Simulation of Heterogeneous CAN-Systems

Matthias Stümpfle, Joachim Charzinski
University of Stuttgart
Institute of Communications Switching and Data Technics
Seidenstr. 36
70174 Stuttgart, Germany
stuempfle/charzinski@ind.uni-stuttgart.d400.de

Abstract *As today's systems are becoming more and more complex, simulation is often the only viable way to verify the functionality of a system, or to estimate its performance. Especially in time and money critical sections it is important to gain information about a designed system before any expensive hardware is to be implemented. Using an object oriented simulation framework eases the solving of this problem. Different CAN components were developed separately and are now available as a CAN part library. Complete and heterogeneous systems can now be simulated and evaluated by taking parts from the library and connecting them using the standardized interface from the simulation framework. Configuration of the simulation is supported by a simple to use description language.*
The gained results are presented in the last section. We use the SAE scenario to show how the simulation results can be used to dimension a CAN network.

1 Introduction

Today's systems are becoming more and more complex. The human mind is unable to comprehend complex systems in their entirety. Therefore, complex systems need to be structured in a way that allows humans to cope with this complexity. This is usually achieved by breaking down the system into a hierarchy of subsystems and modules. In such cases, simulations can help in evaluating different design choices. Simulation programs can be used to verify the functionality of the system as well as to estimate the performance of the target system.

Simulation of systems is also important in time and money critical market sections. Obtaining information about a desired designed system is important before constructing it. This helps to find bottlenecks and gives hints for design improvements. It is also an important addition to analytical results that usually present a worst-case view or an approximation of a system [14] and therefore may be away from reality.

This paper is structured as follows. In Section 2 we will present the key design issues of the used simulation framework. A more detailed description can be found in [4, 5]. Section 3 deals with the modeling of CAN systems. We present here the major aspects of the implemented simulation model. In this section we also introduce the environment of the simulation. This framework consists of the open network description language ODL [13] and a special compiler that generates the necessary input information for the simulation.

The paper ends with a presentation of simulation results regarding the SAE scenario.

2 The Object-Oriented Simulation Framework

The following section describes briefly the overall software architecture of the simulation framework. Two main parts can be distinguished. The simulation support subsystem contains all components that are necessary to control the execution of a simulation program.

The main part of the system is the simulation model. The simulation model can be hierarchically decomposed into submodels and model components. The latter are called entities. Entities communicate with each other by exchanging messages. All messages are derived from an abstract base class that defines some common properties, e.g. the message type. Contents and meaning of a message are user defined. Each entity can evaluate only those aspects of a message which it is interested in.

A port mechanism is used for communication. Transferring messages between entities is as simple as connecting the input and output ports of these entities. A handshake protocol which is part of the simulation framework ensures that both entities are ready to exchange messages before they are actually sent. Entities can be seen as black boxes that communicate with the outside world using ports. This strict encapsulation allows separation of the behavior of an entity from the structural arrangement within the model. Therefore, it is easy to insert a new entity between existing entities without modifying the existing ones.

The simulation framework is based on an event-driven paradigm. In event-driven simulations, events are used to plan future activities. Events are entered into a sorted event list and processed later. The meaning of an event depends on the entity that generated it. Events are passed to the entity for processing. The entity might process the event itself, or may pass it on to its parent entity. In this way hierarchical processing of events is supported.

A model entity is a special entity that has a built-in event list. Usually, the model entity stays at the top level of the simulation model hierarchy. Since a model may be composed of more than one submodel, the framework supports more than one event list. The submodels are responsible for synchronizing distributed event lists.

3 Simulating CAN-Systems

In the following section, we will describe some architectural details of the simulation model. After a short introduction into the CAN elements of the simulation we show how complex simulation scenarios may be constructed using parts of a simulation library. We then focus on the modeling of applications using burst-silence generators. Finally we describe the infrastructure that surrounds the simulation and makes it easy to use.

3.1 Elements of the CAN Model

The simulation of a system has to guarantee functional equivalence to reality. The realization of the CAN protocol in different controller types therefore requests the decomposition of a controller network into two basic parts: the controllers with their duty to manage messages with different priorities either as Full-Can or as Basic-CAN, and the bus with its centralized arbitration and routing functionality. We will present these two parts in the following sections.

