Model-driven Development of a MAC Layer for Ad-hoc Networks with SDL

Dennis Christmann, Philipp Becker, Reinhard Gotzhein, Thomas Kuhn

Computer Science Department, University of Kaiserslautern
Postfach 3049, D-67653 Kaiserslautern, Germany
{christma,pbecker,gotzhein,kuhn}@cs.uni-kl.de

Abstract. This paper outlines the specification and evaluation of a MAC layer for wireless ad-hoc networks in the context of SDL Model-Driven Development (SDL-MDD), a model-driven approach that utilizes SDL as modelling language. The MDD process leads through the complete development of networked systems in a holistic model-driven way with an extensive support for reuse in form of SDL patterns and SDL components, especially micro protocols. We show the incremental steps that transfer our case study, the development of the MAC layer *MacZ*, from a solution-independent model into a platform-specific model with focus on the combination of SDL-MDD with micro protocols. Furthermore, we highlight the advantages of SDL-MDD regarding the fast evaluation of the specified system.

1 Introduction

In recent years, several approaches have been proposed in model-driven development of communication protocols. Their objective is the maintenance of models as central artifacts in every development stage and to define processes transforming these models towards the final model that is used for deployment of the system to concrete hardware. The main advantage of model-driven development is the ability to specify the artifacts of the system in a more platform-independent way. This allows a strict separation from hardware-specific details and makes a fast deployment to different hardware platforms possible [16].

Model-driven approaches also provide support for extensive utilization of reuse methods and can – in combination with supporting tools – improve quality and increase productivity. Concretely, SDL-MDD, our model-driven development process that is based on the ITU-T Specification and Description Language (SDL) [14], allows the integration of both generic and ready-to-use design solutions in recommended stages. E.g., SDL patterns are generic solutions to design problems and can be selected, adapted and composed in the models defined by SDL-MDD. On the other hand, SDL components are examples for concrete, ready-to-use design solutions and can also be applied to the artifacts of SDL-MDD [9].

In this paper, we present the development of *MacZ* in the course of restructuring a complex MAC layer [1, 2, 4]. *MacZ* is a MAC layer for mobile ad-hoc networks, e.g., as a basis for ambient intelligence systems. The development of *MacZ* was guided by SDL-MDD. Here, we focus on the combination of SDL-MDD with the idea of micro protocols. Micro protocols can be understood as a special form of SDL components; they represent self-contained, ready-to-use solutions for a specific design problem. They differ in the granularity and the type of the offered functionality from components in general, because micro protocols contain exactly one protocol functionality [10]. More specifically, micro protocols are distributed components providing an elementary protocol functionality that is not decomposable further [9]. To obtain systems with several functionalities, it is often necessary to compose micro

protocols to larger components. This results in so-called macro protocols, aggregations of micro protocols with additional, necessary *glue* components that connect the single micro protocols with each other.

The remaining part of this paper is structured as follows: In Section 2 we survey SDL Model-Driven Development (SDL-MDD) [9], our holistic model-driven approach for generating networked systems with SDL. Section 3 introduces the MAC layer *MacZ* [2], and a selection of micro protocols that were developed and applied during the MDD process [4] to specify MacZ with SDL. In Section 4, we show results of performance simulations of *MacZ* with *PartsSim* [3], one of the tools that support SDL-MDD. In Section 5, we draw conclusions and point out further developments.

2 SDL-MDD

The SDL Model-Driven Development (SDL-MDD) [16] represents a holistic, domain-specific process for developing communication systems. It consists of several process steps that are based on models as central artifacts. SDL-MDD makes use of the ITU's Specification and Description Language (SDL) [14], which comes with a formal semantic and a graphical and textual notation. SDL allows to specify both structure and behavior of a networked system in a platform-independent way, with the possibility to integrate platform-specific code directly into the model [16]. It is supported by several commercial tool suites that offer, e.g., graphical editors for SDL or SDL-to-C compilers. With the latter, the SDL model can be used to automatically generate platform-specific code that is compiled into binaries. For the case study presented in this paper, we have used the Telelogic Tau tool suite [17], which provides these features.

The complete SDL-MDD process is illustrated in Fig. 1 [9]. It shows six stages that bring the system step-by-step from the functional requirements to the executable platform-specific system. These six stages can be grouped into to 2 major phases:

