extension of IEEE 802.15.4 [Ins11]. The eponymous term real-time chain is defined as a multi-hop data flow and has a specified priority. Data of a real-time chain is transferred on separate channels, whereas best effort traffic is sent with CSMA/CA on a common channel.

To setup a real-time chain, a chain open frame, which is sent on the common channel and with preceding black burst arbitration, is sent by the source node. This frame is forwarded – again with preceding black burst arbitration – along the route to the final destination node.[2] The content of this frame comprises an index, which is increased on each hop, and the priority of the data flow that dictates the duration of black bursts. A node receiving/sending a chain open frame gets a temporary node id, which corresponds to the index in the chain open frame, and is afterwards part of this particular chain until the chain is closed.

After opening a chain, all participating nodes switch to a new set of channels. Nodes with even node id are so-called *one channel nodes* and send/receive on the same channel. Other nodes are *dual channel nodes* and change channels in between receiving and forwarding. To improve throughput by exploiting spatial reuse, a chain can allocate several channels such that the same channel is not reused by neighbored nodes. Data of a real-time chain is transferred after black burst arbitration, where the duration of black bursts depends on the priority of the data flow and the position of the sending node in the chain, and is acknowledged with ACK frames. Thereby, forwarding of frames gets priority over new frames.

The protocol's implementation is evaluated on MICAz nodes [MEMarb] and supports up to four flow priority levels. Since the protocol is fully distributed and unsynchronized, problems arise if several chains are opened at the same time or if there are conflicting chains in neighborhood of each other. To alleviate the second problem, the authors propose an extension, in which chains can be reopened with a different set of channels.

6.1.5 Backoff Preamble (Sequential) MAC

In [KKS09, Kle11, KB11, KB12, Kle12] and [Kle10, Sect. 2.5], Klein et al. present two new MAC protocols – called Backoff Preamble MAC (BP-MAC) and Backoff Preamble Sequential MAC (BPS-MAC) – for unsynchronized single-hop networks. Their objective is the prevention of collisions in case of correlated, event-driven, and bursty traffic, which is insufficiently handled by common CSMA/CA-based MAC protocols of WSNs due to large CCA delays.

[2]The authors assume the availability of an adequate routing protocol, but do not discuss this topic further.

Medium arbitration with BP-MAC is based on black burst arbitration, where the duration of black bursts, which are called backoff preambles by the authors, is calculated randomly [KKS09]. Each black burst is always a multiple of so-called *slots*, whose duration must be sufficient to enable the detection of an occupied medium. Since several stations may compute the same number of slots, uniqueness of winners is not guaranteed, thereby causing collisions of subsequent data frame transmissions. Due to unsynchronized arbitration starts and blind periods during transceiver switchings, collisions may also arise if stations use black bursts of similar but different length. BP-MAC is analyzed in a probability analysis and compared with CSMA/CA by simulations.

In [Kle10, Sect. 2.5] and [Kle11], BPS-MAC is presented as extension of BP-MAC to reduce collision probabilities. The extension consists of the sequential elimination of contention by applying several shorter black burst arbitrations, where the concrete number has to be determined as a trade-off between medium access delay and collision probability. In addition, the author discusses the impact of the duration calculation of blacks burst on the probability of collisions. In this regard, it is shown that calculating the duration of black bursts by random functions with non-uniform distributions achieves less collisions if more than two nodes compete. The derivation of the best distribution, however, requires exact knowledge on the number of competing nodes. Otherwise, the number of collisions increases. BPS-MAC is evaluated by simulations in [KB12, Kle12] and compared to the non-beacon mode of IEEE 802.15.4 [Ins11].

An enhancement of BPS-MAC to support QoS is presented in [KB11] and works as follows: In the first arbitration, the durations of black bursts are no longer calculated randomly but priority-based. In this regard, several static and dynamic priority metrics like energy resources, node ids, and buffer states are discussed. If priorities are not unique, further black burst arbitrations are scheduled subsequently to resolve remaining contention in a probabilistic way. BPS-MAC with priorities is evaluated in simulations and compared to contention-based medium access of IEEE 802.15.4 [Ins11]. With the results, the authors could show that their protocol decreases delays of high-priority transmissions during bursty traffic situations significantly.

BP(S)-MAC is only designed for single-hop networks or multi-hop networks without hidden stations. Because the protocol is unsynchronized and preambles are used for medium arbitration but not to wake up receivers, measures to save energy are hardly compatible. Though some implementation aspects are discussed in [Kle12], no complete implementations of BP-MAC or BPS-MAC are mentioned, and it is questionable whether the used timings are sufficient. In particular, the slot duration, which must be sufficient to switch to receive mode and to detect the correct medium state, is set to 128 µs in simulations. This is, however, only the CCA delay of the CC 2420 transceiver, which is not enough for this purpose as proven in [Eng13].[3]

[3] In Sect. 2.5.1 in [Kle10], it is stated that the "slot duration has to be chosen such that a node is able to switch the transceiver mode and to detect a busy medium within a single slot duration". However, this does not reflect the slot duration in simulations (128 µs) and is contradictory to other descriptions of the protocol.

6.1.6 A Black Burst-Based Protocol for Vehicular Networks

In [KEÖ07], Korkmaz et al. present two broadcast protocols for wireless multi-hop networks, which are called *Ad hoc Multi-hop Broadcast* (AMB) and *Urban Multi-hop Broadcast* (UMB). They are position-based and designed for IEEE 802.11-based [Ins12a] vehicular networks, in which nodes are not aware of the network topology but know their position by GPS. The objective of the protocols is to quickly propagate new information into the network. For this purpose, measures are introduced to solve the hidden station problem and to improve efficiency of multi-hop communication.

To distribute new data, both protocols apply directional broadcasts. For this purpose, a node with new information transmits a Request-To-Broadcast (RTB) frame containing the duration of the upcoming data transfer, its position, and the intended broadcast direction. RTB frames are sent with conventional CSMA/CA. When receiving an RTB, all nodes in the intended broadcast direction calculate the distance to the sender of the RTB based on the GPS-based position information and start a black burst arbitration. Thus, different from previous applications of black burst arbitration, black bursts are now sent by possible receivers. Their duration is proportional to the distance, such that nodes that are furthest from the sender win and are afterwards allowed to send a Clear-To-Broadcast (CTB) frame. Since there can be more than one winner, collisions of CTB frames can occur, which are detected by the sender of the RTB frame by observing a busy medium without frame reception. In this case, the sender of the RTB frame sends a new RTB frame and a new black burst arbitration starts, in which only winners of the previous run participate and distances are separated into smaller granularities to increase the probability of a unique winner. This process is repeated until there is only one winner or a maximal number of attempts is exceeded. After the sender of the RTB frame receives a CTB frame correctly, it broadcasts its information along with the id of the winner, which is afterwards responsible for broadcasting the information further.

The protocols are evaluated by simulations, where they outperform existing IEEE 802.11-based protocols in terms of reliability and efficiency. Drawbacks are the need to arbitrate hop-by-hop and missing guarantees regarding delays due to ambiguous priorities.

6.2 Binary Countdown Protocols

Instead of encoding priorities in the duration of busy tones, binary countdown protocols convert priorities into binary numbers of fixed length and send them bit-by-bit. For this purpose, bits are classified as dominant bits that are implemented by emitting short signals of constant length and recessive bits, which are realized by the absence of a transmission. Thus, stations sending recessively can listen on the medium to detect dominant bits and stop processing their own bit sequence if a dominant bit is detected. As result, nodes sending the numerically greatest sequence arise as winners among all nodes in sensing range. In most cases, binary countdown protocols are adopted as arbitration protocol to resolve contest and to enable collision-free data transmissions subsequently. In the following, the survey presents some wireless representatives of binary countdown protocols, which differ in assumptions on the network topology and the creation of binary numbers.

6.2.1 CSMA/IC and BROADEN

To solve contention in multi-hop networks without hidden stations, You et al. present the *Carrier Sense Media Access protocol with ID Countdown (CSMA/IC)* [YYH03b], which adopts the binary countdown principle and provides priority-based medium access.

CSMA/IC divides time into periodical time frames, which are called competing units. They are further subdivided into four phases: A *medium sensing slot*, a *beacon sending slot*, a *binary competing phase*, and a *data sending period*. In the beacon sending slot, local synchronization is established among neighbors by the transmission of beacons, which are sent by all nodes in a probabilistic way. In the binary competing phase, the binary countdown protocol is executed with unique binary numbers comprising two parts: A priority (e.g., based on the type of the pending data frame) and a uniqueness part (e.g., the node id). Dominant signals – called *buzz* signals – encode binary 1s; binary 0s are encoded recessively. The derivation of bit durations is only sketched and considers maximum propagation, switching, and sensing delays.

Since beacons are sent randomly and may suffer from collisions, there is no upper bound on synchronization offset. To avoid these problems, the authors argue for external synchronization mechanisms like GPS, which is not discussed in detail. CSMA/IC was evaluated in comparison simulations with the CSMA/CA-based arbitration of IEEE 802.11 [Ins12a].

CSMA/IC has been incorporated into BROADEN (Binary countdown – RTS – Object-to-send (OTS) – Agree-to-send (ATS) – Disagree-to-send (DTS) – Ensure-to-send (ETS) – Neaten-to-send (NTS)) [YYH03a]. On a *control channel*, BROADEN subdivides time into time frames as in CSMA/IC and applies a binary countdown protocol before frame transmissions. Transmissions of data frames occur on separate data channels. To avoid collisions on these channels, BROADEN introduces a couple of frame types that are sent on the control channel to negotiate transmissions on a data channel. By maintaining tables with scheduled transmissions, BROADEN gathers information on the neighborhood of a node, which is evaluated to negotiate new transmission schedules and to reduce the exposed station problem. As further measure to improve spatial reuse, the transmission power is adjusted according to the distance to the receiver.

A general drawback of BROADEN is its very high overhead: Besides maintaining state tables for nearby receptions and transmissions, data transmissions require at least four preceding control frames (RTS, ATS, ETS, and NTS). Because CSMA/IC is adopted unaltered on the control channel, all drawbacks of CSMA/IC hold for BROADEN, too – in particular, regarding synchronization. Though BROADEN alleviates the exposed station problem on the data channel, the problem still exists on the control channel. BROADEN is also evaluated in simulations and compared to IEEE 802.11 [Ins12a]. The results show that BROADEN prevents collisions and improves delays and throughput in heavy load situations.

6.2.2 BitMAC

BitMAC is a very comprehensive MAC protocol for multi-hop WSNs with single sinks and has been proposed by Ringwald and Römer in [RR05]. It comprises TDMA elements to multiplex communication between parents and children as well as FDMA to dissolve interference between neighbored subtrees. Synchronization required for TDMA is achieved by an internal tick synchronization protocol, which achieves deterministic accuracy by the collision-protected encoding of bits. Besides synchronization, many other protocol parts of BitMAC also utilize collision-protected transmissions of bit sequences and exploit the fact that useful information can be encoded just in the presence of medium occupancy. As a result, BitMAC enables deterministic up- and downlink communication in the tree topology of the network.

BitMAC is not explicitly advertised as binary countdown protocol. Instead, it introduces several subprotocols, which perform synchronization and integer operations like OR, AND, MIN, and MAX in a collision-protected way. But since the result of a MAX operation is identical to the outcome of the binary countdown protocol, BitMAC implicitly realizes a binary countdown protocol, which is, for instance, used to determine the parent node of a child and to allocate channels to different subtrees. Compared to other binary countdown protocols, a distinctive feature of BitMAC is the repetition of received bits by parent nodes. Thus, the radius of the binary countdown protocol amounts to two sensing range hops and children of the same parent do not need to be in sensing range of each other.

Parts of BitMAC have been implemented with the BTnode3 platform [Tex] and evaluated in experiments. Further delays have been derived by analysis. Besides its deterministic communication, further advantages of BitMAC comprise its internal synchronization and support for duty cycling. An inherent drawback is its limitation to tree structures.

6.2.3 SYNchronized MAC

The SYNchronized MAC (SYN-MAC) protocol also adopts the binary countdown principle and has been proposed in [WUT05, PW07]. Its objective is the reduction of collisions, which are observed with IEEE 802.11 [Ins12a] in multi-hop networks with hidden stations. The protocol requires synchronization among all nodes, which is, however, not further discussed by the authors. Instead, they refer to external solutions like GPS [US 08] or the cellular systems.

SYN-MAC subdivides time into three periodical intervals: *Contention interval*, *hidden station elimination interval*, and *data interval*. The first two intervals solve contention among senders, whereas data frames are exchanged in the third interval.

In the contention interval, a binary countdown protocol is applied. For this purpose, the interval is subdivided into so-called *contention slots*, where the protocol processes one bit in each slot. Binary numbers are either generated randomly to improve fairness or from priority classes to support QoS. Since they are not necessarily unique, several winners can emerge in the contention interval. This problem is, however, neglected under the assumption of sufficiently large binary numbers. Different from other binary countdown protocols, dominant bits are not just sent as buzz signals but as regular MAC frames (referred to as *contention signals*) and contain the MAC address of the receiver as payload. This information can be evaluated by receivers if no collision occurs. Thus, a node reading its MAC address in a received contention signal in the i-th contention slot is aware of its receiver role and generates a so-called *contention bit mask*, where the i-th bit is 1. Afterwards, it ignores further contention signals in this contention interval. A node receiving a contention signal and reading a different MAC address stops participating in the current contention – both as potential sender and receiver.

In the hidden station elimination interval, all receivers that have received a contention signal correctly send their generated bit mask in so-called *hidden station clear messages* (HCMs). All potential senders in range receiving this message compute the binary AND of the received bit mask and the binary number, which they sent during the contention interval. If the result is different from 0, they become transmitters in the subsequent data interval. Otherwise, they are aware of being beaten.

An enhancement of SYN-MAC w.r.t. throughput is presented in [PW07] and requires knowledge about the neighborhood of nodes. Instead of always stopping if a contention signal with a different MAC address is received, a node first checks whether the read MAC address belongs to a node in the direct neighborhood. If this is not the case, the node continues contention, thereby alleviating the exposed station problem.

The protocol is evaluated by numerical analysis and simulations [WUT05, PW07] w.r.t. collision probability, delay, and throughput. Implementations do not exist. A drawback of SYN-MAC is the need for extra hardware to establish synchronization. Furthermore, network resources are potentially wasted if data frames do not fill in data intervals completely or if contention intervals do not produce winners due to identical binary numbers. For that reason, no upper delay bounds can be provided.

6.2.4 Binary Countdown MAC

Binary Countdown MAC (BCMAC) is a cluster-based MAC protocol for WSNs [KBK07]. Its main objective is the reduction of energy consumption, in particular, regarding idle listening and collisions [YHE02]. The protocol is an extension of the cluster-based MAC protocol BMA (Bit-Map Assisted MAC) and incorporates a binary countdown protocol to solve contention of intra-cluster communication. The behavior of BCMAC consists of two recurring phases: In a *setup phase*, cluster heads and followers are determined based on the amount of energy that is required for communication within the cluster. This phase is similar to BMA. The second phase (*steady-state phase*) comprises event-driven communication within the cluster. To resolve contention among nodes of the same cluster before data transfer, a binary countdown protocol is executed, where bit sequences are derived from unique node identifiers. Thus, collisions within the same cluster are avoided entirely. Since nodes without data to send can switch their transceiver off, energy waste due to idle listening is reduced.

BCMAC applies the principle of binary countdown in a simple and – by using node identifiers as priorities – collision-free variant, and transfers it to the context of clustering. As a consequence, it requires that all nodes of a cluster are in sensing range of each other. Conflicts between neighbored clusters are not considered but are implicitly solved due to unique node identifiers. Synchronization, which is required to separate bit times of the binary countdown protocol and to synchronize contest of different clusters, is not discussed. BCMAC is analyzed regarding energy consumption. Implementations do not exist.

6.2.5 Wireless-Dominance

Wireless-Dominance (WiDom) is a binary countdown protocol for sporadic message streams with static priorities [AT05] and is strongly motivated by CAN [Int04]. Enhancements, specializations, and applications have been presented in numerous publications [PAT06, PATR07a, PAT07, PATC09, PGAT09, VA10, VA11, APTG11]. Initially, WiDom comes with an internal decentralized synchronization protocol [AT05], but variants with external or master-based synchronization exist as well [PGAT09]. Variants of WiDom have been proposed for single-hop networks (e.g., [AT05]) and multi-hop networks (e.g., [PATR07a]). The base cycle of WiDom consists of three phases: A *synchronization phase*, a *tournament phase* to determine the node with the highest priority by executing a binary countdown protocol, and the *data transfer phase*. Similar to CAN, WiDom uses message-based priorities and encodes low numbers with high priorities, i.e., 0s are dominant and 1s recessive.

In the first paper [AT05], the protocol is introduced without explicit name and for single-hop networks. The name WiDom is first mentioned in [PAT06], where the implementation of the protocol's single-hop variant is presented, which is based on TinyOS[4] and the MICAz platform [MEMarb]. Similar to the Imote 2 [MEMara], the MICAz is equipped with TI's CC 2420 transceiver [Tex07] (see also Sect. A.2). But different from our implementation of black bursts by irregular MAC frames, WiDom realizes dominant bits with unmodulated carriers, which are supported by the test mode of the transceiver. Details on the implementation of WiDom's multi-hop variant are presented in [PATR07a]. It is developed with Nano-RK[5] and supports the wireless sensor platforms MICAz and Firefly[6]. In [PGAT09, VA11], a more efficient implementation of WiDom is introduced, which is based on two additional hardware extension boards (so-called WiFLEX boards) for MICAz and FireFly nodes. They enable faster tournament phases and are the foundation of WiDom's variant with master-based synchronization (also called *slotted WiDom* [VA11]).

An extensive and summarizing work about WiDom's single-hop variant can be found in [PAT07]. Besides describing mode of operation, implementation, and protocol timings, this paper also provides a response time analysis, which can verify the timely transmission of messages as function of workload. Furthermore, it presents results of experiments with 99.99% success rates. An interesting fact is that in some cases, the response times are higher in experiments than the theoretical bounds. The authors could trace this discrepancy back to false positives that were caused by noise. As countermeasure against false negatives, a further single-hop variant of WiDom is devised [PATC09], in which dominant bits are relayed by receiving nodes.

[4]`http://www.tinyos.net`
[5]`http://www.nanork.org/`
[6]`http://www.ece.cmu.edu/firefly/`

In [PATR07a, PATR07b], WiDom is extended to solve hidden station problems in multi-hop networks. The extension is based on the network-wide forwarding of synchronization pulses and the two-hop propagation of priority bits. Data transfer is still one hop. Due to arbitration radii of two hops, this variant of WiDom allows multiple simultaneous winners, but also suffers from multi-hop competing problems (see Sect. 3.5).

Applications of WiDom – like the distributed calculation of aggregated values (e.g., MIN, MAX) – are presented in [PGAT09] for single- and multi-hop networks. While these calculations are straightforward in single-hop networks, a cluster-based approach is adopted in multi-hop networks, where values are first determined within clusters and then sent to a central master. A further single-hop application of WiDom is in association with the interpolation of sensor readings [APTG11]. Here, WiDom is applied to determine local extrema of differences between interpolated sensor values and actual sensor readings. This application has similarities to the application of ACTP to TOD (see Sect. 4.2.2), where also priorities are derived from dynamic error values.

The presentation of WiDom is very thorough and extensive, and also several implementations, simulations with probability models, and experimental evaluations exist. A drawback of WiDom is the limitation of its arbitration radius to one and two hops, respectively, thereby requiring hop-by-hop arbitration if data must be sent across longer routes. Furthermore, synchronization overhead is high due to very frequent and inefficient resynchronizations. Extended variants of WiDom with WiFLEX boards indeed reduce this overhead, yet they introduce costs of additional hardware and still require resynchronization very frequently. Another drawback is the missing support of duty cycling due to WiDom's event-driven operation mode, which prohibits the switching off of transceivers in periods without transmissions. A further drawback is regarding the efficiency of WiDom's implementations: In [PAT06], for instance, carriers of dominant bits have a duration of 486 µs, since the authors observed a high rate of false negatives with smaller pulses. Because we also obtained very good results with the same transceiver and shorter pulses (see Chapter. 5), the necessity of such large pulse durations is somehow irreproducible and has maybe its origins in additional overhead of the used OS (TinyOS).

6.2.6 The Binary Priority Countdown Protocol

The Binary Priority Countdown (BPC) protocol is a MAC protocol for wireless single-hop networks [KOK13b]. In BPC, arbitration, which is called *contention resolution period*, consists of one to many *contention intervals*, each running a binary countdown protocol with priorities of N bits length. Priorities are calculated randomly and once per contention resolution period, where lower values represent higher priorities. Because the arbitration may consist of several contention intervals, the priority space is very flexible and usually larger than $\{0, \ldots, 2^N - 1\}$.

Contention resolution periods work as follows: Stations with priorities larger than $2^N - 1$ stay passive in the current contention interval, but reduce their priority by $2^N - 1$ if they do not detect any transmission. Other stations or stations, which decreased their priority in previous contention intervals correspondingly, participate actively by running the binary countdown protocol with the two's complement of their priority. After winning the contest, a station is allowed to transmit a data frame. Because priorities are calculated randomly, collisions of data frames can occur, which are detected by the absence of ACK frames and cause doubling the priority space. For a station losing the contest, two alternative strategies are opposed: Either discarding the priority and calculating a new value, or reducing the own priority by the priority of the winner. To shorten the duration of contention resolution phases, the authors present an optimization that is based on the fact that a data frame can additionally serve as dominant bit. Thus, a station, which is aware of winning the contest prematurely[7], can bring its data transmission forward to additionally utilize it as dominant bit.

The protocol and its alternatives are evaluated by analyses and simulations with ns-2. Synchronization, which is required to start contention intervals simultaneously, is not discussed. A big advantage of BPC over other binary countdown protocols is its dynamic priority space.

In [KOK13a], an enhancement of BPC, called *Contention Overhead - Adaptive Binary Priority Countdown* (CO-ABPC), is presented, changing two parts of BPC: First, stations always calculate a new priority if they have lost more than k consecutive times. Thereby, the so-called problem of *collision memory* is decreased. Second, arbitration overhead is reduced by determining the optimal priority space and by using this value after collisions. To determine this value, the current contention overhead in the network is evaluated.

6.3 Discussion

Though busy tone and binary countdown protocols differ in their mode of operation, their objective and outcome are similar. In particular, both protocol classes provide medium arbitration with deterministic result as long as priorities are unique. Basically, even value transfer is possible with both, yet limitations due to multi-hop competing problems have to be considered.

The most important difference between both protocol families lies in the encoding of priorities: For busy tone protocols, priorities are encoded in the duration of a buzz signal. Under the assumption of uniform length distributions, the average arbitration delay is thereby half as large as the worst-case delay. Binary countdown protocols transmit binary sequences and – with the exception of BPC – achieve identical average and worst-case delays. As further difference, delays with busy tone protocols increase linearly with increasing priorities, whereas the increase is logarithmic with binary countdown protocols. Though binary countdown protocols seem to be more efficient at first glance due to the more compact encoding of priorities, it has to be considered that they may suffer from the frequent switching of the transceiver's mode. Particularly, with typical wireless sensor nodes, the resulting delays can be very high. Thus, no universal statement is possible and busy tone protocols may even outperform binary countdown protocols if the priority space is small.

Independent of their category, the discussed protocols differ in following aspects:

- **Synchronization.** Though both busy tone protocols and binary countdown protocols require synchronized starts, synchronization is, in general, more important for binary countdown protocols, since nodes must assign a detected medium occupancy to a bit position unambiguously. With busy tone protocols, the start of the buzz signal can establish on-demand synchronization implicitly (see e.g., BP(S)-MAC). Some of the discussed protocols – like WiDom and CSMA/IC – provide internal synchronization, whereas others like SYN-MAC rely on an external synchronization source like GPS. Protocols with multi-

[7]This is, for instance, the case, if the station is still in contest after the second to last bit and its last bit is dominant.

hop support furthermore vary in the range of synchronization, which can either be local (CSMA/IC) or network-wide (BitMAC).

- **Priorities and guarantees.** By assigning unique priorities, many of the discussed protocols guarantee unique winners. However, there are also other protocols like SYN-MAC, BPC, and BP(S)-MAC, which generate priorities randomly and can consequently not give any guarantee. Protocols with unique winners differ in the derivation of priorities: While some protocols (e.g., ES-DCF or BCMAC) use node ids, others (like WiDom) derive priorities from message types. CSMA/IC and BROADEN incorporate even both: Non-unique message priorities and node ids to break ties.

- **Considered topologies and network models.** Many outlined protocols consider either single-hop networks or multi-hop networks without hidden stations. Multi-hop networks with hidden stations are only supported by some protocols like WiDom, BitMAC, and SYN-MAC. Though these protocols can deal with hidden stations, their arbitration radius is limited to one or two hops. In this regard, limiting the arbitration radius does usually not produce a network-wide maximal number of winners due to multi-hop competing problems (see Sect. 3.5).

- **Traffic pattern.** A further distinctive feature is the addressed traffic pattern. BitMAC, for instance, applies binary countdown only a few times during network setup. Other protocols like the protocol of Sobrinho and Krishnakumar can only deal with strong periodical traffic, whereas others like WiDom target sporadic traffic.

- **Application domain.** The protocols are often advocated as an alternative for IEEE 802.11 DCF or other CSMA/CA-based arbitration protocols. Thus, the main application of them is medium arbitration. Only a few (e.g., BitMAC and WiDom) actually address collision-protected value transfer and data aggregation.

Different from ACTP, all outlined protocols only support arbitration with one or two hop radius. Hence, subsequent data transfer is limited to one hop and several arbitrations are required if data has to be transmitted over several hops. By incorporating ACTP into macro slots with the internal synchronization protocol BBS, ACTP demands no additional hardware and has a lower synchronization overhead than other protocols with internal synchronization (like WiDom or CSMA/IC), which enforce (re-)synchronization before each arbitration. Though also other protocols like WiDom and BP(S)-MAC address sensor networks, these protocols are – different from ACTP – not compliant with duty cycling. A summary and qualitative comparison can be found in Table 6.1, where w.r.t. WiDom, the comparison is based on the protocol's variant without extension boards.

Besides their common origin as binary countdown protocol and their realization with the same transceiver, further commonalities exist between ACTP and WiDom regarding their target applications. In this regard, an example can be found in [PGAT09], where WiDom is – similar to ACTP's application to TOD (see Sect. 4.2.2) – applied with priorities that are derived from error values. In both applications, the protocols realize a MEF message scheduling strategy, yet the utilization of the communicated error values is different (monitoring of a physical environment vs. input of a control algorithm).

	network model	**deterministic time bounds**	**range**	**priorities**	**spatial reuse**	**synchronization**	**duty cycling**
ACTP	multi-hop with hidden stations	✓	n-hop	flexible	✓	internal, periodic, det. offset (BBS)	✓
CSMA/IC, BROADEN	multi-hop w/o hidden stations	✓	1-hop	message-based plus node ids, unique	✓	internal, on demand, probabilistic	X
BitMAC	multi-hop tree structure	✓	2-hop	flexible	✓	internal, periodic, det. offset	✓
SYN-MAC	multi-hop with hidden stations	X	2-hop	random	✓	external (e.g., GPS)	X
BCMAC	multi-hop cluster-based w/o hidden stations	✓	1-hop	node ids, unique	✓	unknown	✓
WiDom	multi-hop with hidden stations	✓	1-hop / 2-hop	message-based, unique	✓	internal, on demand, det. offset	X
BPC, CO-ABPC	single-hop	X	1-hop	random, arbitrary length	irlvt	unknown	X

Table 6.1: Comparison between ACTP and previous binary countdown protocols. (X – not supported, ✓ – supported)

PART II

SDL REAL-TIME TASKS: DESIGN, IMPLEMENTATION, AND EVALUATION

The biggest challenge in the development of a real-time system is the *timely* production of *correct* results. Thus, timeliness is a fundamental requirement for real-time systems and distinguishes them from other types of systems. In some real-time systems, the too late delivery of a result is synonymous with the absence of the result and can have – depending on the concrete scenario – far-reaching implications. In this regard, real-time systems are classified either as *hard* real-time systems, where the miss of a deadline can have a catastrophic outcome, or *soft* real-time systems, which can tolerate time constraint violations to some degree [Liu00]. To meet the requirements of real-time systems, it is not sufficient to use just "very fast" hardware. Instead, the fulfillment of time constraints has to be omnipresent during the entire development process and includes the selection of suitable hardware, adequate scheduling and load control, and time analyses of the system.

Though the SDL standard advertises SDL as language for real-time systems [Int12c], projects with SDL have shown that the language is indeed well-suited for the design of such systems, but badly performs in implementations that are generated from the specification automatically. Referring to this, examples can be found in [BSP11], where different manual and model-driven IEEE 802.15.4 [Ins11] implementations are compared, and in [BCGM14], where an SDL implementation of a synchronization and TDMA-based protocol is evaluated against a manual implementation. In both examples, it is not only shown that manual implementations are more efficient, but it is also demonstrated that SDL implementations are significantly less predictable. The differences between SDL implementations and manual implementations can be traced back to the gap between SDL's view of a perfect world with unlimited resources and concurrency, and the properties of the real world with limited memory and processing units. Here, the biggest gap particularly arises w.r.t. the scheduling of the SDL system, where the strongly concurrent execution model of SDL has to be serialized by the implementation.

This part of the thesis presents a new approach to deal with the mismatch between SDL's perfect world and the real world, and to improve the predictability of SDL implementations. For this purpose, real-time tasks [Kop97], which are a concept of real-time systems to structure a system and its execution, are transferred to SDL and formally incorporated into the syntax and semantics of the language. To furthermore eliminate the scheduling nondeterminism of SDL implementations, a priority-based extension of SDL's execution model is proposed on the basis of SDL real-time tasks and implemented in a novel scheduling strategy for SDL systems.

Though the objective of SDL real-time tasks is the improvement of the applicability of SDL for real-time systems, further considerations are necessary to actually use SDL for (hard) real-time systems. In particular, this thesis does not propose new analytical methods to derive worst-case execution times (WCETs) of SDL real-time tasks and to perform schedulability tests. Instead, it presents results of experimental evaluations to demonstrate the practical benefits of the presented language extensions. While such evaluations are sufficient to compare the new approach with state-of-the-practice solutions, to quantify the benefits, and to derive probabilistic WCETs, they are still purely statistical and, therefore, inadequate if deterministic guarantees regarding delays and runtimes are required.

Contributions

To improve SDL's applicability in real-time systems, language extensions are required to incorporate an adequate priority-based execution model. With SDL real-time tasks, such extensions have been developed, syntactically and semantically integrated into the language, and evaluated. In detail, the following contributions are presented in this part of the thesis:

- The adoption of real-time tasks [Kop97] demonstrates the applicability of an approved concept from real-time systems in SDL. In particular, formal definitions of real-time tasks are given in the context of SDL, and incorporated into SDL's syntax and semantics. For this purpose, a distinction between *code unit* and *execution unit* is introduced.

- By introducing *distributed* SDL real-time tasks, the concept of real-time tasks is extended to accommodate SDL's application in distributed systems and protocol engineering.

- Together with SDL real-time tasks, a new priority-based execution model is introduced in the formal semantics of SDL that outperforms existing prioritization measures like signal priorities and priority inputs. Different from existing measures, priorities in SDL real-time tasks are dynamically associated with transition executions and not statically assigned to structural elements of an SDL specification.

- The design and implementation of a new scheduling strategy – called *task scheduling* – is presented, which implements the priority-based execution model of SDL real-time tasks to order transition executions at runtime. The resulting execution order is dynamically created and orthogonal to SDL systems' static structure of SDL processes. As further difference from prioritization approaches of existing SDL tools, task scheduling serializes the concurrency of SDL systems without relying on a specific (real-time) operating system.

- The new scheduling strategy is integrated into an existing scheduling framework to enable its use in scheduling-aware system specifications, where the designer can select and configure a desired scheduling strategy from within an SDL specification.

- New SDL features are proposed to suspend SDL real-time tasks in a way that is transparent and influenceable by the system designer. Thereby, load of a system becomes better controllable and delays of critical transition executions can be decreased further.

- By implementing SDL real-time tasks in an entire SDL tool chain, it is shown that the approach is compliant with the model-driven development process SDL-MDD [Got07]. It is furthermore demonstrated that priorities of SDL real-time tasks are well-suited to guide the automatic transformation of SDL specifications to implementations and that –

different from state-of-the-practice approaches – no additional implementation phase and implementation-specific priority schemes are necessary.

- Functional and performance evaluations provide evidence that SDL real-time tasks and task scheduling have practical benefits and outperform state-of-the-practice solutions w.r.t. the amount and predictability of WCETs.
- By a survey on related work, it is illustrated that the dynamical assignment of priorities to transition executions and the support of distributed tasks are unique features of SDL real-time tasks. The discussion of related work furthermore shows that there are several orthogonal proposals, which improve SDL's applicability in embedded real-time systems regarding different criteria like energy consumption and can be combined gainfully.

Outline

This part of the thesis is subdivided into five chapters and supplemented by two appendices.

- **Chapter 7** presents a survey of SDL; including its history, syntax, and semantics. Furthermore, the model-driven development process SDL-MDD is outlined. Though this chapter does not contain new results, it introduces foundations for subsequent chapters.
- **Chapter 8** motivates the necessity of SDL extensions for real-time systems. For this purpose, shortcomings of SDL w.r.t. implementability and predictability of SDL implementations are investigated. With real-time tasks, this chapter surveys a concept from the domain of real-time systems to structure the design and execution of time-critical systems, and transfers this concept to the syntax and semantics of SDL. Furthermore, language elements to suspend the execution of real-time tasks are presented. In addition, this chapter illustrates the application of SDL real-time tasks by means of an example with parts of a comprehensive communication protocol and relates SDL real-time tasks to existing prioritizing language elements of SDL.
- **Chapter 9** surveys the implementation of SDL real-time tasks in a tool chain consisting of the code generator ConTraST, the SDL runtime environment SdlRE, and the SDL environment implementation SEnF. The focus of this chapter is on a new SDL scheduling strategy, which is part of SdlRE and has been especially devised to run SDL real-time tasks. This chapter furthermore outlines the simulator framework FERAL and extensions enabling functional and performance evaluations of SDL systems with the support of hardware-in-the-loop simulations.
- **Chapter 10** presents results of evaluations of SDL real-time tasks. W.r.t. their functional evaluation, the chapter outlines an SDL test system, in which a new test category is introduced for SDL real-time tasks, testing their correct creation, scheduling, and suspension. Furthermore, performance evaluations of SDL real-time tasks are presented to provide an experimental proof-of-concept of the practical benefits of SDL real-time tasks. The evaluations realize an adaptive cruise control scenario and a system with an inverted pendulum, and compare both delays and overhead of SDL real-time tasks and task scheduling with state-of-the-practice scheduling solutions.

- **Chapter 11** surveys related work. Since there are actually only very few comparable previous works, the chapter provides a broad overview and includes an outline of SDL implementation variants and other language extensions with different objectives. Many of the surveyed works are not competitive proposals, but can be combined with SDL real-time tasks gainfully.

- **Appendix B** summarizes the incorporation of SDL real-time tasks into the formal syntax of SDL. For this purpose, extensions of Basic SDL-2010 [Int12d] are presented.

- **Appendix C** presents extensions of the semantics of SDL in order to incorporate the behavior of SDL real-time tasks formally. The extensions are based on SDL-2000 [Int00], since it contains the last available version of SDL's formal ASM semantics.

7. Chapter

The Specification and Description Language – An Outline

The Specification and Description Language (SDL) is a well-established and standardized language for the development of distributed systems. This chapter is concerned with foundations for subsequent chapters, which will introduce extensions of SDL and their implementation in a semantically integrated tool chain. In Sect. 7.1, this chapter summarizes the SDL standard and the history and concepts of the language. By outlining language elements of SDL, an overview of the language's syntax and semantics is given as well. Afterwards, a model-driven development process with SDL as design language, called SDL-MDD, is outlined in Sect. 7.2. All language extensions that are introduced in the following chapters of this thesis can be applied in compliance with SDL-MDD.

7.1 SDL in a Nutshell

SDL is a formal language to describe behavior, data, and structure of reactive distributed systems [Int12c]. It is standardized in the Z.100 series of the International Telecommunication Union[1] (ITU). The original target application of SDL is in the telecommunication domain, but it has also been applied to the development of (real-time) communication systems in general [BH93, MT00]. A famous example of the utilization of SDL for communication protocols can be found in Annex J of the IEEE 802.11 WLAN standard [Ins12a].

During its long history of almost 40 years, SDL has evolved from notation guidelines for state machines to a language with several syntaxes and complete formal semantics [Ree00, Ree11a, EGG+01]. Since 1976, the SDL standard has been published in seven major versions and several minor editions. A summary of all versions and changes is given in Table 7.1. With a four year update interval of its standards, SDL was very lively during the first years. Currently, SDL has reached a steady state with a lower publication interval and only minor changes to the latest standards. This particularly holds for its syntax, which is in the core already described in SDL-80 [Ree00].

While the focus of SDL was on a common notation for state transition diagrams at the beginning, it was recognized that structure mechanisms, semantics, and tool support are required to deal with large and complex systems. Thus, more and more features were added to the SDL standard over the years and executability has become a key issue [EGG+01]. In this regard, the SDL-2000 standard, where in Annex F of Z.100 [Int00], the dynamic semantics is formally

[1] http://www.itu.int

standard	year	focus
SDL-76	1976	• First SDL symbols and definitions for state transition diagrams (behavior only)
SDL-80	1980	• (Still) informal but more precise description of SDL • Support of hierarchical structure mechanisms [EHS97]
SDL-84	1984	• First interpretation model based on mathematical graphs
SDL-88 [Int88]	1988	• Formal foundations of today's SDL: SDL-PR, SDL-GR, abstract syntax, semantics • Data definitions based on algebraic models
SDL-92 [Int93]	1993	• Introduction of nondeterminism and object orientation (block-/process types, packages)
	1995	• ASN.1 support [Int95]
	1996	• *SDL-96*: Actually an addendum to SDL-92 with minor changes, corrections, and simplifications • Definition of the Common Interchange Format (CIF) [Int96a]
SDL-2000 [Int99a] [Int02] [Int07]	1999	• Stronger focus on using SDL for design and implementation • Operational dynamic semantics with ASMs • Introduction of *exceptions*, textual algorithms in SDL-GR, *composite states* • New non-axiomatic data model with object orientation • Support of SDL combined with UML [Int99b]
	2002	• Minor updates and corrections, and reorganizations
	2007	• Minor changes and corrections of flaws
SDL-2010 [Int12c]	2011	• Separating a subset of SDL in a *basic SDL definition* • Support of Unicode • Removal of unimplemented features like *exception handling* • Extensions with new language elements such as *input via*, *timer supervised states*, *multiple priority input levels*, and *signal priorities* • (Bindings to other languages like Java, C, and C++)

Table 7.1: The history of SDL [Ree00, Ree11a, Int12c].

specified in an operational way by using Abstract State Machines (ASM) [BS03], represents an important step. Since operational semantics is very close to implementations [Pri00], the dynamic semantics of SDL-2000 is more easily understandable for system developers and was also used as foundation for SDL tools [FGW06, PvL03b, PvL03a]. To further improve SDL's applicability as implementation language, many couplings to other languages – like to the *Abstract Syntax Notation One* (ASN.1, [Int08]) and to the *Unified Modeling Language* (UML, [Obj22]) – and many concepts known from usual programming languages – e.g., temporary variables or loops and conditions – were brought to SDL. Furthermore, the way of defining data types was changed in SDL-2000 from algebraic axioms to a constructive and algorithmic data approach [Ree09, Ree11b, vLoM03, San00]. With the same objective, SDL-2010 was also meant to

document	content
Z.100	Overview of SDL-2010 and its documents [Int12c]
Z.101	Basic SDL: Core features of SDL-2010 [Int12d]
Z.102	Comprehensive SDL: All SDL-2010 features [Int12e]
Z.103	Shorthand notations and annotations [Int12f]
Z.104	Data and action language [Int12g]
Z.105	ASN.1 binding [Int12h]
Z.106	Common Interchange Format (CIF) [Int12i]
Z.107	Object-oriented data [Int12j]
Z.109	UML profile for SDL-2010 [Int12b]

Table 7.2: Documents of the latest SDL standard SDL-2010 [Ree11a, Int12c].

allow bindings to other programming languages like C, C++, or Java [Int12g]. This, however, was not completed at the time of SDL-2010's approval. In summary, SDL is no longer used as description language to visualize and analyze system properties only, but it is used as stand-alone or integrated[2] graphical implementation language for the model-driven engineering of distributed systems.

The current SDL standard is SDL-2010 and has been approved in December 2011. It consists of ten main recommendations and several addenda. An overview of all documents is given in Table 7.2. While the previous SDL-2000 standard came with formal semantics, the semantics is missing in the current SDL-2010 standard due to a lack of resources [Ree11a]. Instead, Z.100 Annex F of the previous SDL-2000 standard is referenced [Int00]. This document is indeed out-of-date, but since changes between SDL-2000 and SDL-2010 are only minor, the document still covers most language parts.

A big advantage of SDL is its support by commercial as well as academic tools. Examples of commercial tools are PragmaDev's *Real-Time Developer Studio* (RTDS) [Praar], IBM's Rational SDL Suite [IBMar], and Cinderella [Cinar, RKL05]. However, a drawback of all these tools is missing support of many features introduced in SDL-2000, i.e., most tools provide SDL-92 or SDL-96 only and there is currently no tool fully implementing SDL-2000 [Ree11a]. As a consequence, a major objective of SDL-2010 was to separate the core features of SDL, which should be supported by all tools, into a separate document (Z.101 [Int12d]).

7.1.1 SDL Syntax

The current SDL standard comes with four different syntaxes [Int12c]: SDL Graphical Representation (SDL-GR) and three levels of Common Interchange Formats (SDL-CIF). SDL-GR is the *visual and default* syntax that is used by developers to specify the structure and behavior of a system in form of diagrams. It is introduced in [Int12d, Int12e, Int12f, Int12g] and specified by context-free grammars in Backus-Naur-Form (BNF) with extensions for graphical language constructs [EGG+01, PvL03a, PvL03b]. Since it is more vivid than the CIF syntaxes, it is usually

[2]An example framework incorporating SDL is TASTE [PCD+11], which combines SDL with several other modeling and implementation techniques like Matlab Simulink, VHDL, and SystemC. Another example is a simulator in [SLO+10, MLON13] that supports SystemC and SDL models.

easier to learn. Though all four syntaxes are human readable, the CIF syntaxes are easier to handle by tools, because they are purely textual. The three SDL-CIF syntaxes are specified in [Int12i]. Level 0 of the syntax is also known as SDL-PR (SDL Phrase Representation). Its language coverage is equivalent to SDL-GR, thereby representing an alternative for developers to specify a system with text only. The next CIF syntax is CIF level 1 (also called CIF-PR), which, different from SDL-PR, also allows partial specifications. SDL-CIF level 2 is also called CIF-GR and is intended for the exchange of SDL-GR diagrams between different tools. It is an extension of CIF-PR with additional elements to describe characteristics of the graphical representation.

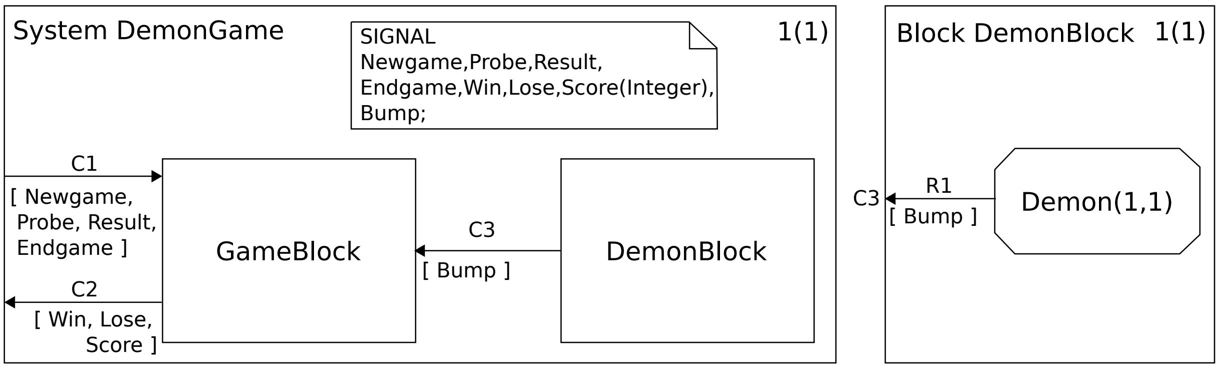
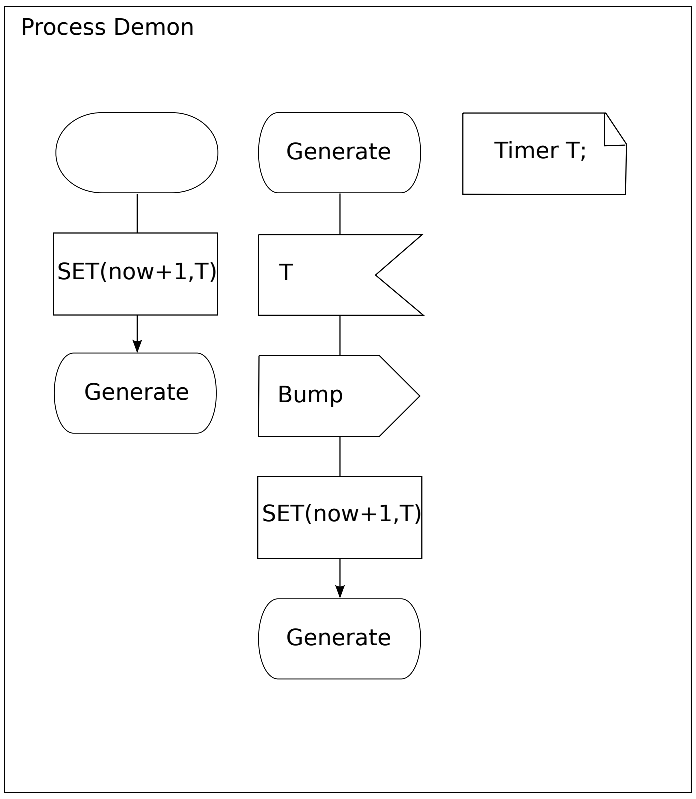


Figure 7.1: Example of the structure of an SDL system with SDL-GR [Int12i].

Figure 7.1 gives an example of the structure of an SDL specification in SDL-GR. The example is taken from [Int12i] and shows signal declarations and two SDL blocks. Signals defined in the text symbol may also have parameters (see signal `Score`) and can be used in the system diagram as well as all referenced block and process diagrams. The SDL blocks `GameBlock` and `DemonBlock` are connected by channel `C3` for the unidirectional transfer of SDL signal `Bump`. `GameBlock` is additionally connected to the environment of the system by two channels. By sending SDL signals to the environment, an SDL system can interact with the underlying hardware/software platform to access hardware peripherals and to communicate with other network nodes. If, vice versa, signals are sent from the environment to the system, the SDL system can be informed about external events, thereby enabling the specification of stimulus–response behavior. The SDL block `DemonBlock` is referenced in the system diagram and specified in the diagram on the right-hand side of Fig. 7.1. It owns only one SDL process `Demon`, which defines (parts of) the behavior of the SDL specification.

SDL processes specify communicating extended finite state machines [EHS97]. In Fig. 7.2, the definition of process `Demon` is depicted exemplarily. Besides the SDL-GR notation, the process definition is also shown in CIF-GR. In the text symbol (top right), the SDL process defines a timer `T` that is set in the start transition (transition on left-hand side) for the first time. The transition consuming this timer (transition on right-hand side) outputs signal `Bump` and sets the timer again with the same interval. In addition to the language elements used in the example, SDL has many further features like decisions, procedures, dynamic process creations, composite states, and priority inputs.



```
/* CIF ProcessDiagram */
/* CIF Page 1 (1400,1000) */
Process Demon;
/* CIF DefaultSize (200,100) */
/* CIF Text (800,100) */
Timer T;
/* CIF End Text */
/* CIF Start (300,100) */
start;
/* CIF Set (300,250) */
SET(Now+1, T);
/* CIF NextState (300,400) */
nextstate Generate;
/* CIF State (550,100) */
state Generate;
/* CIF Input (550,250) */
input T;
/* CIF Output (550,400) */
output Bump;
/* CIF Set (550,550) */
SET(Now+1,T);
/* CIF NextState (550,700) */
nextstate Generate;
/* CIF End ProcessDiagram */
endprocess Demon;
```

Figure 7.2: Example of an SDL process definition in SDL-GR and CIF-GR [Int12i].

7.1.2 SDL Semantics

Though SDL-2010 does not come with a formal semantics, many parts of SDL-2000's formal semantics are still valid. In the following, a summary of the static and dynamic SDL-2000 semantics is given, which is defined in Annex F of SDL-2000 Z.100 [Int00].

Starting from the concrete syntax of an SDL specification, the static semantics checks the SDL specification against well-formedness conditions [Pri00]. These conditions are given by using first order predicate calculus, because they can hardly be expressed with context-free grammars, and include, for instance, visibility checks of variables and SDL signals. If the checks are successful, the concrete SDL syntax is transformed into an abstract syntax tree[3], which is also formally defined in BNF, and again checked against constraints. The transformation consists of several steps: First, irrelevant details, like separators, are removed from the specification. Second, so-called shorthand notations – or sometimes referred to as *syntactic sugar* [PST07] – are removed in favor of their corresponding core SDL expressions. This transformation actually describes a mapping from concrete SDL syntax into concrete SDL syntax. Though shorthand notations do not add additional functionality to SDL, they make SDL more concise and keep the language core and the dynamic semantics small [Ree09, Ree11b]. Examples of a shorthand notation are asterisk states, which stand for all SDL states that are used in the actual context, and lists of output signals that are mapped to a sequence of single outputs. In the last step of the

[3]Actually, the SDL standard distinguishes between two abstract syntaxes, AS0 and AS1 [PvL03b]. Because the only purpose of AS0 is to unify the differences between the concrete syntaxes and is, therefore, very similar to the concrete syntaxes, we still refer to AS0 as concrete syntax and to AS1 as abstract syntax.

transformation, rewrite rules are applied to map elements of the concrete syntax to corresponding constructs of the abstract syntax, which is almost a one-to-one mapping [PvL03a, PST07].

The dynamic semantics defines the behavior of an SDL specification [Pri00, EGG$^+$01]. Its starting point is the abstract syntax tree (AS1) of the specification, which is mapped to primitives of the SDL Abstract Machine (SAM). The SAM is itself defined with an ASM [BS03]. It is executed under the control of the SDL Virtual Machine (SVM), which provides typical operating system functionalities for system initialization and firing of transitions [EGG$^+$01]. ASMs were originally published by Yuri Gurevich [Gur95] under the notion of *evolving algebras*. SAM and SVM fall into the class of asynchronous multi-agent ASMs and consider real-time aspects by providing a notion of time and timers [Int00]. Multi-agent ASMs consist of a set of ASM agents, each running its ASM program autonomously and interacting with other agents by sharing parts of the global state [She12]. ASM programs, in turn, consist of transition rules defining the moves and, thereby, the possible runs of an ASM agent. The overall run of a multi-agent ASM is accordingly defined on a partial order of all moves of ASM agents.

In the dynamic semantics [Int00], The SDL standard distinguishes between three types of ASM agents: *SdlAgentSets*, *Links*, and *SdlAgents*. For each agent type, a different ASM program is provided, which distinguishes between initialization and execution phase. Before initializing an SDL specification, there is only one *SdlAgentSet*, which represents the SDL system type of the specification. During the initialization, the system is recursively unfolded according to its structure of SDL blocks and processes, i.e., *SdlAgentSets* create contained *SdlAgents* and vice versa. Additionally, *Links* are generated to connect gates with each other. To illustrate the result of the unfolding, Fig. 7.3 presents the final system hierarchy of the demon game example.

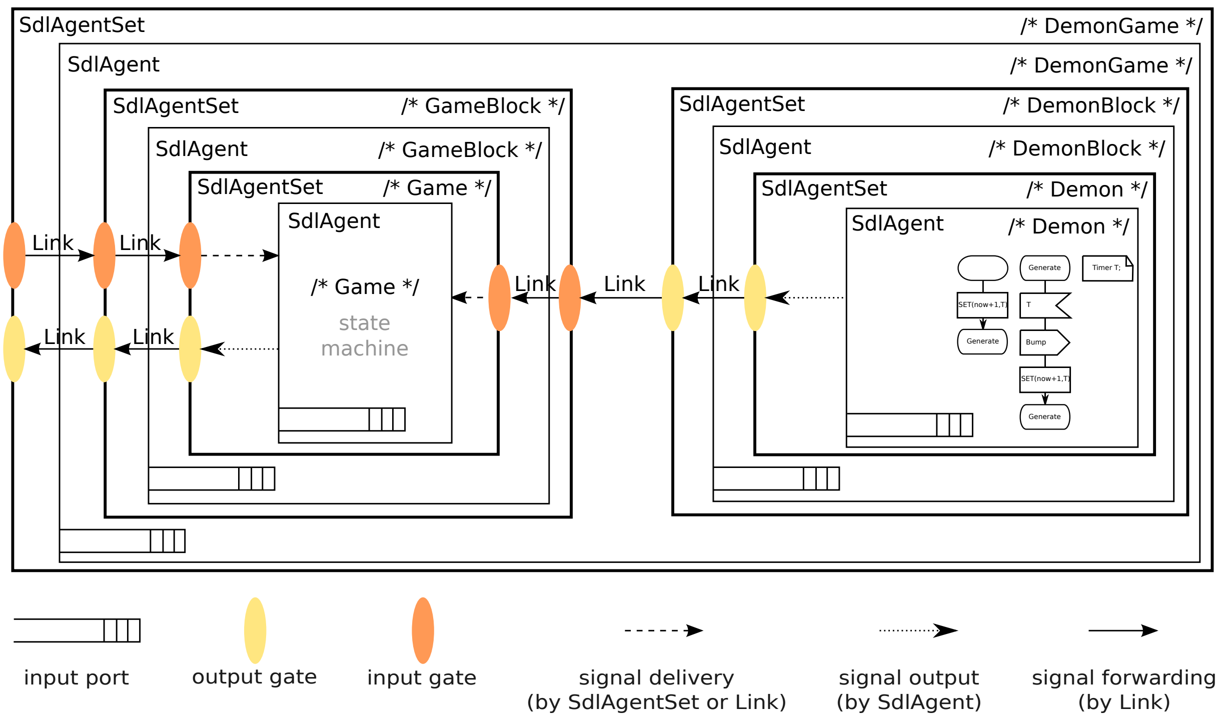


Figure 7.3: Hierarchical nesting of ASM agents after initialization. The notation is based on [Fli09, p. 61].

After initialization, all ASM agents concurrently run the execution phase of their ASM programs. For *Links*, this phase consists of the continuous, unidirectional, and serial forwarding of SDL signals between gates, whereas the task of *SdlAgentSets* is to deliver signals from an input gate to the input port of a contained *SdlAgent*. The state machines that are explicitly given by the system designer in the SDL specification are executed by *SdlAgents*. They have an input port attached that holds all arrived SDL signals sorted by their arrival time. To find the next transition to be executed, this queue is inspected and the corresponding signal is removed from the queue. Usually, this is the first signal of the input port, but SDL additionally provides language constructs to save or privilege signals, thereby changing the signal consumption order. Though the definition of active behavior by state machines is allowed in all *SdlAgents* since SDL-2000 – regardless of their origin as SDL system, block, or process –, no tool actually supports this feature for blocks and systems. Instead, they only support state machines in SDL processes. For this reason, *SdlAgent* and SDL process is often used synonymously, although the statement that only SDL processes define the active behavior of a system is no longer valid since SDL-2000.

As shown in Fig. 7.3, an initialized SDL system may also have gates to the environment to interact with the hardware/software platform. According to the SDL standard, there are only few restrictions on the SDL environment and it is just assumed that it consists of one or more agents [Int99a]. In real systems, this assumption alone is often not sufficient and further constraints, e.g., on the frequency of system stimulus, are stipulated [EHS97].