3.1.1 CAN Controllers

Controllers generally consist of a sender and a receiver part. They provide buffers for messages that have to be sent over or have been received from the bus, respectively. One general difference exists between so called „Basic-CAN“ controllers and „Full-CAN“ controllers. Whilst Basic-CAN controllers provide only one general receiving channel, Full-CAN controllers provide buffers to each message identifier that is proposed to be received. From this fact results the basic structure of our controller model hierarchy that can be seen in Figure 3.1:

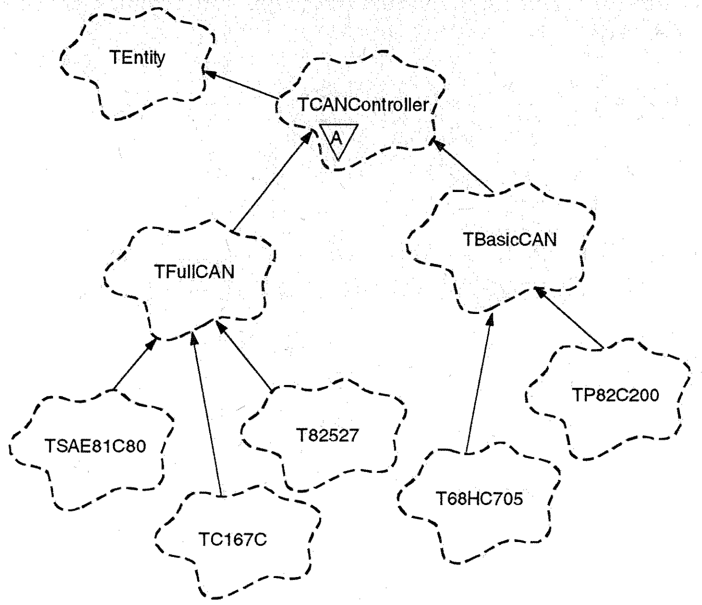

Figure 3.1: *CAN Controller Model Hierarchy*

The `TCANController` class is implemented as an abstract base class that is derived from the basic model class `TEntity` of the simulation framework. By deriving all components from this base class all controllers inherit the ability of statistic measurement capabilities and general connectivity to other components. The derived classes only contain specific implementation details of the corresponding controller.

A further feature is the realization of the extended frame format and the standard frame format specified in the second edition of the CAN specification [9]. Frame formats are supported according to the implementation documentations of the controllers.

The connection to applications is realized by the same port concept that is used within the components. The signaling of having sent or received a message is done by a special interrupt message (IR) that is sent from the controllers to the application.

Basic CAN controllers like the Philips PCA82C200 [8] and the Motorola MC68C705-family [6] use an alternating buffer mechanism as their receiving channel. This allows them to store two different arriving messages: one buffer contains the oldest arrived message and the other contains the newest which has not yet been requested by the application . Figure 3.2 shows the internal structure of our Basic-CAN model.

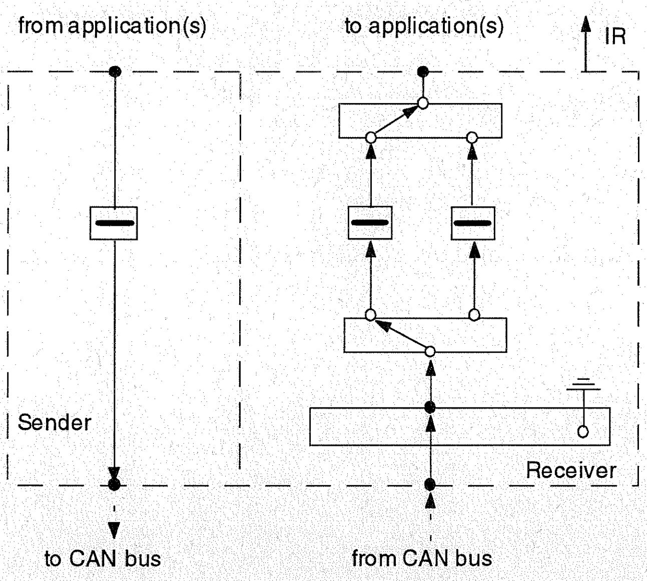

Figure 3.2: *Basic-CAN Controller Model*

Full-CAN Controllers like Intel's 82527 [3] or the Siemens SAE81C90 [11] provide message filtering on-chip. Therefore these controllers have multiple receiving buffers which are assigned to a special CAN identifier.