- The *specification phase* (CIM, PIM, PSM) covers the building of the model containing the complete behavior. It starts with the Computation-Independent Model (CIM), which is specified with Message Sequence Charts (MSCs) [13]. In the second stage, the Platform-Independent Model (PIM) is developed. Here, SDL [14] is used to structure the system and to specify the abstract behavior. The model is independent of a concrete hardware platform to allow a fast migration to different platforms. Nevertheless, it already represents a functionally complete SDL model that can be analyzed and whose behavior can be validated [16]. Platform-specific details are added in a further transformation step to the Platform-Specific Model (PSM). The PSM extends the PIM with behavior and definitions specific to a concrete purpose (e.g., for the deployment on special hardware platforms). In other words, the PIM is completely included in the SDL specification of the PSM.
- The *implementation phase* (RIC, RSC, system in execution and system under simulation respectively) contains the conversion of the SDL model into an executable binary. Starting point of the transformation is the PSM, the result of the specification phase. The Runtime-Independent Code (RIC) is obtained with a code generator that brings the SDL model to a representation in a programming language. E.g., *Cmicro*, which is part of the commercial Telelogic Tau suite, transforms the SDL notation into C-code. The RIC is further processed with a platform-specific compiler into machine instructions called Runtime-Specific Code (RSC). The SDL Engine represents the runtime environment of the SDL system and is, among other things, responsible for the selection and

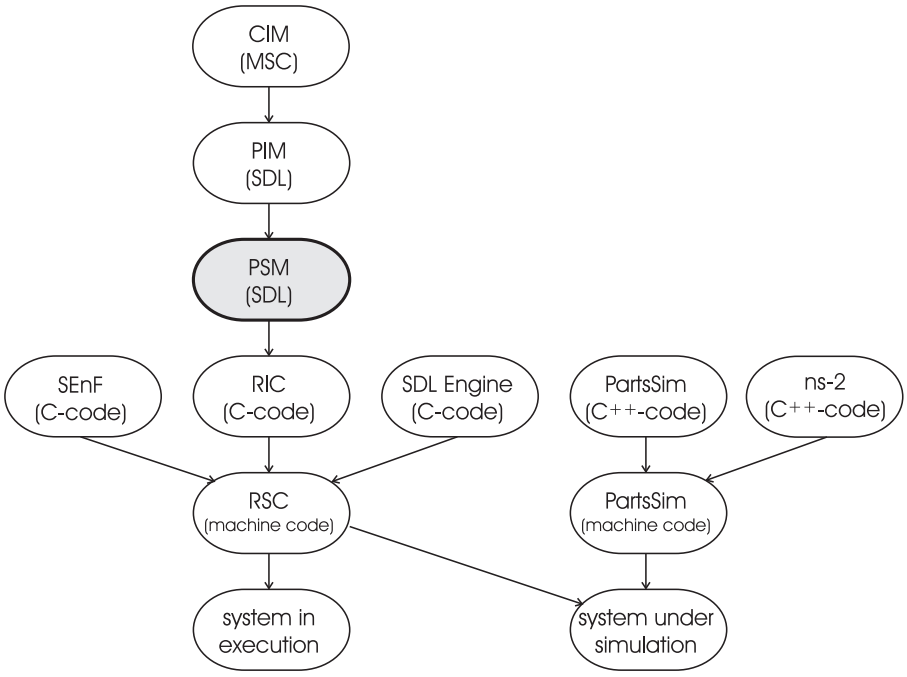


Fig. 1. The model-driven development process SDL-MDD [9, 16].

firing of transitions. The SDL Engine is provided by Telelogic Tau and is composed with the code that is generated out of the model. To interface the open SDL system with its environment, e.g., a concrete hardware platform, operating system, or simulator, the RSC is combined with the SDL Environment Framework (*SEnF*) [9]. Unlike the Telelogic TAU tool chain, which only generates skeletons of environment functions, this step allows for a completely automatic tailoring for a specific platform. In Fig. 1, we consider this tailoring by distinguishing between the system in execution, which goes live on hardware, and the system under simulation, which is loaded and evaluated by a simulator like *PartsSim*.

The presented *MacZ* case study focuses first on the specification of the PIM. Here, we use the concept of micro protocols to decompose the requirements of the MAC layer into functional units, which classify as micro protocols. These requirements were determined in the CIM in a more informal way before.

Our second focus is on the model-driven performance simulation of the SDL specification of *MacZ* with *PartsSim* [3] that utilizes the well-known *ns*-2 network simulator [18]. For that purpose, we automatically generate the executable system, starting with the PSM and using the code generator *Cmicro* [17]. Furthermore, we use *SEnF* to add code to the *Cmicro* generated code. So, the resulting binary includes the required code for the usage in *PartsSim*, and the system can be loaded and evaluated without manual coding steps [9].

3 Platform-Independent Model of MacZ

In this section, we describe the creation of the PIM based on the requirements that were determined in the CIM. I.e., the task is to transform the behavior of the less formal requirements into a strong formal specification with SDL and abstract of the hardware platform initially.

3.1 MacZ – An Overview

Before we present selected aspects of the SDL specification, we outline the functionality of *MacZ* [1, 2, 11]. *MacZ* is a medium access control (MAC) protocol for wireless ad-hoc networks that provides different quality of service mechanisms for the transmission of frames. Beside contention-based transmissions with priorities, *MacZ* offers to transmit frames in a contention-free mode. This requires, because of the incompatibility of both transmission modes, the partitioning of the medium into time slots called *virtual slot regions*. In turn, this requires network-wide tick synchronization, so every node is aware of the current slot region.