7.2 Model-driven Development with SDL

Due to an intuitive graphical representation and a formal semantics, SDL is well-suited for model-driven engineering, where abstract graphical system models build the central artifacts, guide through the entire development process, and are largely transformed and generated automatically [ZW06]. The objectives of such approaches are to achieve productivity gains and platform independence, and to improve quality and maintainability of products by applying approved structuring concepts and component reuse. A crucial paradigm of model-centric engineering is abstraction, i.e., the omission of irrelevant details like technological aspects and the focusing on objects of interest. To derive model-driven implementations, abstract models are refined to a less abstract level in each development step and finally transformed to an executable binary.

A model-driven development process, in which SDL is used as modeling language, is SDL-MDD (SDL Model-Driven Development) [KGW06, Got07, BCGI12]. SDL-MDD is a domain-specific, tool-supported, and iterative process to engineer distributed systems and communication protocols. It is architecture-centric and based on the Model Driven Architecture (MDA) of the Object Management Group (OMG) [Gro03]. The development phases of SDL-MDD describe a holistic approach ranging from capturing requirements to the deployment of the executable to the target platform. A key feature of MDA in general and SDL-MDD in particular is the distinction between platform-independent and -specific aspects in order to deploy the same functional system description on different platforms. Thus, it is also possible to use the same model for performance simulations as well as for the final release.

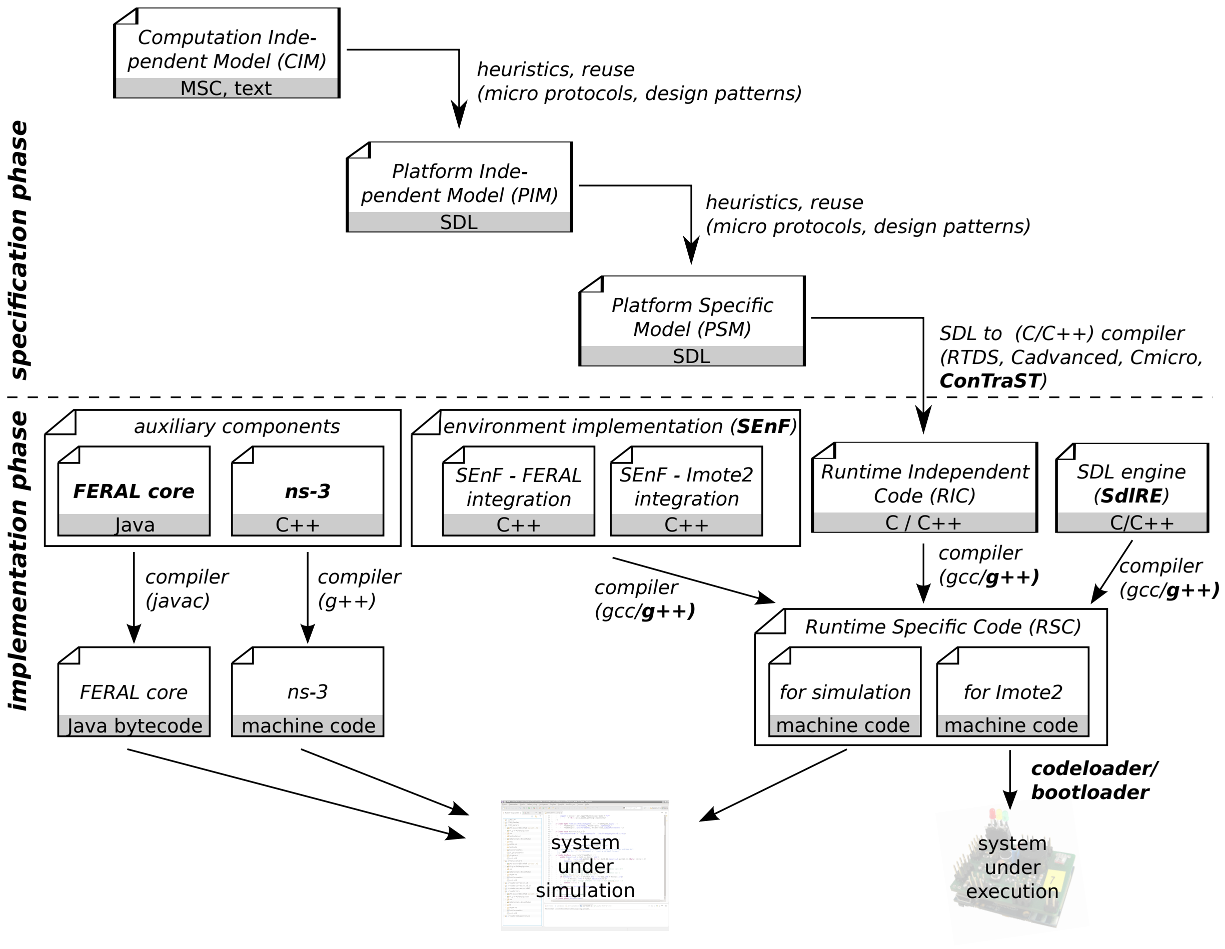

specification phase
implementation phase
Computation Independent Model (CIM)
MSC, text
heuristics, reuse (micro protocols, design patterns)
Platform Independent Model (PIM)
SDL
heuristics, reuse (micro protocols, design patterns)
Platform Specific Model (PSM)
SDL
SDL to (C/C++) compiler (RTDS, Cadvanced, Cmicro, **ConTraST**)
auxiliary components
FERAL core
Java
ns-3
C++
environment implementation (**SEnF**)
SEnF - FERAL integration
C++
SEnF - Imote2 integration
C++
Runtime Independent Code (RIC)
C / C++
SDL engine (**SdlRE**)
C/C++
compiler (javac)
compiler (g++)
compiler (gcc/**g++**)
compiler (gcc/**g++**)
compiler (gcc/**g++**)
FERAL core
Java bytecode
ns-3
machine code
Runtime Specific Code (RSC)
for simulation
machine code
for Imote2
machine code
codeloader/ bootloader
system under simulation
system under execution

Figure 7.4: Model-driven development with SDL-MDD [BCGI12]. Tools and components used in this thesis are written in bold.

The development process with SDL-MDD is shown in Fig. 7.4 and comprises two phases: A *specification phase*, in which models fulfilling all functional and platform-specific requirements are developed, and an *implementation phase*, in which the output of the specification phase is transformed to executable files that are tailored to the final runtime environment.

The specification phase starts with the Computation Independent Model (CIM), in which requirements are collected by means of Message Sequence Charts (MSCs, [Int11]) and informal text. These documents build the input to generate the Platform Independent Model (PIM), which is a formal specification of system structure and behavior with SDL-GR or an equivalent textual representation. Though the PIM is independent of a concrete hardware platform to enable a fast migration to different platforms, it is already functionally complete and allows behavioral validations. The PIM is extended to the Platform Specific Model (PSM), in which abstract interfaces to a specific hardware platform are added and corresponding parameters configured. This transformation step includes, for instance, the determination of a concrete communication technology. PIM and PSM are not built from scratch but by using modularization, approved reusable components – such as SDL micro protocols [GKS02, FGGS04, Fli09, FGG+05] and SDL design patterns [Got03, FGG+05] –, and heuristics. Tools to generate CIMs, PIMs, and PSMs are provided by PragmaDev's RTDS [Praar] and IBM's Rational SDL Suite [IBMar].

The implementation phase starts with the PSM and consists of several automatic transformation steps. Thereby, manual coding is avoided and SDL system and implementation are consistent by design [San00]. In the first step, the textual representation of the PSM is transformed into an imperative or object-oriented language such as C or C++. This step is supported by industrial as well as academic tools. Examples of commercial tools are the C/C++ code generators of PragmaDev's RTDS [Praar] and Cmicro/Cadvanced of IBM's Rational SDL Suite [IBMar]. The *Configurable Transpiler for SDL to C++ Translation* (ConTraST) [FGW06, Fli09] is an example of an SDL-to-C++ transpiler from an academic project (see also Sect. 9.2). The resulting artifact of this first transformation step is independent of the runtime environment of the target platform and called Runtime Independent Code (RIC).

The RIC is further compiled into platform-specific machine code called Runtime Specific Code (RSC). For this step, a corresponding platform-specific compiler is used, which is, for instance, a compiler of GNU's compiler collection[4]. The RSC is additionally augmented by an implementation of the SVM (*SDL engine*) to initialize and control the execution of the SDL system, and an SDL environment implementation providing platform-specific drivers and interface implementations. The SDL engine and SDL environment templates are usually provided by tool chains. Many tool chains also support the incorporation of SDL systems into common operating systems (OSs) like Windows[5] or Linux[6] or Real-Time Operating Systems (RTOSs). Thereby, parts of the SDL engine and the environment are realized by the OS. This particularly affects the scheduling and execution of the SDL system, which can be simplified by mapping SDL processes to tasks/processes of the OS.

To implement and evaluate the SDL extensions that have been developed in the course of this thesis (see Chapter 8), a semantically integrated tool chain consisting of IBM's Rational SDL Suite [IBMar], ConTraST [FGW06, Fli09], the *SDL Runtime Environment* (SdlRE [FGW06, Fli09]), and the *SDL Environment Framework* (SEnF [FGJ+05]) has been used and extended. SdlRE represents an SVM implementation and is further described in Sect. 9.4. SEnF provides implementations of the SDL environment for several target platforms (e.g., PCs with Linux and Imote 2 sensor nodes [MEMara]) and is presented in Sect. 9.5 in more detail.

The SDL-MDD process in Fig. 7.4 outlines the development for two different target platforms exemplarily: Imote 2 sensor nodes and a simulator called *Framework for the Efficient simulator coupling on Requirements and Architecture Level* (FERAL [BCG+13]). The RSC for deployments on Imote 2 is uploaded to the sensor node with help of a code- and bootloader [KE09]. The execution of the RSC for simulation purpose is triggered by the core schedulers of FERAL, which may also interconnect the RSC with other domain-specific simulators like ns-3[7] or Matlab Simulink[8]. Since both RSCs are based on the same RIC, accurate simulative performance and behavior evaluations are possible.

SDL-MDD has been applied in many academic projects. An example with own participation is the micro protocol-based development and performance evaluation of MacZ [CBGK08], a

[4]http://gcc.gnu.org/.
[5]http://windows.microsoft.com/
[6]https://www.kernel.org/.
[7]http://www.nsnam.org.
[8]http://www.mathworks.com/products/simulink/.

MAC layer for ad-hoc networks with protocols for deterministic synchronization and medium access [BGK08]. In [BCG09], we present a second application of SDL-MDD to the micro protocol-based development of deterministic transfer protocols for wireless networks. In this regard, a set of SDL-MDD-compliant design guidelines is introduced to reduce serialization delays. Further examples of SDL-MDD applications by former colleagues are the development of a routing protocol architecture for mobile ad-hoc networks [GGH08] and the interfacing of SDL with the real-time field bus technology FlexRay [BGW10].

8. Chapter

SDL Extensions for Time-critical Systems

Though SDL is a powerful and extensive language, it has limitations regarding the design and realization of real-time systems. Many of these limitations are a result of the gap between SDL's view of a perfect world with unlimited resources and the real world with shortages and delays. In this chapter, extensions are presented to reduce this gap and to improve the expressiveness of SDL and its applicability to real-time systems. The extensions are enhancements of previous work [Chr10], where scheduling-aware system specifications and SDL process priority scheduling has been proposed. However, as it turned out, these extensions were not sufficient, and the introduction of dynamic execution structures is required to bring the requirements of real-time systems and the properties of SDL systems together. By adopting the concept of real-time tasks [Kop97] in SDL, a suitable solution is presented in the following.

The structure of this chapter is as follows: In Sect. 8.1, shortcomings of SDL and the need for improvements are discussed. Thereafter, scheduling-aware system specifications are summarized in Sect. 8.2. The main part is Sect. 8.3, where SDL real-time tasks are formally defined and incorporated into SDL. Their formal incorporation into SDL's syntax and semantics is, however, left out and presented in Appendices B and C. Finally, Sect. 8.4 discusses the results and relates them to existing language elements of SDL.

Results presented in this chapter have been published in [5], [13], [15], [17], and [21].

8.1 Motivation

The correctness of a real-time system does not only depend on functional correct results but also on the point in time, when the results are available [Kop97, Sta88]. To meet this central requirement of timeliness, results must be produced as fast as *necessary*. This does not necessarily imply that results are produced without perceivable delay, which is a common misleading definition of a real-time system in the multimedia domain. Instead, in their original definition, worst-case delays of a real-time system must be predictable and sufficiently low, thereby requiring not just "very fast systems" but an adequate system design and scheduling strategy.

Examples of real-time systems can be found in various application domains. They are very prevalent in embedded systems and (networked) control systems. Their application scenarios range from soft real-time systems, which may miss a deadline, to hard real-time systems, where the violation of time constraints can have a catastrophic outcome [Liu00]. An example from the domain of protocols is TDMA, which is used to communicate over a shared medium reliably and with a desired quality-of-service. If in a system with TDMA-based medium access, nodes violate their transmission slots, frame collisions can occur, which can endanger the functionality of the entire system.

Though SDL is advocated as language for real-time systems [Int12c], it can actually not enforce a desired time behavior [EGG⁺01]. Instead, capabilities of SDL are limited to delaying and timeout mechanisms, thereby being more descriptive than prescriptive. In real-time systems, this is, however, not sufficient, and more control over the time behavior is required in order to fulfill time constraints. The following subsections give more details on these shortcomings.

8.1.1 Non-Determinism in the Semantics of SDL

In the SDL standard [Int12c], there are several parts with explicitly desired nondeterminism. An example are spontaneous transitions [Int12e], which cause transition executions without triggering signal or other precondition. Since there is no priority defined between spontaneous transitions and regular transitions that are triggered by signals, it is up to the implementation to define an order. A further example is the delivery of SDL signals within an *SdlAgentSet* containing several *SdlAgents*. Here, if the sender does not specify the `Pid` of the *SdlAgent* explicitly, the *SdlAgentSet* can forward signals to an arbitrary enclosed *SdlAgent*.

In addition to explicit nondeterminism, the SDL standard contains parts, which are well-defined in the conceptual SDL world, but lead to nondeterministic behavior in implementations. This particularly holds for the scheduling of an SDL system, because different from SDL's concurrent runtime model (see Sect. 7.1.2), a serialized execution model is required on real hardware.

The necessary steps to serialize the execution of agents are illustrated in Fig. 8.1. On the left-hand side, the ASM model of the agent hierarchy from Fig. 7.3 is shown. In this model, all agents run concurrently, executing transitions and processing signals from their input ports or gates. In implementations, the concurrent behavior has to be transformed into a more sequential model. In particular on a single-core hardware, which is currently still standard of embedded systems and shown in the right part of Fig. 8.1, even total serialization is required. This is achieved by defining an order, either based on agents or on signals triggering transition executions. Though the SDL standard defines three types of ASM agents, common implementations consider only *SdlAgents*[1] explicitly; *SdlAgentSets* and *Links* are either not realized as

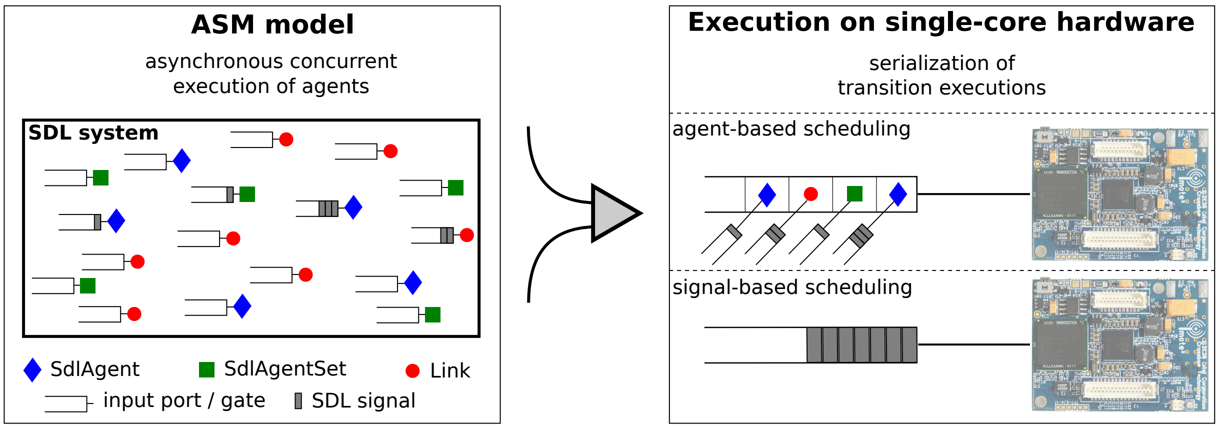


Figure 8.1: Execution according to formal semantics [Int12i] and in implementations.

[1]Often simplified referred to as (SDL) process. In this regard, see also Sect. 7.1.2.

particular agents or implicitly executed when SDL signals are sent. If not stated otherwise, we assume such implementations in the following and do not consider the execution of *Links* and *SdlAgentSets*.

The process of serializing transition executions depends on the SVM implementation and is usually based on a FCFS (First Come, First Served) order. To influence the transition execution order from within the design, there are only limited SDL constructs like priority inputs [Int12e] and signal priorities [Int12d], which affect only the order within *SdlAgents*. In particular, SDL does not provide any constructs to influence transition execution orders between different agents. SDL real-time tasks as presented below remove this disadvantage by introducing a notion of system task and a priority-based execution model in SDL. These extensions are incorporated into SDL's syntax and semantics and pave, together with SDL extensions for scheduling-aware system design, the way for the adequate and transparent execution of SDL systems with consideration of transition urgencies.

8.1.2 Time Progress in SDL

In the conceptual world of SDL, an undefined amount of time may pass during the execution of almost all constructs [Int12c]. It is even valid that the time to execute a construct varies in two runs. The definition of time progress furthermore allows that SDL actions do not require any amount of time, and there are, in particular, two situations, in which time progress is always fixed to zero [GP05]: Sending signals via channels with a **nodelay** attribute and starting execution of an enabled transition. Because of this vague notion of time, several more concrete and more realistic time models have been proposed in the literature (see also Sect. 11.2).

In SDL implementations on real hardware, however, progress of physical time is not controllable and always larger than zero.[2] There are, in general, three sources of delay, which cause a deferral of a transition execution (see Fig. 8.2): First, the consumption of the triggering signal is delayed due to other SDL signals in the same input port. We refer to this type of delay as *queueing delay*. It can be reduced in the SDL specification by using priority inputs, whose applicability is, however, very limited, because they depend on the state of the receiving *SdlAgent*

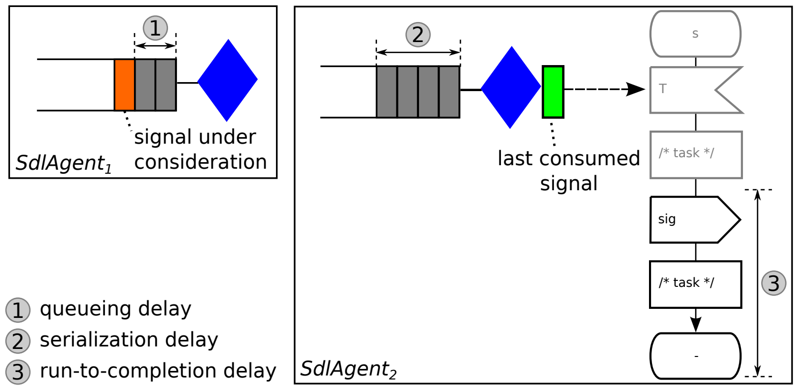


Figure 8.2: The three sources of transition execution delay in SDL.

[2]Yet time progress is possibly not always perceivable if granularity of the clock of the system is coarse, physical amount of time is always required to execute one to many processor instructions.

and the signal type. Thus, two instances of the same signal type cannot be treated differently. The second type of delay is called *serialization delay* and comprises the time to execute transitions in other *SdlAgents*. Because the SDL semantics assumes all *SdlAgents* to run concurrently, this delay cannot be reduced in standard SDL, but their reduction is one of the main objectives of this thesis. A further source of delay is referred to as *run-to-completion delay* and defines the time to finish the execution of a currently running transition. If this transition is in the same *SdlAgent*, there is no way to interrupt the transition execution, because SDL transitions are atomic. If, on the other hand, the transition is executed in a different *SdlAgent*, the execution could be interrupted in favor of the considered signal and continued at a later time. This would, however, require a scheduler with preemption, which is in general more complex and less efficient. SDL real-time tasks address this delay by incorporating task suspension in SDL.

8.1.3 Shortcomings of SDL by Means of a Concrete Example

By means of the SDL system in Fig. 8.3, this section discusses further design limitations of SDL regarding the realization of real-time systems. The system shows a protocol stack consisting of an application layer, a network layer, and a MAC layer, which is taken from the specification of

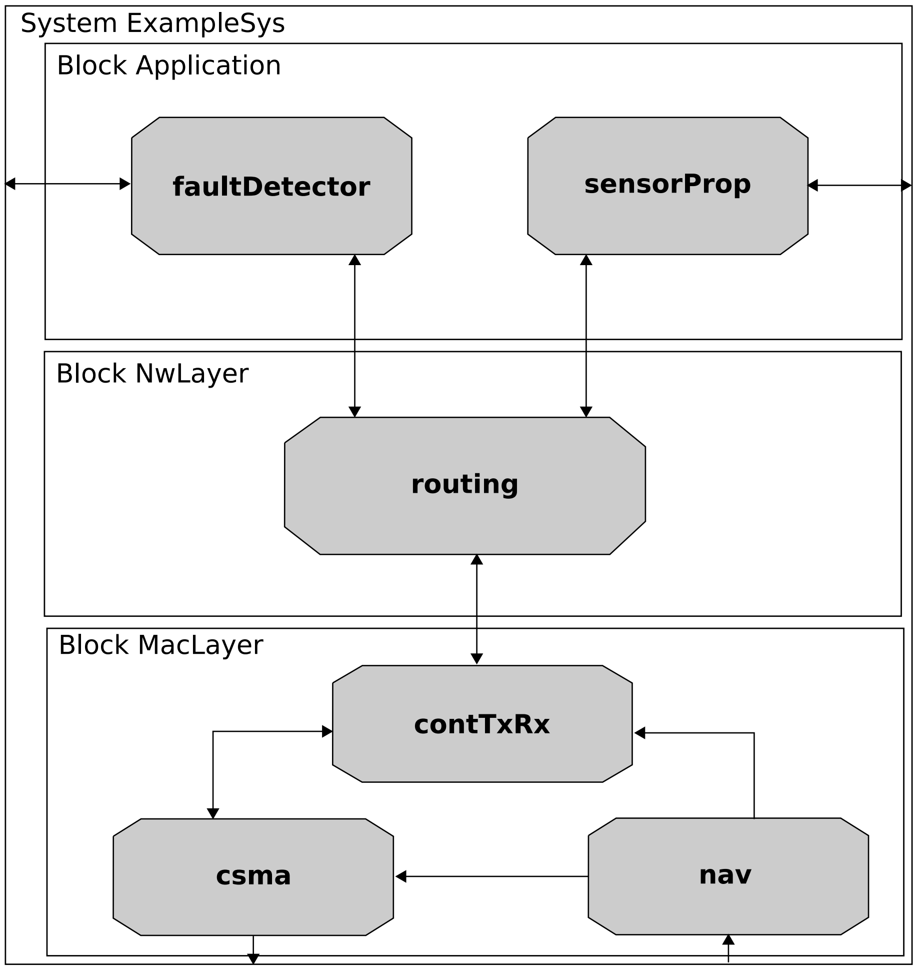


Figure 8.3: Example system illustrating shortcomings in SDL's expressiveness.

MacZ [BGK07, CBGK08]. The system comprises two tasks: First, sensor values have to be created periodically and sent to the network. Second, the system should inform about exceptional situations, which can occur at any time, very quickly. Since the information sent by the second task is much more important, it should take precedence over sensor value propagation.

When trying to realize both tasks and to privilege them adequately, the designer faces two problems. The first problem results from the fact that the dynamic execution structure is in general orthogonal to the static system structure, which is built with concepts like abstraction, modularization, and reusability. This means, in the concrete example, that both tasks are not restricted to a single SDL process but they are realized by numerous transitions spread over the entire system. Hence, it is not possible in SDL to privilege all transition executions that are executed as part of a particular system task. Though there are in fact some tool-specific extensions enabling the prioritization of all transitions of an SDL process or block (see also related work in Chapter 11), which could be used in the example to prioritize the origin process of the second task (`faultDetector`), there is no possibility to privilege signals and transitions that are executed in the context of the second task in a process-spanning way. The second problem are SDL transitions that are shared by several system tasks. For instance, in the example system, the same transitions in blocks `NwLayer` and `MacLayer` are used by both tasks. However, there are no available SDL constructs to give their execution precedence in the context of the second task only, since all available prioritization measures – like priority inputs or signal priorities – are linked to structural elements of the specification.

To sum up, language support is missing in SDL to enable a mapping of transition executions to system tasks, on whose behalf the transition is executed, and to assign adequate priorities to their execution. By introducing SDL real-time tasks, we address these shortcomings in Sect. 8.3.

8.2 Scheduling-aware System Specifications

Running SDL systems on real hardware requires a scheduler in the SVM implementation to serialize the concurrent ASM model of SDL. In general, all dynamic scheduling strategies are feasible to deal with the actor model of SDL. Static offline strategies, on the other hand, are inappropriate due to SDL's capabilities to dynamically create signals and *SdlAgents*.

To make scheduling more transparent and controllable, annotation-based SDL extensions have been introduced to choose and configure a desired scheduling strategy [CBG11].[3] In previous work, three scheduling strategies have been presented for these extensions: *Agent-based round robin (RR)*, *agent-based FCFS*, and *agent-based (SDL process) priority scheduling*. Agent-based RR corresponds most closely to the semantics of SDL, because it executes all agents including *SdlAgentSets* and *Links*. Due to the execution of idling agents, it is also the most inefficient strategy. Agent-based FCFS is more efficient and schedules only *SdlAgents* with enabled transitions. To privilege *SdlAgents*, process priority scheduling has been introduced, enabling the specification of fixed precedence ratings and the suspension of low priority agents.

However, as it turned out, process priority scheduling is in general not sufficient, because transitions of a task are usually not grouped into single *SdlAgents*. Instead, a system task is composed of transitions that are distributed over several SDL blocks and processes. Motivated

[3]These extensions have been published in the master's thesis [Chr10] and have been enhanced to support a new scheduling strategy for SDL real-time tasks.

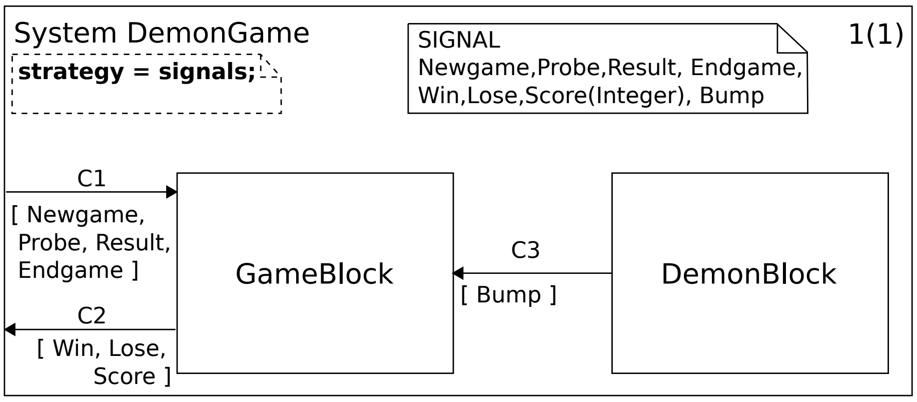


Figure 8.4: Selection of signal-based FCFS scheduling in the demon game example.

by this fact, SDL real-time tasks and a new scheduling strategy for their execution have been elaborated (see Sect. 8.3) and implemented (see Chapter 9). Similar to previous scheduling strategies, the new strategy can be selected by the designer in the system. To enable comparison evaluations with a state-of-the-practice scheduler, a signal-based FCFS strategy has been realized and integrated.

To illustrate the selection of a scheduling strategy within an SDL specification, Fig. 8.4 shows an extension of the demon game system, in which signal-based FCFS has been selected. The underlying formal foundations in terms of syntactical and semantical extensions can be found in Appendices B and C, whereas implementation-specific aspects are summarized in Chapter 9.

8.3 SDL Real-time Tasks

This section introduces SDL real-time tasks, their foundations from real-time systems, their formal definition, and their usage in SDL specifications. Formal extensions of the syntax and semantics of SDL can be found in Appendices B and C. Their implementation is surveyed in Chapter 9, their functional and quantitative evaluation in Chapter 10.

8.3.1 Foundations of Real-time Tasks

Real-time tasks (short: *tasks*) are a concept to structure and schedule the execution of real-time systems [Kop97]. A task is defined as the execution of a sequential program, thereby being a dynamic and nonrecurring unit, which is controlled by an OS or runtime environment. In [Kop97], Kopetz distinguishes between simple tasks (S-tasks) and complex tasks (C-tasks). An S-task is a task without synchronization points, i.e., without interaction with other tasks, whereas a C-task requires synchronization with other tasks. In common programming languages, this is achieved by blocking statements and semaphores. Though a task is unique and nonrecurring, other tasks may execute the same code. Thus, it is reasonable to distinguish between a task as *execution unit* and its implementation as *code unit*. For instance, a system reading temperature values every minute consists of one code unit with an implementation to access the temperature sensor and 60 execution units per hour running the code unit.

If several tasks are running simultaneously, their execution is ordered by their urgencies. For this, priorities are assigned to tasks, either statically or dynamically, e.g., based on deadlines. Depending on the cause of an execution, a task is either event- or time-triggered. Event-triggered tasks start execution after observing a significant state change, which is, for instance, an external event that is detected by a sensor, whereas time-triggered tasks start at determined points in time.

A real-time system always has a clearly defined objective. Thus, system tasks and subtasks as well as structure of hardware and software are known a priori [Sta92]. Different from desktop systems, amount and kind of system stimuli are known or can be estimated with an upper bound. A further property is that several tasks may run concurrently but not competitively, i.e., from a global perspective, all tasks work cooperatively to perform the system's overall job.

8.3.2 Formal Definition of Real-time Tasks in SDL

In this section, the concept of real-time tasks is adopted in SDL. Note that there is already a notion of task in SDL to specify statements in transition bodies. To avoid confusion, we refer to this type of task as *task statement* in the following.

A first step when transferring real-time tasks to SDL is the definition of code and execution units. As a starting point, a code unit can be defined as the specification of a single SDL transition. Accordingly, a simple execution unit in SDL is the execution of this transition by an SVM implementation. Hence, a *simple SDL real-time task* is formally defined as follows:

Definition 8.1. Let T be the set of all transition specifications (code units) of an SDL system. Then, a **simple SDL real-time task** τ is a tuple $(id, t_e, f_{transition}, f_{prio})$, where

- id is a globally unique task identifier,
- t_e is a transition execution (execution unit),
- $f_{transition} : \{t_e\} \rightarrow T$ is a function to map t_e to a code unit, and
- $f_{prio} : \{t_e\} \rightarrow \mathbb{N} \cup \{undef\}$ is a function defining the priority of the transition execution.

Note that t_e is a nonrecurring transition execution and dynamically created at runtime when the SDL real-time task is executed. Furthermore, an SDL real-time task is also unique and nonrecurring, since it comprises nonrecurring execution units, and there are, in particular, no two SDL real-time tasks with the same *id*.

The definition allows that several SDL real-time tasks execute the same transition (a property called *transition sharing*), since there is no restriction on $f_{transition}$. Though a simple SDL real-time task has only one transition execution, it may compete against transition executions of other SDL real-time tasks. To define an order between them on the basis of their urgencies, a priority is optionally assigned by f_{prio} such that lower values represent higher priorities. Accordingly, 0 is the highest priority. If no priority is defined explicitly, $undef$ is assigned and the transition is executed without preference.

In realistic systems, simple SDL real-time tasks are not sufficient. Instead, system functionalities comprise several transition executions, which may run in sequence or concurrently. Here, an example is the protocol stack in Fig. 8.3, where sent/received data traverses several SDL blocks and processes. To enable SDL real-time tasks for such systems, a more general and process-spanning definition is required:

Definition 8.2. Let T be the set of all transition specifications of an SDL system. Then, a **local SDL real-time task** τ is a tuple $(id, T_e, f_{transition}, f_{prio}, <_{eo})$, where

- id is a globally unique task identifier,
- T_e is a set of transition executions,
- $f_{transition} : T_e \rightarrow T$ is a function mapping each transition execution to an SDL transition,
- $f_{prio} : T_e \rightarrow \mathbb{N} \cup \{undef\}$ is a function assigning a priority to each transition execution, and
- $<_{eo} \subsetneq T_e \times T_e$ is an execution order on T_e with the following properties:
  - $<_{eo}$ is a strict partial order, i.e., $<_{eo}$ is irreflexive, transitive, and antisymmetric
  - $\exists t_e \in T_e . \forall t'_e \in T_e . (t'_e \neq t_e \Rightarrow t_e <_{eo} t'_e)$, i.e., there is a least element defining the starting point of the task.

Similar to simple SDL real-time tasks, transition executions of local tasks are nonrecurring events. Since $<_{eo}$ defines a causal order only, there is no restriction on time progress in between two transition executions of the same SDL real-time task. Consequently, a transition execution may also be delayed by using SDL timers or SDL real-time signals [KBCG11]. Though the order $<_{eo}$ demands from SDL real-time tasks to start with a single transition execution, it does not enforce a total order. Thus, concurrent transition executions, i.e., transition executions that are not ordered by $<_{eo}$, become possible. They can be executed in any order or simultaneously, if the hardware provides several processing units. A further implication of this definition is that the same transition can be executed multiple times in the same SDL real-time task (a property called *transition repetition*).

Since priorities are associated with transition executions and not with the SDL real-time task, different transition executions of the same task can run with different priorities. Accordingly, it is also possible to execute the same SDL transition multiple times with different priorities. In this case, priorities are evaluated both to privilege an SDL real-time task over another SDL real-time task and to order transition executions of the same task if they are not ordered by $<_{eo}$.

To apply SDL real-time tasks to networked systems, the definition is further extended:

Definition 8.3. Let N be a set of network nodes and T the set of all SDL transition specifications. Then, an **SDL real-time task** τ is a tuple $(id, T_e, f_{transition}, f_{prio}, f_{node}, <_{eo})$, where

- id is a globally unique task identifier,
- T_e is a set of transition executions,
- $f_{transition} : T_e \rightarrow T$ is a function assigning each transition execution to an SDL transition,
- $f_{prio} : T_e \rightarrow \mathbb{N} \cup \{undef\}$ is a function assigning a priority to each transition execution,
- $f_{node} : T_e \rightarrow N$ is a function allocating each transition execution to a network node, and
- $<_{eo} \subsetneq T_e \times T_e$ is an execution order on T_e with the following properties:
  - $<_{eo}$ is a strict partial order, i.e., $<_{eo}$ is irreflexive, transitive, and antisymmetric
  - $\exists t_e \in T_e . \forall t'_e \in T_e . (t'_e \neq t_e \Rightarrow t_e <_{eo} t'_e)$, i.e., there is a least element defining the starting point of the task.

If there are nodes in the network running different SDL systems, T contains the union of all SDL transition specifications. In consequence, there is a constraint on f_{node}, because a transition can only be executed by a node with a corresponding SDL transition specification. Based on the general definition of SDL real-time tasks, different types of tasks can be distinguished:

Definition 8.4. Let $\tau = (id, T_e, f_{transition}, f_{prio}, f_{node}, <_{eo})$ be an SDL real-time task. Then, τ is called

- **time-triggered**, if the first transition execution of τ (that is the smallest element w.r.t. $<_{eo}$) is either triggered by an SDL timer, SDL signal with activation delay, or real-time signal [KBCG11]. Otherwise, τ is called **event-triggered**.
- **serial**, if $<_{eo}$ defines a total order:

$$\forall t_1, t_2 \in T_e : (t_1 \neq t_2) \implies ((t_1 <_{eo} t_2) \vee (t_2 <_{eo} t_1)).$$

  Otherwise, τ is called **concurrent**.
- **non-terminating**, if T_e contains infinitely many transition executions.
- **distributed**, if the image of f_{node} contains at least two distinct elements:

$$\exists t_1, t_2 \in T_e : f_{node}(t_1) \neq f_{node}(t_2).$$

  Otherwise, τ is a **local** SDL real-time task (see above).
- **cyclic**, if the same transition is executed more than once on the same node:

$$\exists t_1, t_2 \in T_e : (t_1 \neq t_2) \wedge (f_{transition}(t_1) = f_{transition}(t_2)) \wedge (f_{node}(t_1) = f_{node}(t_2))$$

  In this case, neither $f_{transition}$ nor f_{node} are injective. Note that a non-terminating SDL real-time task is always cyclic, since the set of transitions and the set of nodes are finite.

Though a distributed SDL real-time task requires the execution on several nodes, the definition does not describe how nodes communicate. Instead, value passing and synchronization to enforce the execution order $<_{eo}$ is left to the environment and the SVM implementation of the system. They are also responsible for transferring task attributes like task ids between nodes. In addition, they have to take measures to guarantee uniqueness of task ids. This can, for instance, be achieved by subdividing task ids into a node id part and a locally unique part.

Comparing the above definition of real-time tasks with the original definition of Kopetz [Kop97], there is at first glance no *complex* real-time task in SDL, since SDL has no explicit constructs for blocking synchronization. But taking a closer look, there are also situations in SDL, in which the execution of a real-time task has to be delayed due to the occupancy of the executing agent, which can be seen as "shared resource". These situations are caused by the run-to-completion semantics of SDL transitions, because an agent must finish a running transition before another transition of the same agent can be executed. Thus, every SDL real-time task sharing SDL state machines with other tasks can be seen as *complex* within the meaning of Kopetz [Kop97]. Consequently, also transition executions of real-time tasks can suffer from some kind of synchronization delay. On implementation level, there is additionally synchronization required to access shared data, e.g., signal queues of an *SdlAgent*. Since this synchronization depends on the implementation, it is not inherent to real-time tasks.

Though SDL real-time tasks are nonrecurring execution units, they are usually fulfilling recurring system tasks. An example from protocol design are RTS/CTS handshakes, where a single application of a handshake can be described by one SDL real-time task. Since RTS/CTS handshakes are, however, performed before each communication, they are recurring. Accordingly, there are several SDL real-time tasks during the lifetime of the system that execute the same set of transitions. To enable the association of SDL real-time tasks with the objective of their execution, we introduce *task types* and a *type function*:

Definition 8.5. Let $\Phi = \{\tau_1, \tau_2, \ldots\}$ be a set of SDL real-time tasks and $\Gamma = \{\gamma_1, \gamma_2, \ldots \gamma_k\}$ the set of task types of a (distributed) system. Then, a function $f_{type} : \Phi \to \Gamma \cup \{undef\}$ is called **type function**, if it associates each SDL real-time task with the system task it fulfills.

Though task types are formally captured in Γ, a task type itself is more an intuitive than a formal property of a real-time task – e.g., "RTS/CTS handshake". Unlike the set of SDL real-time tasks Φ, the set of task types Γ is always finite.

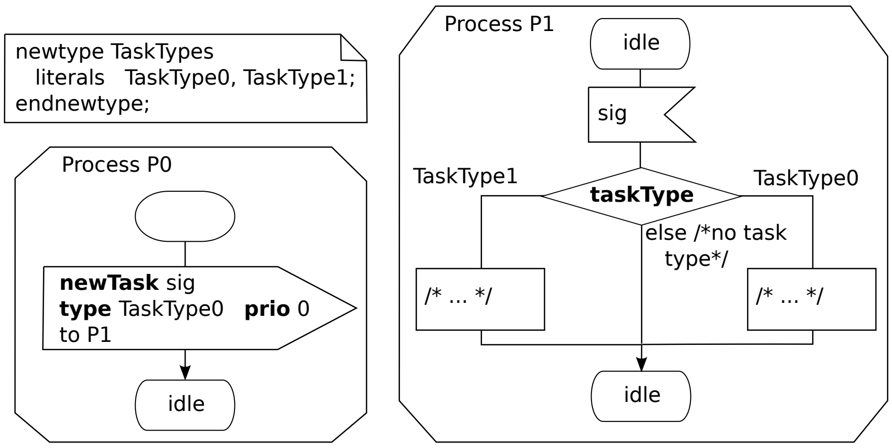
8.3.3 Incorporation of SDL Real-time Tasks into SDL

This subsection presents the incorporation of real-time tasks into SDL. It is shown how SDL real-time tasks are created, how transitions are executed in the context of a task, and how task attributes affect the transition selection order. The presentation focuses on an illustrative description, whereas concrete changes and extensions of SDL's formal syntax and semantics are mostly left to Appendices B and C.

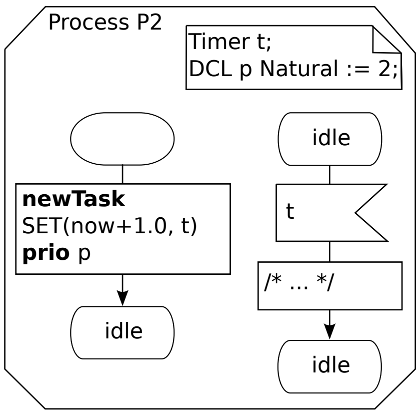
8.3.3.1 Creation and Control of SDL Real-time Tasks

To associate transition executions with real-time tasks, so-called *task signals* are introduced in SDL, which are SDL signals extended with task attributes consisting of task id, priority, and task type. For task id, a new SDL data type `Tid` is composed that is similar to the existing process identifier data type `Pid`. Task signal priorities are natural numbers as defined in Definitions 8.1, 8.2, and 8.3, and influence the transition execution order of *SdlAgents*. Task types, on the other hand, are scenario-dependent, thereby preventing the use of pre-defined data types. Instead, user-defined types are required, which can be created with literals. Besides task signals, plain SDL signals are still available to trigger transition executions outside the scope of an real-time task. They are treated with lower priority than transitions that are triggered by task signals. The declaration of plain SDL signals and task signals is identical, i.e., the same signal type definition can be used as basis for all signal instances.

By means of two examples, Fig. 8.5 shows the creation of two simple SDL real-time tasks by using new syntactical keywords called *task actions*. In both examples, task signals are created by the keyword `newTask`. In Fig. 8.5(a), the task signal is based on a regular SDL signal (event-triggered task), whereas in Fig. 8.5(b), it results from an SDL timer (time-triggered task). Though in both examples, an SDL real-time task is generated immediately, the actual start of the task is delayed until the first task signal is consumed. In Fig. 8.5(b), this is explicitly delayed by specifying `newTask` in a set statement for an SDL timer, which must first expire before a task signal can be consumed. Further alternatives to create time-triggered SDL real-time tasks are signal outputs with activation delay and real-time signals [KBCG11].



(a) Task signal based on regular SDL signal.



(b) Task signal based on SDL timer.

Figure 8.5: Simple real-time tasks in SDL.

When creating a new SDL task with keyword `newTask`, a unique task id of Type `Tid` is automatically generated by the SVM and associated with the task signal. Type of the task and priority of its first transition execution can be provided in the specification by using keywords `type` and `prio`, respectively. The value for `type` is a user-defined literal, whereas `prio` takes an expression evaluating to a natural number (e.g., a natural constant or variable). In the example of Fig. 8.5(a), type `TaskType0` and the largest possible priority 0 are associated with the task signal. In Fig. 8.5(b), only priority `p`, which evaluates to 2, is assigned as task attribute. Since there is no task type provided, the type of the timer-based task signal is set to the predefined value *undef*. If the task signal priority would be omitted (not shown in the examples), the task signal would obtain the lowest possible priority by default.

To access attributes of a consumed task signal in a transition, three nullary functions (called *task operators*) are introduced in SDL: `taskId`, `taskPrio`, and `taskType`. The application of `taskType` is shown in Fig. 8.5(a), where it is used for demultiplexing purpose in the transition consuming signal `sig` to distinguish follow-up actions.

To create non-simple SDL real-time tasks, the keyword `contTask` is introduced to continue a real-time task by forking further transition executions. Its application is illustrated in Fig. 8.6, which is an extension of Fig. 8.5(a). Besides the application of `contTask` in signal outputs,

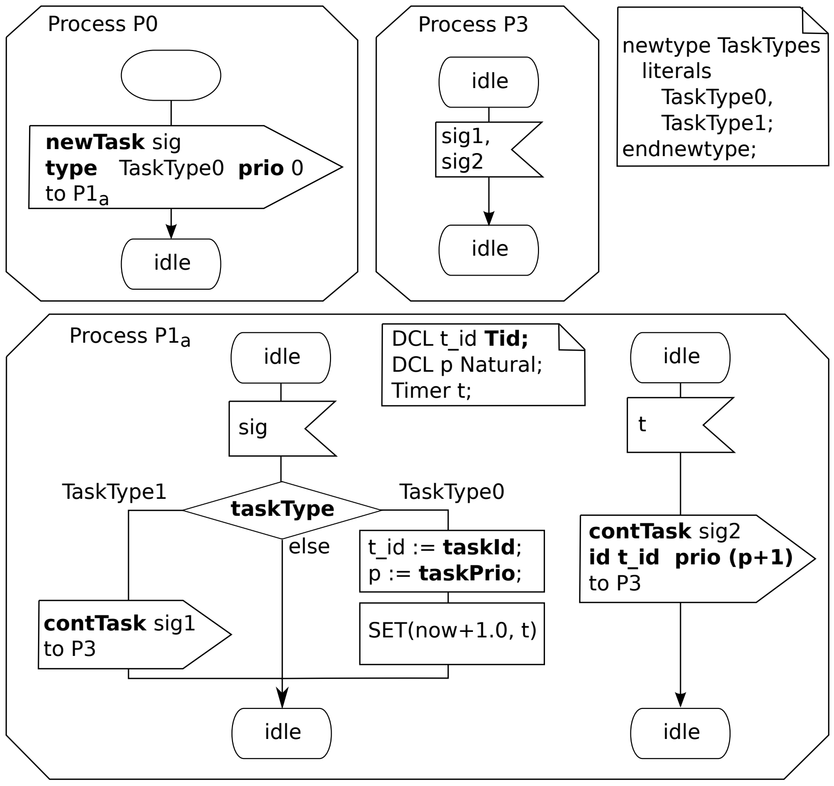


Figure 8.6: Example (continued) of a local SDL real-time task.

which is presented by two examples in process `P1`$_a$, the keyword is also allowed in timer set statements to create timer-based task signals of an existing real-time task. Keyword `contTask` optionally allows to assign a priority and a task id of a previous SDL real-time task to the newly created task signal. If `contTask` is used without any task attribute – like in the left branch of the transition consuming `sig` in process `P1`$_a$[4] –, the created task signal inherits the id and the priority of the consumed signal. In this case, the task continuation is called *implicit task forking*.

If a task id is specified explicitly, a previous real-time task can be continued (*explicit task forking*). This is shown in process `P1`$_a$ in the transition consuming timer signal `t`[5], where the task id of `sig`, which has previously been stored in variable `t_id`, is used. In addition, the priority that has been stored in variable `p` is also reused and increased by one. Thereby, the transition execution triggered by task signal `sig2` continues the real-time task with one priority level less. Different from task signal priorities, task types are a property of the SDL real-time task and not of the task signal, and can, therefore, neither be changed by implicit nor by explicit task forkings.

In the example, the right branch of process `P1`$_a$ is taken, since the consumed signal `sig` is of type `TaskType0`. Due to explicit task forking, the task is continued with signal `sig2` in the transition consuming timer signal `t`. Since there is no further task forking in process `P3`, the real-time task terminates after the consumption of `sig2`. Thus, the real-time task of the example consists of two transition executions, which are triggered by task signals `sig` and `sig2`, respectively.

[4]Note that this left branch is not taken in the example, since the type of the real-time task is `TaskType0`.

[5]Note that explicit task forkings are possible both in transitions executed in the context of an SDL real-time task and in ordinary transition executions.

8.3.3.2 Implications on Transition Execution Order

A transition consuming a task signal is executed in the context of the real-time task, whose attributes are carried by the task signal. In this regard, the priority of the transition execution is also derived from the task signal priority of the consumed signal, which implies that one transition can run with different priorities. In order to support this flexibility and to run transitions according to the task signal priority of consumed signals, the signal consumption order has to be changed. In particular, when selecting the next transition, the input port of an *SdlAgent* has now to be searched for the first task signal with highest priority, which is consumable in the agent's current state.

Figure 8.7 explains this step in more detail by means of an example process `P4`. We assume that the input port is already filled with six signals, which are ordered according to their availability time as defined in the SDL standard [Int12c] (illustrated by characters A to F). For signal `sig3` at position F, it is furthermore assumed that the availability time of the signal is larger than the current system time (now), i.e., the signal is not yet consumable. The signals' consumption order is illustrated by numbers 1 to 6 and is basically determined by the following rules:

1. Task signals take precedence over plain SDL signals.
2. Task signals are consumed according to their priority. In case of a tie, their availability time and insertion order is evaluated.
3. Plain SDL signals are consumed according to their availability time (as in SDL-2010).

Besides these basic rules, there are some exceptions w.r.t. SDL's save construct and implicit consumptions. In this regard, more details are given in the formal semantics in Appendix C.

By applying the extended consumption rules, signal `sig4` at position C, which has highest priority 1, is consumed first. Afterwards, `sig3` at position E, which is part of the same SDL real-time task, is consumed. Thereafter, `sig4` at position D with task signal priority 5 is chosen. It is followed by `sig3` at position B. Though this signal has no priority – i.e., the priority attribute was not specified in the signal output –, it is the only remaining consumable task signal in the input port. Afterwards, the only non-task signal `sig4` at position A is consumed. Finally, `sig3` is consumed as soon as the system time exceeds the availability time of the signal.

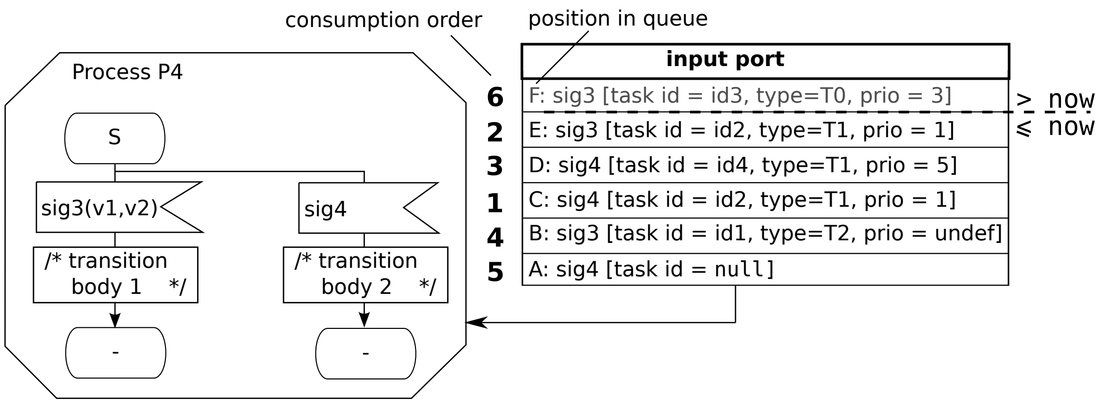

consumption order
position in queue
Process P4
S
sig3(v1,v2)
sig4
/* transition body 1 */
/* transition body 2 */
-
-
input port
6
2
3
1
4
5
F: sig3 [task id = id3, type=T0, prio = 3]
E: sig3 [task id = id2, type=T1, prio = 1]
D: sig4 [task id = id4, type=T1, prio = 5]
C: sig4 [task id = id2, type=T1, prio = 1]
B: sig3 [task id = id1, type=T2, prio = undef]
A: sig4 [task id = null]
> now
⩽ now

Figure 8.7: Changes on signal consumption order due to task signals.

8.3.3.3 SDL Real-time Tasks in Scheduling-aware System Specifications

Following the idea of scheduling-aware system specifications, we introduce a scheduling option for SDL real-time tasks, which can be selected in the head symbol of the system specification. The new scheduling strategy is called *task scheduling*. Its selection is shown in Fig. 8.8 for the demon game example. By selecting task scheduling in the head symbol of a system with task signals, task attributes are interpreted and transition execution orders are derived from the extended signal consumption rules as illustrated above. If, on the other hand, a different scheduling strategy is selected in a system with task signals, task attributes are ignored and task signals are treated as regular (non-task) SDL signals. By making task scheduling compatible with scheduling-aware system specifications, comparative evaluations with other state-of-the-practice scheduling strategies become possible.

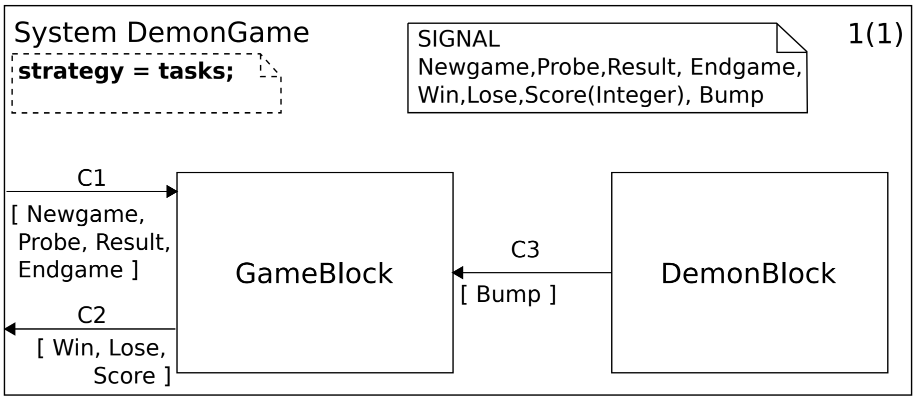


Figure 8.8: Selection of task scheduling to support real-time task-aware transition executions.

In the first instance, the selection of a particular scheduling strategy affects only the transition execution order within *SdlAgents*. The impact of the strategy on the system-wide execution order is up to the implementation and the limitations of the hardware. More details on this topic and a description of all supported scheduling strategies can be found in Chapter 9.

8.3.3.4 Suspension of Real-time Tasks

In [Chr10], process priority scheduling has been introduced with the possibility to temporarily suspend *SdlAgents*. This approach has been adopted for SDL real-time tasks to temporarily prevent the execution of transitions of particular SDL tasks. Thereby, run-to-completion delay (see Sect. 8.1.2), which is in particular a problem in presence of complex and long-running transitions, can be reduced.

The incorporation of task suspension and resumption is illustrated in Fig 8.9, where process `P5` creates a non-terminating real-time task in its start transition with SDL timer `t`, which is after each expiration again scheduled with identical interval. In the transitions consuming the signals of this task, the unary function `suspendTaskType` (`resumeTaskType`) is invoked to suspend (resume) all tasks of type `BgTaskType`, which are assumed to be background tasks with low priority. Thus, application knowledge is exploited to reduce system load in advance of time-critical intervals and transition scheduling can be influenced in a scenario-specific way.

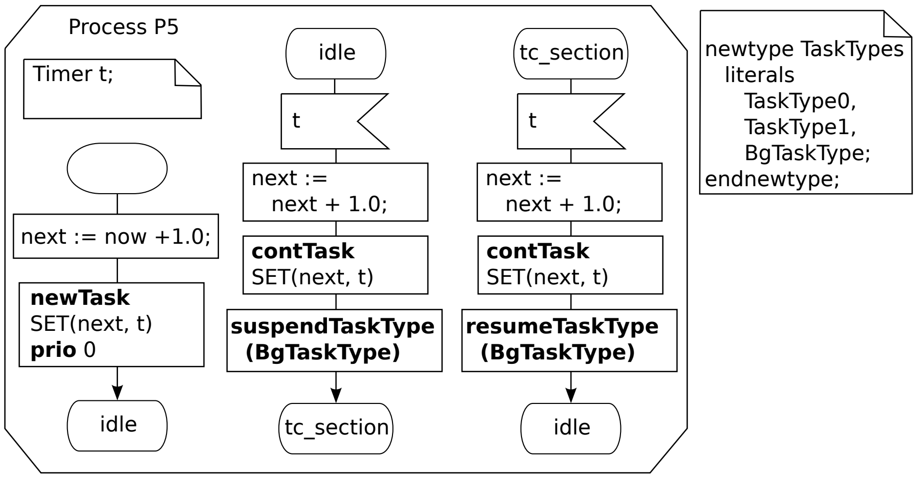


Figure 8.9: Suspension and resumption of transition executions of particular task types.