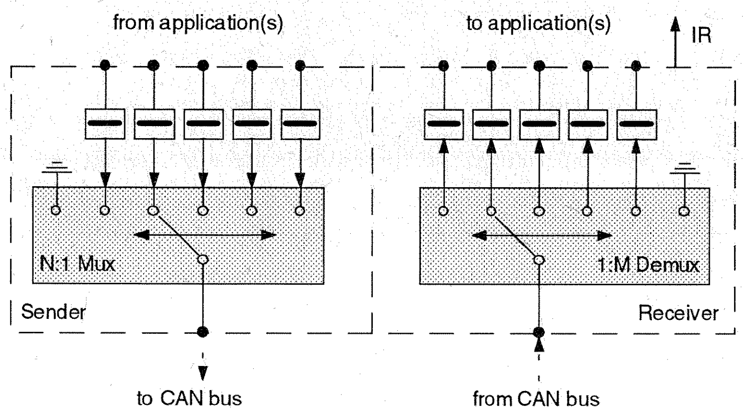

Figure 3.3: *Full-CAN Controller Model*

This prevents the upper layers of a CAN system from spending time in message filtering. Generally one channel of Full-CAN controllers is able to be used as Basic-CAN receiving channel that permits receiving of all messages that are on the bus (according to a specified acceptance mask).

3.1.2 The Bus-Model

The bus model contains the only service phase of the whole model. It models the access of a message to the shared medium and calculates the time that is used to transmit the message. Bus access is gained via an N:1 multiplexer and the routing of messages is done by a 1:N demultiplexer.

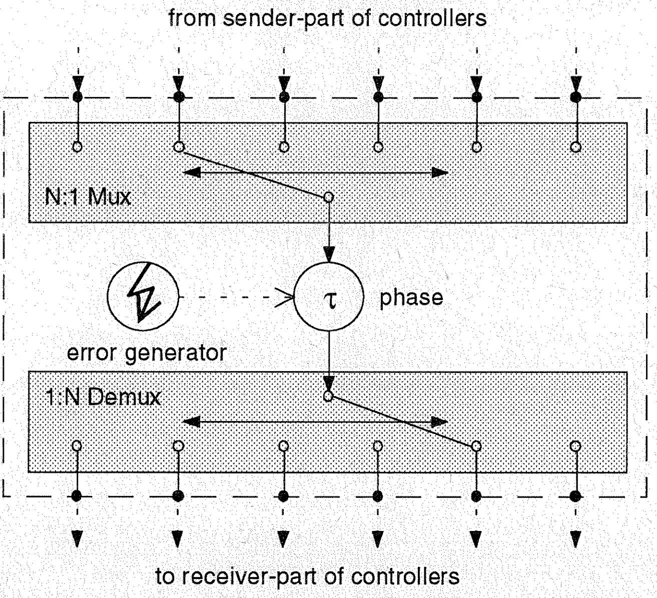
from sender-part of controllers
N:1 Mux
τ
phase
error generator
1:N Demux
to receiver-part of controllers

Figure 3.4: *CAN Bus Model*

Additionally, an error generator may be connected to the system. It generates error-frames according to a continuous limited distribution. Bus access of such errors is also controlled by a continuous distribution.

3.2 Building a Hierarchic and Heterogeneous Simulation Model

The mapping of the model to the simulation is straightforward. A hierarchy of entities can be derived directly from the model. During the development of the program another great benefit of the object-oriented approach became obvious. The library supports an incremental development process. Due to the encapsulation of the entities and the framework that is provided by the library, it is always possible to build a reduced model (single controllers, the bus) which can be tested separately. Later, individual entities are combined to hierarchical entities, and their ports are connected. This has the advantage that an executable program is available during every stage of the development process. The need to integrate a large and complex system in one step does not exist. Since the individual entities are already tested, testing of the whole program could be reduced to validating interactions between controllers and the bus.