Fig. 2 shows an example partitioning of the medium. Beside contention-free and contention-based virtual slot regions, additional idle regions for energy saving purposes are shown. They can be used to switch off the transceiver, which is very important in embedded systems, where small nodes with limited power supply are commonly used. The partitioning into slot regions is based on the slotting into smaller sections with a fixed length called *micro slots*. I.e., a slot region is an aggregation of micro slots; its duration depends on the duration of a single micro slot and the number of micro slots it consists of. Micro slots are numbered consecutively relatively to their synchronization interval [11], so e.g., a contention-based slot region may be present from micro slots numbered 1 to 10.

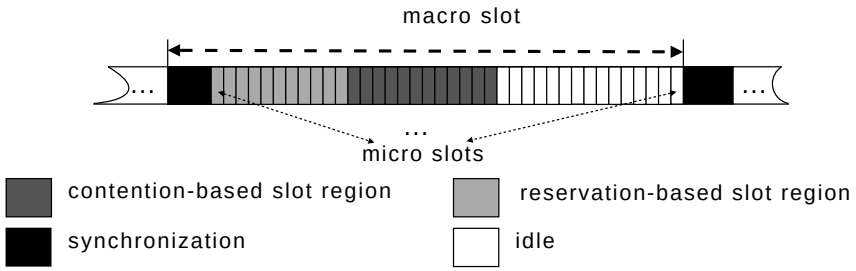


Fig. 2. Partitioning of the medium into virtual slot regions [1, 2, 11].

In contention-based medium access mode, *MacZ* can transmit single frames or frames with preceding RTS/CTS sequence. With RTS/CTS frames, *MacZ* reduces collisions due to hidden station scenarios [15]. To reduce collisions in general, an arbitration of the medium based on CSMA-CA is done with the determination of a backoff interval within a contention window. This is similar to mechanisms used in popular (wireless) MAC protocols like IEEE 802.11e [12]. *MacZ* uses priorities that allow the determination of the lower and upper bound of a contention window. This enables the strict preference of frames by the assignment of priorities that leads to non-overlapping contention windows [2].

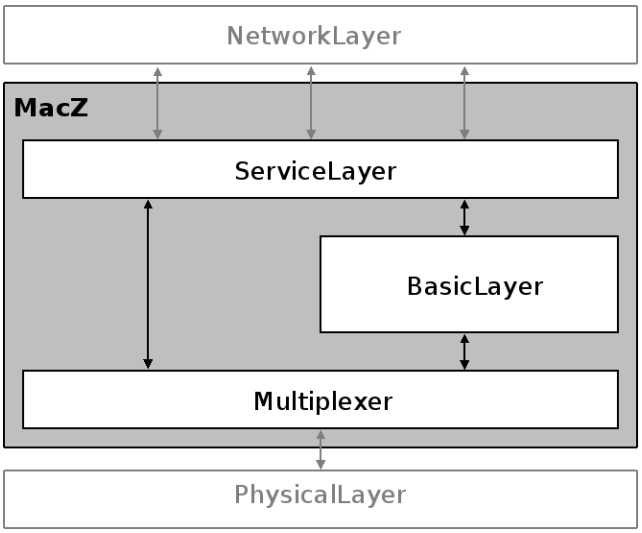


Fig. 3. Overall architecture of *MacZ* [1, 4].

In contention-free mode, *MacZ* does not address the reservation of time slots. Instead, it requires that a reservation is done prior to the transmission by the network layer. Reservations can comprise a single micro slot or a sequence of successive micro slots. The numbers of these reserved slots are provided by the network layer with each transmission request and determine the point of time when *MacZ* transmits the corresponding frame.

3.2 Building up the Architecture

We have analyzed the requirements at our MAC layer to obtain a set of micro protocol definitions. We note that some of these components provide local functionality and therefore may not classify as micro protocols in a strict sense. However, they can be seen as SDL components, i.e. self-contained, ready-to-use building blocks that can be composed and reused. In a second step, we aggregate the identified components and micro protocols to get the PIM, which describes a highly maintainable model, because every building block is self-contained and can be exchanged with other components offering the same functionality.

The architecture is shown in Fig. 3 with the surrounding network and physical layer [1, 4]. The main components `ServiceLayer` and `BasicLayer` are connected to `Multiplexer`, which multiplexes signals between both main components and the physical layer:

- `BasicLayer` is responsible for network-wide tick synchronization between *MacZ* instances and the fragmentation of the medium into micro slots. We will not address this component further in this paper, but concentrate on `ServiceLayer`.
- The `ServiceLayer` component aggregates micro slots provided and numbered by the `BasicLayer` into virtual slot regions. This allows the assignment of different kinds of medium access control methods to special time periods. So, *MacZ* can offer a contention-free and a contention-based transmission mode. These facts result in a further decomposition of the `ServiceLayer` shown in Fig. 4.