Besides keyword `suspendTaskType`, the language has been extended with two further keywords `suspendTaskId` and `suspendTaskPrio` to enable flexible task suspension. With `suspendTaskId`, the execution of one specific real-time task, whose id must be obtained by the `taskId` operator, is paused. Its corresponding counterpart is `resumeTaskId`. The function `suspendTaskPrio` takes one parameter declaring a priority threshold, so that all task signals with a priority value equal to or higher than this parameter are prevented from triggering transition executions. The complement of this function is `resumeTaskPrio`, which takes no parameter and unlocks all task signals that are not suspended by their id or type.

8.3.3.5 Semantical Incorporation of SDL Real-time Tasks

While the full formal incorporation of SDL real-time tasks into SDL's dynamic semantics is presented in Appendix C, this subsection provides a brief overview of necessary changes and extensions. The starting point of the incorporation is Annex F3 of SDL-2000 [Int00], since SDL's formal ASM semantics have not yet been updated in the current SDL-2010 standard [Int12c].

Table 8.1 classifies the necessary changes into four categories and presents the corresponding scope of changes. *Scheduling awareness* summarizes all extensions that are required to enable the selection of task scheduling in the head symbol of the SDL system. Since selecting a scheduling strategy has its major impact on the SDL implementation and not on the formal semantics, semantical extensions of this category are actually minor and consist of seven Lines of ASM

	ASM domains		**ASM functions**		**ASM macros**		**affected**
	new	changed	new	changed	new	changed	**LOCs**
scheduling awareness	1	0	5	0	0	1	7
task attributes	7	3	8	0	1	5	50
suspension/resumption	2	1	3	0	4	1	30
transition selection	0	1	4	0	4	2	138

Table 8.1: Scope of semantical extensions.

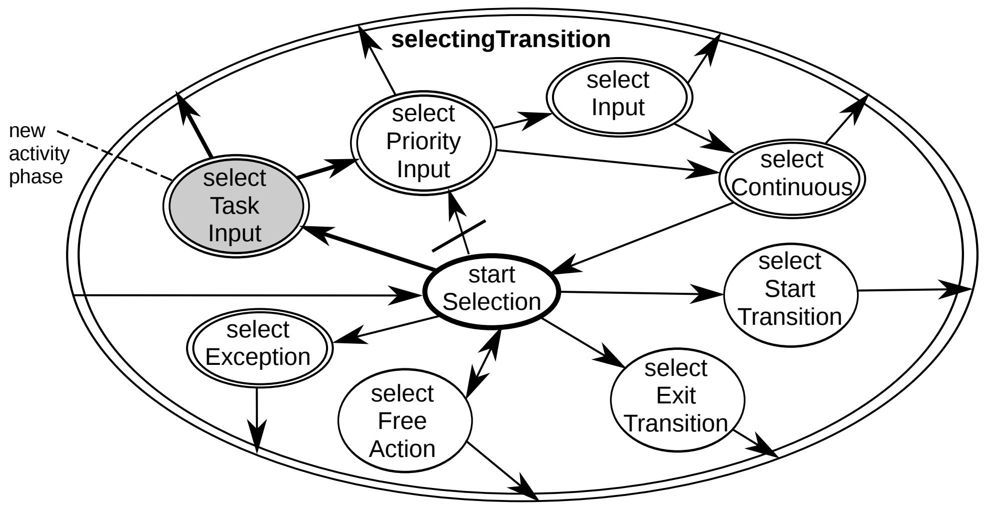


Figure 8.10: New task signal-specific activity phase as subphase of an *SdlAgent*'s transition selection phase (phases with double border are refined in separate diagrams) [Int00].

Code (LOC) only. To formally define *task attributes* in ASM – including their creation with task actions, their association with task signals, and access to them via task operators –, more extensions were required and several new ASM domains, functions, and macros have been added or changed. The same holds for task *suspension and resumption*.

Due to numerous features regarding the selection of transitions – including enabling conditions, saving of signals, and inheritance of transition specifications –, most changes and extensions of SDL affect the category *transition selection*. In this regard, a total of 138 LOCs were added to realize the extended consumption order of task signals as illustrated in Sect. 8.3.3.2. These LOCs mainly implement a new task signal-specific activity phase, which is called `selectTaskInput` and part of the transition selection phase of *SdlAgents* (see Fig. 8.10). Since the new phase precedes all existing phases and, in particular, the selection of priority inputs, it is guaranteed that (consumable) task signals take precedence over any non-task signal. Subphases of the `selectTaskInput` phase and their realization in ASM can be found in Appendix C.

8.3.4 Integration of SDL Real-time Tasks into the SDL Environment

In the first instance, SDL real-time tasks require extensions to SDL signals and changes of the transition selection order. Thus, they are independent of a concrete hardware platform and the realization of the SDL environment. However, to support distributed SDL real-time tasks, extensions of the SDL environment become necessary.

When a task signal, which is created in the SDL system and sent to the system's environment in order to communicate with another node, is received by the SDL environment, task attributes of the signal have to be appended as payload to the outgoing physical signal. Thus, the concrete realization depends on the desired communication technology, which must meet prerequisites to enable the transfer of task attributes as additional payload. On the receiving side, the SDL environment has to gather these task attributes from the received physical signals in order to generate task signals accordingly before sending them to the SDL system.

8.3.5 SDL Real-time Tasks in SDL-MDD

A big advantage of SDL is the availability of holistic model-driven development processes. This advantage is still given with the incorporation of SDL real-time tasks, which can also be applied in compliance with reusability concepts like SDL patterns and micro protocols [Got07]. Though real-time tasks are primarily motivated by the request for more predictable implementations, they improve also the expressiveness of the design. In particular, priorities of task signals are not only instructions for implementations, but introduce general prioritizing extensions of the transition selection, thereby affecting the behavior of SDL systems in analyses, simulations, and implementations. Consequently, it is reasonable to introduce SDL real-time tasks in the PIM of SDL-MDD (see Sect. 7.2).

Compared to prioritization measures of state-of-the-practice approaches, which introduce SDL process, block, or signal priorities in a separate implementation phase, SDL real-time tasks benefit from a priority model at design level that is maintained in implementations. Thereby, assigning priorities in a separate implementation phase becomes obsolete. Additionally, the priority model of SDL real-time tasks enables a more flexible priority assignment that is independent of the system's block and process structure.

To improve the application of SDL real-time tasks in the context of SDL-MDD, existing design guidelines, which have been proposed in conjunction with SDL-MDD [BCG09], can be adopted. In particular, the guideline to split long-running transitions into shorter ones is also beneficial for SDL real-time tasks to reduce run-to-completion delays.

8.3.6 Example: RTS/CTS Handshakes as Distributed SDL Real-time Tasks

In the following, a realistic but simplified example of a distributed SDL real-time task is given. The example realizes an RTS/CTS handshake and utilizes the MAC layer of Fig. 8.3, which is refined in Fig. 8.11 with all relevant transitions.

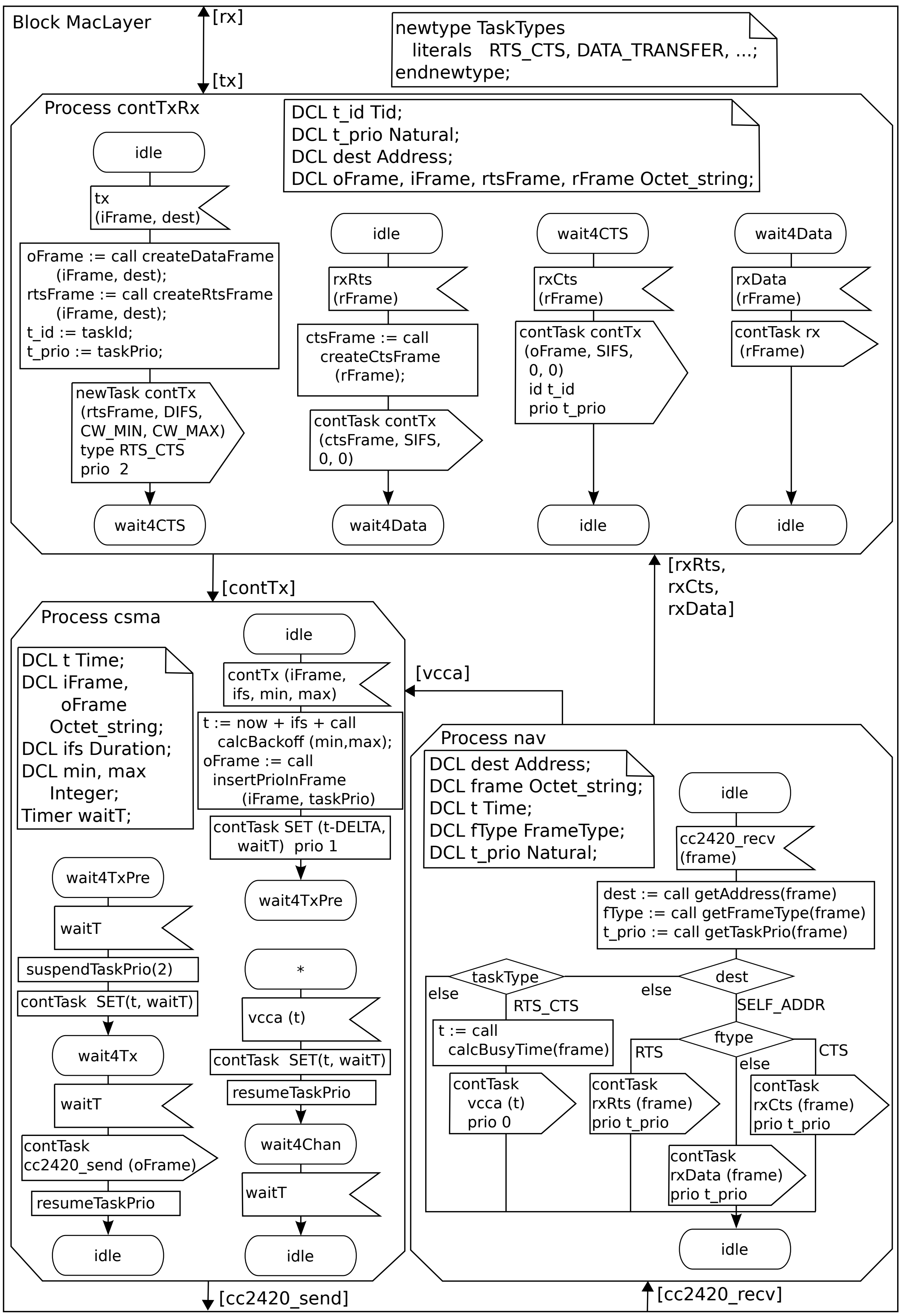
The example assumes a single-hop network of three nodes (N_0, N_1, and N_2), which all run the same SDL system, and that N_0 is going to send data to N_1. When at N_0, the transmission request arrives in process `contTxRx` by signal `tx`, a new SDL real-time task of type `RTS_CTS` is created, where the first signal of this task (`contTx`) is assigned with priority 2. In the same transition, attributes of the previous real-time task, which are included in signal `tx`, are furthermore stored in the state of the *SdlAgent*.

In the transition consuming `contTx` in process `csma`, the priority of the task signal is stored in the content of the data frame, i.e., it is transferred as additional payload. Afterwards, the running real-time task is continued with priority 1 after waiting a random backoff delay. Shortly before the RTS frame is going to be sent, `suspendTaskPrio` is called to suspend real-time tasks with priority values of ≥ 2. Thereby, run-to-completion delay is avoided when the RTS frame is sent to the environment by signal `cc2420_send` shortly afterwards. By calling `resumeTaskPrio`, suspension is again canceled. When the RTS frame arrives at the environment (not shown in the figure), the environment appends task attributes of the running task to the payload of the frame before it leaves the node via the CC 2420 transceiver. Note that at this moment, two priorities are included in the payload of the frame: First, the priority that is explicitly inserted in process `csma` (value 2 in this case) and describes the priority of the frame content. And furthermore, the task signal priority that is implicitly inserted by the environment (value 1), which describes the priority of the transmission itself.

When receiving the RTS frame at nodes N_1 and N_2, the SDL environments of both nodes read the task attributes from the payload of the received frames and accordingly create task signals of type `cc2420_recv`, which are afterwards sent into the SDL system. In process `nav`, this task signal is consumed and the priority of the frame content is derived from the remaining payload of the frame and used to continue the RTS/CTS handshake if the frame is destined for this node. This is the case for node N_1, where task signal `rxRts` is created and sent to process `contTxRx`, thereby causing the response with a CTS frame. On N_2, `vcca` is sent to process `csma` to announce the network allocation vector for virtual carrier sensing. Since this information is very time-critical, `vcca` is sent with maximal priority 0.

The transmission of the CTS frame by node N_1 is similar to the RTS frame but without random backoff. By applying task suspension, non-desired delays are kept minimal, too. After finally receiving the CTS frame in process `contTxRx` at node N_0, the RTS/CTS real-time task terminates and the data transfer task is resumed by consulting the stored task attributes.

The formal definition of the RTS/CTS real-time task is given in Fig. 8.12. Besides explicit transitions, the formal definition of the transition set T includes further transitions like implicit consumption and transitions of the environment. When sending the RTS frame via the environment, the task execution tree, which visualizes the transition execution order of the real-time task, forks into two branches; one branch for transition executions on N_1 and N_2, respectively. Though the real-time task and its transition executions are nonrecurring, other real-time tasks of RTS/CTS handshakes may look similar.



Block MacLayer
[rx]
[tx]
newtype TaskTypes literals RTS_CTS, DATA_TRANSFER, ...; endnewtype;
Process contTxRx
DCL t_id Tid;
DCL t_prio Natural;
DCL dest Address;
DCL oFrame, iFrame, rtsFrame, rFrame Octet_string;
idle
tx (iFrame, dest)
oFrame := call createDataFrame (iFrame, dest);
rtsFrame := call createRtsFrame (iFrame, dest);
t_id := taskId;
t_prio := taskPrio;
newTask contTx (rtsFrame, DIFS, CW_MIN, CW_MAX) type RTS_CTS prio 2
wait4CTS
idle
rxRts (rFrame)
ctsFrame := call createCtsFrame (rFrame);
contTask contTx (ctsFrame, SIFS, 0, 0)
wait4Data
wait4CTS
rxCts (rFrame)
contTask contTx (oFrame, SIFS, 0, 0) id t_id prio t_prio
idle
wait4Data
rxData (rFrame)
contTask rx (rFrame)
idle
[contTx]
[rxRts, rxCts, rxData]
Process csma
DCL t Time;
DCL iFrame, oFrame Octet_string;
DCL ifs Duration;
DCL min, max Integer;
Timer waitT;
idle
contTx (iFrame, ifs, min, max)
t := now + ifs + call calcBackoff (min,max);
oFrame := call insertPrioInFrame (iFrame, taskPrio)
contTask SET (t-DELTA, waitT) prio 1
wait4TxPre
wait4TxPre
waitT
suspendTaskPrio(2)
contTask SET(t, waitT)
wait4Tx
waitT
contTask cc2420_send (oFrame)
resumeTaskPrio
idle
*
vcca (t)
contTask SET(t, waitT)
resumeTaskPrio
wait4Chan
waitT
idle
[vcca]
Process nav
DCL dest Address;
DCL frame Octet_string;
DCL t Time;
DCL fType FrameType;
DCL t_prio Natural;
idle
cc2420_recv (frame)
dest := call getAddress(frame)
fType := call getFrameType(frame)
t_prio := call getTaskPrio(frame)
taskType
else
RTS_CTS
dest
else
SELF_ADDR
t := call calcBusyTime(frame)
ftype
RTS
CTS
else
contTask vcca (t) prio 0
contTask rxRts (frame) prio t_prio
contTask rxCts (frame) prio t_prio
contTask rxData (frame) prio t_prio
idle
[cc2420_send]
[cc2420_recv]

Figure 8.11: Transitions of block `MacLayer` to illustrate an RTS/CTS handshake as example for distributed SDL real-time tasks.

Network nodes and transition specifications:

$$
\begin{array}{rcl}
N = \{ & N_0, N_1, N_2 & \} \\
T = \{ & contTxRx.idle(tx), & \\
& contTxRx.idle(rxRts), & \\
& contTxRx.wait4CTS(rxCts), & \\
& contTxRx.wait4Data(rxData), & \\
& csma.idle(contTx), & \\
& csma.wait4TxPre(waitT), & \\
& csma.wait4Tx(waitT), & \\
& csma.idle(vcca), & \\
& csma.wait4TxPre(vcca), & \\
& csma.wait4Tx(vcca), & \\
& csma.wait4Chan(vcca), & \\
& csma.wait4Chan(waitT), & \\
& nav.idle(cc2420\_recv), & \\
& env.idle(cc2420\_send), & \\
& env.idle(frame), & \\
& implicit\ consumption & \}
\end{array}
$$

Transitions in T are given in the following format:

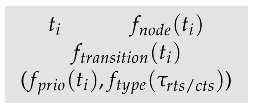
PROCESS_NAME.PROCESS_STATE(TRIGGER)

A real-time task of a successful RTS/CTS handshake looks like follows:

$$\tau_{rts/cts} = (\tau_{id}, T_e, f_{transition}, f_{prio}, f_{node}, <_{eo}),$$

where τ_{id} is a unique id and $T_e = \{t_0, t_1, \ldots, t_{17}\}$ a set of 18 transition executions.

$<_{eo}$ is given by the edges of the transition execution tree on the right-hand side. $f_{transition}, f_{prio}, and f_{node}$ are given in the vertices of the execution tree, which are built as follows.



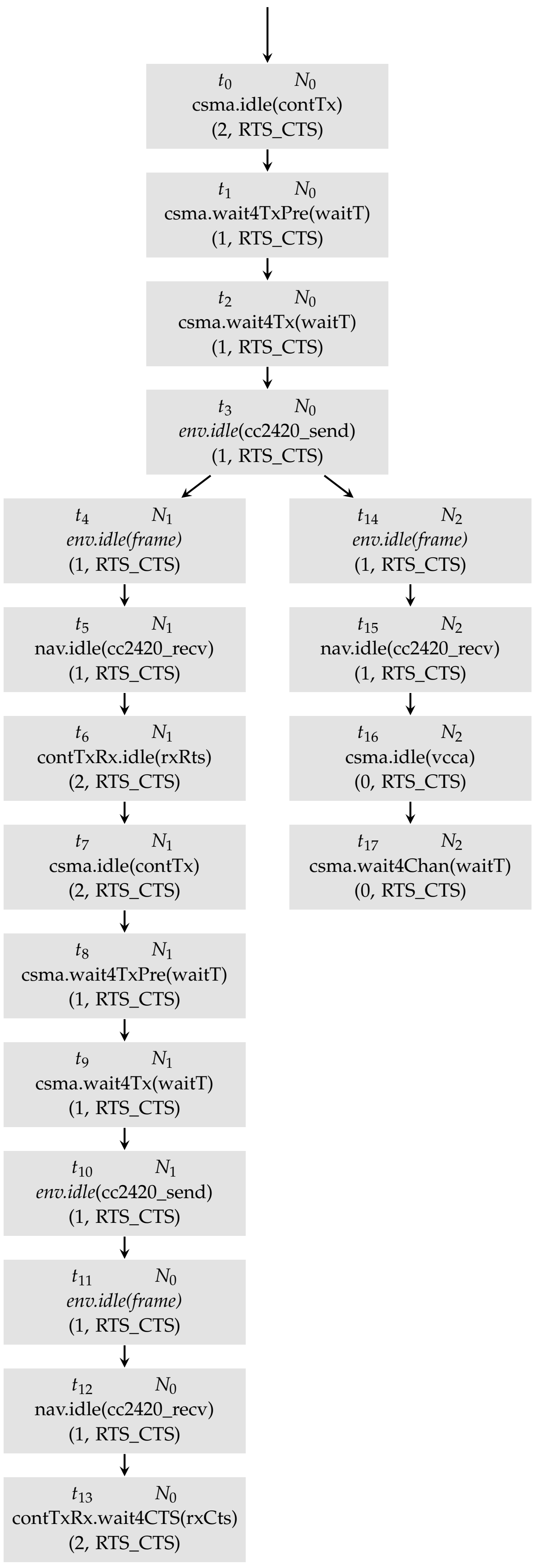

t_0 N_0 csma.idle(contTx) (2, RTS_CTS)
t_1 N_0 csma.wait4TxPre(waitT) (1, RTS_CTS)
t_2 N_0 csma.wait4Tx(waitT) (1, RTS_CTS)
t_3 N_0 *env.idle*(cc2420_send) (1, RTS_CTS)
t_4 N_1 *env.idle(frame)* (1, RTS_CTS)
t_5 N_1 nav.idle(cc2420_recv) (1, RTS_CTS)
t_6 N_1 contTxRx.idle(rxRts) (2, RTS_CTS)
t_7 N_1 csma.idle(contTx) (2, RTS_CTS)
t_8 N_1 csma.wait4TxPre(waitT) (1, RTS_CTS)
t_9 N_1 csma.wait4Tx(waitT) (1, RTS_CTS)
t_{10} N_1 *env.idle*(cc2420_send) (1, RTS_CTS)
t_{11} N_0 *env.idle(frame)* (1, RTS_CTS)
t_{12} N_0 nav.idle(cc2420_recv) (1, RTS_CTS)
t_{13} N_0 contTxRx.wait4CTS(rxCts) (2, RTS_CTS)
t_{14} N_2 *env.idle(frame)* (1, RTS_CTS)
t_{15} N_2 nav.idle(cc2420_recv) (1, RTS_CTS)
t_{16} N_2 csma.idle(vcca) (0, RTS_CTS)
t_{17} N_2 csma.wait4Chan(waitT) (0, RTS_CTS)

Figure 8.12: Definition of a concrete real-time task of a successful RTS/CTS handshake.

8.4 Discussion

SDL real-time tasks address a shortcoming of SDL and all previous prioritization measures, which has limited the applicability of SDL in real-time systems: The lack of prioritization schemes that are orthogonal to the static system structure. Since priorities are assigned to transition executions dynamically, SDL real-time tasks do not suffer from a static mapping of transition specifications to priorities. Prioritization with SDL real-time tasks is very flexible, because the assignment of different priorities to subtasks or to single transition executions of the same task becomes possible. In addition, time-dependent privileges are supported by calculating task signal priorities as a function of the task's urgency. While this is in general less powerful than dynamic scheduling strategies like Earliest Deadline First (EDF), it is sufficient for many considered applications and efficient to implement.

Though the integration of SDL real-time tasks into SDL's semantics, which is presented in this chapter, affects the order of transition executions within single *SdlAgents* only, the presented extensions are also well-suited to direct the agent-spanning serialization of transition executions in implementations. This fact is contemplated in Chapter 9 in detail. By introducing distributed tasks, SDL real-time tasks address another drawback of SDL: The node-spanning identification of system tasks. This is achieved with SDL real-time tasks by communicating task attributes consisting of task identifiers, task types, and task signal priorities.

To highlight differences between SDL real-time tasks and existing prioritization measures of SDL and prior extensions, Table 8.2 presents a comparative survey w.r.t. the three types of delays (see Sect. 8.1.2). Though serialization delay is usually considered in implementations and not on design level, since all agents run concurrently in the conceptual execution model of SDL, it is included in the table for the sake of completeness.

The least powerful language constructs regarding prioritization are SDL signal priorities, which are introduced in SDL-2010 [Int12c]. Because they are only considered if two signals have the same availability time, they are inappropriate to privilege transition executions in implementations, where usually only SDL timers but no regular signals can have identical availability times.

Priority inputs are also included in the SDL standard and have been improved in SDL-2010 [Int12e] by introducing multiple priority levels. Besides the drawback that they are statically

	queueing delay	**serialization delay**	**run-to-completion delay**
SDL signal priorities	X	X	X
priority inputs	✓	X	X
process priorities w/o suspension	X	✓	X
process priorities with suspension	X	✓	✓
SDL real-time tasks w/o suspension	✓	✓	X
SDL real-time tasks with suspension	✓	✓	✓

Table 8.2: Comparison of prioritization measures. (X – delay not addressed, ✓ – reduction of delay possible)

assigned to a structural element, i.e., to a transition in one particular state of an SDL process, their major limitation is their missing impact on serialization and run-to-completion delay.

Process priorities are actually not part of SDL but supported by many tools. In our previous work [CBG11], we also proposed the annotation-based incorporation of process priorities into SDL. As it turned out, their influence on transition execution orders is, however, not sufficient, since they cannot privilege transition executions within an *SdlAgent*. Thereby, they cannot reduce queueing delays. As a further drawback, process priorities are assigned to structural elements (SDL blocks or processes) only. Thus, they reach their limits if transitions within the same SDL process are shared by system tasks with different priorities. In conjunction with process priorities, our previous work [CBG11] introduces process suspension in SDL, which addresses run-to-completion delay for the first time.

By assigning priorities dynamically, SDL real-time tasks outperform existing approaches. In particular, they can deal with transitions that are executed in different system tasks (transition sharing) and allow the re-calculation of priorities depending on the current situation. Since task signal priorities affect the signal consumption order within agents and can additionally be consulted for global serialization, they reduce queueing as well as serialization delay of high priority tasks. By adopting suspension in SDL real-time tasks, run-to-completion delay is addressed as well. Compared with process suspension, suspension with SDL real-time tasks is significantly more flexible since suspension is configurable and based on tasks and not on structural elements of the specification.

9. Chapter

Model-driven Implementation of SDL Real-time Tasks

Implementing SDL is a perseverative topic in academic and industrial projects. Since the concurrent execution model of SDL cannot be realized one-to-one on real hardware, several implementation alternatives have been proposed. Most of them target at the improvement of the efficiency of the system. Prioritization, which is often introduced by the assignment of priorities to SDL blocks or processes, is usually addressed in an additional implementation phase and with support of an OS, thereby requiring an additional development step and deployment diagrams with implementation-specific information. SDL real-time tasks provide a novel approach to guide the implementation and to privilege time-critical transitions at runtime. This approach is more flexible than existing approaches, since prioritization is not based on static elements of the specification. It is furthermore compatible with the model-driven development process SDL-MDD (see Sect. 7.2) and comes without implementation phase, since all relevant information is included in the SDL specification. Because SDL real-time tasks do not depend on a specific hardware or software platform, most parts of their implementation are platform-independent. However, to support distributed SDL real-time tasks, supplemental platform-specific extensions are necessary in the implementation of the SDL environment.

The implementation of SDL real-time tasks is based on a tool chain, which was the result of previous academic projects [Fli09, FGW06, FGJ$^+$05]. This tool chain is compliant with SDL-MDD and consists of the code generator ConTraST, the SVM implementation SdlRE, and the environment implementation SEnF. Besides these tools, which are available with code sources, further proprietary software [IBMar] was used to create SDL specifications. Though the tools allowed extensive modifications, some limitations like missing SDL-2010 support had to be considered. While these features are, however, not essential for SDL real-time tasks, their absence prevented quantitative comparisons between SDL real-time tasks and novel prioritization measures of SDL-2010 like signal priorities or priority inputs with multiple levels.

This chapter is structured as follows: In Sect. 9.1, challenges of implementing SDL are discussed. Thereafter, Sect. 9.2 provides an overview of the used tool chain and realization approach. Sections 9.3, 9.4, and 9.5 summarize changes and extensions of ConTraST, SdlRE, and SEnF, respectively. Afterwards, Sect. 9.6 presents a simulator framework called FERAL, which has been extended to run SDL models and performance evaluations of SDL real-time tasks. Finally, Sect. 9.7 discusses results of this chapter.

The contents of this chapter have been published in [5], [13], [17], [19], [21], and [24].

9.1 Motivation

Besides achieving high-efficient implementations, the major challenge of implementing SDL is the adequate serialization of SDL's concurrent execution model. Since real-world hardware does not offer the same degree of concurrency as SDL systems, the way of serializing transition executions affects the behavior of the system and may lead to undesired delays or unconsidered behavior. Since the standard does not prescribe a default or straightforward approach to order concurrent transition executions, execution orders in implementations are very tool-dependent. To avoid undesired serialization orders – especially in situations with high system load –, some tools provide additional measures to guide the serialization process, e.g., by an additional implementation phase or by annotations in the specification.

To illustrate the impact of serialization on single-core hardware, Fig. 9.1 presents two SDL processes, which run as two concurrent (but not parallel) *SdlAgents* in an implementation. In each process, a timer is set with identical expiration time. Depending on which transition is executed first, either `sig1` or `sig2` is created. Since the created signal is consumed as priority input and cancels the SDL timer `t`, the respective alternative signal is not generated. If no further measures like process priorities are used outside the scope of the SDL specification, the resulting behavior is nontransparent for the designer of the system, nondeterministic, and may, e.g., depend on the names of processes or – even worse – on their positions in the specification.

In real-time systems, the way of serializing concurrent *SdlAgents* is not only a problem w.r.t. transparency but also a problem from a runtime perspective, since delays due to undesired execution orders may not be acceptable. To reduce delays and to obtain a more predictable and transparent behavior without manual implementation phase, priority information of SDL real-time tasks can be used to control the execution and to determine a desired execution order. The required extensions to our SDL tool chain are summarized in the following.

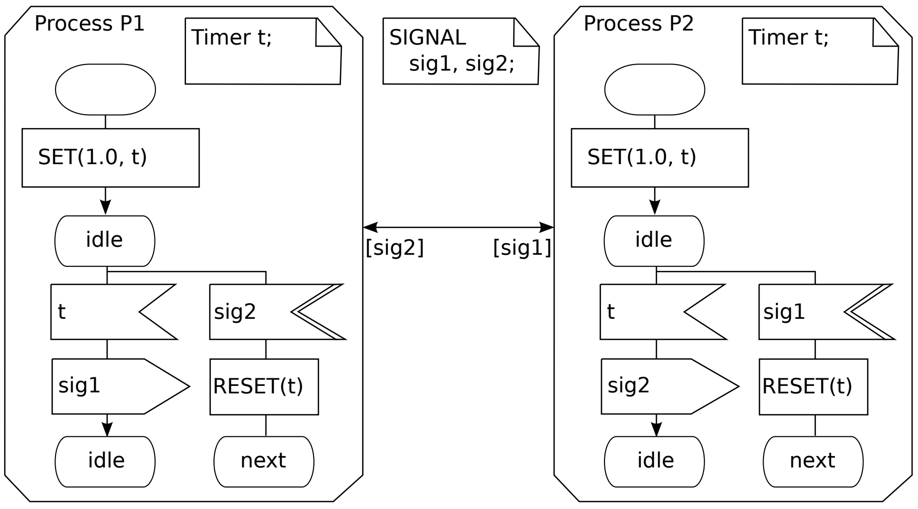


Figure 9.1: Example of nondeterministic transition serialization: Depending on the order, in which the timers are fired in an implementation, different transitions are executed.

9.2 Real-time Tasks and Model-driven Implementations – Outline

In Sect. 7.2, an SDL tool chain is presented in conjunction with SDL-MDD [Got07], consisting of the code generator *ConTraST* [FGW06, Fli09], the SVM implementation *SdlRE* [FGW06, Fli09], and the SDL environment implementation *SEnF* [FGJ$^+$05]. All of these tools have been extended to obtain a prototypical realization of SDL real-time tasks, which is outlined in the following sections. The tool chain supports the following platforms: PCs with Linux, the Imote 2 sensor platform [MEMara], and the network simulator PartsSim [BGK08]. Since the main target of SDL real-time tasks are embedded systems, the focus of the implementation was on Imote 2 sensor nodes. However, most applied extensions are platform-independent and, hence, also available on other platforms. To support simulative evaluations of SDL systems and SDL real-time tasks, the tool chain has been extended in the course of this thesis to support a further simulator called FERAL [BCG$^+$13, BCG$^+$14] (see Sect. 9.6). With FERAL, SDL systems can be simulated either as library, thereby ignoring all runtime delays of the SDL system, or deployed on Imote 2 nodes to conduct Hardware-in-the-Loop simulations. In the second case, delays to execute the SDL system and its runtime environment are taken into account and performance evaluations become possible, which are crucial for the assessment of SDL real-time tasks.

To create SDL specifications, the graphical SDL editor, the SDL-GR to CIF level 0 (SDL-PR) compiler, and the SDL syntax and semantics analyzer of IBM's Rational SDL Suite [IBMar] were used. However, changes and extensions of this tool are very limited, since it is proprietary and closed source. This particularly affects extensions of SDL's concrete syntax as proposed in Chapter 8 and formally defined in Appendix B, which could not be realized in a straightforward way, because they would not be accepted by the analyzer and SDL-PR generator of the SDL Suite. Instead, an annotation-based approach has been implemented. The advantage of annotations is that they are indeed ignored by the SDL Suite and not checked by the analyzer, but copied to SDL-PR without modification, thereby enabling further processing with ConTraST.

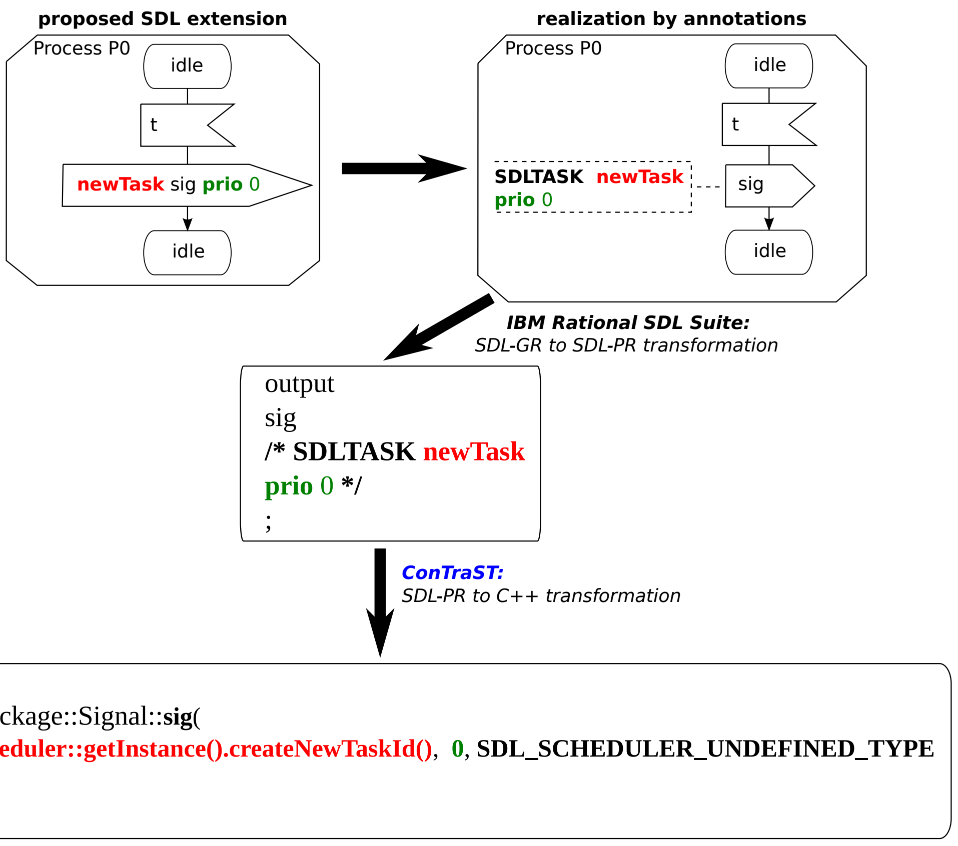
9.3 Incorporation of Task Actions, Attributes, and Operators

To implement SDL real-time tasks, two complementary strategies are applied. On the one hand, annotations are used to specify task actions and to create task signals in the specification. These annotations are then mapped to the implementation by extensions of the code generator ConTraST. On the other hand, task attributes, task operators, and operators for task suspension are realized with built-in language features of SDL like `syntypes` and operators. In the following, both strategies are discussed in more detail.

9.3.1 Extended Code Generation with ConTraST

ConTraST [FGW06, Fli09] is an SDL-2000 to C++ transpiler and a result of an academic project. To pre-process SDL specifications given in SDL-PR, ConTraST utilizes Flex[1], a scanner generator, and Bison[2], a generator for parsers based on context-free grammars.

To stay compatible with IBM's Rational SDL Suite [IBMar], task signals are created by annotation-based task actions. The syntax of the annotations is very similar to the proposed extensions of SDL's concrete grammar. They are ignored by the SDL Suite and preserved when translating the graphical SDL specification to SDL-PR, which is then analyzed and evaluated by ConTraST. The transformation steps and differences between proposed syntax extensions and annotation-based realization are sketched in Figure 9.2. In the example, a new real-time task is created by using keyword `newTask` in the annotations. When ConTraST evaluates this keyword, it adds corresponding task attributes to the C++ representation of the signal. W.r.t. the task identifier, the task scheduler is consulted in order to obtain a new unique identifier.



```
output
sig
/* SDLTASK newTask
prio 0 */
;
```

```
Output(
   new ::SigPackage::Signal::sig(
      TaskScheduler::getInstance().createNewTaskId(), 0, SDL_SCHEDULER_UNDEFINED_TYPE
   )
)
```

Figure 9.2: Annotation-based implementation of task actions.

[1]Fast LEXical analyzer, http://flex.sourceforge.net/
[2]http://www.gnu.org/software/bison/

9.3.2 Realization of Task Attributes and Task Operators with Native SDL

To realize data types for task attributes, SDL type synonyms are used. Though a drawback of type synonyms compared to novel data types is the reduced type safety, the chosen approach is compatible with the editor and analyzer of the SDL Suite and requires very little effort. The required SDL type synonyms are presented in Fig. 9.3. They are specified in a new SDL package, which is dedicated to real-time task-related extensions and can be referenced by all SDL projects in which SDL real-time tasks are applied.

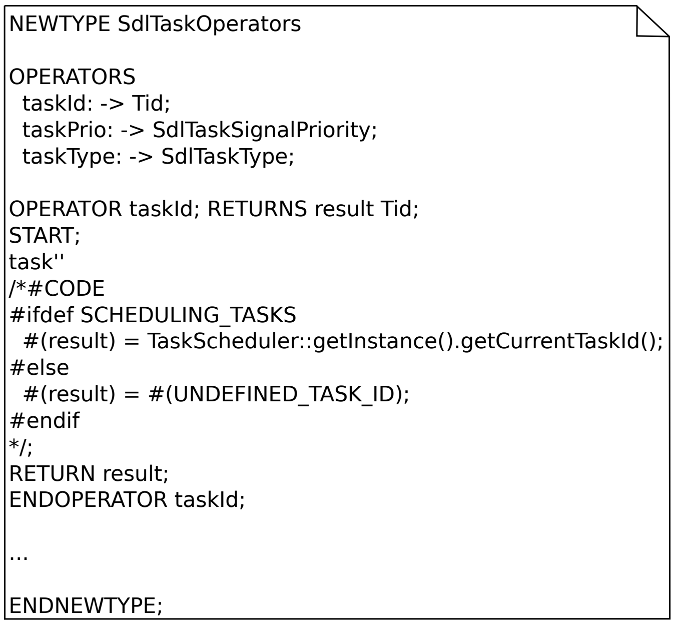
```
SYNTYPE Tid = Natural ENDSYNTYPE;
SYNTYPE SdlTaskSignalPriority = Integer ENDSYNTYPE;
SYNTYPE SdlTaskType = Integer ENDSYNTYPE;
```

```
SYNONYM UNDEFINED_TASK_ID Tid = 0;
SYNONYM UNDEFINED_TASK_PRIORITY SdlTaskSignalPriority = -1;
SYNONYM UNDEFINED_TASK_TYPE SdlTaskType = -1;
```

Figure 9.3: Realization of task attributes as SDL type synonyms.

Compared to the SDL data types that have been introduced for task signal priorities and task types in Sect. 8.3, their syntype-based realization differs in some points. For task signal priorities, integers instead of natural numbers are used, and -1 is selected to refer to undefined task signal priorities (see also element *undefined* of domain TASKPRIORITY in SDL's extended formal semantics in Listing C.2). Regarding task types, SDL's built-in integer type is applied instead of type generators with literals. Accordingly, a designer has not to add a new literal but a scenario-specific integer synonym when introducing a new task type in an SDL specification. In this regard, a synonym `UNDEFINED_TASK_TYPE` is predefined to refer to undefined task types in a scenario-independent way.

The realization of task operators is based on SDL's support for operator definitions and is outlined in Fig. 9.4. In the figure, only `taskId` is shown in detail, whereas for `taskPrio` and `taskType`, only signatures are presented. All operators actually delegate invocations to the implementation of the task scheduler (see also Sect. 9.4.2), which stores context information about the current transition execution and returns the corresponding task attribute. If at the moment of invocation not task scheduling but a different scheduler runs, `taskId` would by default return the value for an undefined task identifier. Operators for task suspension and resumption are also realized by SDL operators, which – similar to task operators – delegate calls to the task scheduler.



```
NEWTYPE SdlTaskOperators

OPERATORS
  taskId: -> Tid;
  taskPrio: -> SdlTaskSignalPriority;
  taskType: -> SdlTaskType;

OPERATOR taskId; RETURNS result Tid;
START;
task''
/*#CODE
#ifdef SCHEDULING_TASKS
  #(result) = TaskScheduler::getInstance().getCurrentTaskId();
#else
  #(result) = #(UNDEFINED_TASK_ID);
#endif
*/;
RETURN result;
ENDOPERATOR taskId;

...

ENDNEWTYPE;
```

Figure 9.4: Realization of task operators.

9.4 SdlRE– An SVM Implementation with SDL Real-time Tasks

SdlRE is an SVM implementation for SDL-2000 and written in C++. It is responsible for system initialization, the transfer of SDL signals, and scheduling of transition executions.

A main objective of SdlRE is to be high compliant with SDL's dynamic semantics as specified in Z.100 Annex F of the SDL-2000 standard [Int00]. However, as a real-world implementation, there are several major differences from the ASM semantics of SDL: The largest difference is the reduced degree of concurrency. Another difference is the compilation of quantifications in ASM into iterations in C++ [PvL03b]. Though this second difference is not really a difference from a functional point of view, it leads to a large mismatch from a performance perspective, since the statement of a simple quantification in ASM may result in expensive loops. To deal with such problems, implementation optimizations – like software caches [Krä13b] – have been proposed. A further difference arises due to the semantics of an ASM move, which requires that each update set is *consistent* [BS03]. This definition forbids the assignment of different values to the same *location*[3] within the same move.[4] This strict limitation does not hold in imperative or object-oriented programming languages and was exploited for further optimizations in SdlRE.

To allow comparison evaluations with different scheduling algorithms, SdlRE includes a scheduling framework [Chr10] with several single-thread schedulers. Currently supported strategies are summarized in Table 9.1. The table also contains the SDL keywords to select the strategy in scheduling-aware system specifications (see also Sect. 8.2). All strategies are online strategies [Liu00] and event-driven, i.e., scheduling is done at runtime without knowledge about future workload and transitions are executed when they are firable and not at predeter-

[3]In broad terms, locations correspond to persistent variables.

[4]To subsequently change the same location to different values, multiple moves are necessary, where each move is the result of one ASM program execution.

Name	Type	Description
RR_{agents}	agent-based round robin	executes every agent (also *Links* and *SdlAgentSets*) in a cyclic manner and independent of open tasks or firable transitions [Chr10]. This scheduling strategy corresponds most closely to SDL's dynamic semantics. Keyword to select strategy in SDL: **non-optimized**
FCFS_{agents}	agent-based FCFS	executes an *SdlAgent* until it has no firable transition [Chr10]. *SdlAgents* becoming executable are enqueued in FCFS order. Signals are transferred between *SdlAgents* directly, and *Links* and *SdlAgentSets* are skipped. Keyword to select strategy in SDL: **default**
$\mathrm{Priorities}_{agents}$	priority scheduling	derives priorities for *SdlAgents* from SDL block and process priorities and orders transition executions based on their *SdlAgent*'s priority [Chr10]. *Links* and *SdlAgentSets* are skipped when transferring a signal. Allows optionally the suspension of *SdlAgents* by means of a priority threshold. Keyword to select strategy in SDL: **static-priorities**
$\mathrm{FCFS}_{signals}$	signal-based FCFS	maintains a global signal queue ordered by the arrival time of signals and executes transitions of *SdlAgents* accordingly. *Links* and *SdlAgentSets* are not executed explicitly and skipped during signal transfer. Keyword to select strategy in SDL: **signals**
$\mathrm{Priorities}_{tasks}$	task scheduling	orders transition executions by task signal priorities. *Links* and *SdlAgentSets* are also skipped when transferring signals. Tasks can optionally be suspended as function of their type, id, or a priority threshold. Keyword to select strategy in SDL: **tasks**

Table 9.1: Scheduling strategies of SdlRE.

mined points in time. Though this class of schedulers generates more overhead at runtime, it can deal with the actor model of SDL best. None of the schedulers supports full preemption, i.e., a transition execution must be finished before another transition execution can start.

The table includes two strategies that have been developed in the course of this thesis: A signal-based FCFS strategy (FCFS$_{signals}$) and task scheduling (Priorities$_{tasks}$), a priority-based strategy for real-time tasks. Different from real-time task-aware transition selection in Chapter 8, task scheduling serializes transition executions system-wide. It is discussed in more detail in Sect. 9.4.2. But at first, Sect. 9.4.1 summarizes extensions of SdlRE regarding creation and control of task signals.

9.4.1 Task Attributes, Task Actions, and Task Operators

To implement task signals, data structures of SDL signals (plain signals as well as timers) are extended to store task attributes. These attributes consist of task identifier, task type, and task signal priority and are now part of all signals. In case of a non-task signal, they are set to an *undefined* values (see also Fig. 9.3). Global uniqueness of task ids is ensured by dividing identifiers into a locally unique part and a part containing the unique node id. Currently, identifiers are of 32 bits size, where 8 bits are reserved for the node id and 24 bits store the local part, which is incremented each time a new task is created. If an overflow occurs at the local part, it starts from beginning. Consequently, clashes of ids may arise in theory but are hardly a problem in practice. In systems with non-terminating or many long-living SDL real-time tasks, further measures are conceivable to guarantee uniqueness, e.g., enlarging local parts or blacklisting ids of *active* real-time tasks.

Regarding task actions, no further adaptations are required in SdlRE, because required code is already generated by ConTraST. Task operators, which allow access to task attributes of consumed signals, only require an interface to the scheduler to get context information about the running transition.

9.4.2 Task Scheduling – A Scheduling Strategy for SDL Real-time Tasks

To execute transitions according to task signal priorities, the scheduling strategy *task scheduling* has been devised. The strategy does not only realize the priority-based transition execution order within an *SdlAgent* but additionally serializes transition executions of concurrent *SdlAgents*. *Links* and *SdlAgentSets* are not explicitly considered but implicitly executed.

9.4.2.1 Mode of Operation

Figure 9.5 presents the mode of operation of task scheduling: The scheduler works on three global queues: The first queue stores all *waiting signals*, i.e., task signals and non-task signals whose arrival time is larger than `now`, and is sorted by the arrival time of the signals. Elements stored in this queue are SDL timers, signals over delaying SDL channels, signals with activation delay, and real-time signals [KBCG11]. Task signals whose arrival time is smaller than or equal to `now` are stored in a queue with *ready task signals*, which is sorted by task signal priorities.

The third queue does not contain signals but *ready SdlAgents* and is sorted by FIFO. By storing *SdlAgents* and not signals, different transition triggers like continuous signals or spontaneous

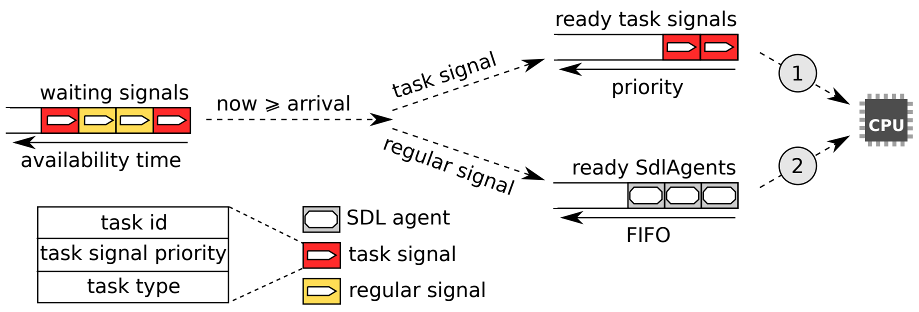


Figure 9.5: Outline of the mode of operation of task scheduling.

transitions are supported. In this regard, *ready* means that the *SdlAgents* have *potentially* firable transitions. Whether there is actually a firable transition is not always decidable at this point in time, if, for instance, enabling conditions or continuous signals come into play. It is even possible that a transition of an *SdlAgent* becomes firable without change of the agent's state, if, for instance, conditions depend on system time.

In addition to the three global queues, each *SdlAgent* holds a signal queue that corresponds to its input port as specified in the SDL semantics [Int00]. Note that in case of task signals, signals are stored both in the agent's input port and in the global task signal queue.[5]

When searching for the next *SdlAgent* to be executed, the dispatcher of the runtime environment consults the scheduler, which first searches for consumable task signals, since they take precedence over other transition triggers. If there is no such signal, an *SdlAgent* with a different transition trigger is executed. Listing 9.1 presents the corresponding function in pseudo code.

```
function getNextAgent() {
  scheduleExpiredSignals();

  // Search for consumable task signal with highest priority
  foreach (s in taskSignals) do {
    if (!isSuspended(s)) {
      switch (getConsumptionType(s)) {
        case transitionInput:
          currentTaskId = s.taskId;
          currentTaskPriority = s.priority;
          currentTaskType = s.type;
          taskSignals.del(s);
          return s.agent; // Return the agent holding this signal
        case implicitTransitionInput:
          taskSignals.del(s);
          break;
        case savedInInport:
          break;
        default:
          break;
      }
    }
  }
```

[5]Strictly speaking, references to the signals and no signal copies are stored to save memory.

```
24
25    // Execute environment or agents with non-task signal triggers
26    currentTaskId = undefined;
27    currentTaskPriority = undefined;
28    currentTaskType = undefined;
29    if (agents.empty())
30      return SDL.getEnvironment();
31    else
32      return agents.begin();
33  }
```

Listing 9.1: Pseudo code determining the *SdlAgent* to be executed

The first step when determining the next agent is removing expired signals from the waiting signal queue. This is done by function *scheduleExpiredSignals()* in line 2, which either inserts expired signals into the list of task signals or enqueues the owning *SdlAgent* in the agent queue. Afterwards, the list of task signals is searched for consumable and non-suspended task signals (lines 5-23). Here, it is additionally checked whether a non-suspended task signal is saved, which can occur either due to an explicit SDL `save` or due to a non-matching enabling condition. If this is the case, the signal is skipped but stays in queue. In contrast, signals implicitly consumed in the agent's current state are removed from the task signal queue. Referring to this, see also Appendix C, where the extended formal semantics gives details on the implicit consumption of task signals. If there is no consumable task signal, either the environment agent or the first *SdlAgent* of the agent queue is executed. The environment, particularly, is executed, if there is no executable *SdlAgent*, and may put the hardware to sleep in order to save energy.

To maintain information about suspended tasks, two sets are introduced in the implementation to store suspended task ids and task types. Additionally, a priority threshold value is used to suspend tasks by priority. For all three data structures, interfaces have been realized to enable modifications via SDL operators as described in Sect. 9.3.2.

9.4.2.2 Implementation

So far, the scheduling framework of SdlRE had only consisted of agent-based scheduling strategies. In order to implement task scheduling, extensions for signal-based schedulers were necessary. As a side effect, a signal-based FCFS strategy has been obtained "for free".

All scheduling strategies in SdlRE are composed of a set of basic schedulers, which are shown in Fig. 9.6 on the left-hand side. In total, there are six basic schedulers, queueing items – i.e., agents or signals – either by priorities, by deadlines, or by FIFO. The basic schedulers indirectly inherit from an abstract template class called `BaseScheduler`, which defines a common interface for all basic schedulers. Thereby, access to an instantiated scheduler is possible without knowledge about its actual implementation.

During compile time, all basic schedulers are determined that are required for the chosen scheduling strategy. For task scheduling, the resulting object structure is presented in Fig. 9.7. The object of class `TaskScheduler` implements the abstract class `SchedulerInterface`, which defines a common interface for all scheduling strategies, and references three basic schedulers: `PrioritySignalScheduler` to queue task signals by priority, `EDFSignalScheduler` to store all

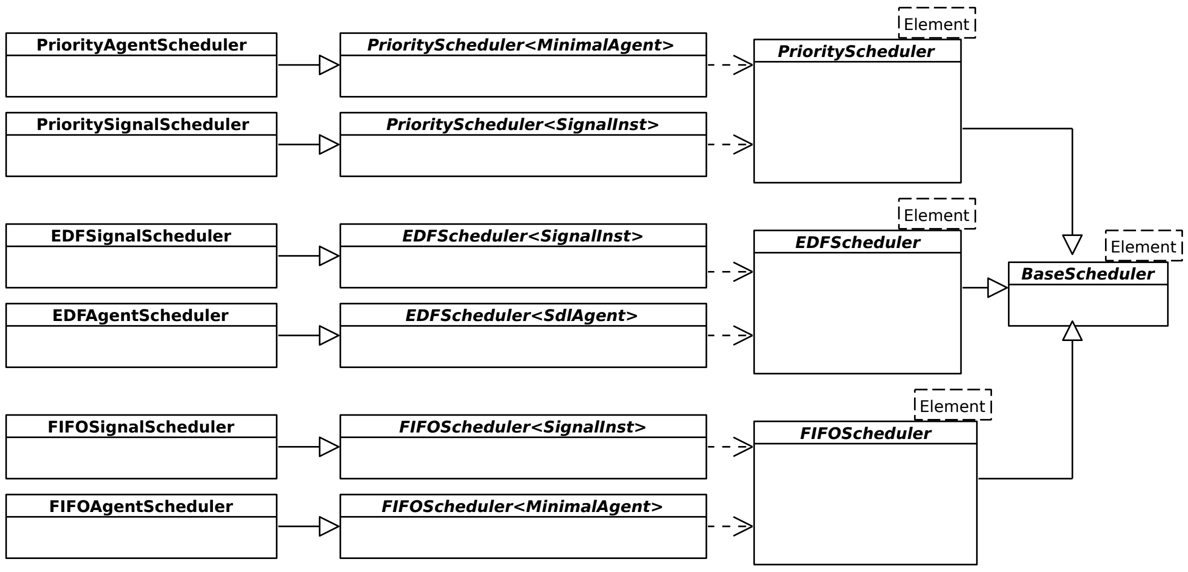


Figure 9.6: Basic schedulers of SdlRE.

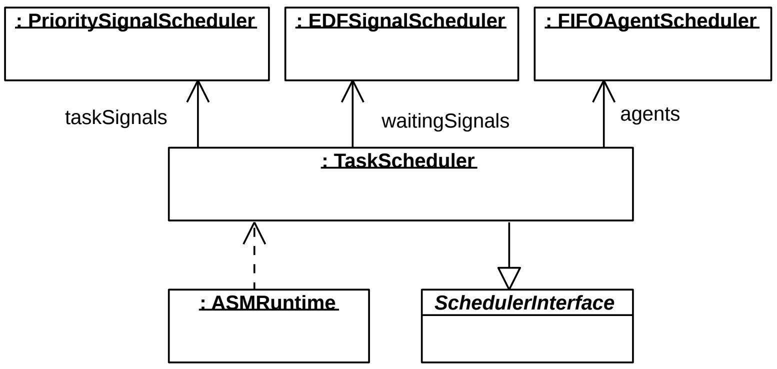


Figure 9.7: Dependencies between task scheduling and basic schedulers.

waiting signals, and `FIFOAgentScheduler` to store *SdlAgents* with non-task signal transition triggers. It is consulted by the dispatcher (`ASMRuntime`) after each transition execution.