During the implementation of the simulation program, we could easily reuse the framework provided by the library. The queues, service phases and generators could either be used directly from the library, or had only to be slightly modified (like the Mux and Demux classes). These modifications could easily be accomplished by deriving new classes from the library entities and by overriding specific methods. The modular architecture also allows to simulate multiple bus systems that are connected via gateway ECUs (electronic control units).

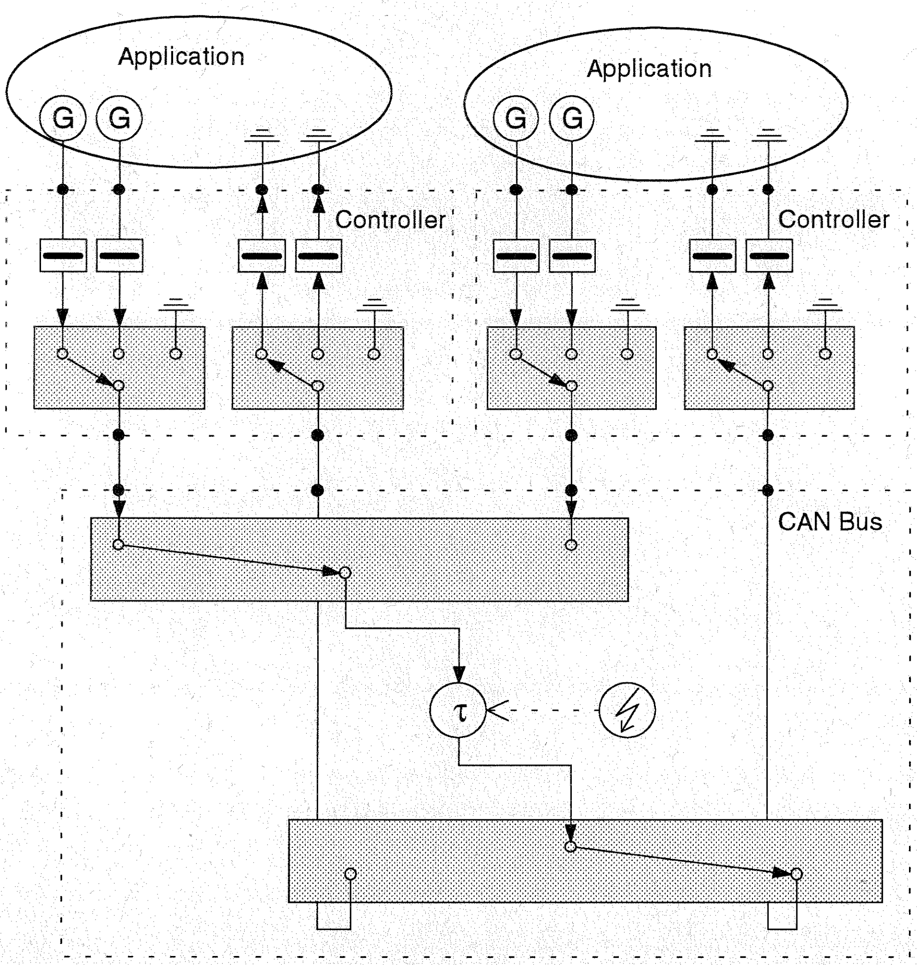

***Figure 3.5:** A complete CAN-Model with two nodes.*

For the moment there are no restrictions to the kind and number of controllers that are attached to the bus but from memory space.

3.3 Using Burst-Silence Generators as an Application Model

Providing the simulation with appropriate generators that represent real attached message sources is a difficult task. In general applications there are predecessor and successor relations, e.g. a message is sent only after another message has arrived.

To cover all possible timing cases we used a burst-silence generator. Figure 3.6 gives the parameters of such a message source.