The sub components in the mentioned Fig. 4 have the following functionality:

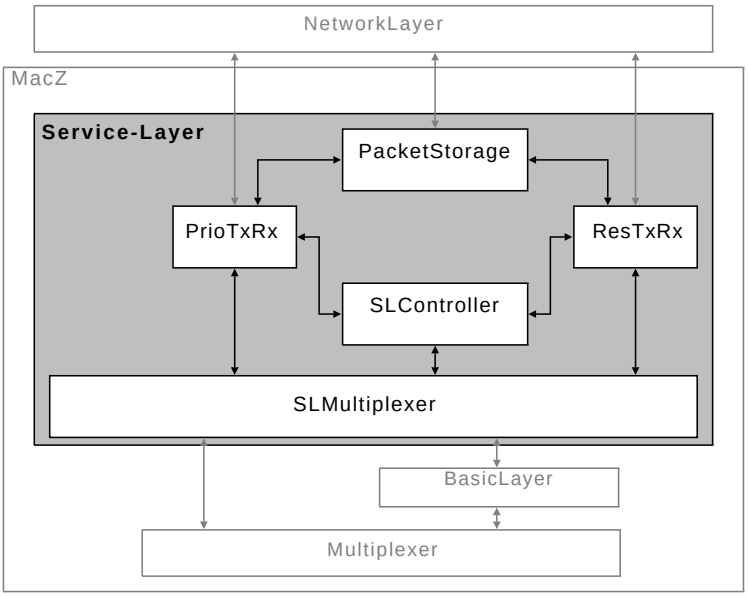


Fig. 4. Architecture of the `ServiceLayer` of *MacZ* [4].

`SLController` is responsible for the synchronous switching between virtual slot regions by every protocol entity, i.e., on every node. This switching is based on the micro slot counting and tick synchronization provided by `BasicLayer`. The component meets all properties of a micro protocol, although it has the characteristic that no messages are exchanged by the protocol entities. It is specified as an SDL process type.

`ResTxRx` is also specified as an SDL process type and provides the transmission and receiving of frames in a contention-free virtual slot region, i.e., it provides a single, symmetric, distributed functionality. Therefore, this component classifies as a typical micro protocol. The frames are sent without any arbitration in the micro slot numbers prescribed by the reservation protocol.

`PrioTxRx` is a composition of several micro protocols, which are enabled in contention-based virtual slot regions only. Thus, `PrioTxRx` encapsulates more than one functionality and represents a macro protocol. The behavior was specified as an SDL block type that consists of the process types shown in Fig. 5. The following micro protocols are distributed among the depicted sub components:

- **MediumArbitration** is responsible for the medium arbitration, i.e., it waits until the medium becomes free and sends frames, if the medium remains free for an amount of time that depends on the lower and upper bound of the contention window, which is given by the priorities of each frame. Its functionality is specified in the two process types `NAV` and `FrameTx`.
- **SimpleSend** sends and receives single data frames. So, it is responsible for the processing of MAC specific headers in frames. The behavior of this micro protocol was specified as a set of SDL procedures and is included in the process type `PrioTxRx` (see also Fig. 6).
- **RTS_CTS** offers the possibility to send frames with a preceding RTS/CTS sequence. Its specification is included as a set of SDL procedures in the process type `PrioTxRx` (see also Fig. 6).