The methods provided by task scheduling are shown in Fig. 9.8. Some of them – e.g., for adding a new signal to the schedule – are prescribed by `SchedulerInterface`. Others – like for accessing the task identifier of the current transition execution – are specific for task scheduling. To guarantee that there is only a single instance of `TaskScheduler`, access to the scheduler is via singleton pattern [GHJV95].

9.4.2.3 Pros and Cons of Preemption

Scheduling strategies with preemption allow the interruption of a task in order to assign the processing unit to a different and possibly more urgent task [Liu00]. In general, preemptive strategies achieve smaller blocking times and can reduce reaction times to critical events significantly. This, particularly, holds in systems with long-running tasks. As a consequence, they can solve more scheduling problems than non-preemptive algorithms.

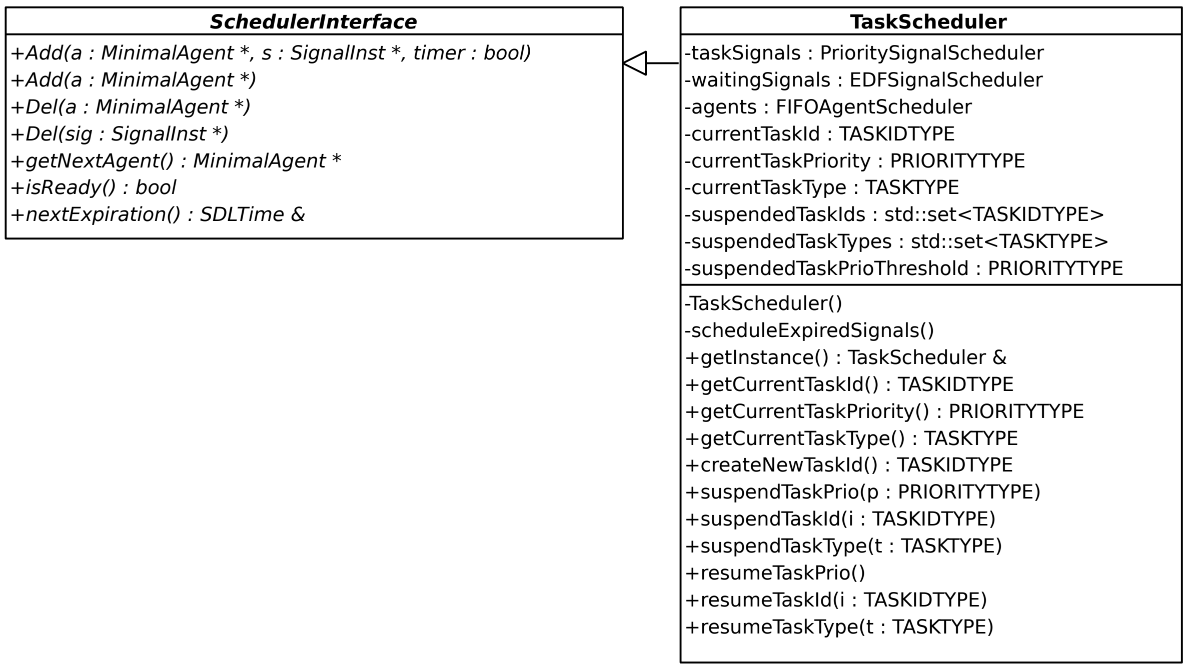

SchedulerInterface
+Add(a : MinimalAgent *, s : SignalInst *, timer : bool)
+Add(a : MinimalAgent *)
+Del(a : MinimalAgent *)
+Del(sig : SignalInst *)
+getNextAgent() : MinimalAgent *
+isReady() : bool
+nextExpiration() : SDLTime &
TaskScheduler
-taskSignals : PrioritySignalScheduler
-waitingSignals : EDFSignalScheduler
-agents : FIFOAgentScheduler
-currentTaskId : TASKIDTYPE
-currentTaskPriority : PRIORITYTYPE
-currentTaskType : TASKTYPE
-suspendedTaskIds : std::set<TASKIDTYPE>
-suspendedTaskTypes : std::set<TASKTYPE>
-suspendedTaskPrioThreshold : PRIORITYTYPE
-TaskScheduler()
-scheduleExpiredSignals()
+getInstance() : TaskScheduler &
+getCurrentTaskId() : TASKIDTYPE
+getCurrentTaskPriority() : PRIORITYTYPE
+getCurrentTaskType() : TASKTYPE
+createNewTaskId() : TASKIDTYPE
+suspendTaskPrio(p : PRIORITYTYPE)
+suspendTaskId(i : TASKIDTYPE)
+suspendTaskType(t : TASKTYPE)
+resumeTaskPrio()
+resumeTaskId(i : TASKIDTYPE)
+resumeTaskType(t : TASKTYPE)

Figure 9.8: Task scheduling and its common and specific interface methods.

Preemption, however, also has a downside: From an overhead perspective, preemption increases system load, since every preemption introduces context switches and deteriorates cache hits [BBY13]. This leads, in general, to a reduced predictability and to a more difficult calculation of WCETs [JSM91]. Furthermore, preemptive algorithms are harder to implement, and require coordination and restrictive synchronization for the access to shared memory.

In SDL, an additional problem can be observed, when running an SDL system with priority-based transition executions and a fully preemptive scheduling strategy, i.e., a scheduling strategy that allows the interruption of transition executions at any part of the transition's body. This problem can lead to situations, in which high priority transition executions are delayed because their activation is deferred due to preemption. To illustrate this problem, consider an SDL transition that runs with priority $x + 1$ and contains two signal outputs. The first signal output triggers a transition with priority x, where the second output triggers a transition execution with a higher priority $x - 1$. If a fully preemptive scheduling strategy is used, the processing of the second output is postponed, since the transition triggered by the first output has a higher priority than the current transition execution and is therefore executed first. Thereby, the activation and execution of the second transition, which is actually most urgent, is delayed. In summary, this problem is a result of SDL's asynchronous communication pattern that both transfers data and activates transitions. It arises, because the order of signal outputs becomes crucial with a fully preemptive execution. A simple solution would be the re-sort of signal outputs w.r.t. priorities. This is, however, in most specifications non-intuitive and painful if, for instance, branches come into play.

Task scheduling avoids these effects by allowing preemption of SDL real-time tasks only with the granularity of transition executions. As a consequence, interruption within a transition execution in favor of a more urgent transition execution is not possible. Thus, task scheduling is neither fully preemptive nor non-preemptive, and corresponds to cooperative scheduling [BBY13], where preemption is allowed at well-defined points, thereby causing substantially lower synchronization overhead than full preemption. In the context of OSs, scheduling with such fixed preemption points is also called *deferred preemption* [Bur94]. With SDL task scheduling, the designer does not need to provide explicit preemption points, since the runtime environment gets control back after each transition execution. Thus, if complex transitions with long execution times are forbidden by guidelines (see also [BCG09]), SDL task scheduling represents a suitable trade-off between a full- and non-preemptive scheduling strategy.

9.5 Demands of SDL Real-time Tasks from the SDL Environment

Though SDL real-time tasks are not an implementation technique for a particular platform, their main application domain targets embedded systems. Thus, the focus of their implementation is on the wireless sensor platform Imote 2 [MEMara] (see Appendix A), which is used as embedded system representatively. The SDL realization for Imote 2 is a bare implementation without further OS. Thereby, SDL real-time tasks can be evaluated on this platform without any side effects like undesired context switches. Though most parts of their implementation are platform-independent, platform-specific extensions of hardware drivers are required to support distributed tasks. In our tool chain, these extensions are provided by SEnF and summarized in Sect. 9.5.1. In addition, a supplementary implementation technique called *early timestamping* is introduced in Sect. 9.5.2, which improves the accuracy of event detection.

9.5.1 Realization of Distributed SDL Real-time Tasks

To realize distributed SDL real-time tasks, task attributes must be appended to outgoing data and derived from incoming data. For this, the device, on which the data is sent/received, must support frame-based transmissions. Frame-based transmissions are, for instance, provided by communication technologies like IEEE 802.3 Ethernet [Ins12b] or IEEE 802.15.4 [Ins11]. The overhead to transfer task attributes in addition to regular payload depends on the supported transmission rates and frame sizes. While it can be neglected for technologies like Ethernet, the overhead can become significant with IEEE 802.15.4-compliant wireless transceiver.

Before transmitting task attributes, they must be serialized and appended to the data frame. To serialize and deserialize attributes, two methods have been introduced in the implementation of SDL signals in SdlRE. The serialization methods returns a byte sequence containing all defined task attributes. It has a minimal length of 1 byte (if signal is no task signal) and a maximal length of 13 bytes (if signal is task signal with all task attributes). On the opposite side, a deserialization method extracts task attributes from a byte sequence.

Distributed SDL real-time tasks have exemplarily been realized in the implementations of the CC 2420 transceiver. Here, task attributes are appended to frames before they are transferred to the transceiver, and derived from received frames. Though there are actually no manual actions required by the designer, the usage of distributed SDL real-time tasks is not entirely transparent. In particular, it has to be avoided a priori that data frames exceed the maximum frame length after appending task attributes. In addition, larger data frames increase transmission times, which has, for instance, to be considered when dimensioning communication slots.

9.5.2 Early Timestamping

A complementary measure to improve monitoring of events is independent of SDL real-time tasks and called *early timestamping* [BCG09]. It requires extensions in interrupt routines of the environment implementation to timestamp events that are the cause of the interrupts. When the detected event is later propagated with a regular SDL signal to the SDL system, the stored timestamp is attached to the SDL signal. This can be realized either by means of an ordinary signal parameter or by adopting extensions of the SVM implementation like the anonymous variable `sendtime` (see related work in Sect. 11.2.8 and [KBCG11]). Without early timestamping, the point in time of an event is usually not taken until the signal informing about the event is consumed in an SDL transition. Due to various delays between the occurrence of the event and the execution of this transition, this timestamp can differ significantly from the actual event, thereby making the timestamp almost useless. By reading and storing hardware clock values in implementation-specific interrupt routines, timestamps become significantly more accurate.

To gather more precisely the times of a medium state change, early timestamping has been applied in the CC 2420 transceiver driver in SEnF. The relevant function is illustrated in Listing 9.2. It is invoked during interrupt handling after the transceiver signals a medium state change via the CCA pin. Here, the driver stores the clock value and the current medium state.

```
/* called when an interrupt on the CCA pin has occurred */
void driver_cc2420_CCA_Interrupt () {
  // store current timestamp and get CCA status from transceiver
  SEnF_CC2420_Data.actual_time = now();
  SEnF_CC2420_Data.actual_CCA = GPLR(CCA_PIN) & GPIO_BIT(CCA_PIN);

  envHasSignals = TRUE;
}
```

Listing 9.2: Early timestamping in an interrupt routine of the CC2420 driver.

In Listing 9.3, the function `send_CCA_signal()` that is invoked when the environment is executed by the SDL scheduler is shown. In the function, the stored timestamp is appended as a regular parameter to an SDL signal, which is then sent to the SDL system to report on the CCA event. Though the shown function is invoked possibly much later than the interrupt routine from Listing 9.2, the accuracy of the timestamp remains unaffected.

```
/* called by \sdlre during environment polling */
void send_CCA_signal() {
  // send only if there's a change of the CCA pin
  if (SEnF_CC2420_Data.actual_CCA != SEnF_CC2420_Data.last_CCA){

    // Create and initialize SDL signal CC2420_CCA
    SENF_DECLARESIGNAL(CC2420_CCA, SignalIn)
    SENF_GETSDLSIGNAL(CC2420_CCA, SignalIn)

    // Assign signal parameters
    SDL_SIGNAL(SignalIn).Param1 = SEnF_CC2420_Data.actual_CCA;
    SDL_SIGNAL(SignalIn).Param2 = SEnF_CC2420_Data.actual_time;

    // Save CCA status for comparison to avoid reports with same CCA status
    SEnF_CC2420_Data.last_CCA = SEnF_CC2420_Data.actual_CCA;

    // Send signal to SDL
    SENF_SENDTOSDL(CC2420_CCA, SignalIn, 0)
  }
}
```

Listing 9.3: Delivery of the early timestamp as parameter of the SDL signal.

9.6 The Simulator Framework FERAL

FERAL is a Java-based simulation framework [BCG+13, BCG+14]. Its objectives are the rapid coupling of domain-specific simulators and virtual prototyping for early testing and evaluation of design alternatives. By incorporating various simulators and models with different abstraction levels, FERAL supports continuous testing of networked systems during different development phases and enables functional and non-functional evaluation of various communication technologies.

FERAL distinguishes between two types of simulation components: *Functional Simulation Components* (FSCs) modeling the behavior of (networked) nodes and *Communication-based Simulation Components* (CSCs) simulating interaction between FSCs. In addition, FERAL supports *bridges* and *gateways* to interconnect different CSCs and to simplify their exchange. FSCs can, for instance, be native components written in Java or Matlab Simulink[6] models. CSCs can be simulated by ns-3[7] – thereby supporting various communication technologies like IEEE 802.3 (Ethernet) [Ins12b] and IEEE 802.11 (WLAN) [Ins12a] – or by specialized bus simulators for CAN [Int04] and FlexRay [Fle10].

To evaluate SDL real-time tasks in the context of FERAL, the simulator and the SDL tool chain have been extended to support SDL-based FSCs, which are automatically derived from SDL specifications. In this regard, extensions have been implemented in SEnF, SdlRE, and FERAL. The required steps to control an SDL FSC's transition executions by FERAL and to interconnect SDL FSCs to CSCs are summarized below. Initially, Sect. 9.6.1 presents the realization of SDL FSCs as shared libraries, where the simulator core has full control over time progress in the SDL system. In this variant, execution times of transitions and the SDL runtime environment are not considered. Thus, this variant is suitable for functional evaluations of SDL models and quantitative assessments of the selected communication technologies. Afterwards (Sect. 9.6.2), a second variant of SDL FSCs is introduced, in which SDL models are executed on Imote 2 platforms by Hardware-in-the-Loop (HiL) simulations. Thereby, execution delays can be evaluated and comprehensive performance assessments of SDL real-time tasks become possible.

[6]http://www.mathworks.com/products/simulink/

[7]http://www.nsnam.org

9.6.1 Library-based SDL FSCs for Functional Evaluations

To support SDL-based FSCs, two steps had to be applied: First, parts of the control over transition executions and time progress had to be moved from the runtime environment of SDL to FERAL. Second, conversions between SDL signals and simulator messages had to be realized to integrate SDL models into the data flow of FERAL. Both steps are explained in more detail below. Since FERAL is implemented in Java but the SDL tool chain is in C++, the *Java Native Interface* (JNI[8]) was used to overcome the language barrier.

Simulation components in FERAL are controlled by directors, where both time- and event-triggered execution semantics are supported and can even be nested. In the following, the focus is on FERAL's event-triggered model, since its semantics is most similar to SDL's execution model.[9] With this integration, the director that is responsible for an SDL FSC executes the SDL system in two situations: First, if a timer expires in the SDL system, or, second, if a simulator message was sent from another FERAL component to the SDL FSC.

The realization of an event-triggered and library-based SDL FSC is illustrated by the simplified class diagram in Fig. 9.9. With the exception of the SDL system, which is loaded as shared library and accessed via JNI, all classes are implemented in Java. Each object of the central `SDLEventTriggeredComponent` class represents one SDL FSC and can possess several input and output ports to interact with other FSCs and CSCs. In order to control an SDL FSC's execution, the methods defined by the `SimulationComponent` interface are implemented. They are invoked consecutively each time the SDL FSC is executed:

- *preFire()* checks and returns whether the SDL component is enabled to avoid the execution of a disabled component in the *fire()* step.

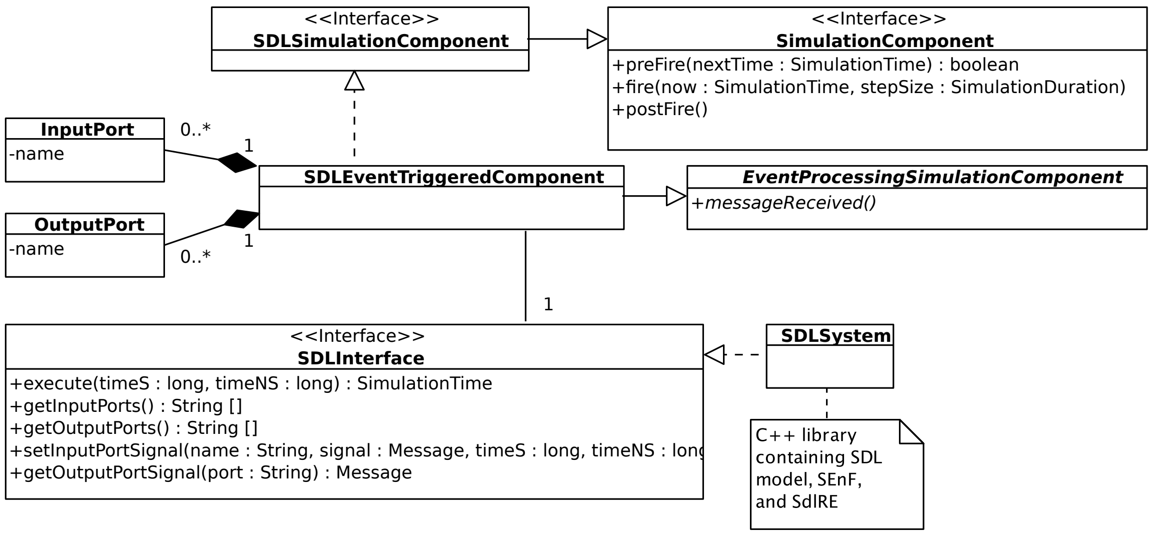

<<Interface>>
SDLSimulationComponent
<<Interface>>
SimulationComponent
+preFire(nextTime : SimulationTime) : boolean
+fire(now : SimulationTime, stepSize : SimulationDuration)
+postFire()
InputPort
-name
0..*
1
SDLEventTriggeredComponent
EventProcessingSimulationComponent
+messageReceived()
OutputPort
-name
1
0..*
1
<<Interface>>
SDLInterface
+execute(timeS : long, timeNS : long) : SimulationTime
+getInputPorts() : String []
+getOutputPorts() : String []
+setInputPortSignal(name : String, signal : Message, timeS : long, timeNS : long
+getOutputPortSignal(port : String) : Message
SDLSystem
C++ library containing SDL model, SEnF, and SdlRE

Figure 9.9: Library-based SDL FSC with event-driven execution semantics.

[8] http://docs.oracle.com/javase/7/docs/technotes/guides/jni/

[9] Actually, SDL's time- and event-triggered FERAL integrations are almost identical for library-based SDL FSCs. With both execution semantics, all firable SDL transitions are executed instantaneously.

- *fire()* updates the time of the SDL system to the current simulation time. Afterwards, it executes all firable transitions of the SDL system (see *execute(. . .)* of `SDLInterface`). After executing the last firable transition, the SDL system returns the time when the next timer (either an SDL timer or an emulated hardware timer) expires and the SDL FSC should be executed again by the director.

- *postFire()* fetches all SDL signals that were sent to the SDL environment during *fire()*, converts them to simulator messages, and queues them in the corresponding output ports of the SDL FSC (see *getOutputPortSignal(. . .)* of `SDLInterface`). Afterwards, the messages are forwarded via links from the SDL FSC's output ports to the input ports of connected simulation components.

To transfer messages between simulator and SDL system, the port concept of FERAL, which is based on named ports, is adopted by SEnF. In more detail, the port concept is reproduced as follows: For each communication medium supported by FERAL, virtual device drivers are introduced in SEnF. Thus, SEnF provides, for instance, a device driver for a Point-to-Point (PtP) medium. Depending on the types of SDL signals that are declared in the SDL specification, corresponding virtual drivers are instantiated during the initialization of the system, where for each instantiated virtual driver, one input port and one output port is created. Their names are fixed and reflect the driver name (e.g., *ptp_rx* for the incoming port of the PtP medium). After initializing all required virtual drivers, the created port structure is copied to the Java side of the SDL FSC to announce available ports in the simulator. This is done by inquiring the names of all created input and output ports (see *getOutputPorts()* and *getInputPorts()* in `SDLInterface`). As result, all ports of the SDL system are also known by the simulator core and can be used to connect an SDL FSC with other simulation components by means of their port names.

If a message is sent from another simulation component to the SDL FSC, this message is indicated by method *messageReceived()* and forwarded from the SDL FSC's input port to the corresponding input port in the SDL system (see *setInputPortSignal(. . .)* of `SDLInterface`). For this purpose, simulator messages are converted into SDL signals and buffered in the SDL environment, until they are further processed when the SDL system is executed.

For illustrative purposes, Fig. 9.10 provides a small simulation scenario with two library-based SDL FSCs that are connected via a PtP communication medium. The shown object diagram is the result of following startup code:

```
// Setup simulator core and director
Scenario scenario = ScenarioImpl.scenario;
Director dir = new DiscreteEventDirector(new BasicTopology());
scenario.setRootComponent(dir);

// Create simulation components and add to root container
SimulationComponent sdl1 = new SDLEventTriggeredComponent("Node1", new
    SimulationTime(1, 0), new SimulationTime(3, 0));
SimulationComponent sdl2 = new SDLEventTriggeredComponent("Node2", new
    SimulationTime(1, 500000000), new SimulationTime(3, 0));
dir.addComponent(sdl1, null);
dir.addComponent(sdl2, null);

// Connect components via ptp
```

```
PTPConfiguration ptp_config = new PTPConfiguration();
ptp_config.setNumberOfPeers(2);
PTPSimulationComponent ptp = new PTPSimulationComponent(ptp_config);
dir.addComponent(ptp, null);

// Create and add links between FSCs and CSC
Link l1, l2, l3, l4;
l1 = new LinkImpl(sdl1.getOutputPort("ptp_tx"), ptp.getInputPort("RX_0"));
dir.addLink(l1, null);
...

// Start simulator
scenario.init();
scenario.setEndCondition(new EndTimeCondition(new SimulationTime(4, 0)));
scenario.execute();
```

Listing 9.4: Configuration and start of simulation scenario.

Both SDL FSCs and the PtP CSC are controlled by an event-driven director. When creating an SDL FSC by generating an object of `SDLEventTriggeredComponent`, a name – „`Node1`“ and „`Node2`“, respectively – is provided, which is also used as reference to the shared library containing the SDL system. At the end of the startup code, the simulation run starts for a duration of 4 s. However, the SDL FSCs start not before 1 s and 1.5 s, respectively, and terminate at 3 s.

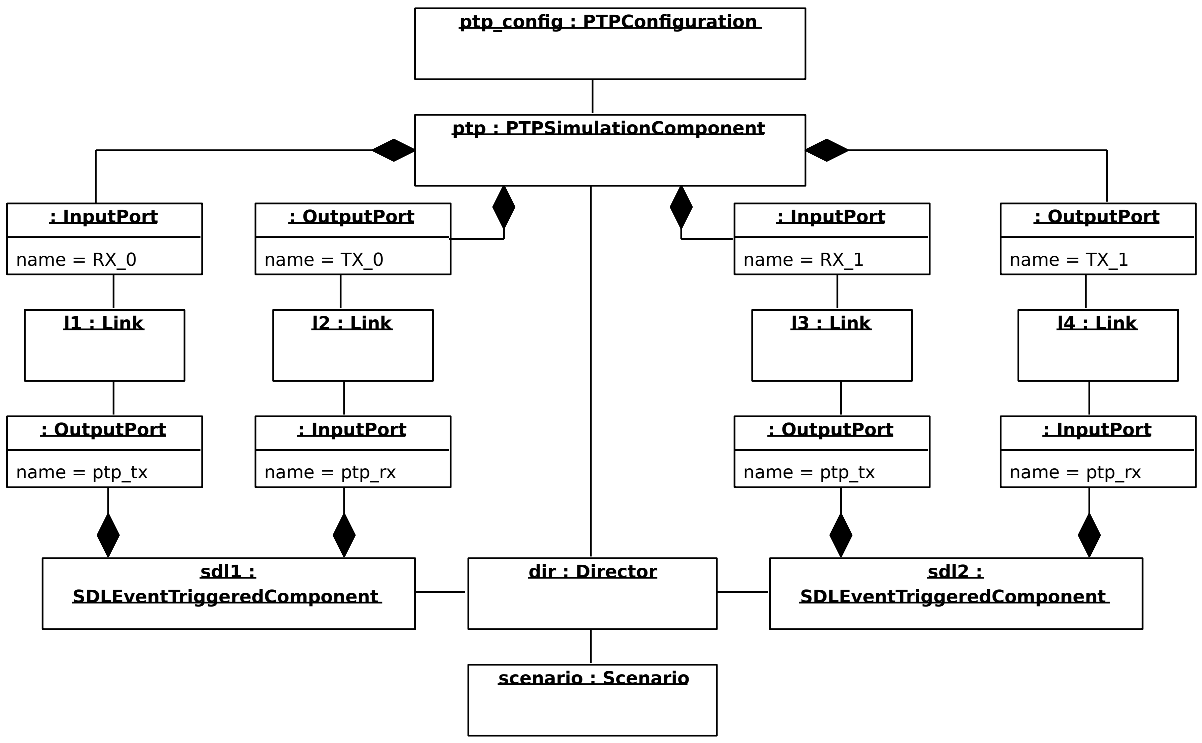


Figure 9.10: Object diagram of an example scenario with FERAL and two SDL FSCs.

9.6.2 SDL FSCs with Hardware-in-the-Loop Simulations

The library-based SDL integration into FERAL is well-suited for functional evaluations of SDL specifications. To consider non-functional properties like execution delays and to assess SDL real-time tasks, this integration variant is, however, not sufficient. Instead, the observation of realistic execution delays is required, which is achieved in a second integration variant by executing SDL models on HiL with the Imote 2 platform. Unlike simulation approaches with specialized platform simulators (e.g., the Atmel AVR simulators Atemu[10] and Avrora[11]), simulations with HiL do not require detailed models of the hardware to obtain realistic delays and to get accurate simulation results. The demand for detailed models is, in particular, a problem for platform simulators if modern hardware comes into play, because the more complex the hardware is the more expensive is an accurate simulation of all platform characteristics and optimizations like caches. Thus, a simulation approach with HiL states a practical alternative for performance evaluations in manageable scenarios.

When simulating an SDL system on HiL with FERAL, the Java part of the SDL FSC only serves as proxy for the actual execution of the SDL system on the Imote 2. On the Imote 2, there is similarly a FERAL stub in the SDL environment implementation that interfaces the SDL model. The concrete interplay between proxy and stub is illustrated in Fig. 9.11. Messages and commands between them are encoded with ASN.1 [Int08] and sent via a gateway, which communicates with the proxy by TCP/IP and with the stub on the Imote 2 by the serial interface UART (Universal Asynchronous Receiver Transmitter). To handle UART frame losses and corruption, timeouts, checksums, and ACKs are incorporated into the communication between proxy and stub. An advantage of introducing a dedicated gateway is that the node hosting the gateway and interconnecting the Imote 2 can be physically different from the node running the simulator, thereby enabling a spatial separation of simulator and HiL.

Abstracting from the communication via TCP/IP and UART, the logical interface of the SDL FSC with HiL is similar to the interface of the library-based SDL FSC as presented in Sect. 9.6.1. In particular, simulator messages are converted from and to SDL signals, signals to the SDL FSC

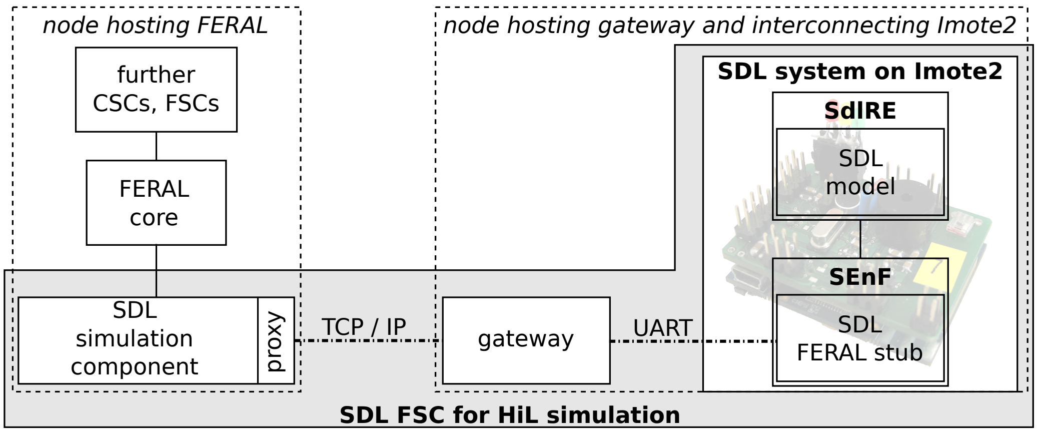


Figure 9.11: SDL FSC for Hardware-in-the-Loop simulations with FERAL.

[10]http://www.hynet.umd.edu/research/atemu/
[11]http://compilers.cs.ucla.edu/avrora/

are buffered in the SDL environment before executing the model, and outgoing SDL signals generated by the SDL model are fetched after the end of the SDL model's execution.

The biggest difference between both variants of SDL FSCs is regarding time progress in the SDL system: While transition executions are instantaneous and time progress is fully controlled by the simulator with library-based FSCs, the time of the SDL system with HiL simulations increases during executions, depending on the speed of the hardware and the workload of the system. In more detail, the time of the SDL system on HiL FSCs is managed during the FSC's *fire()* step as follows:

- Before executing an SDL transition, the SDL system's time is set to the same value as the simulation time, i.e., the clocks of FERAL and Imote 2 get synchronized.
- During the execution of the SDL system, the SDL system's time increases with the same rate as the physical time.
- After execution, the clock of the SDL system is stopped, so that there is no further time progress in the SDL system.

At the end of execution, the time of the SDL system usually precedes simulation time. For semantically correct simulations and to handle the differences between SDL time and simulation time, the following measures are applied: First, the Java simulation component is informed about the SDL system's time at the end of its execution. Second, the SDL system is not executed until the simulation time catches up and is larger than or equal to the SDL time. Additionally, to avoid messages from the future, signals sent by the SDL system are delayed until the simulation time is up to the SDL time.

9.7 Discussion

The implementation of SDL real-time tasks by means of annotations is a trade-off between full syntactical integration as suggested in Appendix B and possibilities given by available tools. Different from the incorporation of SDL real-time tasks into SDL's dynamic semantics (Appendix C), the implementation of the changed transition selection order goes one step further by flattening the concurrent execution model of SDL. In particular, by ordering transition executions as a function of task signal priorities system-wide, all three sources of delay as identified in Sect. 8.1.2 are addressed: Queueing delay by privileging urgent signals within the same *SdlAgent*, serialization delay by ordering transition executions system-wide, and run-to-completion delay by supporting the suspension of SDL real-time tasks. Since task signal priorities are neither derived from signal types nor from any other structural property of the system, they can be assigned flexibly and provide an adequate opportunity to privilege urgent transition executions.

Task scheduling as well as the implementation of task attributes, task operators, and task actions primarily affect the code generator ConTraST and the SVM implementation SdlRE. Platform-specific modifications to the environment implementation SEnF have become necessary to realize distributed tasks only. Thus, the SDL specification extended with SDL real-time task annotations as well as most parts of the implementation are independent of a specific hardware/software platform and compatible with the model-driven development process SDL-MDD. As result, SDL real-time tasks are now available on all supported platforms, yet some implementations to support distributed tasks are missing.

The scheduling strategy for SDL real-time task falls into the class of online [Liu00] and co-operative scheduling strategies with deferred preemption [Bur94, BBY13]. This means that scheduling decisions are made at runtime and preemption of SDL real-time tasks is possible, but only after each transition execution. It is additionally an idling strategy, if suspension is activated. In this case, the execution of firable transitions may be delayed due to the suspension of the containing SDL real-time task. Though the presented implementation of task scheduling realizes a total order on transition executions, such a strong order is not prescribed by SDL real-time tasks but was implemented, since the entire SDL tool chain only supports single-thread execution up to now. With multi-core hardware and after adaptations of the tool chain, parts of SDL's concurrency could be retained in compliance with the priority model of SDL real-time tasks, thereby providing significant benefits w.r.t. performance.

The implementation of distributed real-time tasks additionally requires extensions of SEnF to communicate task attributes. Thereby, SDL real-time tasks become node-spanning and other nodes can continue a running task with retention of its priority. Because task attributes are part of transferred payload, all nodes involved in communication must be aware of the modified frame content. Thus, it is not possible to use distributed SDL real-time tasks in heterogeneous environments, in which some nodes cannot handle the modified frames. However, in time-critical systems – such as found in production environments –, which are the target domain of SDL real-time tasks, node and system deployment are usually easy to control.

The validation of distributed systems is generally many times more expensive than the evaluation of single-node systems. To evaluate SDL real-time tasks in the context of networked systems without assembling the entire system with real hardware, the simulation framework

FERAL has been extended to enable SDL-based simulation components. Here, two variants have been designed: A library-based variant for functional evaluations of SDL specifications and a HiL variant for quantitative assessments. By enabling HiL simulations with SDL real-time tasks on Imote 2 nodes, reproducible performance evaluations on a typical embedded hardware have become possible and large-scale evaluations can be conducted. Due to the incorporation into FERAL, SDL real-time tasks can now be evaluated in the context of simulated networked systems and even with communication technologies that are not directly supported by Imote 2 nodes such as CAN or FlexRay.

10. Chapter

Evaluation of SDL Real-time Tasks

A major objective of model-driven development processes are quality and productivity improvements. With SDL-MDD, this is achieved by the automatic transformation of SDL specifications to implementations, which reduces development effort and enables the usage of the same model on various platforms. As a precondition to achieve this objective, the applied tool chain must be free of errors, thereby requiring systematic validation of all compilers and runtime environments, which are applied in any step of the development process. Thus, also SDL real-time tasks require an exhaustive evaluation of their functional correctness.

In this chapter, we first present the validation of the implementation of SDL real-time tasks in the utilized SDL tool chain. Afterwards, we investigate the main objective of SDL real-time tasks, which is the reduction of critical delays and the improvement of their predictability. This is addressed by performance evaluations, in which average and maximal delays are determined for time-critical tasks of control systems. In this regard, the identification of task scheduling overhead provides additional evidence of the practical benefit of SDL real-time tasks. All performance evaluations are executed on the sensor platform Imote 2 [MEMara], which is used as representative of an embedded system.

The structure of this chapter is as follows: In Sect. 10.1, a motivation is presented discussing the advantages of experimental evaluations. Afterwards, Sect. 10.2 gives a summary on a test system, which is used for functional evaluations of SDL real-time tasks, and test results. Section 10.3 then outlines results of an extensive performance evaluation of a time-critical system from the automotive domain. Thereafter, Sect. 10.4 presents further performance evaluations with a distributed control system and an inverted pendulum, which have been conducted with FERAL and HiL simulations. Finally, Sect. 10.5 summarizes the results and weighs pros and cons of SDL real-time tasks.

Evaluation results presented in this chapter have been published in [17] and [21]. Furthermore, the evaluated SDL system of Sect. 10.3 has also been used in [19] and [24] for the assessment of communication systems.

10.1 Motivation

The main objective of SDL real-time tasks is the improvement of the applicability of SDL in real-time systems. Thus, one purpose of this chapter is the comparison between time-critical reaction delays that are achieved by applying SDL real-time tasks and task scheduling, and corresponding delays that are produced with other state-of-the-practice scheduling strategies. To determine execution times and reaction delays, static analysis and dynamic evaluation techniques can be distinguished [WEE+08]:

- *Static methods* are based on the off-line analysis of the system's code or executable. Their biggest challenge is the architecture of modern hardware, which introduces various optimization techniques like pipelines, caches, and branch prediction to speed-up the system. Thus, WCETs of single statements, which are required by static analysis, do not only depend on the type of statement but also on the history of the system, thereby making calculation of WCETs very complex and often very pessimistic. Today, such optimization techniques can be found on almost all platforms and even on embedded systems: The PXA 271 processor, for instance, that is installed on the Imote 2 node, provides 32 KB instruction cache, 32 KB data cache, a 7 stage pipeline, and branch prediction with 128 entries [Cor05]. A further challenge of static analysis is the correctness of assumptions regarding external events, which serve as input of the system and increase its workload. This is, in general, a problem for all open systems that interact with their environment, but is an even bigger issue for wireless systems, where externally caused load is hard to control.

- Execution times determined by *dynamic methods* are based on measurements during test runs of the system. Though dynamic methods can usually not guarantee to evaluate the worst-case situation, their results are in general more realistic and less pessimistic than static approaches. This is indeed a problem of static techniques, since they often lead to overestimated execution times. In [WEE+08], it is even shown that overestimations increase over the years due to hardware evolution, though there is a permanent improvement of analysis methods and tools. An approach to increase confidence in the results of dynamic methods are the utilization of statistic methods and the determination of probabilistic WCETs [EB01].

After presenting functional evaluations of SDL real-time tasks, this chapter follows a dynamic approach to determine execution delays, where as hardware, the Imote 2 is used as representative of an embedded hardware platform. The main reason for using dynamic and not static methods is reduced effort.[1] Though, as consequence, all delays, which are determined in this chapter, must not be seen as deterministic WCETs, this approach is sufficient to evaluate SDL real-time tasks and to relate task scheduling to other scheduling algorithms.

10.2 Functional Evaluation of SDL Real-time Tasks

To evaluate new functionalities in the SDL tool chain and for the purpose of regression tests, a permanently enhanced SDL test system has been built [Krä13b]. The system is more a synthetical test system than a realistic scenario. Its focus is on testing functionalities of the code generator ConTraST and the SVM implementation SdlRE. Regarding SEnF, only very few parts are evaluated. Consequently, testing mostly affects code, which is identical for all supported target platforms. However, the system is mainly intended to run on Linux, since it provides best possibilities w.r.t. debugging.

[1] Since SdlRE consists of more than 30,000 lines of code, the costs of an analytical approach would be enormous. Due to extensive usage of dynamically allocated memory and pointers, it is even questionable whether static analysis is actually possible with the used tool chain [WEE+08].

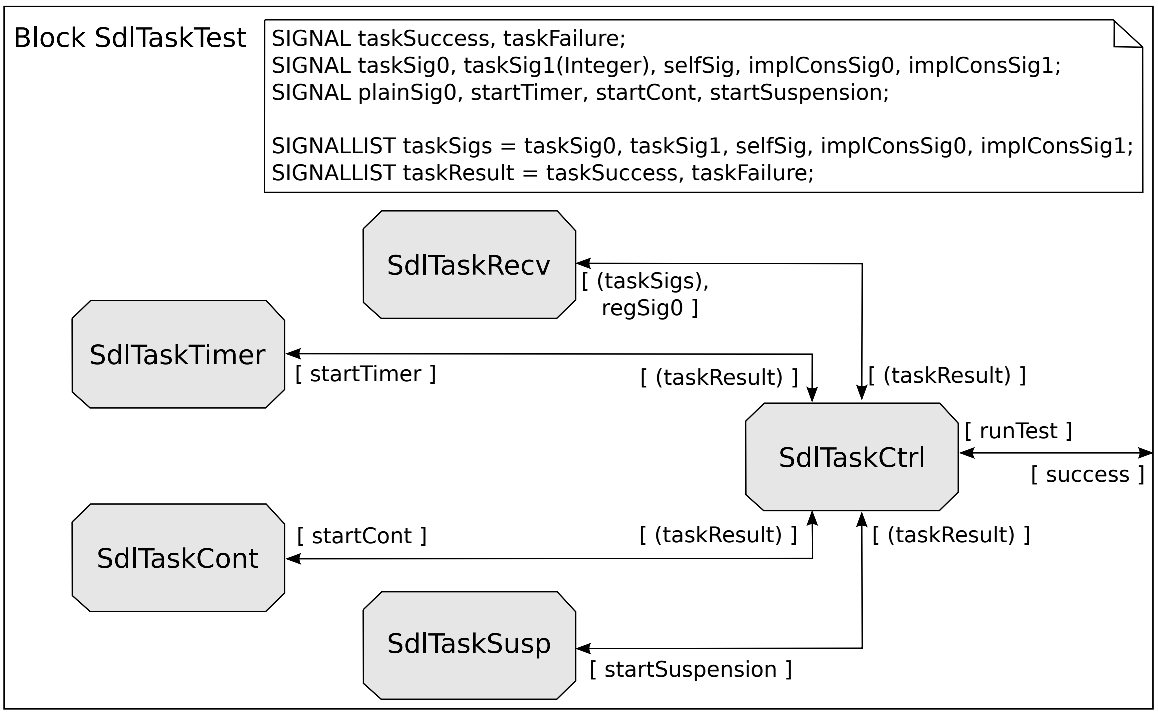

Block SdlTaskTest
SIGNAL taskSuccess, taskFailure;
SIGNAL taskSig0, taskSig1(Integer), selfSig, implConsSig0, implConsSig1;
SIGNAL plainSig0, startTimer, startCont, startSuspension;
SIGNALLIST taskSigs = taskSig0, taskSig1, selfSig, implConsSig0, implConsSig1;
SIGNALLIST taskResult = taskSuccess, taskFailure;
SdlTaskRecv
[(taskSigs), regSig0]
SdlTaskTimer
[startTimer]
[(taskResult)]
[(taskResult)]
SdlTaskCtrl
[runTest]
[success]
SdlTaskCont
[startCont]
[(taskResult)]
[(taskResult)]
SdlTaskSusp
[startSuspension]

Figure 10.1: Overview of SDL real-time task-related tests in the SDL test system.

To validate the implementation of SDL real-time tasks, the test system is extended with a new test category so that the system now consists of 16 test categories. The new tests for SDL real-time tasks are specified in a separate SDL block (see Fig. 10.1). Besides the SDL process `SdlTaskCtrl`, which controls the test of all subcategories, the block comprises four additional SDL processes testing particular subfunctionalities of real-time tasks. In detail, the following tests are performed:

- Processes `SdlTaskRecv` and `SdlTaskTimer` evaluate the implementation of task scheduling w.r.t. (timer-based) task signals. They particularly test, whether priorities of signals are realized correctly and whether the signal consumption order is consistent with the extended SDL semantics (see Appendix C). Furthermore, implicit consumptions and the correct realization of task attributes and task operators are validated.

- In SDL process `SdlTaskCont`, continuation of SDL real-time tasks is validated. Here, tasks are forked both implicitly, i.e. by continuing the currently running task, and explicitly by resuming a previous task by its task id. In addition, correct inheritance of task attributes and access to them by task operators is validated.

- SDL process `SdlTaskSusp` tests the implementation of SDL real-time tasks w.r.t. suspension. In this regard, all ways to suspend and resume tasks are considered; i.e., suspension by id, type, and priority. In this respect, it is also checked, whether suspended task signals are saved in the input port of the *SdlAgent* and prevented from deletion.

When running the test system, the execution of SDL real-time task-related tests is triggered by signal `runTest` that is generated by the overall test coordinator (not shown in Fig. 10.1).

Currently, the validation of SDL real-time tasks is the 15th test category and produces following console output:

```
  ------    -----   --   -            -----
 /  ___/   | ____| |  \ | |          |  ___|
 | |       | |__   |   \| |          | |__
 \___  \   |  __|  | |\   |          |  __|
  ___| |   | |___  | | \  |          | |
 /_____/DL |_____| |_|  \_|VIRONMENT |_|RAMEWORK

 Version 0.5

First test: 15.
Last test: 15.
---------------------------------------
15 - SdlTaskTest started
  .1 - SdlTaskTest - Receiver test successful
  .2 - SdlTaskTest - Timer test successful
  .3 - SdlTaskTest - Task forking test successful
  .4 - SdlTaskTest - Suspension test successful
15 - SdlTaskTest - Successful
All tests finished.
All testcases succeeded.
```

Listing 10.1: Output of the SDL test system regarding SDL real-time tasks.

In the console output, each subtest informs about the successful result. If an error would have occurred, further information would be provided by the responsible SDL process. After all subtests have reported their outcome, the controlling SDL process `SdlTaskCtrl` notifies the overall test coordinator of success/failure.

10.3 Performance Evaluation of SDL Real-time Tasks – Adaptive Cruise Control

Many real-time systems originate from the control system domain. To assess the improvement potential of SDL real-time tasks in such systems, evaluations comparing task scheduling with other state-of-the-practice scheduling strategies are conducted with a scenario from the automotive domain [CBG13]. The scenario is an Adaptive Cruise Control (ACC), whose objective is to retain a reference speed against disturbance variables. The reference speed is prescribed by the driver, and disturbance variables are, for instance, aerodynamic drag or incline of the road. In this context, *adaptive* means that not only the difference between actual speed and reference speed is considered but also the distance to and speed of objects in front of the controlled car. To avoid collisions, a minimal distance to these objects has to be kept by reducing the speed of the car or by initiating an emergency braking.

10.3.1 Evaluation Setup

The scenario is a networked system, which is shown in Fig. 10.2 and consists of six nodes communicate via a CAN [Int04] bus. The SDL system under consideration hosts the control algorithm, which is a Proportional-Integral-Derivative (PID) controller, and is deployed on the node that is referred to as `ACC` in the figure. Amongst others, the `ACC` node receives three types of sensor values: Reference speed (sent by node `refSpeedSensor`), actual speed (`actSpeedSensor`), and distance of and speed to other objects (`RadarSensor`). Based on the received sensor values, the ACC algorithm calculates new control values and sends them to node `Engine`, which adjusts the engine power, or to node `Brake` to start braking.

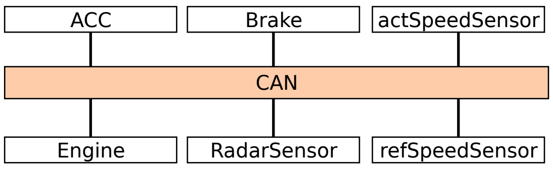


Figure 10.2: Network topology of the evaluated ACC scenario.

Motivated by this scenario, the evaluation of SDL real-time tasks now focuses on node `ACC`. In particular, not the entire networked system is built but only the `ACC` system is specified in SDL and installed on an Imote 2 node. Receptions of sensor values are emulated by the environment of the system. Though we do not run a real CAN bus and do not evaluate characteristics of the communication medium, some properties like minimal interarrival times of CAN messages are taken into account. A positive side effect of running the SDL system as stand-alone system is that evaluation results are not distorted by communication errors. However, if the objective is testing the entire networked system and not evaluating SDL real-time tasks locally, a more comprehensive validation strategy would be required.

The communication schedule of the `ACC` node is shown in Fig. 10.3 schematically. Its basic cycle is dictated by the periodical calculation of new control values (every 20 ms) that are afterwards sent to the engine via the environment of the SDL system. To achieve a high quality of control, sensor values used in the calculation should be very up-to-date and control values should be communicated as fast as possible. This, particularly, requires that all sensor values

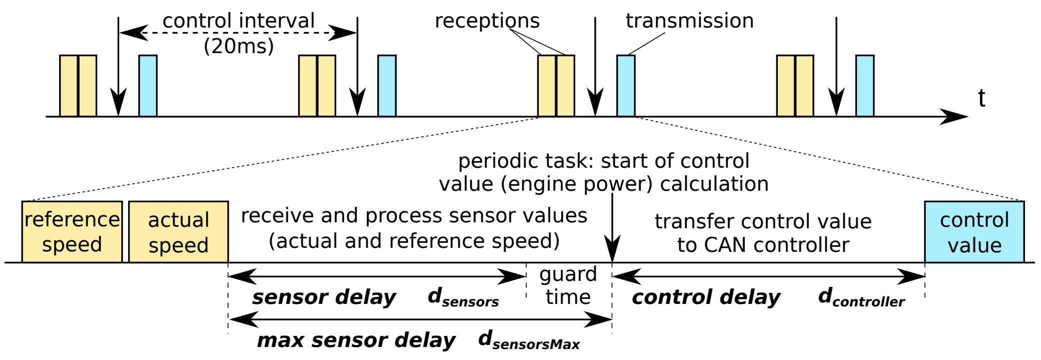

control interval
(20ms)
receptions
transmission
t
periodic task: start of control value (engine power) calculation
reference speed
actual speed
receive and process sensor values (actual and reference speed)
transfer control value to CAN controller
control value
sensor delay $d_{sensors}$
guard time
control delay $d_{controller}$
max sensor delay $d_{sensorsMax}$

Figure 10.3: Periodic message schedule of the `ACC` node (without sporadic messages from/to `RadarSensor`/`Brake`).

are received in time just before control value calculation starts. In more detail, it is crucial to keep the following two delays small:

- The *sensor delay* ($d_{sensors}$) is the time required to process received sensor values and to forward them on the `ACC` node from the SDL environment to the responsible SDL process. In this regard, communication delays to transfer sensor values to the `ACC` node are not considered, but time measurement starts after the last sensor value is available in the SDL environment of `ACC`. In general, sensor delays are variable. However, to guarantee the timely processing of all sensor values, an upper bound $d_{sensorsMax}$ must be found and used to dimension the message schedule. At best, $d_{sensors}$ is very low and without any jitter in order to derive a low and accurate value of $d_{sensorsMax}$.

- The time to calculate a control value and to transfer this value from the SDL process hosting the PID algorithm to the SDL environment is called *control delay* ($d_{controller}$). The faster the control value is sent to the environment – and consequently to the engine –, the faster is the car's adaption to the current situation. Additional delays introduced by the communication system to arbitrate the medium and to transfer the control values over the medium are again ignored for the evaluation.

In addition to the messages shown in the schedule, radar messages are received by the `ACC` node. Their reception is also emulated in the SDL environment and occurs sporadically. They are sent into the SDL system, and can – depending on the reported distances – cause event-triggered reactions with brake messages. The time between receiving a radar message and the response with a brake message is referred to as reaction delay $d_{reaction}$ in the following. In addition to radar and brake messages, sporadic load messages are introduced, which are sent into the SDL system to evaluate the impact of low priority background load to sensor, control, and reaction delays. They also provoke response messages, which inform about the end of load processing.

The SDL system running the control algorithm is presented in Fig. 10.4. The structure is typical for many systems, because it contains concurrent applications with different priorities, which share transitions in lower layers. In this case, the CAN block is used by all applications to fulfill their tasks. In more details, the SDL system comprises the following blocks:

- Block `CAN` is responsible for the en- and decoding of data from CAN messages into SDL data types (process `ConcatCoder`) and the conversion between CAN identifiers and internal message identifiers (process `CANMac`).

- The SDL blocks `Speed` and `Distance` realize the adaptive cruise control, where `Speed` hosts the actual PID controller and `Distance` provides correction values based on received radar messages.

- The `Load` block processes random background load. To determine the impact of load on the evaluated delays, the system is stressed with different load situations. Because load is generated by external messages that are received via block `CAN`, the workload of the system can be increased by increasing the average frequency of these external messages. Thus, the same SDL specification is used for various load situations.

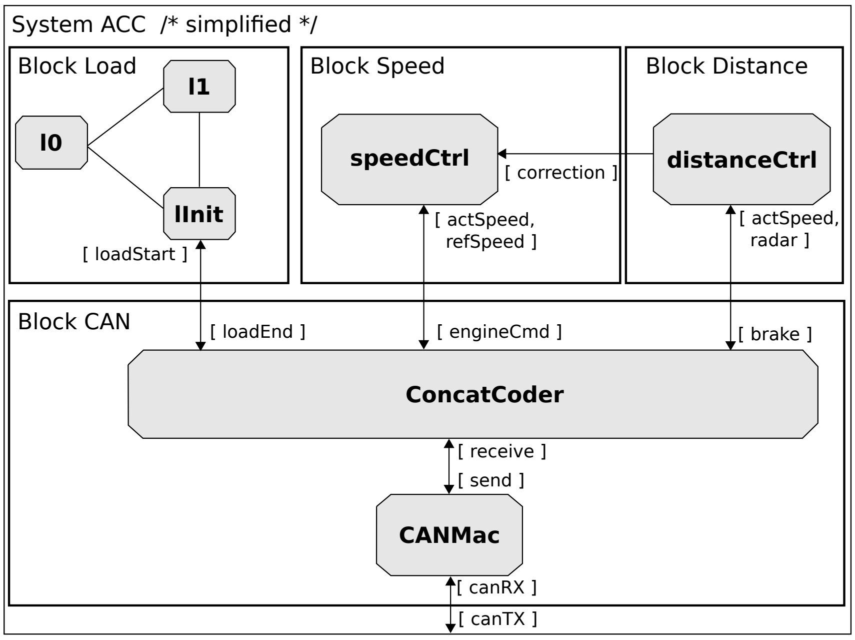


Figure 10.4: Evaluated SDL system of the `ACC` node.

To compare task scheduling with common other scheduling algorithms, a set of experiments has been conducted, in which the SDL system is executed with four different strategies of SdlRE (see Table 9.1 for their description): FCFS$_{signals}$, FCFS$_{agents}$, Priorities$_{agents}$, and Priorities$_{tasks}$. The agent-based round robin strategy (RR$_{agents}$) is not considered, because it is too inefficient by design. The first two strategies, FCFS$_{signals}$ and FCFS$_{agents}$, work with a FIFO queue and do not require further configuration.

For (process) priority scheduling (Priorities$_{agents}$), SDL process priorities have been assigned as presented in Table 10.1. In general, agents responsible for control value calculation obtain a higher priority (i.e., a lower priority value) than load processing agents. The environment runs with highest priority.

SDL process	priority
CANMac, ConcatCoder	3
speedCtrl	2
distanceCtrl	1
lInit, l0, l1	4
environment	0

Table 10.1: Configuration of priorities used by Priorities$_{agents}$.

To configure task scheduling, system tasks have been identified and realized as corresponding types of SDL real-time tasks. They are outlined together with affected signals and priorities in Table 10.2. The table also presents the source and destination of a real-time task, i.e., the processes, in which an SDL real-time task is generated and terminates, respectively. Here, italic names refer to remote systems, thereby pointing out to distributed tasks that would require communication via CAN bus if the entire networked system and transition executions of other nodes would be evaluated. Priorities of task signals are assigned, so that background load is disadvantaged. The possibility to suspend real-time tasks is not utilized in this evaluation.

To exemplify the usage of SDL real-time tasks in this scenario, Fig. 10.5 presents parts of the `speedCtrl` process specification, which provides the PID controller. The figure shows the creation of two SDL real-time tasks: First, a time-triggered and non-terminating real-time task

task type	**source**	**destination**	**priority**	**task signals**
REF_SPEED	*refSpeedSensor*	speedCtrl	4	canRX, receive, refSpeed
ACT_SPEED	*actSpeedSensor*	ConcatCoder	4	canRX, receive
	ConcatCoder	speedCtrl	4	actSpeed
	ConcatCoder	distanceCtrl	5	actSpeed
CTRL_VALUE	speedCtrl	speedCtrl	3	controlTimer
ENGINE_REG	speedCtrl	*Engine*	3	engineCmd, send, canTX
RADAR	*RadarSensor*	distanceCtrl	2	canRX, receive, radar
BRAKE	distanceCtrl	*Brake*	1	brake, send, canTX
	distanceCtrl	speedCtrl	4	correction
LOAD	*loadSimulator*	*loadSimulator*	8	canRX, receive, loadStart loadEnd, send, canTX

Table 10.2: Types and configuration of SDL real-time tasks in the ACC scenario.

of type CTRL_VALUE, which is responsible for periodic control value calculation and triggered by the timer-based task signal `controlTimer`. And second, a real-time task of type ENGINE_REG, which propagates the new control value and is first associated with task signal `engineCmd`. The additional three transitions shown in the figure are also executed in the context of a real-time task; namely, in tasks of type BRAKE, ACT_SPEED, and REF_SPEED. Since they are the last transitions of these tasks, no task-related actions are specified in their bodies.