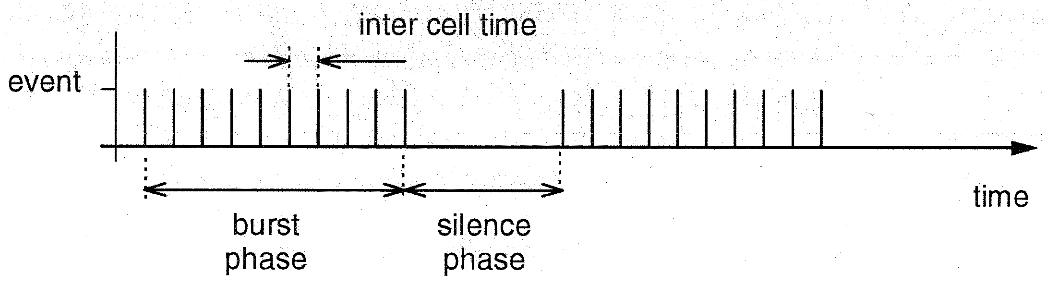

***Figure 3.6:** Burst-Silence Generator*

The burst phase is the active phase of the generator. It is defined by a burst distribution which results in the number of events during the burst phase. Each event is separated by a constant „inter cell time“ that may also be chosen. The third parameter is the silence phase which is again given by a distribution function.

The benefit of using such generators is the possibility to generate all possible phase relations. Therefore a short silence phase is selected that allows small phase dilations.

Typical distributions are a geometric distribution with a mean value of ten for the burst phase and a negative exponential distribution with a mean value of one time unit for the silence phase. The inter cell time is given by the interarrival time of the messages.

3.4 Infrastructure to the Simulation

Preparing a simulation scenario is as easy as describing the controller network with the Open Description Language ODL [13]. That means to define ECUs, the tasks that run on the ECU, the messages with their priorities and choose the desired hardware controller and write it down in a text-based file. A compiler generates the necessary data for the simulation ("Sim-Para“) which then executes automatically: The simulation dynamically parses the description and builds the desired simulation model.

Output is generated by supplying an output-format-file ("Desired Output Specification“). This file contains information about which measurements should be recorded.

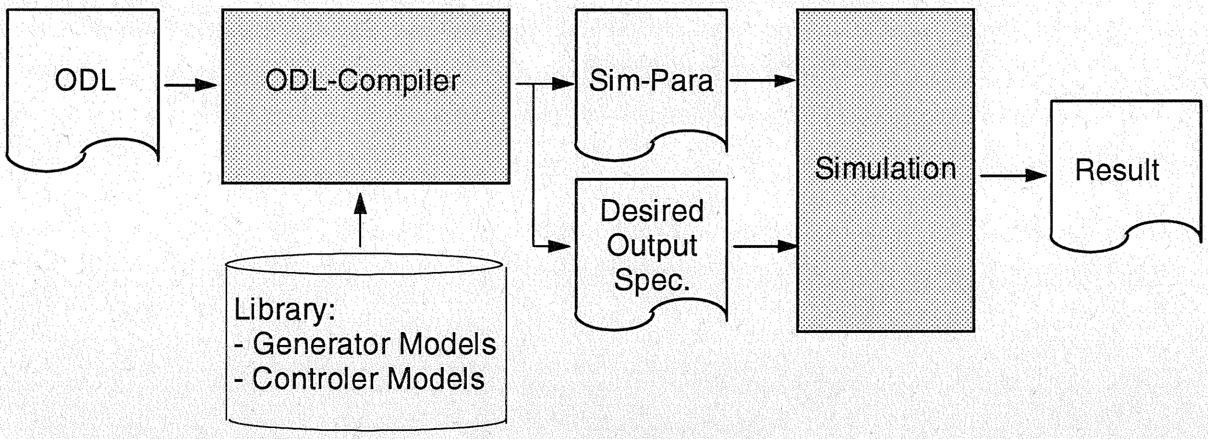

Figure 3.7: *Information flow of simulation*

An extension to the presented model is currently under development: A communications software layer that provides communication mechanisms to the application programmer. Therefore, we see the CAN model as one basic model abstraction and use the input ports of the controllers as interfaces to the overlaying software. The developer of the software model does not need to know how the CAN model works, they just „plug“ the CAN model to their model and then are able to run a whole system simulation. This is also supported by the already mentioned description language that helps to keep the use of the simulation transparent to the user. They just define what kind of generator they want to use and what information is expected from the simulation.