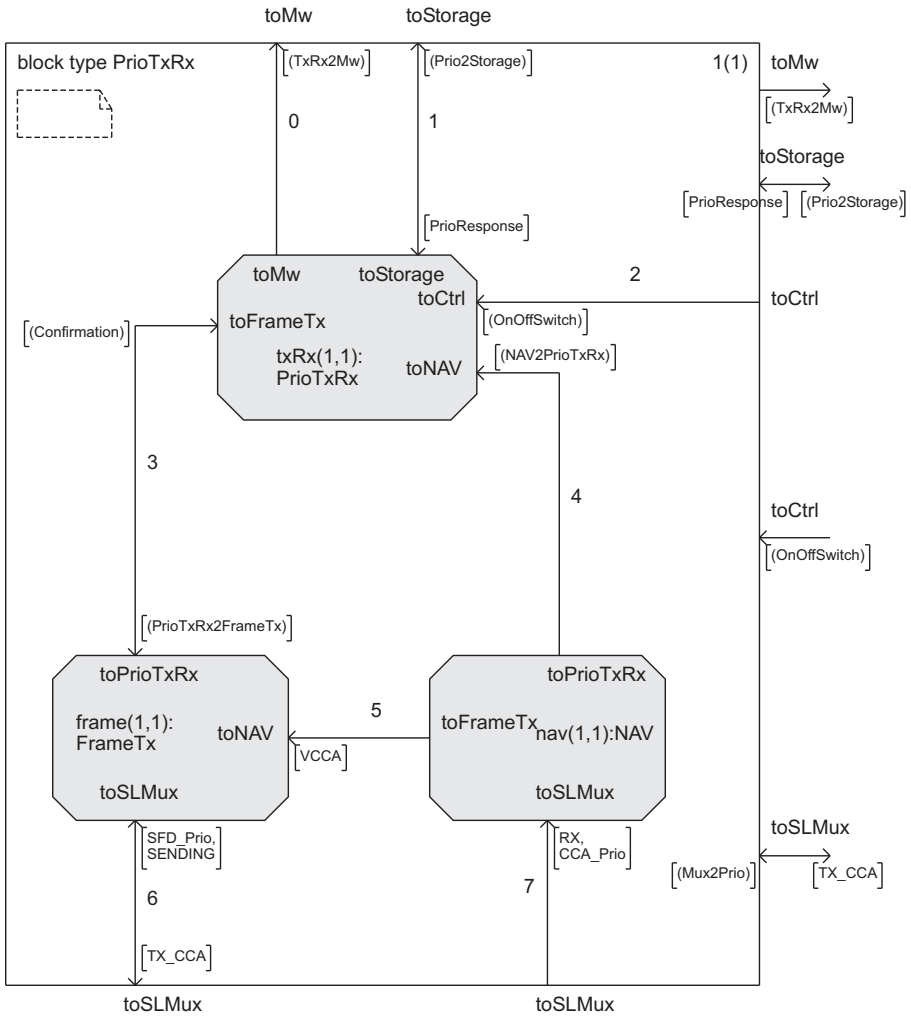


Fig. 5. `PrioTxRx` block type in `ServiceLayer` [4]. `FrameTx` and `NAV` belong to the micro protocol **MediumArbitration**. Process type `PrioTxRx` encapsulates **SimpleSend** and **RTS_CTS**.

The micro protocols **SimpleSend** and **RTS_CTS** were specified as SDL procedures as shown in Fig. 6, because after their composition, these two micro protocols are very close to each other in terms of dependency. More precisely, in case RTS/CTS is desired, the functions of **SimpleSend** are used, if a CTS was received as answer to a sent RTS. Therefore, the two micro protocols are composed in a *possibly sequential* way, which needs some kind of synchronization. Moreover, the sending of data frames must pause, if an RTS was received and a CTS was answered. Hence, with SDL procedures, the micro protocols can synchronize with shared SDL variables defined in the SDL process type, and an extensive exchange of messages is avoided. These variables are introduced during the composition of the micro protocols as a kind of *glue* code. We use SDL procedures instead of SDL service types, because the utilized code generator *Cmicro* does not support service types.

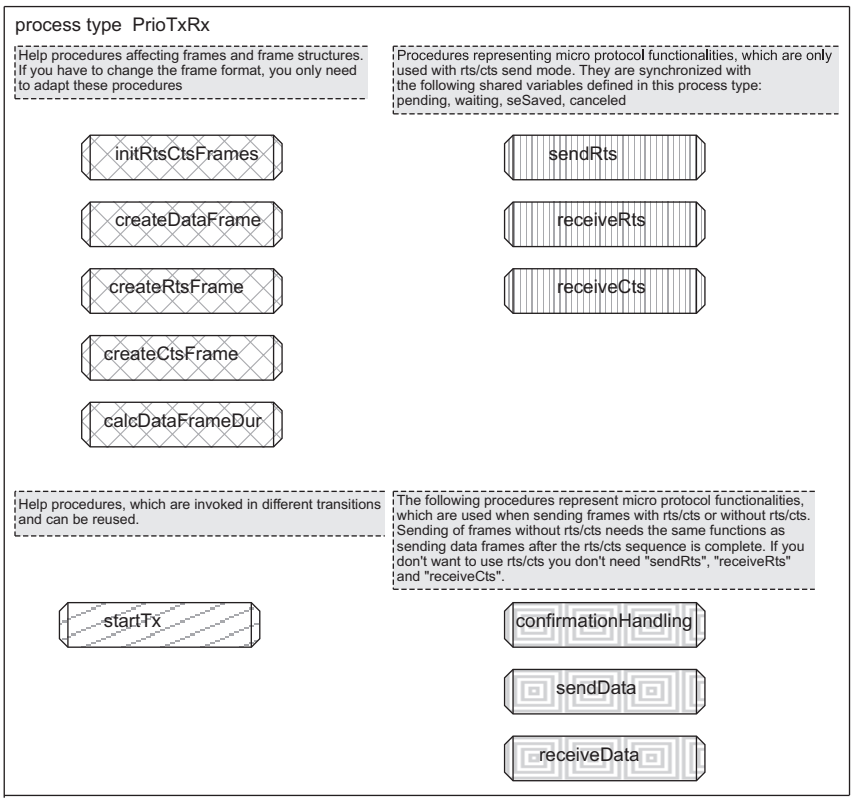


Fig. 6. Procedures in `PrioTxRx` encapsulating **RTS_CTS** and **SimpleSend** [4].