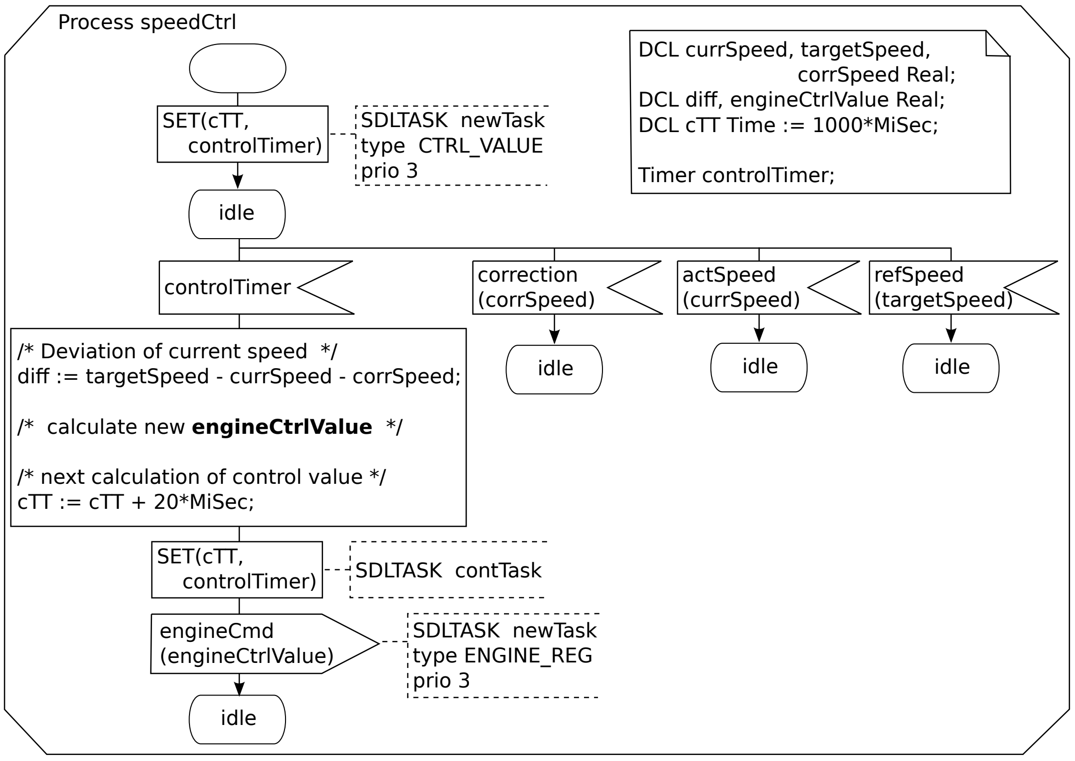

Process speedCtrl
SET(cTT, controlTimer)
SDLTASK newTask type CTRL_VALUE prio 3
DCL currSpeed, targetSpeed, corrSpeed Real;
DCL diff, engineCtrlValue Real;
DCL cTT Time := 1000*MiSec;
Timer controlTimer;
idle
controlTimer
correction (corrSpeed)
actSpeed (currSpeed)
refSpeed (targetSpeed)
idle
idle
idle
/* Deviation of current speed */
diff := targetSpeed - currSpeed - corrSpeed;
/* calculate new engineCtrlValue */
/* next calculation of control value */
cTT := cTT + 20*MiSec;
SET(cTT, controlTimer)
SDLTASK contTask
engineCmd (engineCtrlValue)
SDLTASK newTask type ENGINE_REG prio 3
idle

Figure 10.5: Creation of SDL real-time tasks in process `speedCtrl`.

10.3.2 Evaluation Results

The following evaluation assesses three types of delay: Sensor delay $d_{sensors}$, control delay $d_{controller}$, and reaction delay $d_{reaction}$. In addition, the overhead of the runtime system is determined. The outcome of the evaluation runs are first stored in the memory of the Imote 2 and transferred to a PC via a serial interface after the end of the run.

10.3.2.1 Sensor and Control Delays

In the first series of experiments, $d_{sensors}$ and $d_{controller}$ are determined for 25 different load situations. The load situations range from no additional load up to approximately 80% additional load[2], which brings – together with the regular processing of sensor and control values – the system almost at its limit. Each load situation is repeated five times. In each repetition, 200 sensor values are received and 100 new control values are calculated and sent to the environment. Thus, the first series consists of $5 \cdot 25$ runs for each scheduling strategy. Radar messages are not used in this series. Hence, the system does not generate brake messages as well.

Sensor Delays Figure 10.6 presents results for $d_{sensors}$ with three different load situations. In each plot, the cumulative distribution functions of consumed sensor values, which are carried by SDL signals `actSpeed` and `refSpeed`, are plotted as a function of delay for each scheduling strategy. Points on each line mark the maximal delay after which the latest sensor values are consumed in process `speedCtrl`.[3] This delay has to be considered when determining $d_{sensorsMax}$.

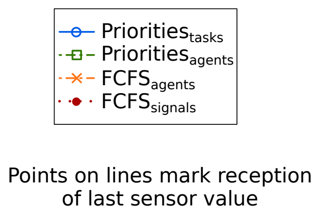
In case of no additional load (Fig. 10.6(b)), there are only very small differences and all schedulers achieve almost identical maximal delays of about 1.7 ms. When looking in more detail, Priorities$_{tasks}$ is the fastest scheduler and delivers the latest signal about 100 µs faster than the slowest scheduler Priorities$_{agents}$. These differences are results of different overhead and a serialization delay, which is caused by process `ConcatCoder` due to the forwarding of signal `actSpeed` to processes `speedCtrl` and `distanceCtrl`.

When adding load to the system, results of the schedulers start to differ. In case of 40% additional load (Fig. 10.6(c)), the maximal sensor delay achieved by Priorities$_{tasks}$ increases to 1.83 ms, whereas the worst scheduler is now FCFS$_{signals}$ and requires 2.59 ms to deliver the latest sensor value. Though all strategies suffer from the background load that constricts the prioritized processing of sensor values, scheduling strategies different than task scheduling are more affected due to inadequate transition execution orders. The reason why sensor delays also increase with Priorities$_{tasks}$ is found in the lack of full preemption and the missing utilization of task suspension. Thus, a transition executed to process background load must first be finished before a transition that becomes fireable in the meantime can start. Another reason for the increase of delays, which takes effect to all scheduling strategies and which must not be neglected, is the higher rate of software and hardware cache misses due to the execution of additional transitions. Their concrete quantitative impact is, however, hard to determine.

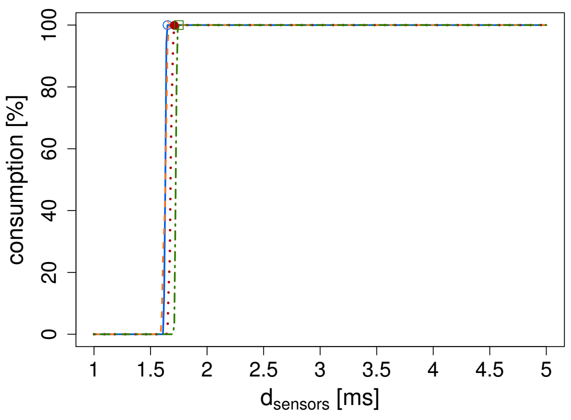
Comparing task scheduling with other schedulers, the increase of delays is much lower. This, particularly, also holds when comparing with (process) priority scheduling, which is a common

[2]The load is given as proportional CPU time.

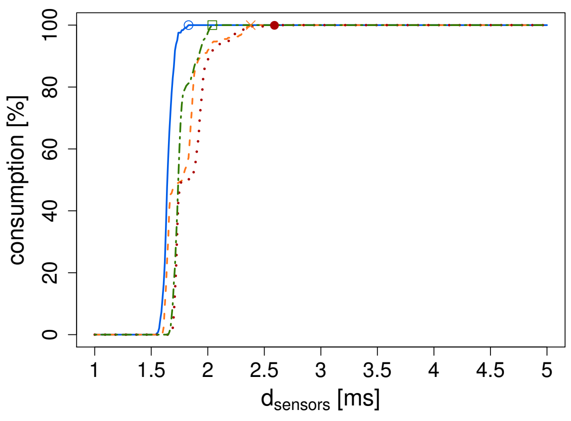
[3]Note that these values are empirical values and no worst-case bounds. They are often also referred to as *maximal observed execution times* [WEE+08] and are not sufficient for hard real-time systems.



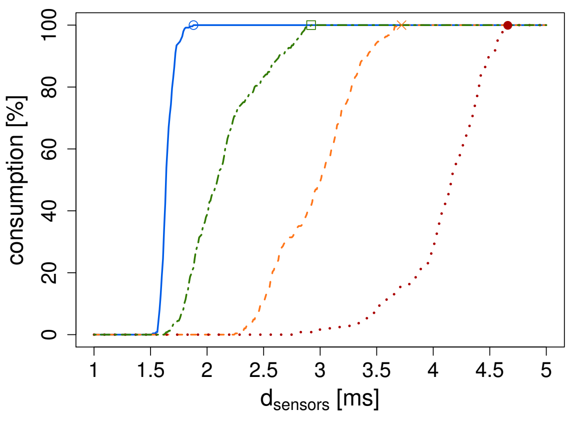
(a) Plot legend



(b) No additional load



(c) 40% additional load



(d) 80% additional load

Figure 10.6: Sensor delays $d_{sensors}$ with three different load situations.

scheduling strategy for SDL implementations and supported by many SDL tools. It is the second best strategy for this load situation with a maximal sensor delay of 2.04 ms. Compared to delays without load, this corresponds to an increase of about 0.3 ms, whereas task scheduling's increase is only about 0.15 ms. The worse results of priority scheduling can be traced back to transitions in block `CAN` that are executed both in the context of load processing and to forward received sensor values. With process priorities alone, this cannot be distinguished and transition executions are (wrongly) prioritized in both cases.

With 80% additional load (Fig. 10.6(d)), the difference becomes even more observable. In this situation, the maximal sensor delay with task scheduling is only 1.88 ms, i.e., the strategy is almost insusceptible to the additional load. Since the second best strategy, which is priority scheduling again, has a maximal sensor delay of 2.92 ms, task scheduling requires less than 65% of the time to bring sensor values to their destination SDL process. A further positive effect of task scheduling is the very low jitter, which is only 370 µs, whereas the jitter with the second best scheduler is 1300 µs . To sum up, sensor delays and their jitter are much lower with task scheduling and the upper bound $d_{sensorsMax}$ can be determined much more accurately. Consequently, the message schedule (see Fig. 10.3) is best realized with task scheduling, thereby achieving the best quality of control.

Control Delays Figure 10.7 presents evaluation results regarding control delays by means of bar plots. The control delay $d_{controller}$ is the time between nominal start of control value calculation in process `speedCtrl` and the reception of the control value by the SDL environment. It should be as small as possible to enable a fast adaption of the engine to the current driving situation. The figure shows four load situations, ranging from no additional load up to 80% background load, where each bar depicts the average, minimal, and maximal control delay.

Without additional load, which is shown in the left part of Fig. 10.7, control values are very similar and approximately 1.1 ms. When load is added to the system, the delays increase similar to the sensor delays before. At 10% additional load, the rise is, however, in the main only observable for maximal delays. In general, these delays increase most for both FCFS strategies and least for task scheduling. The reason for the small rise of the maximal delay with task scheduling of about 50 µs is again the lack of full preemption and suspension.

Comparing results with no load and 10% additional load, a very interesting phenomenon is the reduced best case delay for almost all schedulers. This effect is caused by the absence of *wake-up delays*, when the SDL timer triggering the control value calculation expires and the system just finishes a transition execution of background load.

Increasing the load of the system leads to further deviations between all scheduling strategies. For both FCFS strategies, average as well as maximal delay increase significantly with rising load. Process priority scheduling and task scheduling, on the contrary, are less prone to load and converge almost. However, already with very low background load, task scheduling achieves better control delays than priority scheduling. At 80% additional load, the maximal reaction delay with task scheduling is about 1.2 ms, whereas priority scheduling requires in the worst-case 400 µs longer to transfer control values to the environment.

In summary, the results show that fair strategies are inadequate in systems with time-critical tasks and background load. They also provide evidence that task scheduling is least prone to background load and performs best in terms of amount and variability of control delays. Though process priority scheduling achieves still good control delays in the presence of heavy load, maximal delays with task scheduling are 25% less.

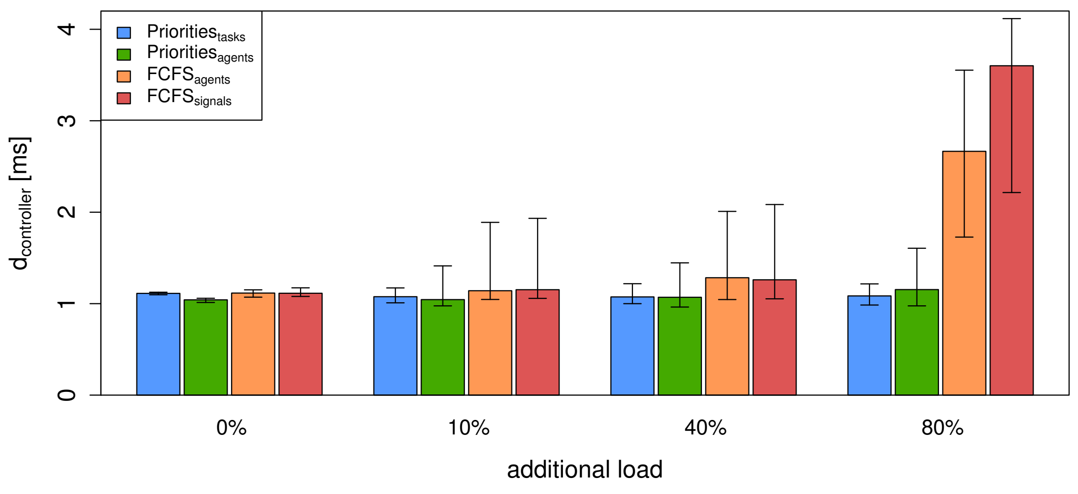


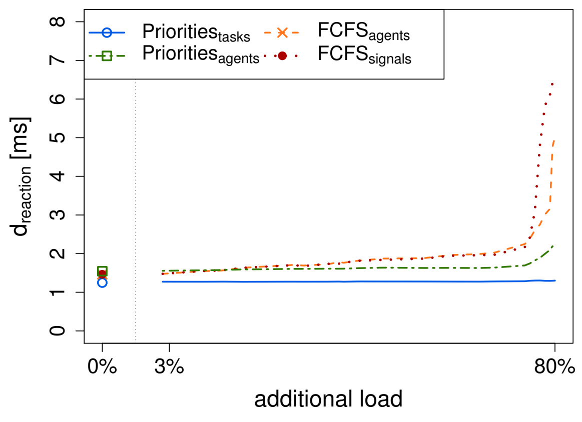
Figure 10.7: Control delays $d_{controller}$ as function of four different load situations.

10.3.2.2 Reaction Delay

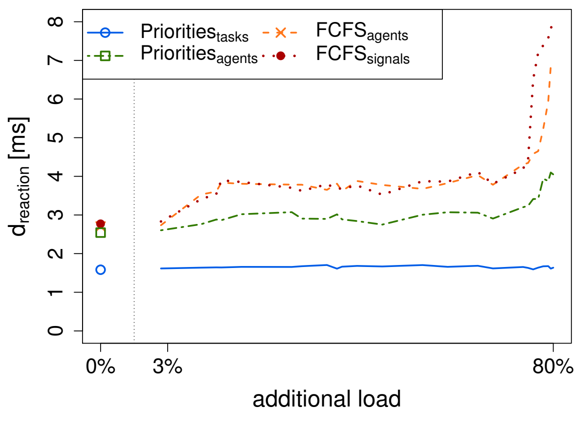
In a second series of experiments with 200 runs, the focus is shifted to sporadic system tasks. In each run, 50 radar messages are generated in the environment of the SDL system in addition to regular sensor values. They are sent sporadically with a minimal interarrival time of 35 ms and require a fast response with brake commands. Thus, their processing is time-critical. To evaluate these reaction delays $d_{reaction}$, the time between the generation of a radar message and the response with the corresponding brake command is monitored. As in the first series of experiments, the 200 runs are split into 25 different load situations.

Since radar and brake messages are – different from sensor and control values – sporadic events, they are harder to realize in message schedules. When relying on reserved transmission slots, pessimistic assumptions must be made on the intervals of the events to provide a sufficient number of transfer possibilities. In the following, challenges regarding communication are not considered, but we allow the arrival of radar messages at any time.

Results of the experiment runs are summarized in Fig. 10.8. The left subfigure presents average values of the reaction delay $d_{reaction}$, whereas the second plot provides maximal reaction delays. In both subfigures, delays are plotted as a function of all load situations. Without additional load, average reaction delays are almost equal for each scheduling strategy, yet the maximal delays already differ significantly. In more detail, the maximal reaction delay with task scheduling is 1.58 ms, whereas the second best scheduler (priority scheduling) has a maximal delay of 2.54 ms. The source of this difference is the regular processing of sensor and control values that constrains the prioritized processing of radar messages and brake commands except for task scheduling. A further difference between task scheduling and the other scheduling strategies is the load independence of reaction delays. This holds for average as well as maximal delays. For instance, the maximal delay at the heaviest load situation is 1.64 ms with task scheduling, whereas priority scheduling achieves 4.05 ms only. To sum up, both average and maximal reaction delays are small and almost constant with task scheduling, thereby improving the empirical determination of an upper bound on $d_{reaction}$.



(a) average delays



(b) maximal delays

Figure 10.8: Reaction delays $d_{reaction}$: Time between reception of radar message and transmission of brake command.

10.3.2.3 Runtime Overhead

In general, an expected drawback of all measures to decrease delays and jitter is the increase of overhead. This indeed does not hold for pure (code) optimization techniques like loop unrolling [ALSU07] but for most privileging OS scheduling strategies, where, for instance, the maintenance of a priority-sorted queue is more expensive than of a FIFO queue.[4] Additional overhead is also generated by sophisticated strategies with preemption due to the costs of context switches [Kop97], and to apply countermeasures against priority inversion with priority inheritance and priority ceiling protocols [SRL90]. To quantify the price of task scheduling, the overhead of the scheduler is determined in the evaluation scenario and compared to the other scheduling algorithms. The used system is executed without radar/brake messages and without background load. For each scheduling algorithm, 10,000 samples are monitored.

The workloads are shown in Fig. 10.9 for each scheduler and subdivided by their type. Note that the results contain overhead as well as regular workload, i.e., not only times spent in the runtime environment are included but also pure transition execution times. In particular, following types of workload are distinguished:

- *agent selection* describes the time the scheduler requires to determine the next executable *SdlAgent*. In case of task scheduling and signal-based FCFS, this workload also contains the time to select the concrete transition that is executed next.
- *transition selection* comprises the time to find the next executable transition after agent selection is complete. For signal-based FCFS and task scheduling, this step is entirely skipped, since transition selection is completed during *agent selection*.[5]
- *transition execution* is the complete workload to execute transitions. This includes SDL statements like assignments to variables as well as timer actions and signal outputs.

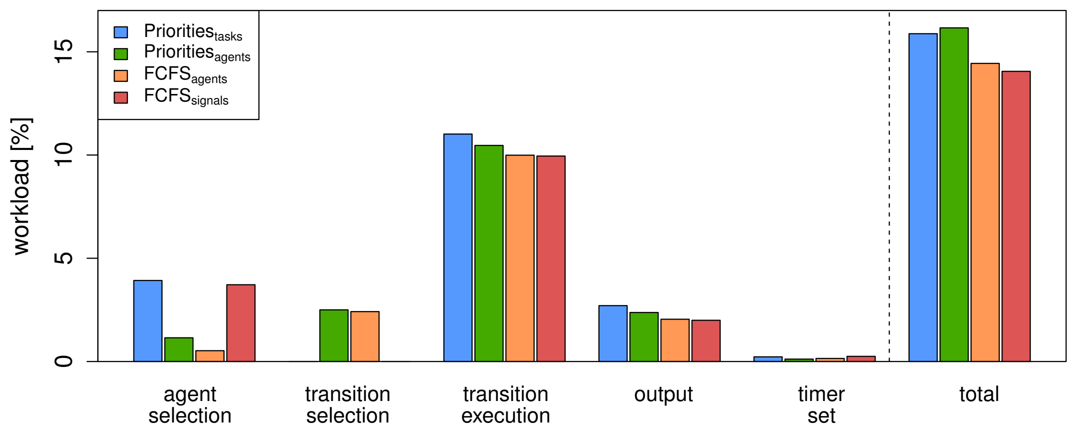


Figure 10.9: CPU utilization as function of workload type and scheduling strategy.

[4]Ignoring possible memory reallocations, complexity of inserting into a FIFO queue is O(1), whereas efficient implementations of a priority-sorted queue require O(log(n)) [Str00].

[5]In the given system, task scheduling is only used together with task signals. If there are other transition triggers like continuous signals or non-task signals, task scheduling would also spend some time in *transition selection*.

- *output* is the proportion of workload of *transition executions* that is spent in signal outputs.
- *timer set* comprises the time to manage SDL timers and is also included in *transition execution*.
- The *total* workload represents the entire time the CPU is occupied. Note that this also includes times that are not part of the delays above. Thus, *total* cannot be considered as summation of the other workload types.

Regarding *agent selection*, the workload with task scheduling and signal-based FCFS is higher than with agent-based FCFS and priority scheduling. The reason for this is the early selection of transitions, which costs time and is postponed to *transition selection* with both agent-based schedulers. Consequently, loads for *transition selections* are higher for both agent-based schedulers (about 2.5%) but 0% for both signal-based schedulers. The time spent for *transition executions* are more similar and have the main impact on the overall load. The cause of Priorities$_{agents}$ and Priorities$_{tasks}$ small increase are more expensive signal outputs, since priority-sorted queues cause more overhead than FIFO queues. In particular, workload due to signal outputs is 2.7% with task scheduling and 2.4% with priority scheduling, whereas it is about 2% with FCFS$_{agents}$ and FCFS$_{signals}$. Regarding timers, the workloads are similar and very low, because there are only few activations of SDL timers in the system.

In total, task scheduling is the second most expensive strategy and occupies the CPU for 15.8%. Only (process) priority scheduling utilizes the CPU more (16.1%). Note that with priority scheduling, each *SdlAgent* must check after each transition execution, whether it has still the highest priority among all executable *SdlAgents* and that these times are included in *total* only. Thus, priority scheduling generates the highest *total* workload, though other workload types let one to assume differently. With the given system stimulus, the most efficient scheduling strategy is signal-based FCFS, which generates a total workload of 14.1%. This may seem surprising at first glance as agent-based FCFS is introduced as an optimization of signal-based FCFS, but is explainable by the low system utilization. In particular, input ports of agents are rarely occupied by more than one signal, thereby making the optimizations of the agent-based FCFS strategy useless. In summary, the overhead of task scheduling is 1.7% higher than the most efficient strategy, which is the price for improvements w.r.t. amount and jitter of delays.

10.4 Performance Evaluation of SDL Real-time Tasks – The Inverted Pendulum

In this section, evaluation series are presented, which have been conducted with FERAL and HiL simulations (see Sect. 9.6). The composed scenario is from the control engineering domain and a popular example of a control loop. In its focus is an inverted pendulum, which consists of a rod with a mass and has a pivot point mounted on a cart. Target of the system is to balance the rod of the pendulum vertically with its mass pointing upwards and to preclude it from toppling by moving the cart horizontally. In the previous project "DFG-Schwerpunktprogramm 1305", this scenario has been realized by a distributed system, in which sensor, controller, and actuator nodes communicate wirelessly and via a service-oriented middleware [Krä13b]. This section now presents the incorporation of SDL real-time tasks into an SDL specification of this existing solution and evaluates differences in communication delays and control quality between task scheduling and other state-of-the-practice scheduling strategies.

10.4.1 Evaluation Setup

While the objective of the original project was the control of a real hardware pendulum, the following paragraphs present a simulative approach, where the inverted pendulum is simulated by Matlab Simulink [Kam13].[6] Similar to the real hardware pendulum, the simulated model provides sensor values in terms of speed/position of the cart, and the rod's angular deflection and velocity. Furthermore, it offers an interface to adjust the cart's movement by an actuator.

The complete simulated network topology is illustrated in Fig. 10.10 and comprises – in addition to the FSC of the inverted pendulum – SDL FSCs for a sensor, actuator, and controller node.

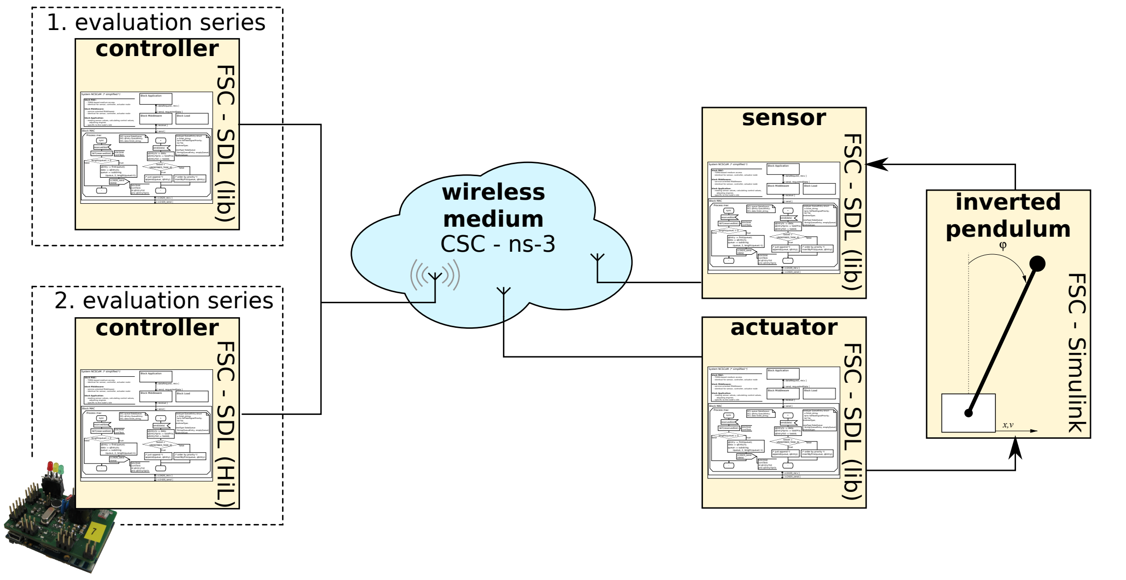


Figure 10.10: Simulated network topology of the inverted pendulum scenario.

[6]The model has been kindly provided by the working group *Automatisierungstechnik* (http://www.eit.uni-kl.de/atplus/).

Furthermore, it includes a CSC with ns-3, simulating CC 2420-based wireless communication among SDL FSCs. FSCs of sensor and actuator node are connected to the FSC of the pendulum directly and realized with FERAL's library-based SDL integration (see Sect. 9.6.1). Thus, they are simulated without consideration of hardware execution times. The focus of the evaluation is on the controller, which hosts a PID controller, whose input is received from the sensor node and whose output is sent to the actuator via the ns-3 CSC. The controller is realized with both SDL integration variants of FERAL: With the library-based integration to perform pure functional evaluations of the considered SDL system (Sect. 10.4.2.1), and with FERAL's support of HiL simulations to conduct performance evaluations on an Imote 2 platform (Sect. 10.4.2.2). In both evaluation series, all FSCs are simulated with FERAL's time-triggered execution semantics and time slices of 10 µs.

To ensure reliable communication, all messages among controller, sensor, and actuator are sent in pre-configured time slots. The concrete slot configuration is shown in Fig. 10.11 and starts with a synchronization slot, in which beacons are sent by the controller. Synchronization slot as well as subsequent transmission slots have an identical duration and are placed equidistantly, such that each node owns one slot every 30 ms.[7] They are used both for management data of the middleware like service announcements and alive messages, and for sensor and control values, which are generated in intervals of 60 ms.

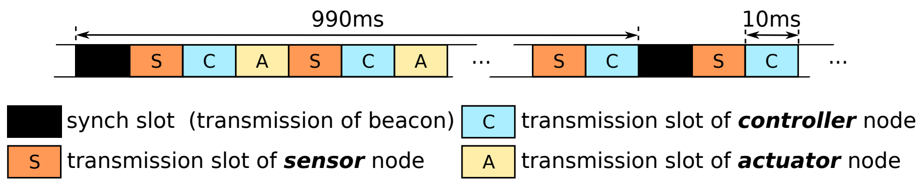

990ms
10ms
S
C
A
S
C
A
...
S
C
S
C
...
synch slot (transmission of beacon)
C transmission slot of ***controller*** node
S transmission slot of ***sensor*** node
A transmission slot of ***actuator*** node

Figure 10.11: Slot configuration of the inverted pendulum scenario.

The communication system realizing this scenario has been introduced under the name *Networked Control System - Communication Middleware* (NCS-CoM) [Krä13b]. Its SDL specification is presented in Fig. 10.12 by a simplified excerpt and has been extended with information about SDL real-time tasks. Though some minor flaws in the specification were fixed in the course of the evaluations – e.g., regarding reliability and accuracy of the beacon-based synchronization –, changes primarily retain the original behavior. Blocks `Middleware` and `MAC` are identical on all nodes; node-specific functionality is only specified in block `Application`, where – depending on the concrete node – either sensor values are read, control values calculated, or the (simulated) engine triggered. Block `Load` is newly introduced to allow an assessment of the system's behavior under load. For block `MAC`, two example SDL transitions are presented to demonstrate the annotation-based incorporation of SDL real-time tasks. The right-hand transition illustrates the queueing of new transmission orders, where task attributes of the consumed signal are stored and transmission orders are sorted by task signal priorities. If the consumed signal is no task signal, the transmission order is appended to the queue.[8] On the left-hand side, the processing of the queue during reserved time slots is presented, where a frame is sent via the simulated CC 2420 interface and continues the previous task by an explicit task forking.

[7]The only exception is the actuator node, which has lower communication demands and cedes one of its slots to the controller for synchronization purpose.

[8]This is also true if the transition is executed by a scheduler different from task scheduling.

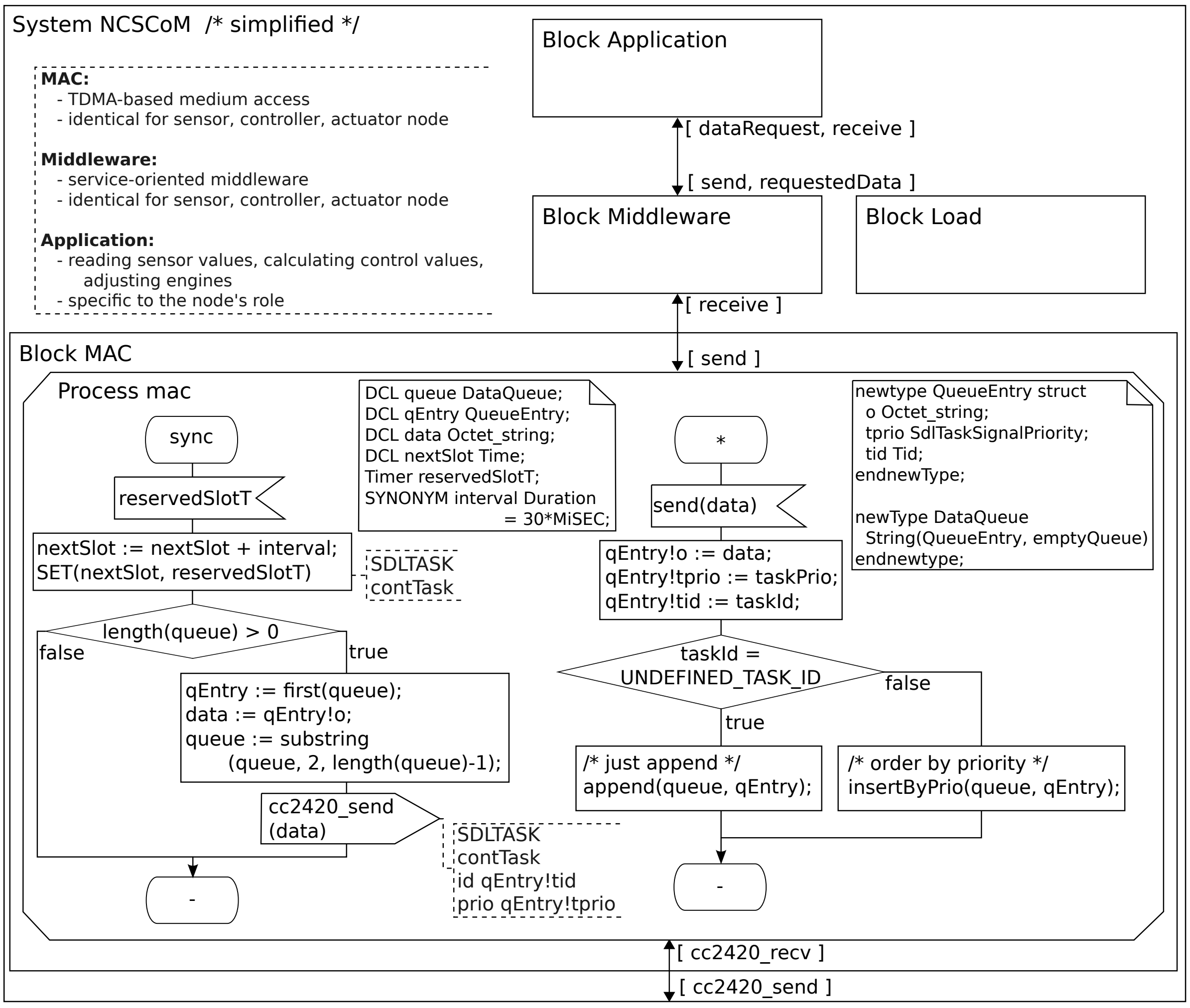


Figure 10.12: Excerpt of the SDL specification of the controller node (based on [Krä13b]).

SDL block	**priority**
MAC	0
Middleware	1
Application	2
Load	5
environment	0

Table 10.3: Block priorities.

Based on the existing SDL specification, types of SDL real-time tasks are identified as shown in Table 10.4. Since task signals of the same task always have the same priority, the table includes one priority value only, which is assigned to all task signals of the particular task type. With the exception of tasks of type LOAD, all tasks are distributed involving all SDL FSCs. In this regard, task attributes are communicated implicitly by the (virtual) driver of the CC 2420 transceiver by appending/retrieving them to/from outgoing/incoming CC 2420 frames. In addition to task scheduling (Priorities$_{tasks}$), the SDL system is executed with agent-based FCFS (FCFS$_{agents}$) and process priority scheduling (Priorities$_{agents}$) to enable comparative assessments. For Priorities$_{agents}$, priorities are assigned as defined by Table 10.3, where processes inherit the priority of the comprising block.

task type	description	priority
MACFCT	MAC layer-specific functionality, including transmission/reception of synchronization beacons and processing of reserved transmission slots.	1
CONTROL	Adjusting the engine with a new control value; including: (1) reading of new sensor values at sensor node (2) calculating new control value at controller node (3) driving engine with new control value at actuator node.	2
MIDMGT	Processing of the middleware's management traffic (subscriptions, announcements, . . .).	3
LOAD	Execution of background load.	5

Table 10.4: Types of SDL real-time tasks in the inverted pendulum scenario.

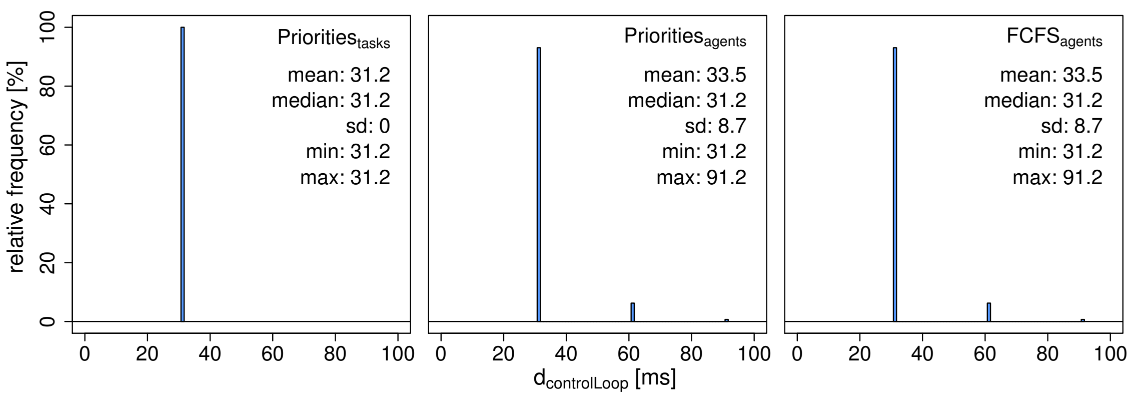
10.4.2 Evaluation Results

The evaluation of this scenario focuses on the controller node and is split into two series: First, differences of the schedulers are compared on design level in Sect. 10.4.2.1. For this purpose, the controller is simulated by a library-based FSC, thereby not taking transition execution times into account. Afterwards, comparative performance evaluations are presented in Sect. 10.4.2.2 with the controller running on HiL. The pendulum's initial state is in both cases identical and given by a standing cart, (yet) without toppling movement of the rod, and an initial angular deflection of 4°, where 0° corresponds to vertically upwards. Since the focus is not on the network's initialization but on the behavior of the network's steady state, delays to discover and subscribe services are not considered but accrue before the actual measurements start. Starting from this state, the system is simulated for eight seconds.

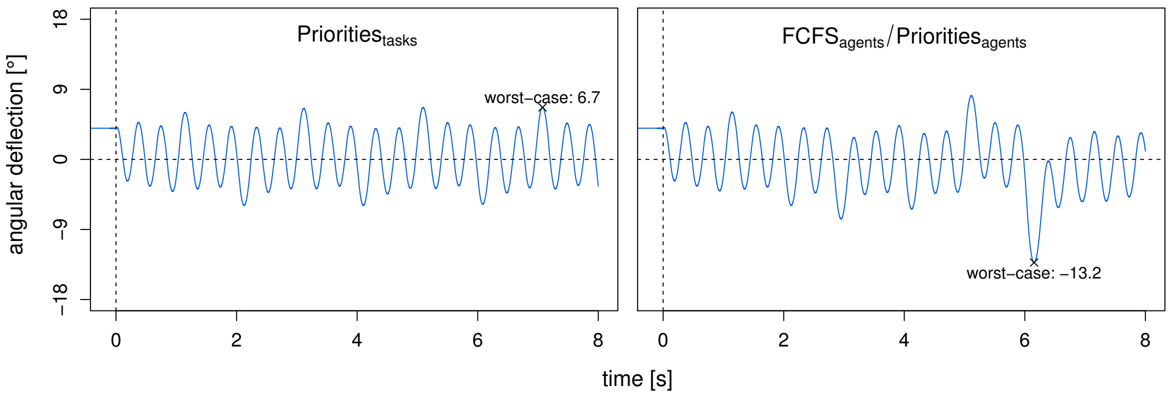
10.4.2.1 Differences on Design Level

In the following, we begin with delays of tasks of type CONTROL, i.e., how long does it take until a new sensor value influences the movement of the cart. We refer to these delays as *control loop delays* ($d_{controlLoop}$), which are – together with the fixed control interval of 60 ms – crucial w.r.t. quality of control. Though execution times of transitions are not considered here, schedulers may still differ in the order of transition executions. This, in turn, can influence the order of frame transmissions, thereby having an effect on communication delays and $d_{controlLoop}$.

In Fig. 10.13(a), $d_{controlLoop}$ is plotted by histograms for all three considered scheduling strategies. For Priorities$_{tasks}$, delays accumulate at 31.2 ms and indicate that both sensor node and controller send their sensor/control values in their respectively next reserved transmission slot. With Priorities$_{agents}$, the most frequent control loop delay is also 31.2 ms. With a frequency of 6%, however, delays are 61.2 ms; and even 91.2 ms with 1% frequency. Similar results can be observed with FCFS$_{agents}$, since frame transmission orders are identical to the order of Priorities$_{agents}$. The impact of the differing control loop delays on control quality is illustrated in Fig. 10.13(b), where the rod's angular deflection is plotted over simulation time. Though all schedulers prevent the rod from toppling, general as well as worst-case deflections are larger



(a) Control loop delays.



(b) Quality of control.

Figure 10.13: Comparison between scheduling strategies on design level without consideration of transition execution times.

with FCFS$_{agents}$ / Priorities$_{agents}$. This temporal extract is indeed inadequate as formal proof for an (in)sufficient control quality, but demonstrates that already few outliers of $d_{controlLoop}$ of about 7% have a serious impact on the rod's stability and should therefore be avoided.

Actually, the presented differences are not caused by differing transition execution orders but by an unequal queuing order of transmissions, where FCFS$_{agents}$ and Priorities$_{agents}$ may prefer (less important) management traffic over sensor/control values. Though an optimized transmission order could also be achieved with these two schedulers by introducing an explicit priority model for transmissions, the utilization of task signal priorities demonstrates the capabilities of SDL real-time tasks to multiplex heterogeneous data flows with implicitly available information about the urgency of a signal. Thus, SDL real-time tasks cope with one priority model on both design and implementation level.

10.4.2.2 Performance Evaluations of the Controller Node

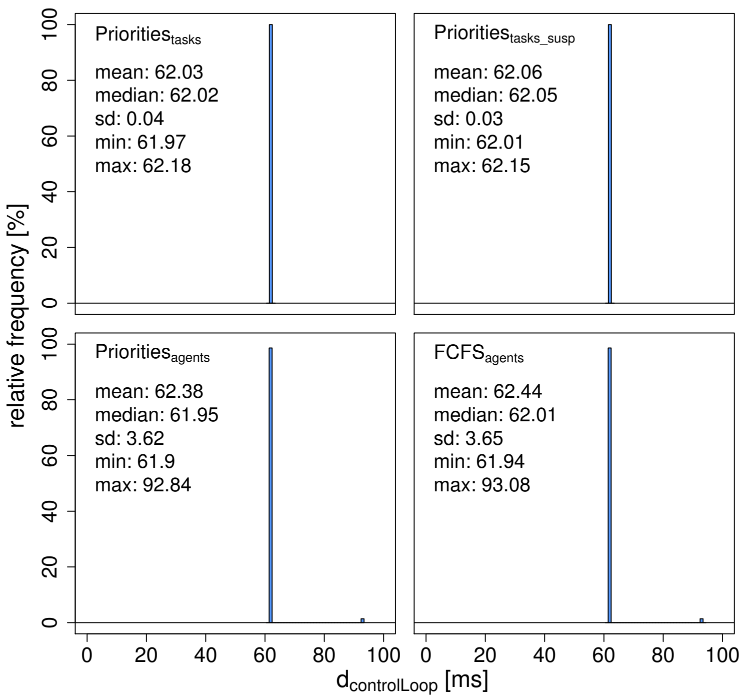
While execution times were neglected in the last subsection, they are now taken into account by executing the controller on HiL. Though the controller's FSC is generated differently and has to be installed on an Imote 2 now, the system's SDL specification remains unchanged. With the controller running on HiL, synchronization offset between controller and sensor/actuator deteriorates in general. But since the focus of the evaluation is on the transmission of sensor/-control values, this deterioration is actually undesired and would impede the interpretation of the results. Therefore, we still aim for perfect synchronization, which is achieved by sending beacons with real-time signaling [KBCG11] and by executing sensor and actuator node still as library-based FSC.

The results of the simulations are presented in Fig. 10.14 and introduce an additional scheduling strategy that is denoted by $\text{Priorities}_{tasks\_susp}$. This strategy corresponds to task scheduling but additionally suspends SDL real-time tasks of priority ≥ 2 during time-critical sections of the MAC layer; e.g., before frame transmissions in reserved time slots. Results of control loop delays (Fig. 10.14(a)) show that for all strategies, the delays significantly increase with an increment of about 30 ms. This indicates that the hardware is not fast enough to receive sensor values, to calculate a new control value, and to enqueue the corresponding transmission order within one slot duration and in time before the start of the next transmission slot of the controller. Though these results do not reveal much further information regarding differences between schedulers compared to the previous results, they point out the importance of evaluations on the target platform to assess the impact of execution delays on control quality (Fig. 10.14(b)). In this regard, HiL simulations state an acceptable trade-off w.r.t. effort.

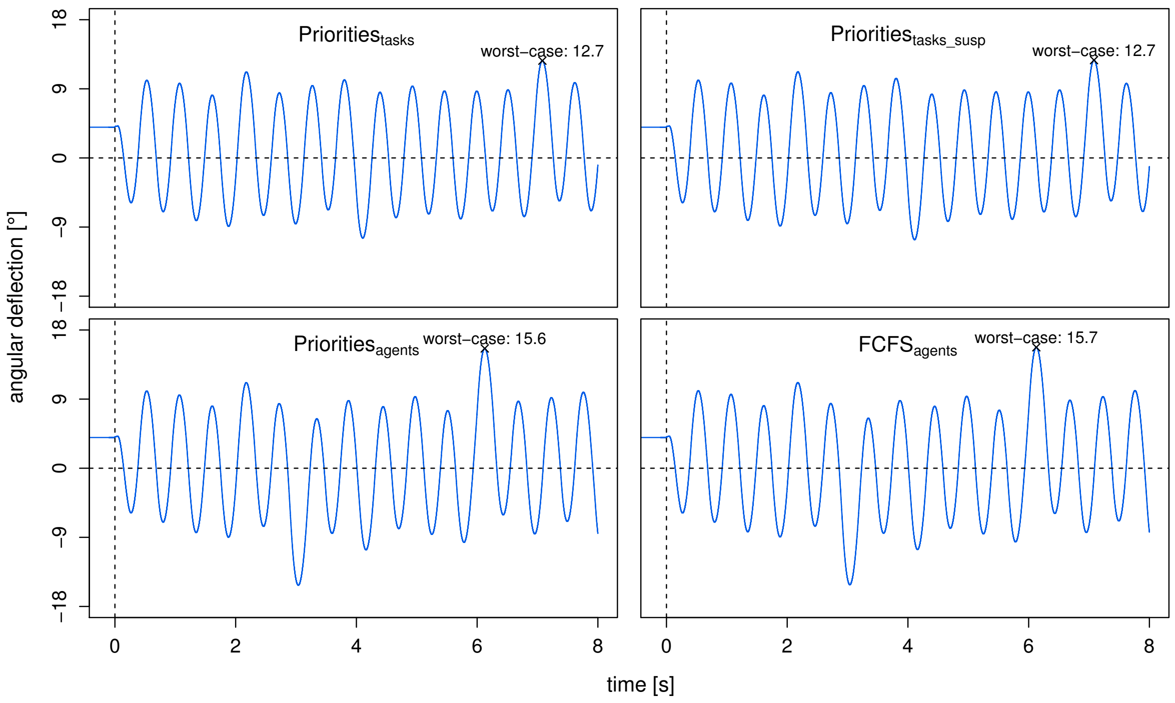
At first glance, a possible measure to decrease control loop delays with task scheduling is the swap of priorities of tasks of type CONTROL and MACFCT. However, this would decrease the controller's compliance with transmission slots, since the transition consuming SDL timer `reservedSlotT` (see Fig. 10.12) would be delayed until the new control value is available in the transmission queue of the MAC process. Because this would endanger the entire communication system, this "optimization" must not be applied. A better solution to reduce the delays is the selection of a communication schedule, which takes the controller's processing delays into account. In this regard, a possible schedule could be achieved by placing the actuator's communication slot in between the slots of sensor node and controller.

To evaluate the system's behavior under load, further simulation runs are performed with random background load of about 300 transition executions per second. This load accrues in the separate SDL block `Load` (see Fig. 10.12) and does not share transitions of other tasks, thereby being very convenient for process priority scheduling.

Corresponding results are presented in Fig. 10.15 by histograms over $d_{controlLoop}$. Though the histograms' general shapes are almost identical to Fig. 10.14(a), exact values (mean, max, etc.) are slightly larger. To investigate this issue in more detail, the controller's conformance with transmission slots is evaluated and plotted in Fig. 10.16. The figure presents the empirical cumulative distribution function of $d_{slotDeviation}$, which is the time difference between the controller's actual transmission start and the nominal start of its reserved communication slot. For most transmissions, the deviation is larger with task scheduling than with the other scheduling strategies. However, task scheduling achieves both with and without load the lowest maximal slot deviation. Furthermore, the difference between best and worst observed deviation is



(a) Control loop delays.



(b) Quality of control.

Figure 10.14: Performance comparison with HiL simulations.

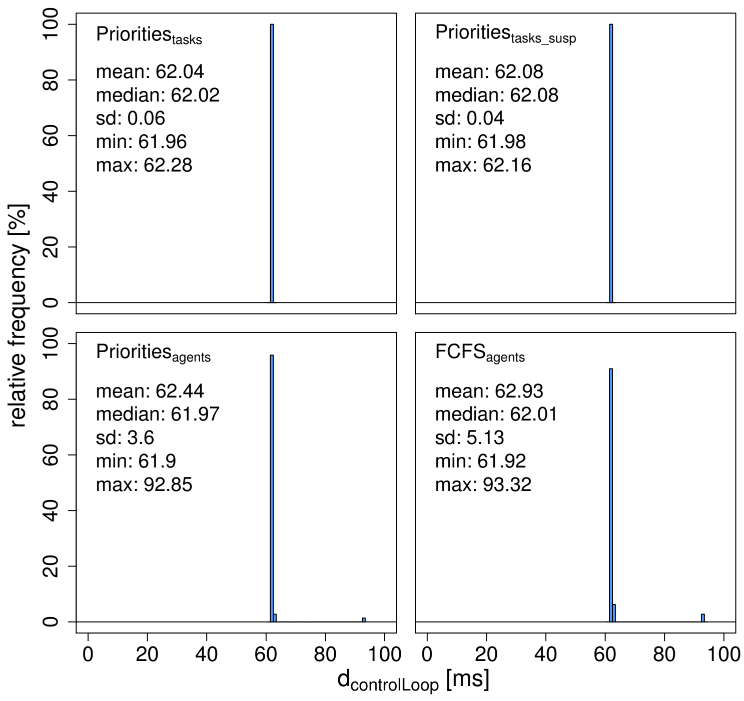
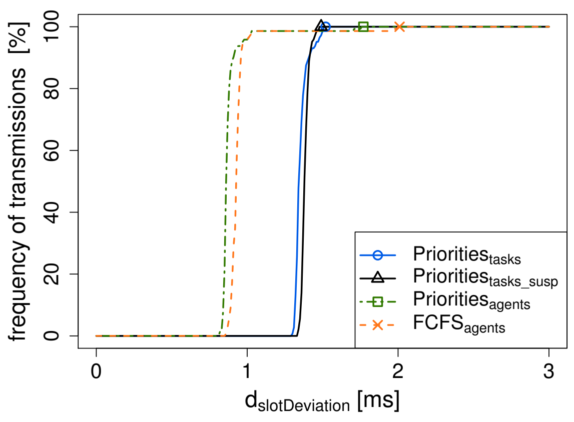


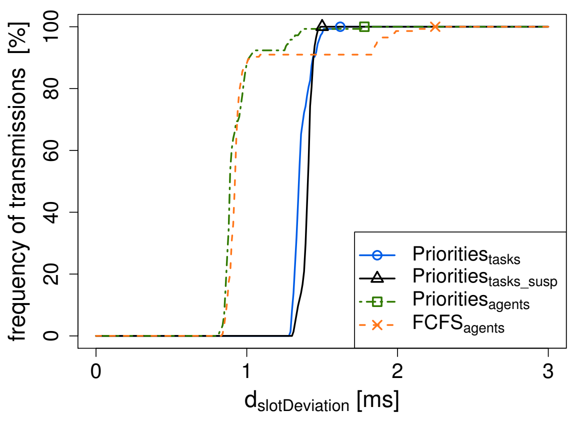
Figure 10.15: Histograms over control loop delays with additional system load.

smaller with task scheduling (1.29 ms → 1.50 ms w/o load; 1.28 ms → 1.60 ms with load). This, particularly, holds when applied with suspension, where $d_{slotDeviation}$ turns out to be (almost) insusceptible to load (1.32 ms → 1.47 ms w/o load; 1.30 ms → 1.48 ms with load). In sum, task scheduling does not emerge as most efficient strategy here but as most predictable one.

To review the property of predictability in more detail, statistical analyses are applied to derive probabilistic worst-case delays, which, in turn, allow a more reliable comparison between the different scheduling strategies. The adopted statistical methodology has been introduced



(a) Without background load.



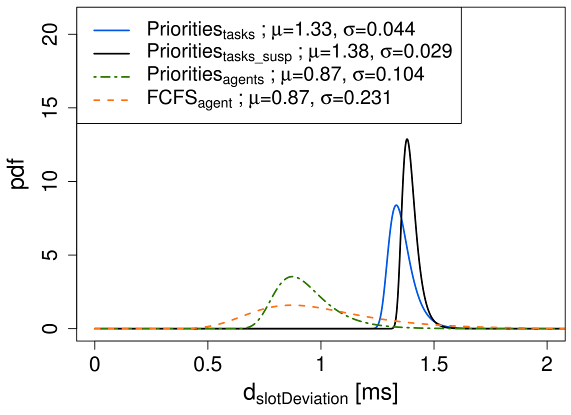
(b) With additional background load.

Figure 10.16: Empirically determined deviation of actual transmission times from nominal start of reserved communication slots.

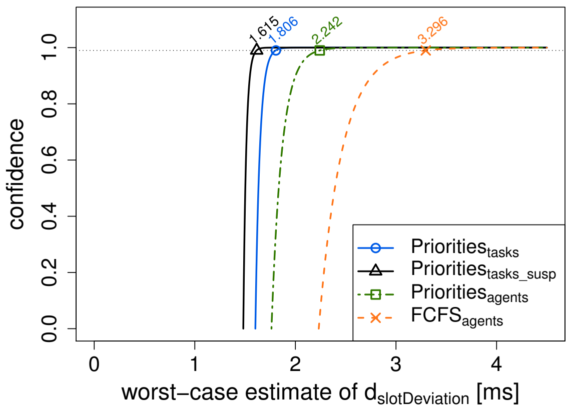
by Edgar and Burns in [EB01] and is based on extreme value theory. Its outcome are estimates of worst-case slot deviations[9] with associated levels of confidence. In more detail, the methodology is adopted as follows: First, a statistical model of slot deviations, which is based on a Gumbel distribution [Gum58], is derived from the empirical data. In the context of this evaluation, one such model is created for each scheduling strategy. Based on these models, pertaining excess distribution functions are built, where the respective maximum of the sample data is used as threshold of the distribution function. This function associates a level of confidence to worst-case estimates of slot deviations and enables comparisons between scheduling strategies w.r.t. to a desired confidence level.

In Fig. 10.17(a), the Gumbel distributions are plotted, which are derived from the outcome of the simulation runs and serve as models of slot deviations. Their parameters in terms of location (μ) and scale (σ) are given in the figure. Derived pertaining excess distributions are illustrated in Fig.10.17(b). They demonstrate that, with the same confidence level, estimates for worst-case slot deviations are significantly lower with task scheduling – and, in particular, with suspension – than with the other scheduling strategies. As examples, concrete values are presented for a confidence of 0.99 numerically: For task scheduling with suspension, this estimate is 1.615 ms, i.e., the probability that the actual possible worst-case deviation is larger than 1.615 ms is only 1%. Process priority scheduling provides the same confidence for an estimate of 2.242 ms, which is more than 600 µs larger.

In sum, the results of the statistical analyses confirm the intuition of the empirical data. On the one hand, they show that task scheduling is less efficient, i.e., slot deviations are on average lower with other state-of-the practice scheduling strategies. However, on the other hand, worst-case deviations with task scheduling are significantly lower; both in the empirical results of the simulations as well as in the probabilistic models. In this regard, the low variations in the deviations enable lower and more accurate worst-case estimates.



(a) Gumbel distributions as models of slot deviations.



(b) Estimates of worst-case slot deviations.

Figure 10.17: Statistical analyses of slot deviations.

[9]The original methodology is applied to execution times, which are equal to slot deviations in this context.