4 Results

In the following section we present some of the obtained results. We implemented the scenario that was proposed by the Society of Automotive Engineers [10]. The scenario originally consists of seven ECUs with 53 messages being exchanged between these ECUs.

To assign all messages to identifiers on real controllers (Intel’s i82527 allows for example for 14 different identifiers) we had to "piggy pack“ some messages. We chose those messages to be put together that have the same deadlines and the same arrival rates. The resulting scenario consists of 42 messages [7]. Selection of priorities follows the "deadline first - jitter“ assignment proposed by Tindell [14]. Release jitter of messages is represented by the bounds for the distribution of the message generators.

Further assumptions regarding the design of the message generators are as follows: we only use cyclic generators. Signals that are not meant to be cyclic but sporadic (e.g. "Emergency Brake") but have a minimal inter arrival time are also considered to be cyclic with the minimal inter arrival time used as cyclic inter cell time.

Due to space limitations we will only present a small selection of our results. For the presentation we picked out the following messages:

CAN ID	SAE-Signal Name	Sent by ECU	Length in byte	ICT in ms	Deadline in ms	Calculated Response Time at 125 kBd in ms
1	Hi&Lo Contactor Open/Close	Battery	1	50	5	1,544
25	Shutdown	I/M_C	1	50	20	exceeds deadl.
40	Motor/Trans Over Temp	Trans	1	1000	1000	exceeds deadl.

Table: 4.1 *Presented Messages*

4.1 Simulating the error-free system

According to the time dilation algorithm [14] worst-case response times can be calculated for these messages. As can be seen in table 4.1 only the first message is supposed to meet its deadline regarding their worst-case. Both other messages are expected to be over their deadlines.

Simulation of the SAE scenario will show that also other messages are supposed to meet their deadlines. Therefore, we extracted the results for the above mentioned message IDs from the simulation.

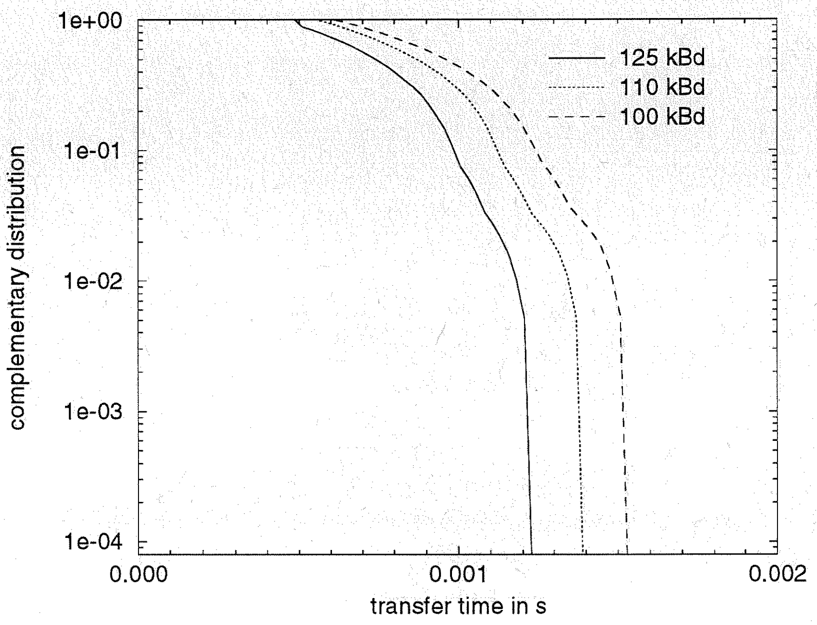

Figure 4.1.a: Message ID 1. Deadline is within 5 ms which will be no problem at any of the simulated bus speeds, because simulation results are always less then 2 ms.