The SDL procedures in Fig. 6 yield a trade-off between synchronization overhead and separation of functionalities. They are grouped into 4 classes:

- Procedures affecting frame structure (represented with crossing lines) contain all access operations to the frame. Those must be adapted, if the frame format changes.
- Help procedures are illustrated with diagonal lines. They are used in different contexts.
- RTS/CTS procedures (vertical lines) contain all the behavior necessary for RTS/CTS handshakes. The procedures are invoked only, if the RTS/CTS mechanism is used.
- General procedures (nested rectangles) handle the transmission of data frames. If a send operation fails, this class is also responsible for possible new attempts.

`SLMultiplexer` (de-)multiplexes signals between `ServiceLayer` and lower layers. However, signals are not just multiplexed according to their type, but also according to the current type of slot region. This allows a strict separation of the presented protocols `ResTxRx` and `PrioTxRx`. At first view, this component does not seem to be a micro protocol, because a multiplexer typically does not have a distributed functionality. Nevertheless, in this case, the multiplexer fulfills the properties of micro protocols, because for the correct behavior, it is essential that all protocol instances forward the signals in the same way. E.g., a frame sent by `PrioTxRx` on one host must not be received by `ResTxRx` on a second host. This crucial primitive represents a distributed functionality. Therefore, the component classifies as a micro protocol, which we have specified as an SDL process type.

3.3 Types of Collaboration

The micro protocols described above collaborate in different ways: The `SLMultiplexer` is composed with the other micro protocols in a sequential collaboration according to the data/control flow through the protocol layers. The `SLController` is concurrently composed with the micro protocol `ResTxRx` and the macro protocol `PrioTxRx`.

`ResTxRx` and `PrioTxRx` are composed with mutual exclusion, so at most one of them is enabled and transmits frames. This exclusive composition requires synchronization, which is accomplished by `SLController` by deactivating certain protocol(s) if the current virtual slot region does not suit their functionality. Thus, this central synchronization solution avoids failures that would come with an error-prone, distributed slot determination.

3.4 Micro Protocol Documentation

To get documentation of the micro protocols, we use *ConTraST* [8], our own SDL-to-C++ code generator that can – in addition to the generation of C++-code out of SDL models [1] – extract annotations from SDL specifications and create a LaTeX file, which includes the interface behavior of the examined micro protocol [7]. With this LaTeX source, documentations in PS or PDF format can be produced automatically. In later developments, such a documentation can be used to find and select micro protocols in a repository, which is crucial for successful reuse.

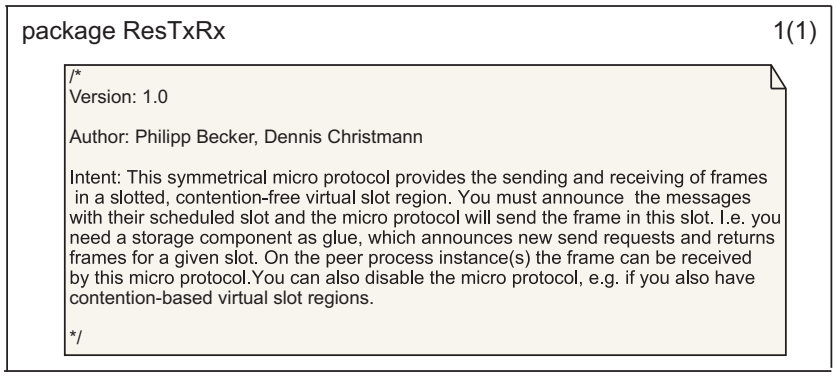
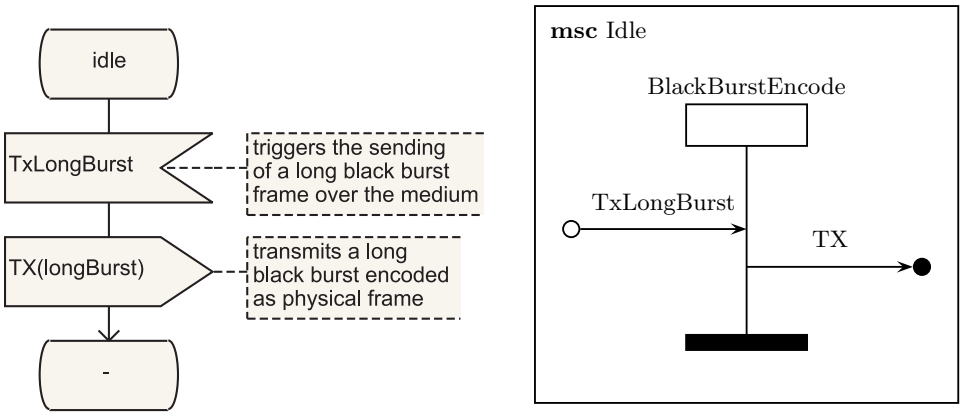


Fig. 7. Annotations showing the overall intention of a micro protocol [4].