10.5 Discussion

This chapter has several contributions: First, it provides a proof-of-concept that SDL real-time tasks are not only a theoretical concept but are implementable and have practical benefits. Second, it demonstrates that task scheduling can – compared to other state-of-the-practice scheduling algorithms – reduce worst-case delays of time-critical tasks significantly. And finally, it provides evidence that costs of task scheduling are indeed larger but acceptable.

By performance evaluations, it is shown that SDL real-time tasks enable a powerful and model-based control of the system's execution at runtime. Because real-time tasks are orthogonal to the static system structure, task scheduling is more flexible than existing scheduling algorithms for SDL implementations and outperforms, in particular, (process) priority scheduling, which is often advocated as adequate prioritization measure. Though SDL process priorities are seen as default prioritizing scheme for SDL systems, they suffer if the same transitions are shared by several system tasks with different urgencies. For instance, in the ACC scenario, maximal reaction delays with task scheduling are only 40% of the worst-case delays with priority scheduling. Furthermore, task scheduling generates less jitter and is less prone to load than other scheduling strategies, thereby being more predictable and allowing the determination of tighter upper bounds. In the ACC scenario, the jitter of sensor delays is 370 µs with task scheduling, whereas priority scheduling produces 1300 µs of jitter.

However, task scheduling does not come for free but is associated with some extra costs. Yet, compared to advantages w.r.t. amount and variability of delays, overhead of task scheduling remains acceptable. In the ACC scenario, for instance, its overhead is approximately 10%-15% higher than the overhead of the most efficient scheduler of SdlRE. Since the overhead of task scheduling was slightly lower than the overhead of process priority scheduling, its efficiency is even put into a better perspective. Nevertheless, concrete values must be seen with great caution, because they are very implementation- and scenario-specific and not universal.

Similar results were observed in the scenario of the inverted pendulum. Though here, task scheduling reveals that it is not the fastest strategy on average and that delays are often smaller with state-of-the-practice solutions, it also demonstrates that it is most predictable and achieves the smallest worst-case delays. For instance, the variability of empirically observed slot deviations in the controller's system under load is with task scheduling 320 µs, whereas process priority scheduling varies in a range of 950 µs. In this regard, further improvements are achieved by task suspensions, which reduce the variability to 180 µs. By means of probabilistic models, it was furthermore confirmed that estimates for worst-case delays are significantly lower with task scheduling.

Though task suspension seems to be a kludge against the lack of full preemption in the first instance, it is actually stronger than preemption, since preemption reaches its limits when transitions share the same agent. In this case, preempting a running low priority transition is not possible due to SDL's run-to-completion semantics, but suspension, on the contrary, can avoid the execution of the transition a priori.

11. Chapter

Related Work: SDL Implementations, Extensions, and Analysis Techniques

SDL looks back to a very long history. Hence, the following survey of state-of-the-practice implementations, language extensions, and analysis techniques is not complete but targets at a broad overview. For this, not only previous works with similar objectives as SDL real-time tasks are considered but also orthogonal and complementary proposals, which can be combined with SDL real-time tasks gainfully. Though this section provides a broad overview, it clearly concentrates on SDL as engineering language for (embedded) real-time systems and excludes works, in which SDL is used for simulation and visualization purpose.

In [Int12c], the SDL standard advocates the language as follows:

> *"The main area of application for SDL is the specification of the behavior of aspects of real-time systems, and the design of such systems."*

And indeed, SDL has many properties that are advantageous for real-time systems [BGK+00] like asynchronous communication and a tight incorporation of time. However, it is also often stated (e.g., in [EGG+01]) that there are limitations, which prevent SDL from being a full real-time-capable design language and restrict application scenarios of SDL implementations. A main drawback is often seen in SDL's scheduling nondeterminism, which is addressed by many related works by incorporating priorities into SDL or by applying system analyses with the assumption of a priority-based execution model.

This chapter summarizes these works and other proposals that target improvements of SDL's expressiveness and applicability for embedded real-time systems. It is structured as follows: In Sect. 11.1, state-of-the-practice SDL implementation approaches and alternatives are outlined. Thereafter, Sect. 11.2 introduces analysis techniques and language extensions of SDL to improve its runtime predictability. Section 11.3 then presents previous approaches to derive SDL specifications from scenario descriptions with MSCs, which can be adopted to identify and illustrate task types of a system. Finally, Sect. 11.4 relates the presented proposals to SDL real-time tasks.

11.1 SDL Implementations

Model-driven development processes with automatically derived implementations have several advantages like consistency between specification and implementation, increase of productivity, and quality improvements [KLK00]. However, it is a common opinion that implementations automatically derived from models are slow, inefficient, and require much more memory than manual implementations [MCR97]. Thus, it is often concluded that automatically generated implementations are not suitable for time-critical systems as found in wireless networks [LL05]. As a consequence, SDL is often used as design language for prototype implementations and analyses or validations but not for final products. To remedy this gap in the engineering process, an impressive number of measures to improve the performance and to configure the generation of implementations – like the mapping of SDL processes to OS tasks – can be found in SDL tools as well as academic proposals. In the following, an outline of these approaches is provided.

11.1.1 General Aspects of Implementing SDL

There are several text books about the efficient implementation and realization alternatives of SDL. In the following, a survey of two comprehensive works is given [BH93, MT00].

In [BH93], Rolv Bræk and Øystein Haugen discuss the implementation of SDL-92 [Int93]. A large part of the book is about trade-offs between hardware and software implementations. Regarding hardware design, guidelines are provided to distribute an SDL system among physical units and to dimension these units appropriately. Referring to this, the gap between SDL systems and real systems is highlighted; including investigation of processing time, errors, and capacity of resources. The software design covers implementations of data types, communication, concurrency, and SDL signals, and compares several alternatives. Concerning signal transfer, an efficient way is the realization by function invocation, thereby removing SDL's asynchronous communication model and dissolving the distinction between communication and scheduling (see also Sect. 11.1.3). For an efficient implementation, Bræk and Haugen propose the omission of hard-to-implement language constructs like SDL's `save`. In addition, they suggest the introduction of signal and process priorities at implementation level to meet real-time constraints, and the suspension of signals from the environment during overload situation. Finally, example C++ implementations of several SDL constructs are shown. In this context, properties of SDL are opposed to properties of modern programming languages, e.g., regarding built-in support of concurrency, time, and communication.

Mitschele-Thiel addresses in [MT00] the implementation of SDL-2000 [Int99a] with the focus on efficiency and performance. The book also outlines differences between SDL and real-world implementations; particularly limitations of queues, processes, and communication. Regarding the interfacing of SDL with OSs, three strategies are compared: *Tight* integration (extensive usage of OS services w.r.t. scheduling and signal transfer), *light* integration (less interaction with OS and more functionality in SDL engine), and *bare* integration (no underlying OS but all functionalities provided by SDL engine). Further presented realization alternatives cover data types, input queues, buffer management, timers, transition activation, and state machines. Additionally, efficiency improvements on the basis of design guidelines and the limitation of SDL constructs are considered. In this regard, loosening of SDL's copy-by-value semantics is presented as a measure to reduce overhead of signal transfer. Furthermore, Mitschele-Thiel presents alternatives to deal with scarce resources and to handle queue overflows.

Besides text books, there are also some papers about efficiency of SDL implementations. In [San00], Sanders discusses alternatives to deal with the mismatch between "SDL's ideal world" and the real world. The presented discussion includes, for instance, limitations of queues, data types, and resources, and the mapping of SDL's concurrent execution model to a serialized execution. To implement SDL efficiently, he suggests to avoid the copying of signal parameters if possible and to realize signal outputs as method calls instead of buffered signal transfers. Mapping signal outputs to method calls equals the activity thread approach (see Sect. 11.1.3) and is only applicable if signal transfer is not circular. If this precondition is not fulfilled, a buffered transfer must be used. In this case, an implementation must, however, consider limited queue sizes and buffer overflows, e.g., by temporarily suspending sending SDL processes. In Sander's opinion, dissolving SDL's concurrent execution semantics is outside the scope of an SDL specification. Thus, he refers to additional tools like the deployment editor of IBM's Rational SDL Suite [IBMar, LEH00], which enables the grouping of SDL processes into (concurrent) OS processes with different priorities. Regarding flexibility in implementations, he suggests configurable code generators – like COCOS [LK99, KLK00] or the RASTA toolkit [MCR97] – to provide optimization parameters (e.g., speed vs. memory usage).

11.1.2 State-of-the-Practice – Commercial SDL Tools

There are currently three vendors offering tool suites with strong SDL support: IBM (Rational SDL Suite [IBMar]), PragmaDev (Real-time Developer Studio [Praar]), and Cinderella ApS (Cinderella [Cinar, RKL05]). Since Cinderella is only available for Windows and was last updated more than ten years ago, the following survey considers only IBM's SDL Suite and PragmaDev's Real-time Developer Studio (RTDS).

IBM's SDL Suite [IBMar] – originally developed by Telelogic – provides various SDL-96, MSC, and ASN.1 tools, and allows a tight collaboration with UML [LEH00]. It supports several target OSs (e.g., Linux, Windows, and various RTOSs like VxWorks[1] and QNX [2]) and bare metal implementations. Besides a heavy-weighted C code generator called Cadvanced/Cbasic, SDL Suite includes with Cmicro a C code generator that is specialized for embedded systems with real-time characteristics. Cmicro enables the (optional) assignment of priorities to SDL processes and signal types, which are directly specified in SDL diagrams by a novel `#PRIO` directive. If process priorities are defined, a preemptive scheduling strategy can be used and scheduling decisions are based on a signal queue for each process priority level that is optionally sorted by signal priorities. Code generated by Cmicro favors static memory allocation and does not include code for SDL blocks or signal routes, i.e., according to SDL's semantics [Int00], only *SdlAgents* are explicitly created and scheduled during runtime. Many optimizations are only possible by omitting support for complex SDL constructs. This includes procedures with states, enabling conditions, continuous signals, inheritance, and SDL services[3].

RTDS [Praar] is currently the most lively SDL tool suite and runs on Windows and Linux. It provides a graphical SDL editor, simulator, debugger, and code generators for both C and C++. Supported target OSs comprehend Windows, Linux, and various RTOSs (VxWorks, FreeRTOS[4], Nucleus[5], ...). With the default mapping strategy, each SDL process becomes a single RTOS task in the implementation (called *threaded* execution), but RTDS also supports grouping of several SDL processes into one RTOS task (called *scheduled* execution). In this case, scheduling

[1] `http://www.windriver.com/products/vxworks/`

[2] `http://www.qnx.com/`

[3] SDL services have been superseded by composite states in SDL-2000 [Int99a].

[4] `http://www.freertos.org/`

[5] `http://www.mentor.com/embedded-software/nucleus/`

within an RTOS task can be performed by a built-in scheduler executing transitions in a signal-based FIFO order. By grouping the entire SDL system into one task, bare implementations are possible. The concrete mapping of SDL processes into tasks is defined via UML deployment diagrams.

Besides SDL, RTDS has also support for SDL-RT (SDL Real-Time [SDL13]), an SDL dialect with the objective to harmonize SDL and RTOS implementations. SDL-RT is currently published in version 2.3 and can basically be described as alignment of naming conventions (e.g., “signal” is renamed to “message”) and the embedding of C into SDL. This includes the incorporation of C data type declarations instead of SDL data types[6], global variables, pointers, and semaphores, which come with a distinct graphical syntax. Parameters of messages can be passed on by reference. In addition, SDL-RT supports combinations of SDL diagrams with UML class diagrams, where classes can either be static (usual C++ classes for data types) or dynamic (becoming active *SdlAgents* in the implementation). SDL-RT also allows the specification of SDL process priorities, which are mapped to priorities of RTOS tasks. Besides language extensions, SDL-RT comes also with some restrictions. This affects, for instance, enabling conditions and priority inputs, which are not part of SDL-RT.

11.1.3 SDL Execution with Activity Threads

An efficient way of implementing SDL signal transfer is by using the activity thread approach [LK99, KLK00]. In this approach, a procedure is provided for each signal input and signal transfers are implemented with procedure calls, i.e., the output of a signal executes the consuming transition immediately. Though this synchronous execution model, in which signals and not processes are seen as active entities, produces much less overhead than a model with signal queues, it differs from SDL’s asynchronous execution model significantly. As result, several semantic conflicts can arise depending on the concrete system structure. These conflicts include, for instance, interference of transition executions of the same process and overtaking of signals. To resolve them and to enforce the run-to-completion semantics of SDL transitions, [LK99, KLK00] propose a technique called *transition reordering*, which has been implemented in a tool called *COnfigurable COmpiler for SDL* (COCOS). Transition reordering is applied at compile time and postpones signal outputs after the transition has reached its successor state. In some cases, it changes additionally the signal output order to enforce a desired transition execution order. Though transition reordering resolves many semantical conflicts, it cannot be applied in all cases, e.g., if signal outputs occur in loops, where the number of iterations is unknown at compile time. Additionally, the activity thread approach is not compatible with SDL’s `save` or if a circular signal output sequence exists, in which the same transition should be executed multiple times. To overcome these problems, the authors suggest a combined implementation, using the activity thread model where possible and a server model providing asynchronous buffered signal transfer in all other cases.

[6]This was also be planned for SDL-2010 [Int12c] but was not completed at the time of the standard’s approval (see also Sect. 7.1).

11.1.4 Directing SDL Code Generators by Annotations

To improve SDL implementations for industrial products, [MCR97] presents extensions of SDL that are classified as fine- or coarse-grained annotations. Fine-grained annotations (called *Target Language Extensions* (TLEs)) are written directly in the SDL specification and support the seamless integration of manually written code into code that is automatically generated from the specification. Examples of TLEs are variable declarations and decisions in implementation language, thereby also supporting data types of the target language and global variables. Coarse-grained annotations are specified in a separate annotation file and guide the code generator regarding structural aspects. Here, examples are the mapping of SDL blocks to OS tasks and the configuration of signals to be implemented as procedure calls.

11.1.5 Implementing SDL on TinyOS

In [DRDK04], Dietterle et al. use the light-weight open source OS TinyOS[7] as foundation for SDL implementations and present an approach to map an SDL system to a set of TinyOS components. In TinyOS, all components run concurrently in the same address space and are controlled by a FIFO scheduler by default. In the presented mapping strategy, each SDL process becomes one TinyOS component, which implements the state machine of the process. Current state and local variables are stored in the frame of the component. A signal queue is additionally introduced in each component to realize SDL's `save`. For optimization purpose, several measures like replacing SDL's asynchronous communication with synchronous procedures and using copy-by-reference data types are discussed. Though the mapping is mostly straight-forward, several limitations exist, since TinyOS does, for instance, not support dynamic component/process creations. Furthermore, it should be considered that TinyOS is no RTOS, thereby limiting its applicability in time-critical systems.

11.1.6 Implementing SDL on RTOSs

Drosos et al. present the realization of an interface to run SDL systems, which are generated with IBM's Rational SDL Suite and Cadvanced [IBMar], on the Virtuoso RTOS [DZM01]. Since the SDL Suite does not come with support for Virtuoso, a new intermediate layer has been developed. This layer provides hardware drivers and interrupt routines, and includes an SDL environment agent, which maps SDL signals to low level function calls and generates SDL signals during interrupt handling. The authors follow a tight integration approach, in which each SDL process is mapped to one RTOS task with a single signal queue. In addition, a timer OS task is generated to handle SDL timers. Scheduling is completely performed by the scheduler of Virtuoso, which supports preemptive and priority-based strategies. A proof-of-concept is given with the DECT protocol on an ARM microcontroller.

A further example of integrating SDL into an RTOS is with the Reflex RTOS [WDEK06]. Starting point of this integration is also Cadvanced of IBM's Rational SDL Suite [IBMar]. The SDL runtime environment is basically realized by the OS and a thin OS integration layer. The integration approach is also tight, i.e., each SDL process becomes a so-called *schedulable activity*

[7] `http://www.tinyos.net/`

of Reflex. Non-schedulable activities of Reflex are used for interrupt routines. By default, activities with higher priority can interrupt other activities and scheduling is performed with an EDF strategy. Communication among processes is via event channels, where a target SDL process must be addressed by its name, thereby reducing flexibility and making dynamic process instances impossible. To realize SDL timers, a global timer queue is introduced. Expired timers are removed from this queue after periodical ticks, which are announced by interrupts. Thus, granularity of SDL timers is limited to the periodicity of ticks. The SDL environment is also realized by a Reflex activity, which is addressed by identifier `xEnv`. Because signals generated by the environment must also be addressed to the receiving SDL process by name, the environment activity must either be specific for each SDL system or processes in the SDL system must follow a fixed naming convention.

11.1.7 Handling Scarce Hardware Resources

Implementations for embedded systems require a well-planned utilization of hardware resources. In the following, two approaches are summarized, which address memory and energy consumption, respectively, for SDL implementations.

The work in [GGK07] is motivated by scarce memory resources of embedded systems. In SDL, a main reason of memory overload is the semantics of input ports, which may store *any number of signals*. Since this is non-realistic in implementations, the authors present extensions of SDL's syntax and semantics to specify bounds on SDL input ports. Here, a distinction is made between the specification of *explicit* and *implicit* input port bounds, where explicit bounds are specified at input ports directly and implicit bounds are associated with incoming signals. The authors prefer implicit bounds and allow their specification globally (at signal type definitions) or locally (at signals in channels or gates). To deal with input port overflow, three strategies are compared: *Discard* (delete new signal), *replace* (replace first signal of same type in input port with new signal), and *delete/append* (delete first signal of same type in input port and append new signal at the end). Since the last strategy is compliant with SDL's semantics regarding signal ordering, it is the preferred solution. In addition, the *delete/append* strategy is also the desired behavior in many systems and, in particular, in Networked Control Systems (NCS). To be compatible with the used SDL editor and code generator Cmicro [IBMar], the extensions have been implemented by annotations.

Not memory constraints but energy resources are addressed in [GKLC09] by energy-aware system designs with SDL. The paper presents two complementary approaches to reduce energy consumption at design time that are called *energy mode signaling* and *energy scheduling*. A precondition for both approaches is the existence of a detailed energy model, in which energy modes, energy consumptions per mode, and transition delays are provided for each hardware component.

- The energy model is explicitly exploited by *energy mode signaling*, where hardware mode transitions are specified in SDL and signaled by SDL procedures or SDL signals to the environment. To distinguish between SDL signals describing the regular behavior and signals triggering transitions between energy modes, naming conventions are introduced. An advantage of energy mode signaling is that scenario knowledge can be utilized, e.g., to realize duty cycling of the transceiver.

- *Energy scheduling* is an implicit approach, which uses system state information to switch between energy modes. Compared to energy mode signaling, it has the advantage that information of the system's runtime environment – like the contents of signal queues or the next timer expiration time – is available.

Both approaches require extensions of the SVM and environment implementation. They are illustrated with an inverted pendulum system using the Imote2 [MEMara] platform.

11.1.8 Hardware/Software Co-Design

A common approach to meet tight timing constraints is the implementation of time-critical functionality in hardware. Due to the concurrent runtime model of SDL, the language is indeed well suited for hardware implementations, which provide a much higher degree of concurrency. As result, several proposals have been submitted covering the usage of SDL for combined hardware/software co-designs.

[SDM01] presents an approach that is based on a fine-grained partitioning of SDL processes into the control flow of state machines and the data operations. To obtain software implementations, C code is generated for selected parts of the system. For hardware implementations, parts of the SDL system are translated into a *Very High speed integrated circuit hardware Description Language* (VHDL) representation and deployed to *Field Programmable Gate Arrays* (FPGAs). The approach enables a flexible assignment of parts of an SDL process to hardware and software. Thus, it is, for instance, possible to implement some transitions in hardware and the remaining transitions and the state machine in software.

In [LL05], an SDL profiling tool called *profSDL* is presented. Its objective is to find potential bottlenecks of the system and to suggest SDL processes which should better be implemented manually or in hardware. For this, the tool provides a static and a dynamic analysis of an SDL specification. The static analysis counts the number of particular SDL constructs that are used in each SDL process. The dynamic analysis evaluates the computational effort by simulating the SDL system and counting the number of executions of particular SDL statements. By measuring concrete execution times of single SDL statements on different target platforms, the counted number of executions can be weighted to obtain a more accurate impression of the computational effort. A further output of profSDL is the overhead of the SVM implementation in terms of inter-process communications. Thus, the tool can also suggest to merge SDL processes in order to reduce communication overhead.

11.2 Predictability and Expressiveness of SDL

The missing capabilities of SDL to express execution precedence and temporal constraints has led to several language extensions, which address SDL's expressiveness w.r.t time and scheduling nondeterminism. While the objective of most of these extensions is the analytical assessment of a system's temporal behavior and non-functional properties, there are also some proposals that target the applicability of SDL as real-time engineering language. Though the objectives of these proposals are completely different, they all want to make SDL implementations more predictable and its runtime behavior more transparent. Therefore, many extensions are related to each other, yet their methodology differs significantly. The following subsections provide a survey of this broad field of related work.

11.2.1 Analysis Based on Markov Chains

Timed SDL (TSDL, [BB90, BB93]) describes extensions of SDL to assess qualitative (e.g., absence of deadlocks) and quantitative (e.g., throughput) system properties. The presented extensions mainly target the evaluation of protocols and can be grouped into two categories: First, there are extensions to mark SDL transitions with probabilities and execution delays. Second, extensions exist to inspect the state of an SDL process and its input port. The proposed approach is based on the mapping of system specifications with TSDL into a more general representation with extended finite state machines. On these state machines, analysis techniques from the area of Markov chains are applied. In this regard, the authors present three analysis algorithms that differ in the way of dealing with state space explosion [BB90]: Non-exhaustive state space exploration, non-exhaustive performance analysis, and probabilistic validation and performance evaluation. All algorithms are implemented in a tool chain and use the system specification with TSDL as input. Additionally, a control file has to be provided to limit the length of signal queues and to mark SDL states, for which performance results should be computed. W.r.t. qualitative evaluation, the output of the analysis distinguishes between errors (e.g., deadlocks), which disallow further performance analyses, and warnings (e.g., implicit consumptions).

11.2.2 Simulation-based Analysis

Queueing SDL (QSDL, [DHMC95, DHMC97]) is an extensive graphical extension of SDL-92 [Int93] for performance evaluation of SDL systems. The concept of QSDL is based on queueing models [Jai91], whereas evaluation results are obtained by discrete simulations. Using QSDL, an SDL specification is extended with so-called machines providing services, properties of load, and the mapping of load to machines. Machines of QSDL look very similar to SDL processes and can be placed in similar places of the specification. They are connected to SDL processes by pipes and some kind of link (similar to SDL channels and SDL links). Type, amount, and priority of load are specified within SDL transitions, either at SDL task statements [DHMC95] or at newly introduced request constructs [DHMC97]. To simulate random load, QSDL provides data types for several probability distributions. QSDL machines are configured with the number of servers and the provided service types. Furthermore, scheduling strategies and speed of servers can be defined. The presented approach is implemented in a tool called QUEST, which transforms extended SDL specifications into executable simulation models. The output of the simulation includes both functional behavior (visualized by MSCs) and performance data (signal wait times, machine utilization, . . .). A drawback of QSDL/QUEST is the neglect of overhead of the runtime environment, e.g., to select a transition or to transfer a signal.

11.2.3 Scheduling SDL with Message Deadlines

In [KF98], Kolloch and Färber present schedulability analyses of SDL systems that are based on the determination of WCETs of transitions by profiling on the target platform, the specification of non-functional properties – in particular, deadlines and worst-case intervals of external events – in *Performance Message Sequence Charts* (PMSCs, [FLMTS97]), and the mapping of the SDL specification to an analyzable task network. Together with properties of the target platform, the resulting documents constitute the Real-Time Analysis Model (RTAM), which is the input of schedulability analyses with a preemptive EDF strategy. Since task networks are less expressive than SDL specifications, the mapping underlies many restrictions like no dynamic process creations, no priority inputs, and no save of input signals. The generated task network consists of several independent task systems. Each system is built by consecutively traversing all SDL transitions that are executed to answer one external system event. Therefore, task systems may share transitions of the same SDL process. As a consequence, task systems have regions with mutual exclusion, since transitions of the same SDL process cannot be preempted.

The work has also become a part of Kolloch's dissertation [Kol02], in which scheduling with signal deadlines (called *Message-based EDF* – MEDF) is incorporated into SDL. The overall objective of his work is the improvement of SDL's predictability and analyzability for reactive real-time systems, in which responses to system stimuli must be created in time. This is achieved by limiting the scheduling nondeterminism of SDL implementations, yet to the detriment of language features (e.g., SDL's `save` and delaying channels are forbidden).

MEDF scheduling is applied to process both external events (signals from the environment) and internal events (SDL timers). In both cases, an absolute message deadline (a so-called *deadline token*) is attached to the signal, which is passed on to outgoing signals during transition executions. Input ports of processes are ordered by these message deadlines and transitions are executed accordingly. Besides describing the impact of MEDF scheduling on SDL's semantics, Kolloch also introduces annotations for the specification of timing constraints such as computation times. Together with further non-functional properties like frequency and dependency of events, the timing constraints build the foundation for system analyses with MEDF scheduling and automatically derived task networks (see above).

MEDF has been implemented with IBM's Rational SDL Suite [IBMar] on the light-weighted RTOS *Real-Time Executive for Multiprocessor Systems* (RTEMS[8]). During code generation, two alternative strategies have to be balanced: The *Server Model* generating one RTOS task for each SDL process and the *Activity Thread Model* grouping all transitions that belong to one event stream into a single RTOS task. The RTEMS scheduler has been extended to execute RTOS tasks w.r.t. signal deadlines and to apply deadline ceiling and deadline inheritance protocols.

[8]http://www.rtems.org/

11.2.4 Analysis of SDL Specifications with Timed Automata

In [BFG+99, BGM01a], Bozga et al. present an approach to interconnect various tool suites in order to evaluate SDL specifications with existing analyzers, validation tools, and model checkers. The resulting integration tool is named *IF environment*, because it is based on an representation called *Intermediate Format* (IF). SDL specifications are generated with ObjectGEODE, which is no longer available on market and was originally developed by VERILOG.[9] Starting from the abstract syntax tree provided by ObjectGEODE, the transpiler SDL2IF translates SDL into IF. Though most of the translation steps are straight-forward, some language constructs of SDL like dynamic process instantiation are not supported. The IF specification of the system is, unlike the hierarchical structure of the SDL specification, flat and consists of a set of timed automata. They asynchronously communicate via buffers, which can – different from SDL – be configured to be lossy and/or bounded and to apply different queuing policies. The semantics is based on a labeled transition system, where transitions can additionally be attributed with different urgencies: *eager*, *delayable*, or *lazy*. Thereby, the designer can control whether time may progress infinitely (*lazy* – default behavior in SDL implementations), restrictedly (*delayable*), or not at all (*eager* – default in SDL simulators) in situations with firable transitions. To cope with state explosion, the authors present countermeasures, which are realized with the IF2IF compiler and based on static analysis.

In [BGK+00, BGM+01b, Gra02], the same authors encourage a broader discussion on SDL's deficiencies w.r.t. missing programming and modeling constructs. Regarding implementation, they propose several extensions like the introduction of cyclic and interruptive timers, and an operator to access timer values. In addition, they suggest language support for atomicity and mutual exclusion, which is argued to be required in real-time systems. To achieve this, standardized SDL packages providing an interface to semaphores and similar synchronization mechanisms are considered [BGM+01b]. On the specification side, the authors criticize the weak time semantics of SDL and missing support to express assumptions on system properties. To mitigate these shortcomings, they present a set of annotations and propose the introduction of local clocks [Gra02], which are deduced from SDL's `now` with a maximal offset and drift. Furthermore, they introduce a new time semantics based on timed automata with three types of urgencies: lazy, delayable, and eager (see above). They are realized by annotations that are associated with signal inputs and continuous signals. Further annotations are connected to SDL channels to attribute communication delays, loss probabilities, and signal order. Since the authors do not believe in a single SDL semantics fulfilling all requirements from system analysis over code generation to test case definitions, they recommend the introduction of semantical profiles [BGK+00]. An overview of the extensions and annotations, and their application in the *Reliable Multicast Transfer Protocol 2* case study can be found in [PVB03].

A detailed description of mapping SDL specifications to timed automata with urgencies is available in [OK01]. The overall objective of this work is the definition and verification of temporal properties of an SDL system. Examples of such properties are invariances, the absence of deadlocks, and *non-zenoness*, i.e., the absence of runs without time progress. Though the last property first describes a theoretical problem and no implementation issue, it is actually also

[9] In 1999, VERILOG has been acquired by Telelogic AB (`http://feed.ne.cision.com/Commands/File.aspx?id=45103`), which has, in turn, been acquired by IBM in 2008 (`http://www-03.ibm.com/press/us/en/pressrelease/23792.wss`). Thus, parts of ObjectGEODE can be found in IBM's Rational SDL Suite [IBMar].

a problem on implementation level, because it indicates a system with permanent overload. Properties are defined with the property specification language GOAL (Geode Observation Automata Language), which is automata-based and has an SDL-like syntax. The verification step is performed with an extended variant of the ObjectGEODE tool. Since most problems and properties that are tackled by the authors are undecidable with full SDL, restrictions of language constructs, variable ranges, signal queue lengths, and the usage of SDL time are made.

11.2.5 Improving SDL's Expressiveness with Native SDL

To enrich SDL specifications with temporal constraints, [ABS01] presents a set of SDL extensions, which are actually no extensions of SDL's syntax but use available language constructs and naming conventions. The presented approach brings temporal knowledge about the implementation into SDL specifications and relies on typed signals and a standardized architecture of embedded real-time systems. In the center of the architecture is an application that has to produce answers to system stimuli from the environment within given deadlines. It is additionally triggered by a periodic clock signal to perform recurrent calculations. The typed signals follow a special naming scheme and have pre-defined parameters to specify timing constraints. Examples are interrupt signals that are augmented by the minimal intervals between subsequent interrupts and deadlines. Because the presented extensions have not been implemented, they only improve the expressiveness of specifications without affecting final implementations.

Motivated by the weak definition of time progress in SDL, [AKLN99] presents a new interpretation of SDL's time model. The proposed model does actually not violate the standard, but limits time progress by totally controlling SDL's `now`. The model defines time progress only in between timed events, which are either SDL timers, signals with enabled conditions, or continuous signals. Executions of transitions do not require any time. If a continuous signal or an enabling condition contains a time interval, the transition is executed at a point in time, which is randomly taken from the interval (comparable with *delayable* transitions in Sect. 11.2.4). Due to this random behavior, the execution is still nondeterministic. All timing constraints are specified with native SDL. They can only be enforced in simulations and cannot be guaranteed in implementations.

11.2.6 SDL* – Extensions for SDL Time

[MSDH01] addresses the problem that SDL's time model is inappropriate to express timing constraints. It is, particularly, criticized that SDL time has no unit and that SDL's definition of time progress is too general. To solve this issue, the authors propose the usage of physical clock models in SDL, which are introduced by an abstract data type *Clock* and corresponding operators to access clock values and to set timers. Physical properties of clocks are specified with the annotation-based extension SDL*. Thus, designers can add several clocks with different physical properties in an SDL specification. The authors additionally add a new abstract data type *Eventclass* to monitor events, to validate timing constraints, or to run performance evaluations. By providing a flexible interface to this abstract data type, complex dependencies between events can be evaluated. Though the presented approach reduces the gap between SDL's time model and physical clocks, it does not improve the efficiency or predictability of implementations.

11.2.7 Transition Priorities and Rate Monotonic Analysis

In [ÁDL+00, ÁDL+01, ÁDL+03], Álvarez et al. present a priority-based execution model of SDL for reactive systems. The presented scheduling strategy is based on fixed transition priorities, which are defined as annotations and connected to input symbols of SDL transitions. Transition priorities are used to derive the dynamic priority of a process, which is defined as the highest priority of all currently firable transitions of the process. The proposed scheduling strategy is fully preemptive, i.e., a firable transition can interrupt the execution of a transition with lower priority as long as both transitions are not in the same SDL process. The changed execution model is used as foundation for a single-core Rate-Monotonic Analysis (RMA). Inputs of the RMA are the WCET of each transition, periodicity of events, deadlines of expected responses, and the extended SDL specification. The response time of an event is defined as the execution time of all transition executions that are involved in processing the event plus *interference* time (time to execute transitions with higher priority) and *blocking* time (time to wait until a transition execution of the same process with lower priority terminates). If the RMA reveals violations of deadlines, the authors suggest the application of redesign rules such as replication of SDL transitions and the movement of transitions to newly created processes.

In [ÁDL+03], language extensions in terms of two pre-defined functions – `time_sent` and `time_received` – are presented to improve the transparency of time progress during signal transfer. They return the signal creation time (`time_sent`) and the time of signal consumption (`time_received`). To deal with hardware interrupts and to control system load, [ÁDL+03] furthermore presents a way to integrate hardware drivers into SDL specifications. For each driver, two processes are introduced: A *passive process* providing access to the hardware and transitions that are directly executed after hardware interrupts, and an *active process* bridging between the passive process and the rest of the SDL system. By assigning appropriate transition priorities, the priority of hardware components can be specified.

The implementation of the priority-based execution model is called Sched_SDL [ÁDL+01]. It maps SDL transitions to POSIX threads and provides a scheduler to enforce the priority-based transition execution order. Though the authors state that their approach is compliant with SDL's semantics [ÁDL+03], it actually supports only a subset of SDL and includes violations of the semantics like the removal of implicit consumptions. It is also unclear how language constructs like enabling conditions are compatible with the priority-based transition selection. This even holds when comparing the proposed transition priorities with SDL-2010 [Int12c], where due to the introduction of multiple input priority levels less semantics violations exist.

The approach of Álvarez et al. is adopted by Diaz et al. to allow the utilization of RT-CORBA (Real Time-Common Object Request Broker Architecture, [Gro05]) from within SDL specifications [DGLT09]. The general objective behind CORBA is enabling method invocations on remote objects and to hide the required communication from the caller. For this purpose, client stubs and server skeletons are used to serialize method parameters and return values, and to abstract from communication. A drawback of standard CORBA is the missing upper delay bound on method invocations, which is improved by RT-CORBA by introducing a priority model that supports both server- and client-defined priorities. To enable the use of RT-CORBA in SDL, [DGLT09] presents a set of SDL design patterns, which introduce caller (client) as well as callee (server) in the specification. If the developer selects server-defined priorities, the priority model of Álvarez et al. [ÁDL+00, ÁDL+01, ÁDL+03] is applied.

11.2.8 Real-time Signaling – A Generalization of SDL Timers

In [KBCG11], Krämer et al. present SDL extensions to introduce the concept of remote timers. The extension is called *real-time signaling* and enables the association of two additional timestamps with ordinary SDL signals to specify time intervals, in which SDL signals should be received. The first timestamp is the *arrival time* that postpones signal delivery until the specified point in time. The second timestamp is the *expiry time*, which allows the removal of "outdated" signals in situations with high load. In addition to real-time signaling, the authors present an anonymous variable called `sendtime` to access the point in time when an SDL signal was created (similar to `time_sent` in Sect. 11.2.7). The extensions are illustrated in an implementation with a centralized scheduler, which activates tasks in a time-triggered way. Compared to solutions with standard SDL, the SDL design with real-time signaling produces less overhead and improves timeliness significantly. Real-time signaling has been incorporated into SDL by extending SDL's concrete syntax and semantics, and implemented in a tool chain consisting of SdlRE [Fli09] and SEnF [FGJ$^+$05]. The idea behind real-time signaling has been adopted in SDL-2010 [Int12c] in terms of activation delays, yet with a different syntax.

11.2.9 Hierarchical Scheduling Strategies for SDL

To close the gap between the behavior of SDL systems in simulations and implementations, Boutet et al. present an approach to integrate scheduling policies on design level [BCRL00]. In a first step, SDL processes of the specification are mapped to nodes, which represent concurrent execution entities. In an implementation, a node can, for instance, also be an OS process. Furthermore, SDL signals are grouped into internal and external events, where internal events are signals between SDL processes of the same node, and external events are signals traversing several nodes (including signals from/to the environment) and SDL timer signals, respectively. Within a node, internal as well as external events are processed with FIFO order. Internal events, however, have always privilege over external events, thereby creating a transition execution order that is different from the SDL semantics. To define a precedence among nodes, each node is attributed with a priority. As result, SDL's scheduling nondeterminsm is replaced by a less concurrent execution model and a priority-based scheduling policy, thereby making system analysis and validations by simulations more implementation-related. The presented approach has been implemented in the ObjectGEODE simulator, in which a GOAL observer implements the proposed inter- and intra-node scheduling policies.

11.2.10 Further Model-based Analysis Techniques for SDL

Model checking with SDL as input language has a long tradition and combines the strength of two worlds: On the one hand, SDL with its formal semantics and its foundation of finite state machines, and, on the other hand, the sophisticated functionalities of established verification tools. Different from the simulation of a scenario, model checking is based on state space exploration. Thus, it generally explores a larger state space than simulations. Since model checking with SDL uses the SDL specification as input, it can, however, not guarantee correctness of the system due to possible errors in implementations or wrong assumptions.

In [RB98], Regensburger and Barnard from Siemens present a model checking tool suite called *System Verification Environment* (SVE). It is based on binary decision diagrams (BDDs) and targets the verification of protocol properties of mobile communication systems. The input of SVE is a set of SDL processes, which are specified with ObjectGEODE and translated into general finite state machines. Since the translation process is severely limited regarding support of SDL constructs and data types, designers of the SDL systems must follow rigid guidelines. Properties to verify and assumptions about relevant system runs are specified in the SDL Property Language (SPL) and translated into temporal logic. The behavior of the environment is modeled by pseudo SDL processes. To keep the state space smaller, a deterministic, single-core, and non-preemptive scheduling strategy with process priorities is applied.

The usage of SDL for post-mortem model checking of distributed systems is presented by Hallal et al. in [HPUB01]. Their approach starts with execution traces, which are recorded during test runs of the system and contain send, receive, and local events. The traces are mapped to signal outputs, inputs, and SDL task statements, respectively. The resulting system consists of one SDL block, where for each entity of the distributed system, one SDL process is included. The events of one system entity are reflected by a total order inside the corresponding SDL process, where SDL's `save` is used to guarantee the correct order of signal inputs. Consequently, the generated SDL system may have very few similarities with the original system. Properties to validate are specified in GOAL and checked by GOAL observers during exhaustive simulation runs with ObjectGEODE. A drawback of post-mortem model checking is the limited validity of results, because a system can still be faulty, though the model checking process does not reveal any error. Thus, choosing the *right* execution traces is crucial.

In [BDHS00, SS01], a model checking approach for SDL systems is proposed with DTSpin, a discrete-time extension of the Spin (Simple Promela INterpreter[10]) model checker. The input is an SDL specification, which is first transformed into IF with the SDL2IF tool (see also [BFG+99, BGM01a]). After reducing the state space by applying static analysis techniques, relevant parts of the model are further transformed into a representation with DTPromela (Discrete Time Process Meta Language), which is the input language of DTSpin and an extension of Promela with a new data type for timers. Different from Spin, DTSpin executes the system with time slices, where time proceeds only if the system cannot perform an action (cf. *eager* in Sect. 11.2.4). Besides explicitly modeling the environment to turn the system from an open to a closed system [BDHS00], automatic code transformations can be applied to introduce spontaneous transitions emulating signals from the environment [SS01]. To verify only subparts of an SDL system, similar steps are performed. Transitions are assumed to be atomic and instantaneous. To deal with large and complex systems, concepts from *safe abstraction* are integrated besides the measures provided by Spin natively [SS01]. The presented approach is demonstrated in a case study with a wireless ATM MAC protocol called Mascara.

Model checking based on the translation of SDL into high-level Petri nets is presented by Aalto et al. in [AHV03]. The transformation step is done automatically by a tool called *SDL2PN*, which generates a single Petri net transition for each SDL statement (like SDL input, task statement, ...). For each variable, Petri net places are introduced, storing tokens with the current value of the variable. The input queue of a process is also carried by a token that is stored in a special place. For this, the authors present a queue data type and operators to access elements

[10] `http://spinroot.com/spin/whatispin.html`

of the queue. An additional Petri net place is also created for each SDL process to store the current program counter. The generated Petri net model is used as input for *Maria*[11], a model checking tool with an elaborated type system and with support for reachability analyses of large systems.

11.3 System Design with SDL and MSCs

Though Message Sequence Chart (MSC) is an own language standardized by ITU-T [Int11], it is often used in combination with SDL to specify system requirements or to illustrate particular use cases. Thus, most SDL tools support MSC as well. W.r.t. SDL real-time tasks, MSCs can, for instance, be used to identify task types of a system. This section outlines design techniques that automatically build SDL specifications by means of MSCs.

In [DGLS99], Dulz et al. present an automatic transformation of MSCs to SDL specifications to enable early performance estimations. The approach is motivated by the fact that the structure of a system — i.e., the distribution of functionality across blocks and processes — has an influence on the overall performance. To guide the transformation from MSCs to SDL, the authors propose an architecture specification language. In order to enable accurate performance evaluations, the designers additionally have to provide annotations in the MSCs. This includes the specification of CPU times and the amount of signal payload. The generated SDL specifications are synthetic and for evaluation purpose only, i.e., they are not intended for later reuse. They are automatically transformed into prototype implementations with the same tool chain that is later used for the final implementation. The implementations are executed on the target platform and monitored. The required system stimulus is generated by an artificial environment, enabling a black box evaluation of the system.

To guarantee consistency between design documents, [KV00] describes a method to automatically enrich existing SDL specifications with behavior specified in MSCs. A main objective of the approach is the extension of SDL process specifications without violating existing behavior. Starting with an existing SDL specification and the new behavior in MSCs, the enrichment process consists of two steps: First, the architecture is completed by adding new SDL processes and channels. Because the decision on the concrete placement of new structural elements cannot be made automatically, this step requires manual effort by the designers. In the second step, the behavior in the MSC — in particular, signal inputs and outputs — is automatically integrated into the extended SDL specification by a tool called MSC2SDL.

Enhancements of the MSC to SDL translation and details of the MSC2SDL tool are presented in [KZ05]. Different from the prior work, two additional MSC language features are supported now: High-level MSCs (HMSCs), which are available since MSC-96 [Int96b] and enable the composition of MSCs, and time constraints, which are introduced in MSC-2000 [Int99c] and are translated into continuous signals or enabling conditions. Though the MSC2SDL tool still requires the target architecture as input, it can now check the consistency between the given set of MSCs and the architecture, and can notify the designer if a mismatch is detected. Furthermore, the tool can automatically examine the implementability of the MSCs, because they may rely on global knowledge that is not available in an SDL system.

[11]http://www.tcs.hut.fi/Software/maria/index.en.html

11.4 Discussion

Though there is much related work on SDL, there are only few works concerning more predictable implementations. Instead, most works found in literature are about efficiency improvements of code generators, realization of time-critical parts in hardware, and system analyses. Whereas efficient software and hardware realizations have their focus on performance, analyses approaches have in common that they indeed assume a priority-based execution of the system, which is mostly based on process priorities, but do not describe how to enforce it in the implementation. In total, there are only very few proposals like Kolloch's MEDF scheduling [Kol02] that actually control transition execution orders in SDL implementations by SDL extensions on design level.

Many outlined proposals regarding efficient code generation – like activity threads [LK99, KLK00] – do not consider predictability of SDL implementations but performance improvement. To schedule SDL systems with priorities, it is often suggested to map an SDL system to processes of an RTOS and to apply a priority-based scheduler of the OS [Praar, IBMar]. Thereby, OS services like multi-thread support and preemption can be utilized, and a wide range of hardware is supported. Though a mapping of SDL systems to OS tasks is provided by many tools, they usually do not support all features of SDL. In particular, the usage of dynamic SDL process creation is often prohibited. In addition, scheduling decisions are very limited with such tight OS integrations, because priorities of RTOS processes can only be derived from the structure of the SDL system. Though the implementation of SDL real-time tasks does not provide the same extensive functionality like an OS, it does, on the contrary, not suffer from these limitations. SDL real-time tasks particularly allow dynamic process creation and enable the execution of a transition with the priority of the task it fulfills and not based on the transition's placement in the specification. If deadlines are violated, it is also state-of-the-practice to outsource parts of the system to hardware or hand-written code [LL05, SDM01]. Although this step is unavoidable in some cases due to efficiency reasons, it does not increase the transparency of the execution of an SDL system and is not consistent with holistic model-driven development processes (see also Sect. 7.2 about SDL-MDD).

Approaches for system analyses, verification, and model checking rely on the transformation of SDL specifications with diverse extensions to different representations and the application of approved methods and tools. To make this possible, such approaches often restrict the set of supported language elements. The methods can be classified according to their degree of realism: On the one hand, there are analyses methods to verify qualitative properties, where the designer decides on time progress. Here, an example is the specification of transition urgencies (eager, delayable, lazy) and the transformation into timed automata [BFG+99, BGM01a]. While such approaches are useful for the verification of the design, they are hardly relevant for implementations. On the other hand, there are approaches like QSDL [DHMC95, DHMC97] and TSDL [BB90, BB93], which are more related to implementations by considering hardware limitations. However, these approaches are also for analyses only and do not affect the behavior of SDL systems during runtime, i.e., they assume a particular system behavior but do not describe how to enforce it in an implementation. A further drawback of almost all presented analysis approaches is the neglect of SVM overhead.

Besides taking advantage of OS process priorities, some commercials tools like IBM's SDL Suite [IBMar] also provide built-in extensions for SDL process or signal type priorities. However, such prioritization measures are very limited, because priorities are statically assigned in the SDL specification. More sophisticated approaches are fixed transition priorities of Álvarez et al. [ÁDL+00, ÁDL+01, ÁDL+03] and MEDF scheduling of Kolloch [Kol02]. Similar to built-in priorities of tools, fixed transition priorities rely on structural elements only, thereby being less powerful and flexible than SDL real-time tasks. Additionally, it can be assumed that SDL real-time tasks produce less overhead, because priorities have not to be re-calculated after each transition execution as it is required by the execution model of Álvarez et al.. MEDF scheduling has most similarities with SDL real-time tasks but has only been applied to process and privilege local events. Distributed tasks are not considered and it is not obvious how to extend MEDF scheduling for this purpose. From a schedulability perspective, EDF-based priorities are superior to fixed priorities as transported by task signals of SDL real-time tasks.[12] This comes, however, to the detriment of larger overhead.

Most of the related works are no competitors of SDL real-time tasks, but are orthogonal and can be combined gainfully. A well-suited example is real-time signaling [KBCG11], which makes time-triggered task executions more flexible. By adopting energy-aware system design [GKLC09] and input port bounds [GGK07], the applicability of SDL real-time tasks in embedded systems could also be improved. A further example is SDL with RT-CORBA [DGLT09], where the two priority models of RT-CORBA (client- vs. server-defined) are currently realized differently in SDL. By incorporating SDL real-time tasks, a homogeneous integration of both models could be achieved. Since SDL real-time tasks realize a particular system task, it is also possible and beneficial to specify SDL real-time tasks in MSCs and to apply MSC to SDL transformations [KV00, KZ05]. Up to now, SDL real-time tasks have been semantically integrated into SDL, prototypically implemented, and evaluated by experiments, but there are no explicit schedulability tests or analyses available for systems with SDL real-time tasks. However, some of the outlined analysis techniques are good candidates to be adopted; for instance, RMA [ÁDL+03] or analyses with task networks [KF98]. Yet, if such methods are applied, gains must be traded off against inherent limitations w.r.t. supported language elements. Furthermore, additional overhead of the SVM implementation should be considered as well.

[12] Note that task signal priorities can also be derived from time-dependent expressions; e.g., by computing priorities based on SDL's `now`.

12. Chapter

Conclusions and Future Work

Depending on the concrete scenario of a networked system, communication primitives have not only to enable data exchange, but must additionally guarantee a desired degree of QoS. This particularly holds for distributed real-time systems, which place two key requirements on the communication system and its development: First, the behavior of the protocols must be deterministic, i.e., their delays and outcome must be predictable if the configuration is known. Second, implementations must comply with timing constraints that are required by the protocols in order to retain their determinism. This thesis is concerned with both requirements: The first part proposes a novel binary countdown protocol for wireless multi-hop networks, providing deterministic value-based medium arbitration and data transfer with bounded delays. The second part presents extensions of SDL to improve the language's applicability to distributed real-time systems and time-critical protocols. The following sections discuss the results of both parts, open questions, and future work.

12.1 Conclusions and Contributions

Part I of this thesis is concerned with ACTP, a decentralized binary countdown protocol for wireless multi-hop networks. As an instance of a binary countdown protocol, ACTP resolves contest among nodes dynamically, deterministically, value-based, and with constant delay. Furthermore, it is robust against node movements and does not rely on topological information like the neighborhood of nodes. Though ACTP is not the first representative of this protocol class, it outperforms previous ones w.r.t. flexibility and configurability and is, particularly, the first one with configurable and even network-wide application range. Different from previous binary countdown protocols, it explicitly comes with a cooperative operation mode, enabling the collision-protected transfer of bit sequences.

The key foundation of ACTP is its collision resistance, which is achieved by the incorporation of black bursts and treated in Chapter 2. In the course of this thesis, a generalization of black bursts is presented to enable the encoding of larger symbols. In addition, optimization measures are proposed to enhance their robustness. In this regard, the focus is on the widely-used IEEE 802.15.4-compliant [Ins11] CC 2420 transceiver [Tex07] and its CCA mechanism, which is crucial w.r.t. the detection of black bursts. Though this transceiver is not optimal concerning the efficiency of black burst implementations, it is sufficient as proof-of-concept. Furthermore, it offers configuration options to optimize the duration of black bursts, which are realized as irregular MAC frames with a duration of 160 µs, and their robustness, which could be increased by transmitting unique synchronization words.

Besides introducing ACTP's mode of operation, bit timing, and realizations in terms of an SDL specification and two manual implementations, Chapter 3 discusses possible limitations and threats in the application of ACTP. On the one hand, they are protocol-inherent and affect restricted applications of ACTP, where the multi-hop competing problem limits the meaning of the outcome of an ACTP run and does not produce the maximal possible number of winners. Yet, also with these limitations, restricted applications of ACTP guarantee that a winner is the only winner within arbitration radius. On the other hand, they are caused externally and arise if the single-network property is violated. In this case, external interference may cause false positives or false negatives, and can lead to wrong winners and the reception of corrupted bit sequences. In general, the validity of the single-network property is essential, yet some limitations – like asymmetrical links – are acceptable as discussed in Sect. 3.6.

Due to small data rates, ACTP is no all-purpose protocol and no replacement of established communication schemes like TDMA, which is, for instance, best suited if communication demands are strictly periodical. Nevertheless, ACTP provides attractive solutions to a wide range of problems of distributed (real-time) systems that are currently only insufficiently solved by available protocols. Examples are presented in Chapter 4 and comprise general problems like leader election and negative acknowledgments as well as control system-specific topics like Try-Once-Discard (TOD). In the context of TOD, ACTP disproves previous papers [TNT07, NL09], which wrongly conclude that TOD's wired solution, which is based on CAN [Int04], cannot be transferred to wireless systems. An important precondition of ACTP that is discussed in the course of its applications is synchronization, which is required to subdivide time into bit rounds and to start ACTP runs synchronously. In this regard, an integrated yet ACTP-independent solution is presented, which is based on the internal synchronization protocol BBS and the establishment of super slots, in which runs of ACTP are placed in virtual slot regions relative to synchronization phases. Different from existing binary countdown protocols, this approach offers configuration options to find an acceptable trade-off between overhead, synchronization offset, and duty cycling.

In experiments with the CC 2420 transceiver, it is shown that ACTP is actually implementable and that success rates of >99.9% are achievable even with customary hardware and violations of the single-network property. Corresponding results are presented in Chapter 5. By investigating the impact of overlapping black burst transmissions, it is furthermore shown that multiple black bursts interfere additively without indication of mutual extermination. In further experiments, proposed optimizations w.r.t. the detection of black bursts are validated and demonstrate that the average detection accuracy can be improved significantly by observing the gradient of the RSSI register. Thereby, black burst detection becomes less dependent from signal strengths and the CCA threshold of the transceiver.

In Part II, this thesis proposes the incorporation of real-time tasks, which are an established concept in the design of real-time systems [Kop97], into SDL. The rationales behind these steps have their origins in the mismatch between SDL's conceptual perfect world, where all SDL processes run in parallel, and the real-world, where transition executions have to be serialized. With SDL real-time tasks, the necessary serialization steps become better controllable and are less nondeterministic than in existing SDL implementations. Since controlling SDL real-time tasks should occur on design level within SDL specifications to maintain compliance with model-driven development processes like SDL-MDD, language extensions became necessary, which are devised in terms of task actions, task signals, task attributes, and task operators in Chapter 8, and are formally transferred into SDL's syntax and semantics in Appendices B and C. To take into account that SDL's application domains are networked systems, distributed real-time systems, and protocol engineering, a notion of distributed SDL real-time task has been introduced, which enables the specification of node-spanning tasks.

The implementation of SDL real-time tasks is covered by Chapter 9. In this regard, an annotation-based realization is presented, which is syntactically similar to the proposed SDL syntax extensions and allows the re-utilization of existing SDL tools. These annotations are the starting point for real-time task-aware implementations, which are automatically generated from SDL specifications and executed by a novel scheduling strategy called task scheduling. Since task scheduling works with implicitly available information of SDL real-time tasks, no additional implementation phase is required, which is usually applied for conventional SDL implementations to map SDL processes to processes of the target OS.

Different from existing priority-based SDL scheduling strategies, priorities of task scheduling are not statically derived from structural elements of the specification but deduced from SDL real-time tasks and dynamically associated with transition executions. As a consequence, the priority model of task scheduling is orthogonal to the static system structure and, in particular, not based on SDL process priorities, which are advocated as adequate prioritization measure by industrial SDL tools, but significantly less flexible. By enabling executions of the same transition with different priorities, the priority model of task scheduling furthermore copes with the fact that SDL processes are designed w.r.t. abstraction and reuse, and that, consequently, the same transition may be executed by different system tasks and with unequal urgencies. Task scheduling is devised as scheduling strategy with deferred preemption, i.e., a task can preempt another task with the granularity of transition executions. By supporting task suspension, delays caused by long-running low priority transitions can be avoided entirely. This also goes beyond the capabilities of existing scheduling solutions, which may indeed support full preemption, but must not violate the run-to-completion semantics of SDL transitions.

Altogether, SDL real-time tasks and task scheduling address all three sources of transition execution delays in an SDL system: Queueing delay by selecting transitions w.r.t. task signal priorities, serialization delay by introducing a global queue of task signals, and run-to-completion delay by enabling the suspension of tasks. Thereby, they outperform existing prioritization measures of SDL and tool extensions, which only address at most two sources of delay.

In Chapter 10, evaluation results are presented that demonstrate the implementability and practical benefits of real-time tasks in SDL. In particular, by two performance evaluations from the domain of control systems, it has been shown that task scheduling, although being slightly less efficient than conventional scheduling solutions, can reduce worst-case delays of critical tasks significantly. For instance, in a scenario with an Adaptive Cruise Control, the worst-case delay with task scheduling is only 40% of the delay that is achieved by a state-of-the-practice SDL scheduler. In a second example with an inverted pendulum and hardware-in-the-loop simulations, task scheduling could reduce the variability of execution times from 950 µs (process priority scheduling) to 180 µs. In summary, these results show that SDL implementations become substantially more predictable with SDL real-time tasks, which is the key requirement towards the application of SDL in (distributed) real-time systems.

12.2 Limitations, Open Issues, and Future Work

At the beginning of the work with ACTP, an objective was to develop the protocol in a model-driven way with SDL and to automatically generate an Imote 2 implementation. In this regard, the aim has not been achieved completely, since ACTP's SDL specification is applicable in simulations only and both available Imote 2 implementations are hand-written. However, the main reason of missing the aim is now no longer the unpredictability of SDL implementations, but their reduced performance, which makes an SDL-based implementation unattractive. The applied and extended tool chain – consisting of ConTraST, SdlRE, and SEnF – has been devised w.r.t. compliance with SDL's formal semantics and not to provide highly efficient SDL implementations. Thus, an open issue is the realization of SDL real-time tasks with a more efficient tool chain. Referring to this, a first step has been made in [BCGM14], where the industrial tool chain of PragmaDev has been incorporated into BiPS. Since the software is proprietary, it is, however, an open issue whether real-time tasks will actually be realized with this tool chain.

Regarding ACTP, further effort has to be paid to investigate violations of the single-network property and to develop countermeasures. Concerning the second point, a possible approach is redundancy increase, which can improve the protocol's success rates, but is only sketched in this thesis. Though ACTP is more susceptible to interference than ordinary transmission techniques, the problem is not inherent to ACTP. In this regard, [PC11], for instance, reports that the performance of WirelessHART [Int10] degrades with neighbored IEEE 802.11 networks [Ins12a]. Thus, there should be a general interest to allocate channels carefully and to separate best effort traffic from traffic with more stringent QoS constraints. A further conceivable improvement of ACTP is the incorporation of compression schemes. Thereby, arbitration delays of critical messages could be decreased by encoding higher priorities with less bits. Since the compression must comply with the mode of operation of binary countdown, finding an adequate compression scheme is, however, not straightforward. Another point of future work has been revealed by the evaluation results of Sect. 5.4 and affects the detection of black bursts with offsets larger than the maximal delay of the CCA mechanism. To guarantee the correct assignment of detected black bursts to bit rounds and to guarantee that the bounds of the synchronization offset of BBS are not exceeded, these outliers have to be identified and filtered.

Concerning SDL real-time tasks, a limitation is the single-thread implementation of task scheduling. Due to the increasing popularity of multi-core architectures, this limitation is no longer contemporary; in particular, since the task-based structuring of the execution opens up several opportunities, e.g., to assign a task to a dedicated core. Such steps implicate, however, a large amount of effort, since the entire used SDL tool chain is currently not designed for multi-thread executions. Though the presented SDL implementations are inappropriate for performance demanding protocols like ACTP, their improved predictability allows their use for time-critical higher-level functionality. In this regard, examples are control systems as demonstrated in the evaluation chapter. To enable their utilization in (hard) real-time systems, schedulability tests become necessary, which, in turn, require knowledge about platform-specific WCETs of transitions and the SVM implementation. Since these delays are very hard to determine analytically on modern hardware [But05], probabilistic WCETs may be an alternative and have exemplarily been derived in Sect. 10.4. However, further work is required to extend this approach to sets of real-time tasks and schedulability tests.

A Appendix

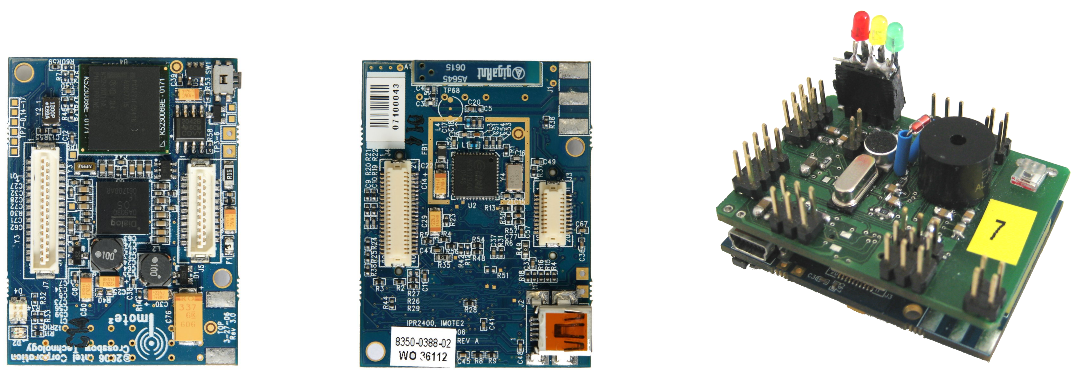
The Imote 2 Platform

The Imote 2 platform is an embedded system for Wireless Sensor Networks (WSNs). In the context of this thesis, it is used to provide evidence of the implementability of both ACTP and SDL real-time tasks and to conduct performance evaluations. Section A.1 presents a general survey of the hardware. Afterwards in Sect. A.2, its CC 2420 wireless transceiver is introduced in detail due to its importance for black bursts and ACTP.

A.1 Overview

The Imote 2 sensor node [MEMara] was originally developed by Intel[1] and last distributed by MEMSIC[2]. It is no longer available on market, yet it is still one of the most sophisticated wireless sensor nodes w.r.t. computational power and memory. In Fig. A.1, the node is shown with and without an extending sensor board. The Imote 2 is equipped with an XScale PXA 271 processor, supporting clock frequencies ranging from 13 Mhz to 416 MHz. The XScale CPU is an ARM[3]-based architecture and supports the ARM-v5TE instruction set. It has built-in memory of 256 KB SRAM and 32 MB SDRAM and non-volatile memory of 32 MB flash.

The PXA 271 has numerous input/output connectors to stake further peripheral hardware. This includes several General Purpose Input/Output (GPIO) pins, Universal Asynchronous



(a) Front (PXA 271 processor). (b) Back (CC 2420, antenna). (c) Imote 2 with sensor board.

Figure A.1: Imote 2 sensor node with and without additional sensor board.

[1] http://www.intel.de
[2] http://www.memsic.com/
[3] http://www.arm.com/

Receivers/Transmitters (UARTs), and Serial Peripheral Interface (SPI) buses. One of the SPI buses and several GPIO pins are used to interconnect the microprocessor and the CC 2420 wireless transceiver [Tex07].

A.2 The CC2420 Transceiver

The CC 2420 [Tex07] is an IEEE 802.15.4-compliant [Ins11] transceiver that is built and distributed by Texas Instruments. It operates on the 2.4 GHz Industrial, Scientific, and Medical (ISM) band and supports in total 16 orthogonal channels in between 2405 MHz and 2480 MHz, where each channel has a bandwidth of 2 MHz and a distance of 5 MHz to neighboring channels. Its data rate is 250 kbps and the maximal PHY payload 127 bytes. The output power of the transceiver is programmable and supports 32 levels with a maximal transmission power of 0 dBm. The receiver sensitivity of the CC 2420 is about -90 dBm, which is slightly better than the sensitivity threshold of -85 dBm that is prescribed by the IEEE 802.15.4 standard. The transceiver is typically used for transmission ranges of up to 30 m, whereas the actual transmission range depends on the used antenna and environment. With external antennas and line-of-sight, ranges of up to several hundred meters are possible [RCM+06].

An IEEE 802.15.4 standard-compliant transceiver must support turnaround times between reception and transmission mode of at most 12 symbol periods (192 µs). The CC 2420 transceiver meets this demand and provides even better values: According to its data sheet [Tex07], the turnaround time from reception to transmission mode can be shortened to 8 symbol periods (128 µs) by configuration. By means of experiments [Eng13], it could be shown that switching from transmission to reception mode also requires 128 µs only and is thus in contradiction to the data sheet.

The CC 2420 transceiver supports three CCA modes that are described in the IEEE 802.15.4 standard. They are discussed in more detail in Sect. 2.3.2. The maximal time to perform CCA is prescribed by IEEE 802.15.4 with 8 symbol periods (128 µs). Two of the CCA modes utilize an energy detector. This detector compares the 8 bit RSSI, which is averaged over 8 symbols and determined with an accuracy of ± 6 dB, against a threshold and reports on a busy medium if this threshold is exceeded. The current CCA state is provided by the CC 2420 transceiver via a special CCA output pin. On the Imote 2, this pin is connected to a GPIO pin of the PXA 271 processor. Thereby, implementations of black bursts and other protocols can use hardware interrupts, which are triggered by changes of the CCA pin, to be informed about a medium state change quickly.

To determine the quality of a link after receiving a frame, the CC 2420 transceiver provides two metrics: First, the RSSI value, which has been measured over the first 8 symbols after the SFD, is appended to the received frame. Second, Link Quality Indication (LQI) is supported, which has – in accordance with IEEE 802.15.4 – a length of 8 bits with at least 8 unique values. In general, the LQI can be implemented based on the RSSI, by a signal-to-noise ratio estimation, or by a combination of both. In the CC 2420 transceiver, LQI can be calculated on the basis of an averaged chip sequence correlation value, which is determined for the first 8 symbols after the SFD and represents some kind of chip error rate.

The IEEE 802.15.4 standard defines several variants of physical layers, where the CC 2420 transceiver implements the *Offset Quadrature Phase Shift Keying* (O-QPSK) PHY. This physical

Parameter	Value	Comment
$s_{minFrame}$	5 bytes	minimal size of a (irregular) frame, used by black bursts
r	250 kBit/s	transmission rate
$d_{switchTx}$	128 µs	switching time to send mode (data sheet)
$d'_{switchTx}$	129 µs	switching time to send mode (by experiments)
$d_{switchRx}$	192 µs	switching time to receive mode (data sheet)
$d'_{switchRx}$	128 µs	switching time to receive mode (by experiments)
$d_{accessRx}$	320 µs	switching time to receive mode until CCA is valid (data sheet)
d_{maxCCA}	128 µs	maximal delay until medium status change is detected
d_{pause}	16 µs	minimal pause between subsequent black bursts
e_{rx}	18.8 mA	energy consumption in reception mode
$e_{tx,max}$	17.4 mA	maximal energy consumption in transmission mode
e_{idle}	0.4 mA	energy consumption in idle mode

Table A.1: Summary of hardware characteristics of TI's CC 2420 transceiver. Values from experiments are taken from [Eng13, ECG14].

layer adopts Direct Sequence Spread Spectrum (DSSS), which maps 4 bits (= 1 symbol) to one of 16 nearly orthogonal chip sequences of length 32. Consequently, the symbol rate of the CC 2420 transceiver is 62.5 kSymbol/s and the chip rate 2000 kChip/s. The generated chip sequences are then modulated onto the carrier with O-QPSK, where chips with an even index are sent with the in-phase carrier and chips with an odd index with the quadrature-phase carrier. Since each chip is shaped into a half-sine pulse before modulating it onto the carrier, the modulation scheme of the CC 2420 is also called Minimum Shift Keying (MSK).

A summary of relevant hardware properties of the CC 2420 transceiver is provided by Table A.1. Values from experiments have been determined during a student's master thesis and are published in [Eng13, ECG14].

B Appendix

SDL Real-Time Tasks – Formal Syntax

This appendix summarizes extensions of SDL's concrete syntax to incorporate SDL real-time tasks formally. They are based on Basic SDL-2010 [Int12d].

To create *task signals*, the signal output and timer set statement of SDL have to be extended. In particular, language constructs to specify *task actions* are required in order to control SDL real-time tasks. Their syntactical incorporation is shown in the following listing, where existing rules are colored in gray and new rules in black. New terminal symbols are written in bold.

```
<output body item> ::= [<task action>] <signal identifier>  [<actual
    parameters>] [<task parameters>]  [<activation delay>] [<signal priority
    >]
<set statement> ::= [<task action>] set <set body> [<task parameters>]

<task action>  ::=  newTask | contTask
<task parameters> ::= <new task parameters> | <cont task parameters>
<new task parameters> ::= [<task type specification>] [<task signal
    priority>]
<cont task parameters> ::= [<task identifier>] [<task signal priority>]
<task type specification> ::=  type  <task type>
<task type> :: = <variable identifer> | <literal identifier>
<task identifier> ::= id < tid expression0>
<task signal priority> ::= prio < Natural expression>
```

Listing B.1: Incorporation of task actions into SDL's formal syntax.

Besides using task actions with plain SDL signals and timers, task actions can also be specified with real-time signaling [KBCG11], which goes beyond the built-in capabilities of activation delays in SDL-2010 [Int12d]. Extensions w.r.t. real-time signaling are, however, not included in Listing B.1.

Depending on the concrete `<task action>`, i.e., whether a real-time task is created or continued, it is either possible to specify `<task type specification>` (when creating a new task) or `<task identifier>` (when explicitly forking a task). This is, however, not prescribed by the syntax rule but has to be checked by an additional well-formedness condition. `<task type>`, `<task identifier>`, and `<task signal priority>` are built under consideration of possible data types. A full check of data type compatibility is, however, also outside the scope of grammars but requires semantical considerations.

Though it is less useful in practice, the BNF syntax allows to specify both an (ordinary) SDL signal priority (`<signal priority>`) and a task signal priority (`<task signal priority>`) for a single signal instance. Because (ordinary) signal priorities are only evaluated if two signals have the same arrival time [Int12d], their impact on the transition execution order is – in contrast to task signal priorities – very limited.

To obtain task attributes of consumed task signals from within an SDL transition, *task operators* are introduced in the formal syntax of SDL:

```
<imperative expression> ::= <now expression> | .... | <tid expression> | <
   tprio expression> | <ttype expression>
<tid expression>  ::= taskId
<tprio expression> ::= taskPrio
<ttype expression> ::= taskType
```

Listing B.2: Incorporation of task operators into SDL's formal syntax.

`<imperative expression>` is introduced in Basic SDL-2010 [Int12d] to access parts of the system state. By adding terminals to access task attributes, information about a consumed task signal that is known to the SVM can be obtained. Similar to SDL's `now`, the `taskId`, `taskPrio`, and `taskType` expressions are nullary functions, which can be used on the right-hand side of assignments or in expressions. For instance, by accessing the identifier of an SDL real-time task with `taskId` and by storing the return value in a variable of type `Tid`, tasks can be forked explicitly at a later point in time. For this purpose, the new `Tid` data type has to be introduced in SDL's syntax:

```
<sort>  ::= <basic sort>  | <pid sort> | <tid sort>

<tid sort> ::= <sort identifier>
```

Listing B.3: Syntactical incorporation of `Tid` sort.

In most cases, `<sort identifier>` is `Tid`. However, to support syntypes of `Tid`, other sort identifier words are possible for `<tid sort>`, too.

To suspend and resume SDL real-time tasks from within a specification, the following new grammar rules are required in SDL's syntax:

```
<suspend statement> ::= <suspend kind>  <suspend body>
<suspend kind> ::= suspendTaskType | suspendTaskPrio | suspendTaskId
<suspend body> ::= ( <suspend parameter> )
<suspend parameter> ::=  <task type> | < Natural expression> | < tid
   expression0>

<resume statement> ::= <resume kind>  [ <resume body> ]
<resume kind> ::=  resumeTaskType | resumeTaskPrio | resumeTaskId
<resume body> ::= ( <suspend parameter> )
```

Listing B.4: Incorporation of task suspension into SDL's formal syntax.

In total, six new terminals are introduced to suspend and resume real-time tasks in a flexible way. Depending on which tasks should be suspended/resumed, the functions accept a task type, a task id, or a priority value as parameter. An exception is `resumeTaskPrio` that does not take a parameter, yet the grammar would allow one.

To enable scheduling-aware system specifications, further extensions are made in the head symbol of the system. The following listing includes all strategies that are currently supported by the tool chain (see also Table 9.1 in Chapter 9):

```
<agent additional heading> ::= [<agent formal parameters>] [<scheduling
    parameters>]
<scheduling parameters> ::= strategy = <scheduling strategy>;
<scheduling strategy> ::= default | non-optimized | static-priorities |
    signals | tasks
```

Listing B.5: Support for scheduling-aware system specifications by selecting a scheduling strategy in the head symbol.

Some of the extensions and scheduling strategies are results of the master thesis [Chr10].[1] To support *task scheduling* and to enable comparisons with *signal-based FCFS*, terminals **tasks** and **signals** have been added. Regarding transition selection order and signal transfer between agents, **non-optimized** (*agent-based round robin*) is most compliant with SDL's semantics. Strategies **default** (*agent-based FCFS*) and **static-priorities** ((*process*) *priority scheduling*) are similar to **non-optimized** regarding transition selection order within an *SdlAgent*, but transfer signals directly between *SdlAgents* without scheduling *Links* and *SdlAgentSets* explicitly. The difference between **default** and **static-priorities** is not in the transition selection order of a single *SdlAgent* but in the serialization of transition executions of different *SdlAgents*. Signal-based FCFS also skips *Links* and *SdlAgentSets*, but additionally eliminates SDL priority inputs, since they are hard to accommodate with a system-wide FCFS signal order. Consequently, **signals** is not compliant with SDL's semantics regarding transition selection order in single *SdlAgents*. The same holds for strategy **tasks**, which changes transition execution orders significantly by executing transitions in the order of task signal priorities. Hence, it does not only influence the transition selection order between several *SdlAgents* but also changes the execution order within a single *SdlAgent*. Thus, strategy **tasks** does not only affect implementation aspects, but requires an incorporation into the formal semantics of SDL (see Appendix C).

[1] In the thesis [Chr10], the extensions are realized by annotations, which allow additional parameters for some scheduling strategies (not shown in listing). For task scheduling, no additional parameters are required, since the scheduling is controlled by task attributes only.

C Appendix

SDL Real-Time Tasks – Formal Semantics

The following adaptations and extensions of SDL's semantics have been applied to incorporate real-time tasks formally into SDL. The extensions are based on SDL-2000 Annex F3 [Int00], which is the latest available version of SDL's formal ASM semantics. It should be noted that the semantics does not directly refer to elements of the concrete syntax, which has been extended w.r.t. SDL real-time tasks in Appendix B. Instead, it relates to the abstract syntax tree AS1. The formal transformation from concrete syntax to AS1 has, however, been omitted, but the relation should be intuitive and self-explanatory.

In Listing C.1, extensions regarding scheduling-aware system design are shown. Existing parts of the semantics are colored in gray and new parts in black.

```
 1 SCHEDULINGSTRATEGY =def {nonOptimized, defaultScheduling, signalFifoScheduling,
       taskScheduling}
 2
 3 controlled schedStrategy: {system} → SCHEDULINGSTRATEGY
 4
 5 initially
 6   behaviour = rootNodeAS1.compile ∧
 7   if rootNodeAS1.s-Agent−definition ≠ undefined then
 8     system.nodeAS1 = rootNodeAS1.s-Agent−definition ∧
 9     system.owner = undefined ∧
10     system.agentMode1 = initialisation ∧
11     system.program = Agent−Set−Program
12     if rootNodeAS1.s-Schedule−definition ≠ undefined then
13       system.schedStrategy = rootNodeAS1.s-Schedule−definition
14     else
15       system.schedStrategy = defaultScheduling
16     endif
17   else
18     system.program = undefined
19   endif
```

Listing C.1: Incorporation of scheduling-aware system specifications into SDL's semantics.

In the listing, a new domain SCHEDULINGSTRATEGY is introduced in line 1 to distinguish between different scheduling algorithms. The scheduling strategy, which can be selected by the designer in the head symbol of the system, is here given as part of the abstract syntax tree *rootNodeAS1*. It is evaluated in line 12 and assigned to *schedStrategy*, which is an attribute of the system agent. If no strategy is selected, the corresponding part of the syntax tree is undefined and the default strategy is assigned.

Different from the five scheduling strategies that are available in the syntactical extensions in Listing B.5 and supported by SdlRE, there are only four strategies in the new ASM domain. The reason is that scheduling strategies **static-priorities** ($\text{Priorities}_{agents}$) and **default** ($\text{FCFS}_{agents}$), which are two different strategies in SdlRE, execute the same types of agents and have the same transition selection order within single *SdlAgents*. Thus, they are identical on the semantical level of SDL, since in the formal semantics – including the extensions of this chapter –, all *SdlAgents* run concurrently. They are subsumed by *defaultScheduling* here and their differentiation is left to the implementation. *defaultScheduling* is, however, different from *nonOptimized*, because it does not treat *Links* and *SdlAgentSets* as independent agents. Adaptations of the semantics required by *signalFifoScheduling* (signal-based FCFS strategy, $\text{FCFS}_{signals}$) are minor and consist of "treating priority inputs like regular signal inputs". In contrast, task scheduling – represented by element *taskScheduling* in the new ASM domain SCHEDULINGSTRATEGY –, task attributes, task operators, and task signals require additional modifications that are discussed in more detail below.

Listing C.2 shows extensions to incorporate task attributes.

```
 1 // New domains to capture task attributes
 2 shared domain TASKID
 3       initially  TASKID = { undefined }
 4 TASKPRIORITY =def NAT ∪ { undefined }
 5 TASKTYPE =def SDLLITERALS ∪ { undefined }
 6
 7 // New functions
 8 shared  taskId : SIGNALINST → TASKID
 9 controlled  taskId : SDLAGENT → TASKID
10 shared  taskPriority : SIGNALINST → TASKPRIORITY
11 controlled  taskPriority : SDLAGENT → TASKPRIORITY
12 shared  taskType : TASKID → TASKTYPE
13
14 INITAGENTCONTROLBLOCK(sa: SDLAGENT, ow:SDLAGENTSET, pa:PID, atd:
      Agent−type−definition) ≡
15   sa.nodeAS1 := atd
16   ...
17   sa.taskId := undefined
18   sa.taskPriority := undefined
19   ...
```

Listing C.2: Incorporation of task attributes into SDL's formal semantics.

Each of the task attributes is introduced by its own ASM domain. Additionally, one function on SDLAGENT and SIGNALINST, respectively, is devised for task id and task signal priority to store the corresponding attributes in SDL signals and in the state of an *SdlAgent*. Since task type is an attribute of a real-time task and not task signal-specific, it is introduced as function of TASKID. Domain TASKID and all functions on SIGNALINST are declared as **shared**, since they can be extended and changed by the environment of the SDL system. Other domains and functions are fully controlled by the system. During the creation of *SdlAgents*, the agents' task attributes are initialized with *undefined* (lines 17 and 18). They are updated when consuming task signals, which is later described in Listings C.6 and C.7.

Task attributes of task signals are set during the creation of the signal, which requires the following extensions:

```
TASKACTION =_def { newTask, contTask, undefined}

// Extensions of tuples to consider task action and task attributes
OUTPUT =_def SIGNAL × VALUELABEL* × VALUELABEL × VIAARG × TASKACTION ×
    TASKIDLABEL × TASKPRIOLABEL × TASKTYPELABEL × CONTINUELABEL
SET =_def TIMELABEL × TIMER × VALUELABEL* × TASKACTION × TASKIDLABEL ×
    TASKPRIOLABEL × TASKTYPELABEL × CONTINUELABEL

TASKIDLABEL =_def TASKTYPELABEL =_def TASKPRIOLABEL =_def VALUELABEL

// Changed macros regarding extension of ordinary signals
EVALOUTPUT(a:OUTPUT) ≡
  SIGNALOUTPUT(a.s-SIGNAL, values(a.s-VALUELABEL-seq, Self), value(a.s-VALUELABEL, Self), a.s-
      VIAARG,a.s-TASKACTION, value(a.s-TASKIDLABEL, Self), value(a.s-TASKPRIOLABEL, Self),
      value(a.s-TASKTYPELABEL, Self))
  Self.currentLabel := a.s-CONTINUELABEL

SIGNALOUTPUT(s:SIGNAL, vSeq:VALUE*, toArg:TOARG, viaArg:VIAARG, taskAction:TASKACTION,
    taskId:TASKID, taskPriority:TASKPRIORITY, taskType:TASKTYPE) ≡
  ...
    choose g: g ∈ Self.outgates ∧ Applicable(s, TOARG, VIAARG, g, undefined)
      extend PlainSignalInst with si
        ...
        if system.schedStrategy = taskScheduling then
          CONFIGTASK(si, taskAction, taskId, taskPriority,taskType)
        endif
        INSERT(si, now, g)
      endextend
    endchoose

// Changed macros regarding timers
EVALSET(a:SET) ≡
  SETTIMER(a.s-TIMER, values(a.s-VALUELABEL-seq, Self), value(a.s-TIMELABEL,Self), a.s-
      TASKACTION, value(a.s-TASKIDLABEL, Self), value(a.s-TASKPRIOLABEL, Self), value(a.s-
      TASKTYPELABEL, Self))
  Self.currentLabel := a.s-CONTINUELABEL

SETTIMER(tm:TIMER, vSeq :VALUE*, t:TIMER, taskAction:TASKACTION, taskId:TASKID, taskPriority:
    TASKPRIORITY, taskType:TASKTYPE) ≡
  let tmi = mk−TimerInst(Self.self, tm, vSeq ) in
    ...
    if system.schedStrategy = taskScheduling then
      CONFIGTASK(tmi, taskAction, taskId, taskPriority, taskType)
    endif
  endlet

// New help macro to insert task attributes into signals
```

```
40 ConfigTask(si:SignalInst, taskAction:TaskAction, taskId:TaskId, taskPriority:TaskPriority,
      taskType:TaskType) ≡
41  if  taskAction = newTask then
42    extend TaskId with taskId
43      taskId.taskType := taskType
44      si.taskId := taskId
45    endextend
46    si.taskPriority := taskPriority
47  elseif taskAction = contTask then
48    if taskId = undefined then
49      // implicit task forking
50      si.taskId := Self.taskId
51      if taskPriority ≠ undefined then
52        si.taskPriority := taskPriority
53      else
54        si.taskPriority := Self.taskPriority
55      endif
56    else
57      // explicit task forking
58      si.taskId := taskId
59      si.taskPriority := taskPriority
60    endif
61  endif
```

Listing C.3: Extensions of signal outputs and SDL timers to support task signals.

In line 1, a new domain is introduced containing keywords for task actions. Here, *undefined* is used if no task action is given in the SDL specification, i.e., if a plain SDL signal or timer is created. In addition, tuples of output and timer set statements are extended to cover task actions and task attributes. Since task attributes may be given as expression (e.g., **taskPrio + variable_name**), they are first introduced as ValueLabel and derived by the value function (lines 11 and 28). In macros SignalOutput and SetTimer, it is then checked whether task scheduling is selected in the specification. If this is the case, task attributes are stored in the signal by applying macro ConfigTask. If task scheduling is not used, task attributes in signals remain *undefined* and are, consequently, not evaluated during transition selection (see Listing C.7 below).

In macro ConfigTask, domain TaskId is extended with a new element if a new task is created. Additionally, a type is assigned to the new task (possibly *undefined*) and the task priority is copied to the signal. Note that if no signal priority is provided, *undefined* is assigned in line 46, which represents the lowest possible priority. If an existing task is continued, it is distinguished between implicit and explicit task forking. With implicit task forking, the currently executed task is continued (lines 49 to 55). If no priority is provided, the priority of the signal is set to the priority that is stored in the state of the *SdlAgent* and corresponds to the task signal priority of the lastly consumed signal. In case of explicit task forking (lines 57 to 59), the task with the provided id is continued. If the task should be continued with a priority different from *undefined*, an explicit priority has to be provided. Since the type of a task is fixed, it cannot be changed when forking tasks.

Task operators can be invoked within a transition to obtain task attributes of the lastly consumed signal. Their incorporation is presented in Listing C.4:

```
SYSTEMVALUE =def VALUEKIND × CONTINUELABEL
VALUEKIND =def { kNow, kSelf, kParent, kOffspring, kSender, kTaskId, kTaskType, kTaskPriority }

EVALSYSTEMVALUE(a: SYSTEMVALUE) ≡
  value(Self.currentLabel, Self) :=
    case a.s-VALUEKIND of
    | kNow: now.semvalue
    ...
    | kSender: Self.sender.semvalue
    | kTaskId: Self.taskId
    | kTaskType: Self.taskId.taskType
    | kTaskPriority: Self.taskPriority
    otherwise undefined
    endcase
  Self.currentLabel := a.s-CONTINUELABEL
```

Listing C.4: Incorporation of task operators into SDL's formal semantics.

In the listing, domain VALUEKIND is extended with a new element for each task operator. They are used as argument of macro EVALSYSTEMVALUE to obtain the corresponding task attribute, which is stored in the state of the *SdlAgent* when consuming a task signal. The transfer of task attributes from signals to the state of an agent is part of Listing C.7.

To realize suspension and resumption of real-time tasks, several extensions regarding transition selection and the storage of information about suspended tasks become necessary. To begin with, Listing C.5 shows the interface to access and modify the lists of suspended tasks. Ignoring suspended tasks during transition selection is covered in Listing C.6.

```
// New system-wide functions to store suspended task ids, types, or priority.
// Note that functions return by default "undefined" if no value is set.
shared suspended: TASKID → BOOLEAN
shared suspended: TASKTYPE → BOOLEAN
shared taskPrioThreshold: → TASKPRIORITY

// New actions for suspension and resumption
ACTION =def VAR ∪ ... ∪ LEAVESTATENODE ∪ SUSPEND ∪ RESUME

SUSPEND =def TASKIDLABEL × TASKTYPELABEL × TASKPRIOLABEL × CONTINUELABEL
RESUME =def TASKIDLABEL × TASKTYPELABEL × CONTINUELABEL

// Consider new actions
EVAL(a: ACTION) ≡
  if a ∈ VAR then EVALVAR(a)
  ...
  elseif a ∈ LEAVESTATENODE then EVALLEAVESTATENODE(a)
  elseif a ∈ SUSPEND then EVALSUSPEND(a)
  elseif a ∈ RESUME then EVALRESUME(a)
  endif

```

```
22 // Suspend task id, type, or priority depending on invoked function
23 EVALSUSPEND(a: SUSPEND) ≡
24   SUSPENDTASK(value(a.s-TASKIDLABEL, Self), value(a.s-TASKTYPELABEL, Self), value(a.s-
         TASKPRIOLABEL, Self))
25   Self.currentLabel := a.s-CONTINUELABEL
26
27 SUSPENDTASK(tid:TASKID, ttype:TASKTYPE, tprio:TASKPRIORITY) ≡
28   if tid ≠ undefined then
29     // suspendTaskId is invoked
30     tid.suspended := True
31   elseif ttype ≠ undefined
32     // suspendTaskType is invoked
33     ttype.suspended := True
34   else
35     // suspendTaskPrio is invoked
36     taskPrioThreshold := tprio
37   endif
38
39 // Resume task based on its id, type, or priority
40 EVALRESUME(a: RESUME) ≡
41   RESUMETASK(value(a.s-TASKIDLABEL, Self), value(a.s-TASKTYPELABEL, Self))
42   Self.currentLabel := a.s-CONTINUELABEL
43
44 RESUMETASK(tid:TASKID, ttype:TASKTYPE) ≡
45   if tid ≠ undefined then
46     // resumeTaskId is invoked
47     tid.suspended := False
48   elseif ttype ≠ undefined
49     // resumeTaskType is invoked
50     ttype.suspended := False
51   else
52     // resumeTaskPrio is invoked
53     taskPrioThreshold := undefined
54   endif
```

Listing C.5: Introducing further actions for task suspension and resumption.

In total, three new functions are introduced in lines 3 to 5 to store suspended tasks ids, task types, and task signal priorities. These functions are **shared**, because they are observed and modified by all ASM agents. They are accessed via two new actions, SUSPEND and RESUME. Parameters of both actions include task id and task type. SUSPEND additionally has a priority parameter to suspend task signals by their priority.

Comparing the semantical extensions in Listing C.5 with the syntactical extensions in Appendix B, six terminal function symbols in the syntax are mapped to two ASM macros in the semantics. Depending on which function is invoked in the SDL specification, one argument of SUSPENDTASK and RESUMETASK, respectively, is defined and other arguments are given as *undefined*. Similar to the creation of task signals in Listing C.3, arguments of the macros are provided as VALUELABEL first and evaluated with the *value* function (lines 24 and 41). In macros SUSPENDTASK and RESUMETASK, it is checked which argument is actually provided and the corresponding suspension state of the task id, type, or priority threshold is updated.

The major impact of SDL real-time tasks is on transition selection, which has to privilege task signals and to order them by their priority. The relationship between task signal selection and existing selection phases of *SdlAgents* was already outlined in Sect. 8.3.3.5 and illustrated in Fig. 8.10. In the figure, the selection of transitions consuming task signals is introduced as a new activity phase (called `selectTaskInput`), which precedes the selection of priority inputs. Since the activity phase `startSelection`, which is the entry point of the transition selection phase, only initializes some variables of *SdlAgents*, `selectTaskInput` is the first activity phase in steady state – i.e., after running the start transition of an *SdlAgent*— that actually searches for firable transitions. To execute the new activity phase and to manage task attributes of consumed task signals, the following extensions of the `startSelection` activity phase are necessary:

```
AGENTMODE =_def { ... , selectTaskInput, ... } // New element added

// New function for task signal queue (sorted by task signal priorities)
controlled taskSignalsChecked: SDLAGENT → SIGNALINST*

SELECTTRANSITIONSTARTPHASE ≡
  if Self.currentExceptionInst ≠ undefined then
  ...
  else
    Self.inputPortChecked := Self.inport.queue
    // Store all task signals ordered by priority in taskSignalsChecked
    Self.taskSignalsChecked := collectCurrentTaskSignals(Self.inport.queue, empty)
    // Reset task attributes of agent
    Self.taskId := undefined
    Self.taskPriority := undefined
    // Search first for task signals, not for priority inputs
    Self.agentMode3 := ~~selectPriorityInput~~ selectTaskInput
    Self.agentMode4 := startPhase
  endif

// Function to copy task signals from input port to temporary task signal queue
collectCurrentTaskSignals(inQueue: SIGNALINST*, taskQueue: SIGNALINST* ): SIGNALINST* =_def
  if inQueue = empty then taskQueue
  else collectCurrentTaskSignals(inQueue.tail, insertTaskSignal(inQueue.head, taskQueue))
  endif

// Function to insert a task signal in a queue of task signals sorted by task priority
insertTaskSignal(si: SIGNALINST, taskQueue: SIGNALINST*): SIGNALINST* =_def
  if si.taskId = undefined then taskQueue // signal is no task signal
  elseif
    // Insert task signal on correct position if task is not suspended
    if ¬ isSuspended(si) then
      if taskQueue = empty then <si> ∩ taskQueue
      elseif si.taskPriority = undefined then
        <taskQueue.head> ∩ insertTaskSignal(si, taskQueue.tail)
      elseif taskQueue.head.taskPriority = undefined then <si> ∩ taskQueue
```

```
37          elseif  si.taskPriority  < taskQueue.head.taskPriority then <si> ∩ taskQueue
38          else   <taskQueue.head> ∩ insertTaskSignal(si, taskQueue.tail)
39          endif
40        else
41          taskQueue
42        endif
43      endif
44
45  // Function to check whether the signal's task is suspended
46  isSuspended(si: SIGNALINST): BOOLEAN $=_{\text{def}}$
47    // Check suspension by (0) is task signal? (1) id, (2) type, (3) priority
48    if  si.taskId = undefined then False
49    elseif  si.taskId.suspended = True then True
50    elseif  si.taskId.taskType.suspended = True then True
51    elseif  taskPrioThreshold $\neq$ undefined $\wedge$ ( si.taskPriority  = undefined $\vee$ si.taskPriority  $\geq$
          taskPrioThreshold) then True
52    else  False
53    endif
```

Listing C.6: Collect task signals in a separate queue before start of transition selection.

In addition to the introduction of a new agent mode, a new signal queue is introduced in line 4 to temporarily store all task signals of the *SdlAgent*. This list is filled in line 12 before start of transition selection by calling the recursive function *collectCurrentTaskSignals*, which is defined in lines 22 to 25 and takes two arguments: The first argument (*inQueue*) contains signals – plain SDL signals as well as task signals –, which are stored in the *SdlAgent*'s input port but not yet in the temporary task signal queue. The second argument (*taskQueue*) is a list of task signals, which have already been copied from the *SdlAgent*'s input port, and is sorted by priority. This argument is also the result of the function, if there are no more signals in *inQueue*.

The function *insertTaskSignal* is used in *collectCurrentTaskSignals* to insert a signal into the list of task signals and to sort the list by priority. The function is also recursive and takes the signal to be added and the task signal queue as parameters. A signal is only inserted, if it is actually a task signal (see line 29) and if the signal is not suspended by its task id, type, or priority (see line 32). This is checked by help function *isSuspended* in lines 46 to 53. When inserting a task signal into the queue (lines 33-38), it has to be considered that task signals may not necessarily have a priority assigned. In this case, *taskPriority* is *undefined*, which is synonymous to the lowest possible priority.

The actual selection of a transition is described in the subactivities of `selectTaskInput` (see Fig. C.1). The formal definition can be found in Listing C.7. The subactivities are very similar to the transition selection with plain SDL signals; in particular, regarding gathering SDL states and possible transitions. Thus, some of the submacros in Listing C.7 could be transferred from the standard [Int00]. Different from transition selection with plain SDL signals, where spontaneous transitions are possible at all times, spontaneous transitions are ignored during transition selection with task signals. Instead, `selectTaskInput` has in total only three subactivities: A start phase to initialize required functions, a selection phase to find possible transitions, and an evaluation phase to examine enabling conditions.

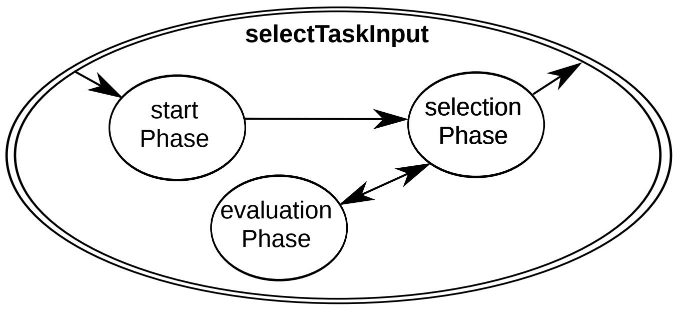


Figure C.1: Subactivities of `selectTaskInput`.

```
SELECTTRANSITION ≡
  ...
  elseif $Self.agentMode3 = selectTaskInput$ then
    SELECTTASKINPUT
  ...

SELECTTASKINPUT ≡
  if $Self.agentMode4 = startPhase$ then
    SELTASKINPUTSTARTPHASE
  elseif $Self.agentMode4 = selectionPhase$ then
    SELTASKINPUTSELECTIONPHASE
  elseif $Self.agentMode4 = evaluationPhase$ then
    SELTASKINPUTEVALUATIONPHASE
  endif

SELTASKINPUTSTARTPHASE ≡
  // Start searching for consumable task signals if task signal queue is not empty.
  if $Self.taskSignalsChecked \neq empty$ then
    $Self.signalChecked := Self.taskSignalsChecked.head$
    $Self.stateNodesToBeChecked := collectCurrentSubStates(\{Self.topStateNode\})$
    $Self.stateNodeChecked := undefined$
    $Self.transitionsToBeChecked := \emptyset$
    $Self.agentMode4 := selectionPhase$
  else
    $Self.agentMode3 := selectPriorityInput$
    $Self.agentMode4 := startPhase$
  endif

SELTASKINPUTSELECTIONPHASE ≡
  if $Self.stateNodeChecked = undefined$ then
    NEXTSTATENODETOBECHECKED
  elseif $Self.transitionsToBeChecked \neq \emptyset$ then
    choose $t$: $t \in Self.transitionsToBeChecked$
      $Self.transitionsToBeChecked := Self.transitionsToBeChecked \setminus \{t\}$
      if $t$.s-$LABEL \neq undefined$ then
        EVALUATEENABLINGCONDITION$(t)$
      else
```

```
        // Transition consuming task signal found
        Self.currentSignalInst := Self.signalChecked
        Self.sender := Self.signalChecked.signalSender
        // Copy task attributes to sdl agent's attributes
        Self.taskId := Self.signalChecked.taskId
        Self.taskPriority := Self.signalChecked.taskPriority
        // Delete task signal from inport.
        // No deletion from Self.taskSignalsChecked required, since
        // this queue is rebuilt during next transition selection.
        DELETE(Self.signalChecked, Self.inport)
        TRANSITIONFOUND(t)
      endif
    endchoose
  else
    Self.stateNodeChecked := undefined
  endif

  where
    // Identical to EvaluateEnablingCondition in SelInputSelectionPhase
    // (SDL-2000 Z.100 Annex F.3)
    EVALUATEENABLINGCONDITION(t:TRANSITION) ≡
      Self.transitionChecked := t
      Self.currentStateId := Self.stateNodeChecked.parentStateNode.stateId
      Self.currentLabel := t.s-LABEL
      Self.agentMode4 := evaluationPhase

    NEXTSTATENODETOBECHECKED ≡
      if Self.stateNodesToBeChecked $\neq \emptyset$ then
        if Self.stateNodeChecked = undefined then
          SELECTNEXTSTATENODE
        else
          CHECKFORINHERITEDSTATENODES
        endif
      else
        // No transition found. Would be an implicit consumption for regular signals.
        // But for task signal: Just delete from temporary task signal queue
        // and let signal in agent's inport.
        DELETE(Self.signalChecked, Self.taskSignalsChecked)
        NEXTSIGNALTOBECHECKED
        Self.stateNodesToBeChecked := collectCurrentSubStates({Self.topStateNode})
        Self.stateNodeChecked := undefined
      endif

    SELECTNEXTSTATENODE ≡
      let sn = Self.stateNodesToBeChecked.selectNextStateNode in
        if sn.stateNodeKind = procedureNode then
          Self.stateNodesToBeChecked := Self.stateNodesToBeChecked \ collectCurrentSubStates({sn.
              getPreviousStatePartition})
        elseif sn.stateNodeKind = statePartition then
          Self.stateNodesToBeChecked := Self.stateNodesToBeChecked \ {sn}
        elseif sn.stateNodeKind = stateNode then
          Self.stateNodeChecked := sn
          Self.stateNodesToBeChecked := Self.stateNodesToBeChecked $\setminus$ {sn}
          Self.transitionsToBeChecked := { t $\in$ (sn.stateTransitions.inputTransitions $\cup$ sn.
              stateTransitions.priorityInputTransitions): t.s-SIGNAL = Self.signalChecked.
              signalType }
          if Self.signalChecked.signalType $\in$ sn.nodeAS1.s-Save−signalset then
            // No save required, since task signals in task signal queue are
            // copied from inport. I.e., delete signal from task signal queue.
            DELETE(Self.signalChecked, Self.taskSignalsChecked)
          endif
        endif
      endlet

    // Identical to CheckForInheritedStateNodes in SelInputSelectionPhase
    // (SDL-2000 Z.100 Annex F.3)
    CHECKFORINHERITEDSTATENODES $\equiv$
      Self.stateNodeChecked := undefined
      let sn1 = Self.stateNodeChecked in
        if Self.signalChecked.signalType $\in$ { in.s-Signal−identifier | in $\in$ sn1.nodeAS1.s-
            Input−node−set } $\cup$ sn1.nodeAS1.s-Save−signalset then
          Self.stateNodesToBeChecked := Self.stateNodesToBeChecked $\setminus$ { sn2 $\in$ Self.
              stateNodesToBeChecked | InheritsFrom(sn1,sn2) }
        endif
      endlet

    NEXTSIGNALTOBECHECKED $\equiv$
      // The next task signal to be checked is always the head of the task signal
      // queue, since task signals are removed from task signal queue if no
      // transition is found or the signal is saved.
      if Self.taskSignalsChecked $\neq$ empty then
        Self.signalChecked := Self.taskSignalsChecked.head
      else
        Self.agentMode3 := selectPriorityInput
        Self.agentMode4 := startPhase
      endif
  endwhere

SELTASKINPUTEVALUATIONPHASE $\equiv$
  if Self.currentLabel $\neq$ undefined then
    choose b: b $\in$ behaviour $\wedge$ b.s-LABEL = Self.currentLabel
      EVAL(b.s-PRIMITIVE)
    endchoose
  elseif semvalue(value(Self.transitionChecked.s-LABEL,Self)) then
    // Transition consuming task signal found
    Self.currentSignalInst := Self.signalChecked
    Self.sender := Self.signalChecked.signalSender
    // Copy task attributes to sdl agent's attributes
    Self.taskId := Self.signalChecked.taskId
    Self.taskPriority := Self.signalChecked.taskPriority
```

```
133      // Delete task signal from inport
134      DELETE(Self.signalChecked, Self.inport)
135      TRANSITIONFOUND(Self.transitionChecked)
136    else
137      Self.agentMode4 := selectionPhase
138    endif
```

Listing C.7: Searching a consumable task signal with highest priority.

In lines 16 to 27, it is checked in the start phase if there is a task signal in the *SdlAgent*'s input port, for which possible transitions have to be searched. For this purpose, the *taskSignalsChecked* queue, which was generated in macro SELECTTRANSITIONSTARTPHASE before (see Listing C.6), is inspected from high priority task signals to low priority task signals. Since this queue is temporary and contains copies of signals of the agent's input port, signals are removed if the signal is saved in the agent's current state (see line 94) or if there is no matching transition (see line 75). If there is no (further) task signal, the agent continues with searching for priority inputs (see line 25).

Compared with transition selection for plain SDL signals, which is mainly defined in macros SELINPUTSELECTIONPHASE and SELINPUTEVALUATIONPHASE in [Int00], SELTASKINPUTSELECTIONPHASE and SELTASKINPUTEVALUATIONPHASE differ as follows: First, not the agent's input port but the temporary task signal queue is processed (see lines 19 and 114). Second, not only regular signal inputs but also priority inputs are considered when searching for available transitions (see line 90), since a task signal may also be consumed by a priority input. Third, task attributes are copied from the signal to the *SdlAgent*'s state when a matching transition is found (see lines 42, 43, 131, and 132). In this case, the signal is also removed from the agent's input port (see lines 47 and 134).

Only minor changes are required in existing activity phases of transition selection, which follow the activity phase `selectTaskInput`. By treating task signals and non-task signals in the same way, a task signal may particularly also be consumed implicitly in the existing SELECTTASKINPUT macro. To avoid implicit consumptions of suspended task signals, the existing SELINPUTSELECTIONPHASE macro (see Z.100 Annex F.3 of SDL-2000 [Int00]) has to be extended by the invocation of *isSuspended* (see line 46 in Listing C.6) to check whether a signal should really be consumed implicitly. Besides this minor extension, which is not shown in this appendix, no further modifications of existing activity phases are required.

List of Figures

List of Tables

Bibliography

[ABS01] Ahmad Alkhodre, Jean-Philippe Babau, and Jean-Jacques Schwarz. Preparing SDL Code Generation for Real-time Embedded Systems Modeling. In *IEEE Real-Time Embedded System Workshop*, december 2001.

[ÁDL+00] José M. Álvarez, Manuel Díaz, Luis Llopis, Ernesto Pimentel, and José M. Troya. SDL and Hard Real-Time Systems: New Design and Analyze Techniques. In Sherratt [She00], pages 92–107.

[ÁDL+01] José M. Álvarez, Manuel Díaz, Luis Llopis, Ernesto Pimentel, and José M. Troya. Deriving Hard Real-time Embedded Systems Implementations Directly from SDL Specifications. In Madsen et al. [MHH01], pages 128–133.

[ÁDL+03] José M. Álvarez, Manuel Díaz, Luis Llopis, Ernesto Pimentel, and José M. Troya. Integrating Schedulability Analysis and Design Techniques in SDL. *Real-Time Systems*, 24(3):267–302, 2003.

[AHV03] Annikka Aalto, Nisse Husberg, and Kimmo Varpaaniemi. Automatic Formal Model Generation and Analysis of SDL. In Reed and Reed [RR03], pages 285–299.

[AKLN99] R. Ashour, F. Khendek, and T. Le-Ngoc. Formal Description of Real-time Systems Using SDL. In *Real-Time Computing Systems and Applications, 1999. RTCSA '99. Sixth International Conference on*, pages 190 –197, 1999.

[ALSU07] Alfred V. Aho, Monica S. Lam, Ravi Sethi, and Jeffrey D. Ullman. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, Boston, 2 edition, 2007.

[APTG11] Björn Andersson, Nuno Pereira, Eduardo Tovar, and Ricardo Gomes. Using a Prioritized Medium Access Control Protocol for Incrementally Obtaining an Interpolation of Sensor Readings. Technical Report HURRAY-TR-110101, Polytechnique Institute of Porto (ISEP-IPP), January 2011.

[ASSC02] Ian F. Akyildiz, Weilian Su, Yogesh Sankarasubramaniam, and Erdal Cayirci. Wireless Sensor Networks: A Survey. *Computer Networks*, 38(4):393–422, 2002.

[AT05] Björn Andersson and Eduardo Tovar. Static-Priority Scheduling of Sporadic Messages on a Wireless Channel. In James H. Anderson, Giuseppe Prencipe, and Roger Wattenhofer, editors, *OPODIS*, volume 3974 of *Lecture Notes in Computer Science*, pages 322–333. Springer, 2005.

[Atm09] Atmel. Datasheet AT86RF230. http://www.atmel.com/dyn/resources/prod_documents/doc5131.pdf, 2009.

[BB90] Falko Bause and Peter Buchholz. Protocol Analysis Using a Timed Version of SDL. In Juan Quemada, José A. Mañas, and Enrique Vázquez, editors, *FORTE*, pages 239–254. North-Holland, 1990.

[BB93] Falko Bause and Peter Buchholz. Qualitative und Quantitative Analysis of Timed SDL Specifications. In Nina Gerner, Heinz-Gerd Hegering, and Joachim Swoboda, editors, *Kommunikation in Verteilten Systemen*, Informatik Aktuell, pages 486–500. Springer, 1993.

[BBCG11] Philipp Becker, Martin Birtel, Dennis Christmann, and Reinhard Gotzhein. Black-Burst-Based Quality-of-Service Routing (BBQR) for Wireless Ad-Hoc Networks. In *11th International Conference on New Technologies in Distributed Systems (NOTERE'2011), Paris, France*, pages 1–8. IEEE, 2011.

[BBY13] Giorgio C. Buttazzo, Marko Bertogna, and Gang Yao. Limited Preemptive Scheduling for Real-Time Systems. A Survey. *IEEE Trans. Industrial Informatics*, 9(1):3–15, 2013.

[BCG09] Philipp Becker, Dennis Christmann, and Reinhard Gotzhein. Model-Driven Development of Time-critical Protocols with SDL-MDD. In Reed et al. [RBG09], pages 34–52.

[BCG$^+$13] Tobias Braun, Dennis Christmann, Reinhard Gotzhein, Anuschka Igel, Thomas Forster, and Thomas Kuhn. Virtual Prototyping with Feral – Adaptation and Application of a Simulator Framework. In *The 24th IASTED International Conference on Modelling and Simulation*, 2013.

[BCG$^+$14] T. Braun, D. Christmann, R. Gotzhein, A. Igel, and T. Kuhn. Virtual Prototyping of Distributed Embedded Systems with FERAL. *International Journal of Modelling and Simulation*, 2(34), 2014.

[BCGI12] Tobias Braun, Dennis Christmann, Reinhard Gotzhein, and Anuschka Igel. Model-driven Engineering of Networked Ambient Systems with SDL-MDD. *Procedia Computer Science*, 10(0):490 – 498, 2012. ANT 2012 and MobiWIS 2012.

[BCGM14] Tobias Braun, Dennis Christmann, Reinhard Gotzhein, and Alexander Mater. SDL Implementations for Wireless Sensor Networks - Incorporation of PragmaDev's RTDS into the Deterministic Protocol Stack BiPS. In Daniel Amyot, Pau Fonseca i Casas, and Gunter Mussbacher, editors, *System Analysis and Modeling: Models and Reusability - 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings*, volume 8769 of *Lecture Notes in Computer Science*, pages 271–286. Springer, 2014.

[BCRL00] F. Boutet, E. Choveau, G. Rieux, and Y. Lejeune. Scheduling in SDL Simulation. Application to Future Air Navigation Systems. In Sherratt [She00], pages 320–332.

[BDHS00] Dragan Bosnacki, Dennis Dams, Leszek Holenderski, and Natalia Sidorova. Model Checking SDL with Spin. In Susanne Graf and Michael I. Schwartzbach, editors, *TACAS*, volume 1785 of *Lecture Notes in Computer Science*, pages 363–377. Springer, 2000.

[BFG⁺99] Marius Bozga, Jean-Claude Fernandez, Lucian Ghirvu, Susanne Graf, Jean-Pierre Krimm, Laurent Mounier, and Joseph Sifakis. IF: An Intermediate Representation for SDL and its Applications. In *SDL Forum*, pages 423–440, 1999.

[BGK⁺00] Marius Bozga, Susanne Graf, Alain Kerbrat, Laurent Mounier, Iulian Ober, and Daniel Vincent. SDL for Real-Time: What is Missing? In Sherratt [She00], pages 108–121.

[BGK07] Philipp Becker, Reinhard Gotzhein, and Thomas Kuhn. MacZ – A Quality-of-Service MAC Layer for Ad-hoc Networks. In *HIS '07: Proceedings of the 7th International Conference on Hybrid Intelligent Systems*, pages 277–282. IEEE Computer Society, Sept. 2007.

[BGK08] Philipp Becker, Reinhard Gotzhein, and Thomas Kuhn. Model-driven Performance Simulation of Self-organizing Systems with PartsSim. *PIK – Praxis der Informationsverarbeitung und Kommunikation*, 31(1):45–50, 1/2008.

[BGK14] Tobias Braun, Reinhard Gotzhein, and Thomas Kuhn. Mode-based Scheduling with Fast Mode-Signaling – A Method for Efficient Usage of Network Time Slots. *Journal of Advances in Computer Networks (JACN)*, 2:48–57, 2014.

[BGM01a] Marius Bozga, Susanne Graf, and Laurent Mounier. Automated Validation of Distributed Software using the IF Environment. *Electr. Notes Theor. Comput. Sci.*, 55(3):370–381, 2001.

[BGM⁺01b] Marius Bozga, Susanne Graf, Laurent Mounier, Iulian Ober, Jean-Luc Roux, and Daniel Vincent. Timed Extensions for SDL. In Reed and Reed [RR01], pages 223–240.

[BGS07] M. Bertocco, G. Gamba, , and A. Sona. Experimental Optimization of CCA Thresholds in Wireless Sensor Networks in the Presence of Interference. In *Proceedings of the IEEE EMC Europe*, 2007.

[BGW10] Tobias Braun, Reinhard Gotzhein, and Matthias Wiebel. Integration of FlexRay into the SDL-Model-Driven Development Approach. In Frank Alexander Kraemer and Peter Herrmann, editors, *System Analysis and Modeling: About Models – SAM 2010, 6th International Workshop on System Analysis and Modeling, Oslo, Norway, October 4–5, 2010, Co-located with MODELS 2010*, volume 6598 of *LNCS*, pages 56–71. Springer, 2010.

[BH93] Rolv Bræk and Øystein Haugen. *Engineering Real Time Systems*. Prentice Hall, 1993.

[BPC⁺07] Bach Duy Bui, Rodolfo Pellizzoni, Marco Caccamo, Chin F. Cheah, and Andrew Tzakis. Soft Real-Time Chains for Multi-Hop Wireless Ad-Hoc Networks. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 69–80. IEEE Computer Society, 2007.

[BS03] Egon Börger and Robert Stärk. *Abstract State Machines – A Method for High-Level System Design and Analysis*. Springer, 2003.

[BSP11] T. Basmer, H. Schomann, and S. Peter. Implementation Analysis of the IEEE 802.15.4 MAC for Wireless Sensor Networks. In *Mobile and Wireless Networking (iCOST), 2011 International Conference on Selected Topics in*, pages 7–12, 2011.

[Bur94] Alan Burns. Preemptive Priority Based Scheduling: An Appropriate Engineering Approach. In *PRINCIPLES OF REAL-TIME SYSTEMS*, pages 225–248. Prentice Hall, 1994.

[But05] Giorgio C. Buttazzo. *Hard Real-time Computing Systems*. Springer, New York, NY, 2 edition, 2005.

[BYAH06] Michael Buettner, Gary V. Yee, Eric Anderson, and Richard Han. X-MAC: A Short Preamble MAC Protocol for Duty-cycled Wireless Sensor Networks. In Campbell et al. [CBH06], pages 307–320.

[CBG11] Dennis Christmann, Philipp Becker, and Reinhard Gotzhein. Priority Scheduling in SDL. In Ober and Ober [OO11], pages 200–215.

[CBG13] Dennis Christmann, Tobias Braun, and Reinhard Gotzhein. SDL Real-Time Tasks - Concept, Implementation, and Evaluation. In Khendek et al. [KTGR13], pages 239–257.

[CBGK08] Dennis Christmann, Philipp Becker, Reinhard Gotzhein, and Thomas Kuhn. Model-driven Development of a MAC Layer for Ad-hoc Networks with SDL. In *Joint ITU-T and SDL Forum Society workshop on ITU System Design Languages, Geneva, Switzerland*, 2008.