The documentation generator extracts inter alia annotations of packages showing the overall purpose of the micro protocol (Fig. 7), and of transitions including triggers, outputs, and decisions. This leads to a documentation that shows the complete behavior of the micro protocol from an outside viewpoint.

Fig. 8 shows on the left a small excerpt of the SDL specification of `BasicLayer` that is responsible for synchronization and medium slotting. The annotations are added to the

[1] One could ask why we use *Cmicro* instead of *ConTraST* for generating code out of the SDL model. This is due to the fact that *ConTraST* was not developed for embedded systems with scarce hardware resources like the *MicaZ* mote [5], for which *MacZ* was developed originally.



Description:

The signal `TxLongBurst` triggers the sending of a long black burst frame over the medium. The signal `TX` transmits a long black burst encoded as physical frame.

Fig. 8. Excerpt of the SDL specification with annotations and the resulting documentation compiled with LaTeX [4, 7]. The example is out of a sub component of `BasicLayer`.

specification as comments, following a predefined format. On the right, the documentation that is generated out the model is shown.

The example demonstrates that beside the benefit of getting a separate documentation, the annotations also provide a description of the behavior in the SDL model itself. This helps developers in understanding the intention of a specification that was created by other developers, and improves maintainability.

4 Model-driven Performance Simulation

To simulate the functional behavior, we take the PSM of *MacZ* and extend it to a complete SDL system with an application layer, which performs stimuli to the system. For all that, the SDL model of *MacZ* is unchanged, so the behavior of the system under test and the system after deployment is the same. The RIC is generated afterwards by *Cmicro* and compiled with a platform-specific compiler, supplemented with additional code of *SEnF* and the SDL Engine. The resulting RSC is then simulated with *PartsSim* [3], which is based on ns-2 [18]. Therefore, constraints by the network like message loss, e.g., caused by weak signals or collisions, are simulated.

Below, we show the results of three simulations [2, 4]. They are generated with the same partitioning of the medium into micro slots, the same configuration of virtual slot regions, the same system stimuli, and the same topology consisting of six nodes. We also use the same transceiver providing a total bandwidth of 250 Kbps. The difference is the used transmission mode, because every simulation was performed with one of the following modes provided by *MacZ*: Contention-free with reservations, contention-based with strict priorities and contention-based without strict priorities. Thus, we can compare the benefits and impacts of the medium access mechanisms to the performance results. Due to the SDL-MDD

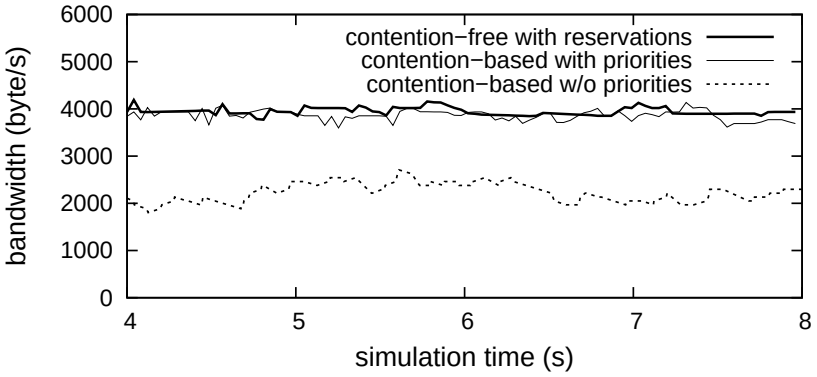


Fig. 9. Comparison of transmission bandwidth with different transmission modes [2, 4].

process, we can adapt the SDL system with very little effort to change the desired transmission mode. Afterwards, the executables, which are loaded by *PartsSim*, can be generated automatically.

Fig. 9 presents the attained bandwidth of an audio stream transmitted in the simulations [2, 4]. In each simulation, the audio stream is generated by a single node with a data rate of 2 kB/s. The audio data is divided into fragments of 40 ms, so every frame contains 82 bytes payload. The frames with the audio data are transmitted over 2 hops, i.e., an intermediate node forwards the audio data to the destination node. This results in a total bandwidth requirement of 4 kB/s to transmit the audio stream completely. Two of the remaining three nodes fill the rest of the bandwidth with sporadic events and other background traffic. Thus, up to 4 nodes compete with each other, which leads to potential collisions and strong competition when the contention-based medium access without strict priorities is used.

The best results are obtained with contention-free transmissions, because all frames are sent with exclusive access to the medium. This prevents collisions of frames, and competition for network bandwidth. All aberrations between the obtained bandwidth and the required bandwidth of 4 kB/s can be traced back to the loss of frames due to weak signal strength, because the size of the contention-free slot region is configured large enough, so every audio frame is assigned sufficient resources and can be sent in time.

With contention-based access without strict priorities, we get the worst results, because all sending nodes compete with the same conditions for medium access. This results both in the deferral of frames with audio data and collisions. Collisions occur, if at least two stations choose the same backoff interval that is determined between the lower and upper bound of the contention window randomly.

The resulting bandwidth of contention-based transmissions with priorities is almost as perfect as the results given with transmissions based on reservations. The only difference comes with the non-preemptive nature of frame transmissions, because proceeding transmissions can delay the sending of audio frames.

5 Conclusions

In this paper, we have presented the development and performance evaluation of *MacZ*, a MAC layer for wireless ad-hoc networks, in the context of SDL-MDD. SDL-MDD uses SDL as design language to specify the central artifacts of the model-driven development process.

We have shown the possibilities of reusablity when applying SDL-MDD, which improves productivity and quality. Especially, micro protocols were identified and composed to the Platform-Independent Model of *MacZ*. The idea behind micro protocols is the specification of self-contained, ready-to-use components that provide a single, distributed protocol functionality. When following this idea, in the course of time, a repository of micro protocols can be created, which enables the fast composition of larger protocols with existing design solutions. So, experiences of prior development processes are kept, and only marginal specification effort remains. With a well thought-out documentation (as provided by *ConTraST*), the identification of matching micro protocols is simplified.

Furthermore, we have presented the benefits of SDL-MDD regarding model-driven performance simulations. Here, the additional effort of the transformation between SDL design and the test application is very small. This allows the evaluation of SDL models during early development stages. For this purpose, we have used *PartsSim*, an extension of ns-2, to simulate the SDL specification. This approach has the second advantage that the system-under-test is based on the same SDL model as the binary that is deployed to a concrete hardware platform.

Up to now, *MacZ* has been developed for the *MicaZ* mote [5]. In ongoing work, we are porting *MacZ* to the *Imote2* [6]. With the model-driven approach presented in this paper, the porting is possible with very little effort.

References

1. Philipp Becker. Entwicklung des MacZ Service Layers. Master's thesis, Computer Science Department, University of Kaiserslautern, 2006.
2. Philipp Becker, Reinhard Gotzhein, and Thomas Kuhn. MacZ - a quality-of-service mac layer for ad-hoc networks. Proceedings of 7th Conference on Hybrid Intelligent Systems (HIS), Kaiserslautern, Germany, 2007.
3. Phillipp Becker, Reinhard Gotzhein, and Thomas Kuhn. Model-driven performance simulation of self-organizing systems with partssim. *Praxis der Informationsverarbeitung und Kommunikation*, pages 45–50, 1/2008.
4. Dennis Christmann. Mikroprotokoll-basierte Restrukturierung, toolgestützte Dokumentation und Evaluation von MacZ. Bachelor's thesis, March 2008.
5. Crossbow Technology Inc. Micaz data sheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/MICAz_Datasheet.pdf`.
6. Crossbow Technology Inc. mote2 data sheet. `http://www.xbow.com/Products/Product_pdf_files/Wireless_pdf/Imote2_Datasheet.pdf`.
7. Ingmar Fliege. Documentation of micro protocols. Technical Report 358/07, Computer Science Department, University of Kaiserslautern, 2007.
8. Ingmar Fliege, Rüdiger Grammes, and Christian Weber. ConTraST - a configurable SDL transpiler and runtime environment. In Reinhard Gotzhein and Rick Reed, editors, *SAM*, volume 4320 of *Lecture Notes in Computer Science*, pages 216–228. Springer, 2006.
9. Reinhard Gotzhein. Model-driven with 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.

10. 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.
11. Reinhard Gotzhein and Thomas Kuhn. Tick synchronization for multi-hop medium slotting in wireless ad hoc networks using black bursts. IEEE SECON 2008, San Francisco, California, USA, June 16-20 2008.
12. IEEE Std. 802.11e. *Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specifications: Amendment 8: Medium Access Control (MAC) Quality of Service Enhancements*. IEEE Computer Society, 2005.
13. ITU-T. *Message Sequence Charts (MSC)*. ITU-T Recommendation Z.120. International Telecommunication Union (ITU), 2001.
14. ITU-T Recommendation Z.100 (11/2007). *Specification and description language (SDL)*. International Telecommunication Union (ITU), 2007.
15. Phil Karn. MACA - a new channel access method for packet radio. In *Proceedings of the ARRL/CRRL Amateur Radio 9th Computer Networking Conference*, 1990.
16. 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.
17. Telelogic Tau. Tau sdt. `http://www.telelogic.com/products/tau/index.cfm`.
18. USC Information Sciences Institute. The network simulator - ns-2. `http://www.isi.edu/nsnam/ns/`.