[CBH06] Andrew T. Campbell, Philippe Bonnet, and John S. Heidemann, editors. *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems, SenSys 2006, Boulder, Colorado, USA, October 31 - November 3, 2006*. ACM, 2006.

[CGK09] Dennis Christmann, Reinhard Gotzhein, and Thomas Kuhn. Multi-hop Clock Synchronization in Wireless Ad-Hoc Networks. *ECEASST*, 17, 2009.

[CGKW10] Dennis Christmann, Reinhard Gotzhein, Marc Krämer, and Martin Winkler. Flexible and Energy-efficient Duty Cycling in Wireless Networks with MacZ. In Khalil Drira, Ahmed Hadj Kacem, and Mohamed Jmaiel, editors, *NOTERE*, pages 121–128. IEEE, 2010.

[CGKW13] Dennis Christmann, Reinhard Gotzhein, Marc Krämer, and Martin Winkler. Flexible and Energy-efficient Duty Cycling in Wireless Networks with MacZ. *Concurrency and Computation: Practice and Experience*, 25(2):218–233, 2013.

[CGR12] D. Christmann, R. Gotzhein, and S. Rohr. The Arbitrating Value Transfer Protocol (AVTP) - Deterministic Binary Countdown in Wireless Multi-Hop Networks. In *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, pages 1 –9, aug 2012.

[CGSW14] D. Christmann, R. Gotzhein, S. Siegmund, and F. Wirth. Realization of Try-Once-Discard in Wireless Multi-hop Networks. *Industrial Informatics, IEEE Transactions on*, 10(1):17–26, 2014.

[Chr10] Dennis Christmann. Spezifikation und automatisierte Implementierung zeitkritischer Systeme mit TC-SDL. Master's thesis, TU Kaiserslautern, 2010.

[Cinar] Cinderella ApS. Cinderella. `http://www.cinderella.dk/`, 2015.

[CM10] Dennis Christmann and Ivan Martinovic. Experimental Design and Analysis of Transmission Properties in an Indoor Wireless Sensor Network. In *WiOpt*, pages 342–347. IEEE, 2010.

[CMS10] Dennis Christmann, Ivan Martinovic, and Jens B Schmitt. Analysis of Transmission Properties in an Indoor Wireless Sensor Network Based on a Full-Factorial Design. *Measurement Science and Technology*, 21(12):124003, 2010.

[Cor05] Intel Corporation. Intel PXA27x Processor Family – Electrical Mechanical, and Thermal Specification, 2005.

[CS97] Ana R. Cavalli and Amardeo Sarma, editors. *SDL '97 Time for Testing, SDL, MSC and Trends – 8th International SDL Forum, Evry, France, 23-29 September 1997, Proceedings*. Elsevier, 1997.

[CTN07] D. Carnevale, A.R. Teel, and D. Nešić. A Lyapunov Proof of an Improved Maximum Allowable Transfer Interval for Networked Control Systems. *IEEE Trans. Autom. Control*, 52/5:892–897, 2007.

[DGLS99] Winfried Dulz, S. Gruhl, Lennard Lambert, and M. Söllner. Early Performance Prediction of SDL/MSC Specified Systems by Automated Synthetic Code Generation. In *SDL Forum*, pages 457–472, 1999.

[DGLT09] Manuel Díaz, Daniel Garrido, Luis Llopis, and José M. Troya. Designing Distributed Software with RT-CORBA and SDL. *Computer Standards & Interfaces*, 31(6):1073–1091, 2009.

[DHMC95] Marc Diefenbruch, Jörg Hintelmann, and Bruno Müller-Clostermann. QSDL: Sprache und Werkzeuge zur Leisungsanalyse von SDL-Systemen. In *Tagungsband des 5. GI/ITG-Fachgespräch: Formale Beschreibungstechniken für verteilte Systeme*, june 1995.

[DHMC97] Marc Diefenbruch, Jörg Hintelmann, and Bruno Müller-Clostermann. QUEST Performance Evalution of SDL System. In Klaus Irmscher, Christian Mittasch, and Klaus Richter, editors, *MMB (Kurzbeiträge)*, pages 126–132. TU Bergakademie Freiberg, 1997.

[DRDK04] Daniel Dietterle, Jerzy Ryman, Kai F. Dombrowski, and Rolf Kraemer. Mapping of High-Level SDL Models to Efficient Implementations for TinyOS. In *DSD*, pages 402–406. IEEE Computer Society, 2004.

[DSJ07] Shu Du, Amit Kumar Saha, and David B. Johnson. RMAC: A Routing-Enhanced Duty-Cycle MAC Protocol for Wireless Sensor Networks. In *INFOCOM*, pages 1478–1486, 2007.

[DZM01] Christos Drosos, M. Zayadine, and Dimitris Metafas. Real-Time Communication Protocol Development - Using SDL for an Embedded System On Chip Based on ARM Microcontroller. In *ECRTS*, pages 89–94. IEEE Computer Society, 2001.

[EB01] Stewart Edgar and Alan Burns. Statistical Analysis of WCET for Scheduling. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001), London, UK, 2-6 December 2001*, pages 215–224. IEEE Computer Society, 2001.

[ECG14] Markus Engel, Dennis Christmann, and Reinhard Gotzhein. Implementation and Experimental Validation of Timing Constraints of BBS. In A.L. Murphy abd N. Trigoni, editor, *EWSN*, volume 8354 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2014.

[EGG$^+$01] Robert Eschbach, Uwe Glässer, Reinhard Gotzhein, Martin von Löwis, and Andreas Prinz. Formal Definition of SDL-2000 – Compiling and Running SDL Specifications as ASM Models. *J. UCS*, 7(11):1024–1049, 2001.

[EHD04a] Amre El-Hoiydi and Jean-Dominique Decotignie. WiseMAC: An Ultra Low Power MAC Protocol for Multi-hop Wireless Sensor Networks. In *ALGOSENSORS*, volume 3121 of *Lecture Notes in Computer Science*, pages 18–31. Springer, 2004.

[EHD04b] Amre El-Hoiydi and Jean-Dominique Decotignie. WiseMAC: An Ultra Low Power MAC Protocol for the Downlink of Infrastructure Wireless Sensor Networks. In *ISCC*, pages 244–251. IEEE Computer Society, 2004.

[EHS97] Jan Ellsberger, Dieter Hogrefe, and Amardeo Sarma. *SDL – Formal Object-oriented Language for Communication Systems*. Prentice Hall, 1997.

[Eng13] Markus Engel. Optimierung und Evaluation Black Burst-basierter Protokolle unter Verwendung der Imote 2-Plattform. Master's thesis, TU Kaiserslautern, 2013.

[Ett13] Ettus Research. USRP: Universal Software Radio Peripheral 2. `https://www.ettus.com/product/category/USRP-Networked-Series`, 2013.

[FGG$^+$05] Ingmar Fliege, Alexander Geraldy, Reinhard Gotzhein, Thomas Kuhn, and Christian Webel. Developing Safety-critical Real-time Systems with SDL Design Patterns and Components. *Computer Networks*, 49(5):689–706, 2005.

[FGGS04] Ingmar Fliege, Alexander Geraldy, Reinhard Gotzhein, and Philipp Schaible. A Flexible Micro Protocol Framework. In Daniel Amyot and Alan W. Williams, editors, *Proceedings of 4th SAM (SDL and MSC) Workshop, Ottawa, Canada*, volume 3319 of *Lecture Notes in Computer Science*, pages 224–236. Springer, 2004.

[FGJ+05] Ingmar Fliege, Alexander Geraldy, Simon Jung, Thomas Kuhn, Christian Webel, and Christian Weber. Konzept und Struktur des SDL Environment Framework (SEnF). Technical Report 341/05, TU Kaiserslautern, 2005.

[FGW06] Ingmar Fliege, Rüdiger Grammes, and Christian Weber. ConTraST – A Configurable SDL Transpiler and Runtime Environment. In Reinhard Gotzhein and Rick Reed, editors, *System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, May 31 – June 2, 2006, Revised Selected Papers*, volume 4320 of *Lecture Notes in Computer Science*, pages 216–228. Springer, 2006.

[Fle10] FlexRay Consortium. *FlexRay Communication System Protocol Specification Version 3.0.1*. 2010.

[Fli09] Ingmar Fliege. *Component-based Development of Communication Systems*. PhD thesis, University of Kaiserslautern, 2009.

[FLMTS97] Nils Faltin, Lennard Lambert, Andreas Mitschele-Thiel, and Frank Slomka. An Annotational Extension of Message Sequence Charts to Support Performance Engineering. In Cavalli and Sarma [CS97], pages 307–322.

[GGH08] Alexander Geraldy, Reinhard Gotzhein, and Christoph Heidinger. Model-driven Development of Complex Routing Protocols with SDL-MDD. In *Joint ITU-T and SDL Forum Society workshop on ITU System Design Languages, Geneva, Switzerland*, 2008.

[GGK07] Reinhard Gotzhein, Rüdiger Grammes, and Thomas Kuhn. Specifying Input Port Bounds in SDL. In *Proceedings of the 13th international SDL Forum conference on Design for dependable systems*, SDL'07, pages 101–116, Berlin, Heidelberg, 2007. Springer-Verlag.

[GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns – Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1995.

[GK11a] Reinhard Gotzhein and Thomas Kuhn. Black Burst Synchronization (BBS) – A Protocol for Deterministic Tick and Time Synchronization in Wireless Networks. *Computer Networks*, 55(13):3015–3031, 2011.

[GK11b] Reinhard Gotzhein and Thomas Kuhn. Method, Computer Program Product, and System for the Tick Synchronization of Nodes in a Wireless Multi-Hop Network, august 2011. European Patent EP2195949(B1).

[GKLC09] Reinhard Gotzhein, Marc Krämer, Lothar Litz, and Alain Chamaken. Energy-aware System Design with SDL. In Reed et al. [RBG09], pages 19–33.

[GKS02] Reinhard Gotzhein, Ferhat Khendek, and Philipp Schaible. Micro Protocol Design: The SNMP Case Study. In Edel Sherratt, editor, *SAM*, volume 2599 of *Lecture Notes in Computer Science*, pages 61–73. Springer, 2002.

[GKW+02] Deepak Ganesan, Bhaskar Krishnamachari, Alec Woo, David Culler, Deborah Estrin, and Stephen Wicker. Complex Behavior at Scale: An Experimental Study of Low-power Wireless Sensor Networks. Technical report, UCLA/CSD-TR 02, 2002.

[Got03] Reinhard Gotzhein. Consolidating and Applying the SDL-Pattern Approach: a Detailed Case Study. *Information & Software Technology*, 45(11):727–741, 2003.

[Got07] Reinhard Gotzhein. Model-driven by SDL – Improving the Quality of Networked Systems Development (Invited Paper). In *Proceedings of the 7th International Conference on New Technologies of Distributed Systems (NOTERE 2007), Marrakesh, Morocco*, pages 31–46, June 4-8 2007.

[GP05] Susanne Graf and Andreas Prinz. Time in State Machines. In *Abstract State Machines*, pages 217–232, 2005.

[Gra02] Susanne Graf. Expression of Time and Duration Constraints in SDL. In Edel Sherratt, editor, *SAM*, volume 2599 of *Lecture Notes in Computer Science*, pages 38–52. Springer, 2002.

[Gro03] Object Management Group. MDA Guide Version 1.0.1. http://www.omg.org/cgi-bin/doc?omg/03-06-01, 2003. www.uml.org.

[Gro05] Object Management Group. Real-time CORBA SPecification Version 1.2 formal/05-01-04. http://www.omg.org/spec/RT/1.2/PDF, 2005.

[Gro12] Robert Groh. Contributions to the Integration of CC2420 and SDL into ns-3. Bachelor thesis (in german), University of Kaiserslautern, Computer Science Department, 2012.

[Gum58] Emil Julius Gumbel. *Statistics of Extremes*. Columbia University Press, 1958.

[Gur95] Yuri Gurevich. Specification and Validation Methods. chapter Evolving Algebras 1993: Lipari Guide, pages 9–36. Oxford University Press, Inc., New York, NY, USA, 1995.

[HPUB01] Hesham Hallal, Alex Petrenko, Andreas Ulrich, and Sergiy Boroday. Using SDL Tools to Test Properties of Distributed Systems. In *in Proc. of Formal Approches to Testing of Software (FATES'01), Workshop of the Int. Conference on Concurrency Theory (CONCUR'01)*, pages 125–140, 2001.

[HTvdWN09] W.P.M.H. Heemels, A.R. Teel, N. van de Wouw, and D. Nešić. Networked Control Systems with Communication Constraints: Tradeoffs between Transmission Intervals and Delays. In *European Control Conference*, pages 732–747, Budapest, Hungary, July 2009.

[IBMar] IBM Corp. Rational SDL Suite. http://www-01.ibm.com/software/awdtools/sdlsuite/, 2015.

[IEE03] IEEE. *Part 15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Computer Society, New York, NY, USA, Oct. 2003.

[IEE05] IEEE. *Part 15.1: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Wireless Personal Area Networks (WPANs)*. IEEE Computer Society, New York, NY, USA, June 2005.

[IG13] A. Igel and R. Gotzhein. A CC2420 Transceiver Simulation Module for ns-3 and its Integration into the FERAL Simulator Framework. In *The Fifth International Conference on Advances in System Simulation*, pages 156–164, 2013.

[Ins11] Institute of Electrical and Electronics Engineers. *IEEE Standard 802 Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)*. IEEE Computer Society, New York, NY, USA, June 2011.

[Ins12a] Institute of Electrical and Electronics Engineers. *IEEE Standard 802 Part 11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications*. IEEE Computer Society, New York, NY, USA, Feb 2012.

[Ins12b] Institute of Electrical and Electronics Engineers. *IEEE Standard for Ethernet*. IEEE Computer Society, New York, NY, USA, Dec 2012.

[Int88] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 (11/88): Specification and Description Language (SDL). http://www.itu.int/rec/T-REC-Z.100-198811-S, 1988.

[Int93] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 (03/93): Specification and Description Language (SDL). http://www.itu.int/rec/T-REC-Z.100-199303-S, 1993.

[Int95] International Telecommunication Union (ITU). ITU-T Recommendation Z.105 (03/95): SDL Combined with ASN.1 (SDL/ASN.1). http://www.itu.int/rec/T-REC-Z.105-199503-S, 1995.

[Int96a] International Telecommunication Union (ITU). ITU-T Recommendation Z.106 (10/96): Common Interchange Format for SDL. http://www.itu.int/rec/T-REC-Z.106-199610-S, 1996.

[Int96b] International Telecommunication Union (ITU). ITU-T Recommendation Z.120 (10/1996): Message Sequence Charts (MSC). https://www.itu.int/rec/T-REC-Z.120-199610-S, 1996.

[Int99a] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 (11/99): Specification and Description Language (SDL). http://www.itu.int/rec/T-REC-Z.100-199911-S, 1999.

[Int99b] International Telecommunication Union (ITU). ITU-T Recommendation Z.109 (11/99) - Specification and Description Language - SDL Combined with UML. http://www.itu.int/rec/T-REC-Z.109-199911-S, 1999.

[Int99c] International Telecommunication Union (ITU). ITU-T Recommendation Z.120 (11/1999): Message Sequence Charts (MSC). https://www.itu.int/rec/T-REC-Z.120-199911-S, 1999.

[Int00] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 Annex F: Formal Semantics Definition. http://www.itu.int/rec/T-REC-Z.100-200011-I!AnnF1; http://www.itu.int/rec/T-REC-Z.100-200011-I!AnnF2; http://www.itu.int/rec/T-REC-Z.100-200011-I!AnnF3, 2000.

[Int02] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 (08/2002): Specification and Description Language (SDL). http://www.itu.int/ITU-T/2005-2008/com17/languages/Z100.pdf, 2002.

[Int04] International Standardization Organization. Controller Area Network (CAN), ISO 11898, 2004.

[Int07] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 (11/2007): Specification and Description Language (SDL). http://www.itu.int/rec/T-REC-Z.100-200711-I, 2007.

[Int08] International Telecommunication Union (ITU). Abstract Syntax Notation One (ASN.1) Recommendations (Overview). http://www.itu.int/ITU-T/studygroups/com17/languages/, 2008.

[Int10] International Electrotechnical Commission. Industrial Communication Networks - Wireless Communication Network and Communication Profiles - WirelessHART (IEC 62591 ed 1.0), april 2010.

[Int11] International Telecommunication Union (ITU). ITU-T Recommendation Z.120 (02/2011): Message Sequence Charts (MSC). http://www.itu.int/rec/T-REC-Z.120-201102-I/en, 2011.

[Int12a] International Electrotechnical Commission. Industrial Communication Networks - Wireless Communication Network and Communication Profiles - ISA 100.11a (IEC 62734 ed 1.0), march 2012.

[Int12b] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 (04/12) - Specification and Description Language - Unified Modeling Language Profile for SDL-2010. http://www.itu.int/rec/T-REC-Z.109-201204-I, 2012.

[Int12c] International Telecommunication Union (ITU). ITU-T Recommendation Z.100 (12/11) - Specification and Description Language - Overview of SDL-2010. http://www.itu.int/rec/T-REC-Z.100/en, 2012.

[Int12d] International Telecommunication Union (ITU). ITU-T Recommendation Z.101 (12/11) - Specification and Description Language - Basic SDL-2010. http://www.itu.int/rec/T-REC-Z.101-201112-I, 2012.

[Int12e] International Telecommunication Union (ITU). ITU-T Recommendation Z.102 (12/11) - Specification and Description Language - Comprehensive SDL-2010. http://www.itu.int/rec/T-REC-Z.102-201112-I, 2012.

[Int12f] International Telecommunication Union (ITU). ITU-T Recommendation Z.103 (12/11) - Specification and Description Language - Shorthand Notation and Annotation in SDL-2010. http://www.itu.int/rec/T-REC-Z.103-201112-I, 2012.

[Int12g] International Telecommunication Union (ITU). ITU-T Recommendation Z.104 (12/11) - Specification and Description Language - Data and Action Language in SDL-2010. http://www.itu.int/rec/T-REC-Z.104-201112-I, 2012.

[Int12h] International Telecommunication Union (ITU). ITU-T Recommendation Z.105 (12/11) - Specification and Description Language - SDL-2010 Combined with ASN.1 Modules. http://www.itu.int/rec/T-REC-Z.105-201112-I, 2012.

[Int12i] International Telecommunication Union (ITU). ITU-T Recommendation Z.106 (12/11) - Specification and Description Language - Common Interchange Format for SDL-2010. http://www.itu.int/rec/T-REC-Z.106-201112-I, 2012.

[Int12j] International Telecommunication Union (ITU). ITU-T Recommendation Z.107 (04/12) - Specification and Description Language - Object-oriented Data in SDL-2010. http://www.itu.int/rec/T-REC-Z.107-201204-I, 2012.

[Jai91] Raj Jain. *The Art of Computer Systems Performance Analysis*. WILEY Professional Computing, 1991.

[JLS+10] A. Jentzen, F. Leber, D. Schneisgen, A. Berger, and S. Siegmund. An Improved Maximum Allowable Transfer Interval for L^p-stability of Networked Control Systems. *IEEE Trans. Automatic Control*, 55(1):179–184, 2010.

[JSM91] Kevin Jeffay, Donald F. Stanat, and Charles U. Martel. On Non-Preemptive Scheduling of Periodic and Sporadic Tasks. In *Real-Time Systems Symposium*, pages 129–139. IEEE, 1991.

[Kam13] Alain Tierry Chamaken Kamde. *Model-Based Cross-Design for Wireless Networked Control Systems*. PhD thesis, 2013.

[Kar90] Phil Karn. MACA – A New Channel Access Method for Packet Radio. In *Proceedings of the ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, 1990.

[KB11] Alexander Klein and Lothar Braun. A Preamble-Based Approach for Providing Quality of Service Support in Wireless Sensor Networks. In Sacchi et al. [SBV+11], pages 144–155.

[KB12] A. Klein and L. Braun. Performance Study of a Preamble Based MAC protocol in Multi-hop Wireless Networks. In *Wireless Advanced (WiAd), 2012*, pages 132–137, June 2012.

[KBCG11] Marc Krämer, Tobias Braun, Dennis Christmann, and Reinhard Gotzhein. Real-Time Signaling in SDL. In Ober and Ober [OO11], pages 184–199.

[KBK07] Srinivasa Reddy Kasu, Santosh Kumar Bellana, and Chiranjeev Kumar. A Binary Countdown Medium Access Control Protocol Scheme for Wireless Sensor Networks. In *ICIT*, pages 122–126. IEEE Computer Society, 2007.

[KdI07] Thomas Kuhn and José Irigon de Irigon. An Experimental Evaluation of Black Burst Transmissions. In Albert Y. Zomaya and Sherali Zeadally, editors, *MOBIWAC*, pages 163–167. Proceedings of the Fifth ACM International Workshop on Mobility Management & Wireless Access, MOBIWAC 2007, Chania, Crete Island, Greece, October 2007.

[KE09] Marc Krämer and Markus Engel. New Bootloader for the Imote2 Platform. Technical Report 373/09, TU Kaiserslautern, 2009.

[KEÖ07] Gökhan Korkmaz, Eylem Ekici, and Füsun Özgüner. Black-Burst-Based Multihop Broadcast Protocols for Vehicular Networks. *IEEE T. Vehicular Technology*, 56(5):3159–3167, 2007.

[KF98] Thomas Kolloch and Georg Färber. Mapping an Embedded Hard Real-Time Systems SDL Specification to an Analyzable Task Network – A Case Study. In Frank Mueller and Azer Bestavros, editors, *LCTES*, volume 1474 of *LNCS*, pages 156–165. Springer, 1998.

[KGW06] Thomas Kuhn, Reinhard Gotzhein, and Christian Webel. Model-Driven Development with SDL – Process, Tools, and Experiences. In Oscar Nierstrasz, Jon Whittle, David Harel, and Gianna Reggio, editors, *MoDELS*, volume 4199 of *Lecture Notes in Computer Science*, pages 83–97. Springer, 2006.

[KKS09] Alexander Klein, Jirka Klaue, and Josef Schalk. BP-MAC: A High Reliable Backoff Preamble MAC Protocol for Wireless Sensor Networks. *Electronic Journal of Structural Engineering (EJSE)*, Special Issue on Sensor Network for Building Monitoring: From Theory to Real Application:35–45, 2009.

[Kle10] Alexander Klein. *Performance Issues of MAC and Routing Protocols in Wireless Sensor Networks*. PhD thesis, Julius Maximilian University of Würzburg, 2010. http://d-nb.info/1009425110.

[Kle11] Alexander Klein. BPS-MAC: Backoff Preamble Based MAC Protocol with Sequential Contention Resolution. In Sacchi et al. [SBV$^+$11], pages 39–50.

[Kle12] Alexander Klein. *Wireless Sensor Networks – Technology and Protocols: Chapter 7 – Preamble-Based Medium Access in Wireless Sensor Networks*. InTech, 2012.

[KLK00] Hartmut König, Peter Langendörfer, and Heiko Krumm. Improving the Efficiency of Automated Protocol Implementations Using a Configurable FDT Compiler. *Computer Communications*, 23(12):1179–1195, 2000.

[KNE03] David Kotz, Calvin Newport, and Chip Elliott. The Mistaken Axioms of Wireless Network Research. Technical Report TR2003-467, Dartmouth College, July 2003.

[KNG+04] David Kotz, Calvin C. Newport, Robert S. Gray, Jason Liu, Yougu Yuan, and Chip Elliott. Experimental Evaluation of Wireless Simulation Assumptions. In Simonetta Balsamo, Carla-Fabiana Chiasserini, and Lorenzo Donatiello, editors, *MSWiM*, pages 78–82. ACM, 2004.

[KOK13a] I. Kedzo, J. Ozegovic, and A. Kristic. Contention Overhead - Adaptive Binary Priority Countdown protocol. In *Software, Telecommunications and Computer Networks (SoftCOM), 2013 21st International Conference on*, pages 1–5, Sept 2013.

[KOK13b] Ivan Kedžo, Julije Ožegović, and Ante Kristić. BPC – A Binary Priority Countdown Protocol. *Ad Hoc Networks*, 11(3):747 – 764, 2013.

[Kol02] Thomas Kolloch. *Scheduling with Message Deadlines for Hard Real-Time SDL Systems*. PhD thesis, Technische Universität München, 2002.

[Kop97] Hermann Kopetz. *Real-Time Systems – Design Principles for Distributed Embedded Applications*. Kluwer Academic Publishers, 1997.

[Kra13a] Christopher Kramer. Ermittlung des Netzzustands von Funk-Netzwerken . Projektarbeit, Technische Universität Kaiserslautern, Fachbereich Informatik, 2013.

[Krä13b] Marc Krämer. *Modellgetriebene Entwicklung von Kommunikationsprotokollen für drahtlos vernetzte Regelungssysteme*. PhD thesis, University of Kaiserslautern, 2013.

[Kra14] Christopher Kramer. Drahtlose Kommunikationssysteme für den Produktionsbereich . Master's thesis, TU Kaiserslautern, 2014.

[KSM08] Alexey Kiryushin, Aleksandr Sadkov, and Alan Mainwaring. Real-World Performance of Clear Channel Assessment in 802.15.4 Wireless Sensor Networks. *Sensor Technologies and Applications, International Conference on*, 0:625–630, 2008.

[KTGR13] Ferhat Khendek, Maria Toeroe, Abdelouahed Gherbi, and Rick Reed, editors. *SDL 2013: Model-Driven Dependability Engineering - 16th International SDL Forum, Montreal, Canada, June 26-28, 2013. Proceedings*, volume 7916 of *Lecture Notes in Computer Science*. Springer, 2013.

[Kuh09] Thomas Kuhn. *Model Driven Development of MacZ – A QoS Medium Access Control Layer for Ambient Intelligence Systems*. PhD thesis, University of Kaiserslautern, 2009.

[KV00] Ferhat Khendek and Daniel Vincent. Enriching SDL Specifications with MSCs. In Sherratt [She00], pages 305–319.

[KZ05] Ferhat Khendek and Xiao Jun Zhang. From MSC to SDL: Overview and an Application to the Autonomous Shuttle Transport System. In Stefan Leue and

Tarja Systä, editors, *Scenarios: Models, Transformations and Tools*, volume 3466 of *Lecture Notes in Computer Science*, pages 228–254. Springer, 2005.

[LCL07] HyungJune Lee, Alberto Cerpa, and Philip Levis. Improving Wireless Simulation through Noise Modeling. In Tarek F. Abdelzaher, Leonidas J. Guibas, and Matt Welsh, editors, *Proceedings of the 6th International Conference on Information Processing in Sensor Networks, IPSN 2007, Cambridge, Massachusetts, USA, April 25-27, 2007*, pages 21–30. ACM, 2007.

[LEH00] Philippe Leblanc, Anders Ek, and Thomas Hjelm. Telelogic SDL and MSC tool families. In *Telektronikk 4.2000, Languages for Telecommunication Applications*. Telenor, 2000.

[Liu00] Jane W. S. Liu. *Real-Time Systems*. Prentice Hall, Upper Saddle River, NJ, 2000.

[LK99] Peter Langendörfer and Hartmut König. Automated Protocol Implementations Based on Activity Threads. In *ICNP*, pages 3–10, 1999.

[LL05] Peter Langendörfer and Martin Lehmann. Implementation Independent Profiling of SDL Specifications. In Peter Liggesmeyer, Klaus Pohl, and Michael Goedicke, editors, *Software Engineering*, volume 64 of *LNI*, pages 155–166. GI, 2005.

[Mat14] Alexander Mater. Entwicklung einer Schnittstelle für die Imote2-Plattform mit PragmaDev RTDS und BiPS. Bachelorarbeit, Technische Universität Kaiserslautern, Fachbereich Informatik, 2014.

[MCR97] Nikolai Mansurov, Andrew V. Chernov, and Alexei S. Ragozin. Industrial Strength Code Generation from SDL. In Cavalli and Sarma [CS97], pages 415–430.

[MEMara] MEMSIC Inc. Imote2 datasheet. `http://www.memsic.com.cn/index.php?option=com_phocadownload&view=category&download=138`, 2015.

[MEMarb] MEMSIC Inc. MICAz datasheet. `http://www.memsic.com/support/\discretionary{-}{}{}documentation/\discretionary{-}{}{}wireless-sensor-networks/category/7-datasheets.html?download=148%3Amicaz`, 2015. Revision B.

[MHH01] Jan Madsen, Jörg Henkel, and Xiaobo Sharon Hu, editors. *Proceedings of the Ninth International Symposium on Hardware/Software Codesign, CODES 2001, Copenhagen, Denmark, 2001*. ACM, 2001.

[MLK06] Pulat Matkurbanov, SeungKi Lee, and Dong-Sung Kim. A Survey and Analysis of Wireless Fieldbus for Industrial Environments. In *Proc. Int SICE-ICASE Joint Conf*, pages 5555–5561, 2006.

[MLON13] Pavel Morozkin, Irina Lavrovskaya, Valentin Olenev, and Konstantin Nedovodeev. Integration of SDL Models into a SystemC Project for Network Simulation. In Khendek et al. [KTGR13], pages 275–290.

[MSDH01] Ralf Münzenberger, Frank Slomka, Matthias Dörfel, and Richard Hofmann. A General Approach for the Specification of Real-Time Systems with SDL. In Reed and Reed [RR01], pages 203–222.

[MT00] Andreas Mitschele-Thiel. *Engineering with SDL – Developing Performance-Critical Communication Systems*. John Wiley & Sons, 2000.

[NL09] D. Nesic and D. Liberzon. A Unified Framework for Design and Analysis of Networked and Quantized Control Systems. *Automatic Control, IEEE Transactions on*, 54(4):732–747, April 2009.

[NT04] D. Nešić and A. R. Teel. Input-output Stability Properties of Networked Control Systems. *IEEE Trans. Automatic Control*, 49(10):1650–1667, 2004.

[Obj22] Object Management Group (OMG). Unified Modelling Language (UML) Specification: Version 2.4.1, iso/iec 19505. `http://www.omg.org/spec/UML/`, 20122.

[OK01] Iulian Ober and Alain Kerbrat. Verification of Quantitative Temporal Properties of SDL Specifications. In *SDL '01: Proceedings of the 10th International SDL Forum Copenhagen on Meeting UML*, pages 182–202, London, UK, 2001. Springer-Verlag.

[OO11] I. Ober and I. Ober, editors. *SDL 2011: Integrating System and Software Modeling, Toulouse, France, Jule 5–7, 2011, Proceedings*, volume 7083 of *LNCS*. Springer, 2011.

[PAT06] Nuno Pereira, Björn Andersson, and Eduardo Tovar. Implementation of a Dominance Protocol for Wireless Medium Access. In *RTCSA*, pages 162–172. IEEE Computer Society, 2006.

[PAT07] Nuno Pereira, Björn Andersson, and Eduardo Tovar. WiDom: A Dominance Protocol for Wireless Medium Access. *IEEE Trans. Industrial Informatics*, 3(2):120–130, 2007.

[PATC09] Nuno Pereira, Björn Andersson, Eduardo Tovar, and Paulo Carvalho. Improving the Reliability of WiDom in a Single Broadcast Domain. In *SIES*, pages 144–147. IEEE, 2009.

[PATR07a] Nuno Pereira, Bjorn Andersson, Eduardo Tovar, and Anthony Rowe. Static-Priority Scheduling over Wireless Networks with Multiple Broadcast Domains. In *RTSS '07: Proceedings of the 28th IEEE International Real-Time Systems Symposium*, pages 447–456, Washington, DC, USA, 2007. IEEE Computer Society.

[PATR07b] Nuno Pereira, Björn Andersson, Eduardo Tovar, and Anthony Rowe. Static-Priority Scheduling over Wireless Networks with Multiple Broadcast Domains. Technical Report HURRAY-TR-070118, Polytechnique Institute of Porto (ISEP-IPP), 2007.

[PC11] S. Petersen and S. Carlsen. WirelessHART Versus ISA100.11a: The Format War Hits the Factory Floor. *Industrial Electronics Magazine, IEEE*, 5(4):23–34, Dec 2011.

[PCD$^+$11] Maxime Perrotin, Eric Conquet, Julien Delange, Andre Schiele, and Thanassis Tsiodras. TASTE A Real-time Software Engineering Tool Chain: Overview, Status, and Future. In Ober and Ober [OO11], pages 22–44.

[PDO02] Abhishek Pal, Atakan Dogan, and Füsun Özgüner. MAC Layer Protocols for Real-Time Traffic in Ad-Hoc Wireless Networks. In *ICPP '02: Proceedings of the 2002 International Conference on Parallel Processing*, page 539, Washington, DC, USA, 2002. IEEE Computer Society.

[PGAT09] Nuno Pereira, Ricardo Gomes, Björn Andersson, and Eduardo Tovar. Efficient Aggregate Computations in Large-Scale Dense WSN. In *IEEE Real-Time and Embedded Technology and Applications Symposium*, pages 317–326. IEEE Computer Society, 2009.

[Praar] PragmaDev SARL. Real Time Developer Studio. `http://www.pragmadev.com`, 2015.

[Pri00] Andreas Prinz. Formal Semantics of Specification Languages. In *Telektronikk 4.2000, Languages for Telecommunication Applications*. Telenor, 2000.

[PRML06] M. Petrova, J. Riihijarvi, P. Mahonen, and S. Labella. Performance Study of IEEE 802.15.4 Using Measurements and Simulations. In *WCNC 2006: IEEE Wireless Communications and Networking Conference*, pages 487–492. IEEE, April 2006.

[PST07] Andreas Prinz, Markus Scheidgen, and Merete Skjelten Tveit. A Model-Based Standard for SDL. In Emmanuel Gaudin, Elie Najm, and Rick Reed, editors, *SDL Forum*, volume 4745 of *Lecture Notes in Computer Science*, pages 1–18. Springer, 2007.

[PVB03] Benoit Parreaux, Daniel Vincent, and Gérard Babonneau. RMTP2: Validating the Interval Timed Extension for SDL with an Industrial-Size Multicast Protocol. In Rick Reed and Jeanne Reed, editors, *SDL 2003: System Design, 11th International SDL Forum, Stuttgart, Germany, July 1-4, 2003. Proceedings*, volume 2708 of *Lecture Notes in Computer Science*, pages 218–233. Springer, 2003.

[PvL03a] Andreas Prinz and Martin von Löwis. Engineering the SDL Formal Language Definition. In Elie Najm, Uwe Nestmann, and Perdita Stevens, editors, *FMOODS*, volume 2884 of *Lecture Notes in Computer Science*, pages 47–63. Springer, 2003.

[PvL03b] Andreas Prinz and Martin von Löwis. Generating a Compiler for SDL from the Formal Language Definition. In Reed and Reed [RR03], pages 150–165.

[PW07] Lei Pan and Hongyi Wu. Performance Evaluation of The SYN-MAC Protocol in Multihop Wireless Networks. In *ICCCN*, pages 265–271. IEEE, 2007.

[RB98] Franz Regensburger and Aenne Barnard. Formal Verification of SDL Systems at the Siemens Mobile Phone Department. In Bernhard Steffen, editor, *TACAS*, volume 1384 of *Lecture Notes in Computer Science*, pages 439–455. Springer, 1998.

[RBG09] Rick Reed, Attila Bilgic, and Reinhard Gotzhein, editors. *SDL 2009: Design for Motes and Mobiles, 14th International SDL Forum, Bochum, Germany, September 22–24, 2009, Proceedings*, volume 5719 of *LNCS*. Springer, 2009.

[RCM$^+$06] Bhaskaran Raman, Kameswari Chebrolu, Naveen Madabhushi, Dattatraya Y. Gokhale, Phani K. Valiveti, and Dheeraj Jain. Implications of Link Range and (In)stability on Sensor Network Architecture. In *WiNTECH '06: Proceedings of the 1st International Workshop on Wireless Network Testbeds, Experimental Evaluation & Characterization*, pages 65–72, New York, NY, USA, 2006. ACM.

[Ree00] Rick Reed. The Evolution of SDL-2000. In *Telektronikk 4.2000, Languages for Telecommunication Applications*. Telenor, 2000.

[Ree09] Rick Reed. SDL-2010 Route Map (TAG08) – TD0469. `http://www.sdl-forum.org/ftp/pub/SDL-2010/T09-SG17-090916-TD-PLEN-0469!!MSW-E.doc`, september 2009.

[Ree11a] Rick Reed. SDL-2010: Background, Rationale, and Survey. In Ober and Ober [OO11], pages 8–29.

[Ree11b] Rick Reed. SDL-2010 Route Map (TAL08) – TD2116 Rev.1. `http://www.sdl-forum.org/ftp/pub/SDL-2010/T09-SG17-110824-TD-PLEN-2116!R1!MSW-E.doc`, august/september 2011.

[RFM12] RFM. TR1000 Datasheet. `http://www.rfm.com/products/data/tr1000.pdf`, 2012.

[RKL05] Yosef Rauchwerger, Finn Kristoffersen, and Yair Lahav. Cinderella SLIPPER: An SDL to C-Code Generator. In Andreas Prinz, Rick Reed, and Jeanne Reed, editors, *SDL Forum*, volume 3530 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2005.

[RR01] Rick Reed and Jeanne Reed, editors. *SDL 2001: Meeting UML, 10th International SDL Forum Copenhagen, Denmark, June 27-29, 2001, Proceedings*, volume 2078 of *Lecture Notes in Computer Science*. Springer, 2001.

[RR03] Rick Reed and Jeanne Reed, editors. *SDL 2003: System Design, 11th International SDL Forum, Stuttgart, Germany, July 1-4, 2003. Proceedings*, volume 2708 of *Lecture Notes in Computer Science*. Springer, 2003.

[RR05] Matthias Ringwald and Kay Römer. BitMAC: A Deterministic, Collision-free, and Robust MAC Protocol for Sensor Networks. In *Proceedings of Second European Workshop on Wireless Sensor Networks (EWSN 2005), Istanbul, Turkey*, pages 57–69, january 2005.

[RR06] Iyappan Ramachandran and Sumit Roy. On the Impact of Clear Channel Assessment on MAC Performance. In *GLOBECOM*, 2006.

[San00] Richard Sanders. Implementing from SDL. In *Telektronikk 4.2000, Languages for Telecommunication Applications*. Telenor, 2000.

[SBV+11] Claudio Sacchi, Boris Bellalta, Alexey V. Vinel, Christian Schlegel, Fabrizio Granelli, and Yan Zhang, editors. *Multiple Access Communications - 4th International Workshop, MACOM 2011, Trento, Italy, September 12-13, 2011. Proceedings*, volume 6886 of *Lecture Notes in Computer Science*. Springer, 2011.

[SDL13] SDL-RT Consortium. SDL-RT – Specification & Description Language – Real Time V2.3). `http://www.sdl-rt.org/standard/V2.3/pdf/SDL-RT.pdf`, 2013.

[SDM01] Frank Slomka, Matthias Dörfel, and Ralf Münzenberger. Generating Mixing Hardware/Software Systems from SDL Specifications. In Madsen et al. [MHH01], pages 116–121.

[SDTL06] Kannan Srinivasan, Prabal Dutta, Arsalan Tavakoli, and Philip Levis. Understanding the Causes of Packet Delivery Success and Failure in Dense Wireless Sensor Networks. In Campbell et al. [CBH06], pages 419–420.

[She00] Edel Sherratt, editor. *SAM 2000, 2nd Workshop on SDL and MSC, Col de Porte, Grenoble, France, June 26-28, 2000*. VERIMAG, IRISA, SDL Forum, 2000.

[She12] Edel Sherratt. Relativity and Abstract State Machines. In Øystein Haugen, Rick Reed, and Reinhard Gotzhein, editors, *SAM*, volume 7744 of *LNCS*, pages 105–120. Springer, 2012.

[SJK07] Changsu Suh, Jung-Eun Joung, and Young-Bae Ko. New RF Models of the TinyOS Simulator for IEEE 802.15.4 Standard. In *WCNC*, pages 2236–2240. IEEE, 2007.

[SK96] J.L. Sobrinho and A.S. Krishnakumar. Real-Time Traffic over the IEEE 802.11 Medium Access Layer. *Bell Labs Technical Journal*, Autumn 1996:172–187, autumn 1996.

[SK99] J. L. Sobrinho and A. S. Krishnakumar. Quality-of-Service in Ad Hoc Carrier Sense Multiple Access Wireless Networks. *IEEE Journal on Selected Areas in Communications*, 17(8):1353–1368, August 1999.

[SL06] Kannan Srinivasan and Philip Levis. RSSI is Under Appreciated. In *Proceedings of the Third Workshop on Embedded Networked Sensors (EmNets)*, 2006.

[SLO+10] Alexander Stepnaov, Irina Lavrovskaya, Valentin Olenev, Alexey Rabin, Sergey Balandin, and Michel Gillet. SystemC and SDL Co-Modelling Implementation. In *Proceedings of the 7th Conference of Open Innovations Association FRUCT*, pages 80–85, Saint-Petersburg, Russia, April 2010.

[SLWT04] Jang-Ping Sheu, Chi-Hsun Liu, Shih-Lin Wu, and Yu-Chee Tseng. A Priority MAC Protocol to Support Real-time Traffic in Ad Hoc Networks. *Wireless Networks*, 10(1):61–69, 2004.

[SRL90] Lui Sha, Ragunathan Rajkumar, and John P. Lehoczky. Priority Inheritance Protocols: An Approach to Real-Time Synchronization. *IEEE Trans. Computers*, 39(9):1175–1185, 1990.

[SS01] Natalia Sidorova and Martin Steffen. Verifying Large SDL-Specifications Using Model Checking. In Reed and Reed [RR01], pages 403–420.

[Sta88] John A. Stankovic. Misconceptions About Real-Time Computing: A Serious Problem for Next-Generation Systems. *Computer*, 21(10):10–19, 1988.

[Sta92] John A. Stankovic. Real-Time Computing, 1992.

[Str00] Bjarne Stroustrup. *Die C++ Programmiersprache*. Pearson Education Deutschand GmbH, München, Germany, 4 edition, 2000.

[Tex] Texas Instruments. BTnodes - A Distributed Environment for Prototyping Ad Hoc Networks (web page). http://www.btnode.ethz.ch/.

[Tex07] Texas Instruments. CC2420 Datasheet. http://focus.ti.com/lit/ds/symlink/cc2420.pdf, 2007. Revision SWRS041b.

[TNT07] Mohammad Tabbara, Dragan Nesic, and Andrew R. Teel. Stability of Wireless and Wireline Networked Control Systems. *IEEE Trans. Automat. Contr.*, 52(9):1615–1630, 2007.

[US 08] US Department of Defense. *Global Positioning System Standard – Positioning Service Performance Standard*. 4 edition, 2008.

[VA10] Maryam Vahabi and Björn Andersson. New Schedulability Analysis for WiDom. Technical Report HURRAY-TR-100507, Polytechnique Institute of Porto (ISEP-IPP), May 2010.

[VA11] Maryam Vahabi and Björn Andersson. Slotted WiDom: Schedulability Analysis and its Experimental Validation. Technical Report HURRAY-TR-110403, Polytechnique Institute of Porto (ISEP-IPP), April 2011.

[vDL03] Tijs van Dam and Koen Langendoen. An Adaptive Energy-efficient MAC Protocol for Wireless Sensor Networks. In *SenSys*, pages 171–180, 2003.

[vLoM03] Martin von Löwis of Menar. *Formale Semantik des Datentypmodells von SDL-2000*. PhD thesis, Humboldt-Universität zu Berlin, 2003.

[VVL$^+$11] Frank Vanheel, Jo Verhaevert, Eric Laermans, Ingrid Moerman, and Piet Demeester. Automated Linear Regression Tools Improve RSSI WSN Localization in Multipath Indoor Environment. *EURASIP Journal on Wireless Communications and Networking*, 2011(1):1–27, 2011.

[WDEK06] Gerald Wagenknecht, Daniel Dietterle, Jean-Pierre Ebert, and Rolf Kraemer. Transforming Protocol Specifications for Wireless Sensor Networks into Efficient Embedded System Implementations. In Kay Römer, Holger Karl, and Friedemann Mattern, editors, *EWSN*, volume 3868 of *Lecture Notes in Computer Science*, pages 228–243. Springer, 2006.

[WEE$^+$08] Reinhard Wilhelm, Jakob Engblom, Andreas Ermedahl, Niklas Holsti, Stephan Thesing, David B. Whalley, Guillem Bernat, Christian Ferdinand, Reinhold Heckmann, Tulika Mitra, Frank Mueller, Isabelle Puaut, Peter P. Puschner, Jan Staschulat, and Per Stenström. The Worst-case Execution-time Problem - Overview of Methods and Survey of Tools. *ACM Trans. Embedded Comput. Syst.*, 7(3), 2008.

[WLS14] M. Wilhelm, V. Lenders, and J.B. Schmitt. On the Reception of Concurrent Transmissions in Wireless Sensor Networks. *IEEE Transactions on Wireless Communications*, 13(12):6756–6767, december 2014.

[WMW05] A. Willig, K. Matheus, and A. Wolisz. Wireless Technology in Industrial Networks. *Proceedings of the IEEE*, 93(6):1130–1151, June 2005.

[WUT05] Hongyi Wu, Anant Utgikar, and Nian-Feng Tzeng. SYN-MAC: A Distributed Medium Access Control Protocol for Synchronized Wireless Networks. *MONET*, 10(5):627–637, 2005.

[WYB02] G. C. Walsh, Hong Ye, and L. G. Bushnell. Stability Analysis of Networked Control Systems. *IEEE Transactions on Control Systems Technology*, 10(3):438–446, 2002.

[YHE02] Wei Ye, John S. Heidemann, and Deborah Estrin. An Energy-Efficient MAC Protocol for Wireless Sensor Networks. In *INFOCOM*, June 2002.

[YHE04] Wei Ye, John Heidemann, and Deborah Estrin. Medium Access Control with Coordinated, Adaptive Sleeping for Wireless Sensor Networks. *ACM/IEEE Transactions on Networking*, 12(3):493–506, June 2004.

[YYH03a] Tiantong You, Chi-Hsiang Yeh, and H. Hassanein. A New Class of Collision Prevention MAC Protocols for Wireless Ad hoc Networks. In *Communications, 2003. ICC '03. IEEE International Conference on*, volume 2, pages 1135 – 1140 vol.2, may 2003.

[YYH03b] Tiantong You, Chi-Hsiang Yeh, and Hossam Hassanein. CSMA/IC: A New Class of Collision-free MAC Protocols for Ad Hoc Wireless Networks. In *ISCC '03: Proceedings of the Eighth IEEE International Symposium on Computers and Communications*, pages 843–848, Washington, DC, USA, 2003. IEEE Computer Society.

[ZW06] Klaus Zeppenfeld and Regine Wolters. *Generative Software-Entwicklung mit der MDA*, volume 1. Elsevier GmbH, München, 2006.

List of Publications

[1] Dennis Christmann. Mikroprotokoll-basierte Restrukturierung, toolgestützte Dokumentation und Evaluation von MacZ. Bachelor's thesis, TU Kaiserslautern, 2008.

[2] Dennis Christmann, Philipp Becker, Reinhard Gotzhein, and Thomas Kuhn. Model-driven Development of a MAC Layer for Ad-hoc Networks with SDL. In *Joint ITU-T and SDL Forum Society workshop on ITU System Design Languages, Geneva, Switzerland*, 2008.

[3] Dennis Christmann, Reinhard Gotzhein, and Thomas Kuhn. Multi-hop Clock Synchronization in Wireless Ad-Hoc Networks. *ECEASST*, 17, 2009.

[4] Dennis Christmann. Duty Cycling in drahtlosen Multi-Hop-Netzwerken. Technical Report 372/09, TU Kaiserslautern, 2009. http://vs.cs.uni-kl.de/en/publications/2009/Ch09/.

[5] Philipp Becker, Dennis Christmann, and Reinhard Gotzhein. Model-Driven Development of Time-critical Protocols with SDL-MDD. In Rick Reed, Attila Bilgic, and Reinhard Gotzhein, editors, *14th International SDL Forum*, volume 5719 of *LNCS*, pages 34–52. Springer, 2009.

[6] Martin Winkler, Dennis Christmann, and Marc Krämer. Customized Duty Cycling with MacZ. In M. Beigl and F. Cazorla-Almeida, editors, *23rd International Conference on Architecture of Computing Systems (ARCS 2010) – Workshop Proceedings, Hannover, Germany, 2010*, 2010.

[7] Dennis Christmann. Spezifikation und automatisierte Implementierung zeitkritischer Systeme mit TC-SDL. Master's thesis, TU Kaiserslautern, 2010.

[8] Dennis Christmann. On the Behavior of Black Bursts in Tick-Synchronized Networks. Technical Report 377/10, University of Kaiserslautern, 2010. http://vs.cs.uni-kl.de/publications/2010/Ch10/.

[9] Dennis Christmann, Reinhard Gotzhein, Marc Krämer, and Martin Winkler. Flexible and Energy-efficient Duty Cycling in Wireless Networks with MacZ. In Khalil Drira, Ahmed Hadj Kacem, and Mohamed Jmaiel, editors, *NOTERE*, pages 121–128. IEEE, 2010.

[10] Dennis Christmann and Ivan Martinovic. Experimental Design and Analysis of Transmission Properties in an Indoor Wireless Sensor Network. In *WiOpt*, pages 342–347. IEEE, 2010.

[11] Dennis Christmann, Ivan Martinovic, and Jens B. Schmitt. Analysis of Transmission Properties in an Indoor Wireless Sensor Network Based on a Full-Factorial Design. *Measurement Science and Technology*, 21(12):124003, 2010.

[12] Dennis Christmann and Reinhard Gotzhein. Kommunikationsanforderungen in zukünftigen Ad-Hoc-Netzwerken . In *Proceedings of 10. GI/ITG KuVS Fachgespräch „Drahtlose Sensornetze“*, September 2011.

[13] Dennis Christmann, Philipp Becker, and Reinhard Gotzhein. Priority Scheduling in SDL. In I. Ober and I. Ober, editors, *SDL 2011: Integrating system and software modeling, Toulouse, France, Jule 5–7, 2011, Proceedings*, volume 7083 of *Lecture Notes in Computer Science (LNCS) 7083*, pages 200–215. Springer, 2011.

[14] Philipp Becker, Martin Birtel, Dennis Christmann, and Reinhard Gotzhein. Black-Burst-Based Quality-of-Service Routing (BBQR) for Wireless Ad-Hoc Networks. In *11th International Conference on New Technologies in Distributed Systems (NOTERE'2011), Paris, France*, pages 1–8. IEEE, 2011.

[15] Marc Krämer, Tobias Braun, Dennis Christmann, and Reinhard Gotzhein. Real-Time Signaling in SDL. In I. Ober and I. Ober, editors, *SDL 2011: Integrating system and software modeling, Toulouse, France, Jule 5–7, 2011, Proceedings*, volume 7083 of *Lecture Notes in Computer Science (LNCS) 7083*, pages 184–199. Springer, 2011.

[16] Dennis Christmann, Reinhard Gotzhein, and Stephan Rohr. The Arbitrating Value Transfer Protocol (AVTP) - Deterministic Binary Countdown in Wireless Multi-Hop Networks. In *Computer Communications and Networks (ICCCN), 2012 21st International Conference on*, pages 1 –9, aug 2012.

[17] Dennis Christmann and Reinhard Gotzhein. Real-time Tasks in SDL. In Øystein Haugen, Rick Reed, and Reinhard Gotzhein, editors, *SAM 2012*, Lecture Notes in Computer Science (LNCS) 7744, pages 53–71. Springer, 2012.

[18] Tobias Braun, Dennis Christmann, Reinhard Gotzhein, and Anuschka Igel. Model-driven Engineering of Networked Ambient Systems with SDL-MDD. *Procedia Computer Science*, 10(0):490 – 498, 2012. ANT 2012 and MobiWIS 2012.

[19] Tobias Braun, Dennis Christmann, Reinhard Gotzhein, Anuschka Igel, Thomas Forster, and Thomas Kuhn. Virtual Prototyping with Feral – Adaptation and Application of a Simulator Framework. In *The 24th IASTED International Conference on Modelling and Simulation*, 2013.

[20] Dennis Christmann, Reinhard Gotzhein, Marc Krämer, and Martin Winkler. Flexible and Energy-efficient Duty Cycling in Wireless Networks with MacZ. *Concurrency and Computation: Practice and Experience*, 25(2):218–233, 2013.

[21] Dennis Christmann, Tobias Braun, and Reinhard Gotzhein. SDL Real-Time Tasks - Concept, Implementation, and Evaluation. In Ferhat Khendek, Maria Toeroe, Abdelouahed Gherbi, and Rick Reed, editors, *SDL Forum*, volume 7916 of *Lecture Notes in Computer Science*, pages 239–257. Springer, 2013.

[22] Dennis Christmann, Reinhard Gotzhein, Stefan Siegmund, and Fabian Wirth. Realization of Try-Once-Discard in Wireless Multi-hop Networks. *Industrial Informatics, IEEE Transactions on*, 10(1):17–26, 2014.

[23] Markus Engel, Dennis Christmann, and Reinhard Gotzhein. Implementation and Experimental Validation of Timing Constraints of BBS. In A.L. Murphy abd N. Trigoni, editor, *EWSN*, volume 8354 of *Lecture Notes in Computer Science*, pages 84–99. Springer, 2014.

[24] Tobias Braun, Dennis Christmann, Reinhard Gotzhein, Anuschka Igel, Thomas Forster, and Thomas Kuhn. Virtual Prototyping of Distributed Embedded Systems with Feral. *International Journal of Modelling and Simulation*, 2(34), 2014.

[25] Tobias Braun, Dennis Christmann, Reinhard Gotzhein, and Alexander Mater. SDL Implementations for Wireless Sensor Networks - Incorporation of PragmaDev's RTDS into the Deterministic Protocol Stack BiPS (*best paper award*). In Daniel Amyot, Pau Fonseca i Casas, and Gunter Mussbacher, editors, *System Analysis and Modeling: Models and Reusability - 8th International Conference, SAM 2014, Valencia, Spain, September 29-30, 2014. Proceedings*, volume 8769 of *Lecture Notes in Computer Science*, pages 271–286. Springer, 2014.

[26] Christopher Kramer, Dennis Christmann, and Reinhard Gotzhein. Automatic Topology Discovery in TDMA-based Ad Hoc Networks (accepted paper). In *11th International Wireless Communications and Mobile Computing Conference*, 2015.

Dennis Christmann

Curriculum Vitae

Vocational Experience

since 05/2013	**research assistant**, *Technische Universität Kaiserslautern, Arbeitsgruppe "Vernetzte Systeme"*.
11/2010 – 04/2013	**(PhD) student assistant**, *Technische Universität Kaiserslautern, Arbeitsgruppe "Vernetzte Systeme"*, support of exercises and research.
10/2007 – 02/2008	**student assistant**, *Technische Universität Kaiserslautern, Arbeitsgruppe "Vernetzte Systeme"*, holding exercises of the lecture "Vernetzte Systeme".

Education

Academic Studies

09/2010 – 04/2013	**PhD program computer science (doctoral studies)**, *Technische Universität Kaiserslautern*. 10/2010 – 04/2013: scholarship of the Carl-Zeiss foundation
04/2008 – 09/2010	**Master of Science (M.Sc.)**, *Technische Universität Kaiserslautern*, postgraduate studies in computer science. Title of the master's thesis: "Spezifikation und automatisierte Implementierung zeitkritischer Systeme mit TC-SDL"
04/2008 – 09/2010	**PhD program computer science (graduate studies)**, *Technische Universität Kaiserslautern*.
04/2005 – 03/2008	**Bachelor of Science (B.Sc.)**, *Technische Universität Kaiserslautern*, undergraduate studies in computer science. Title of the bachelor's thesis: "Mikroprotokoll-basierte Restrukturierung, toolgestützte Dokumentation und Evaluation von MacZ"

School

08/1995 – 03/2004	**secondary school**, *Burggymnasium Kaiserslautern*, Abitur.
08/1991 – 07/1995	**primary school**, *Grundschule Katzweiler*.