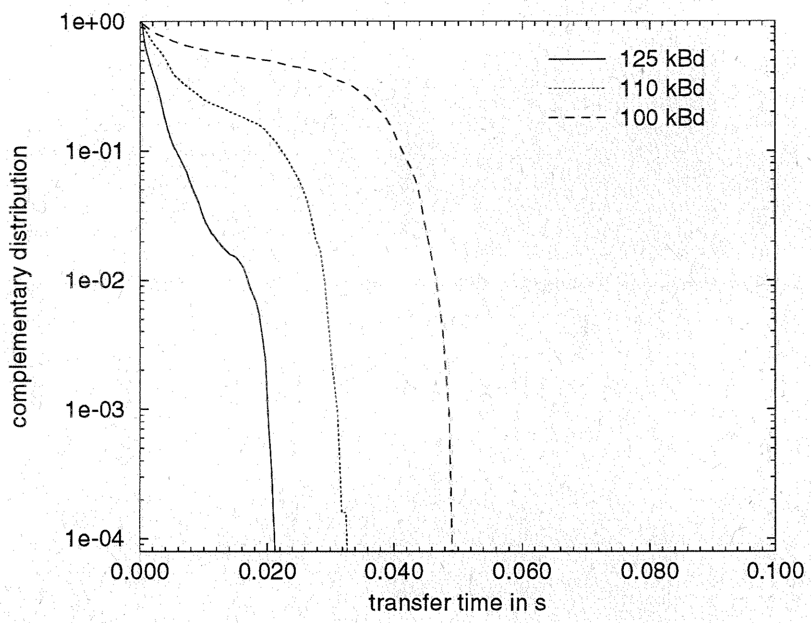

Figure 4.1 b) *Message ID 25. The message's deadline is at 20ms. Simulation gives evidence to the worst-case analysis: the probability to meet its deadline at the simulated speed is very little: at any of the simulated bus speeds there are cases when the deadline cannot be held.*

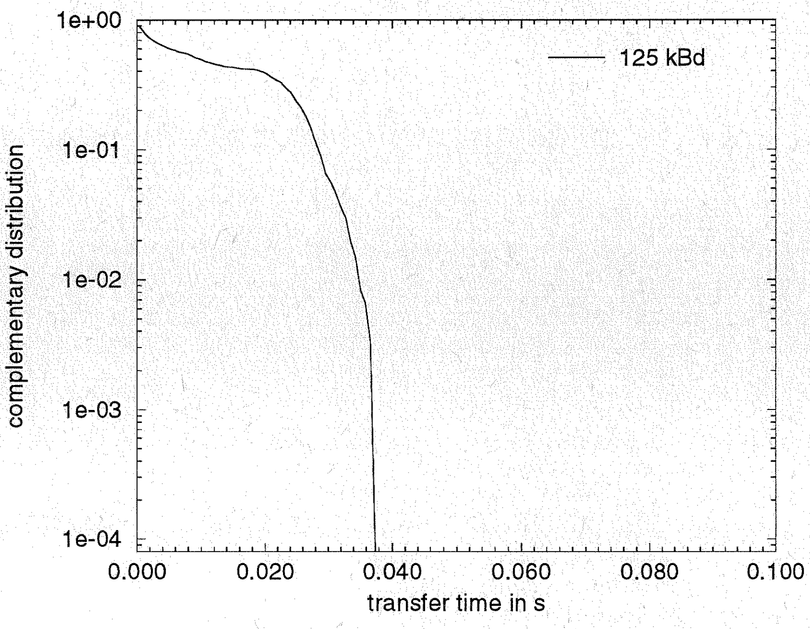

Figure 4.1 c) *Message ID 40. The message's deadline is at 1s which will be met without problems at 125 kBd! But any lower bus rate will result in a loss distribution of 1, i.e. no message at all will be successfully transmitted. The edge between successful transmission and full loss is very steep.*

Figure 4.1 a-c*: Three messages at different bus speeds: 100 kBd, 110 kBd, 125 kBd*

4.2 Simulation with error source

When errors are introduced, the deadlines may not be met any more. To find out at what rate the system will stop working correctly, we increased the disturbance rate. The results are shown for one selected message ID, ID 40.

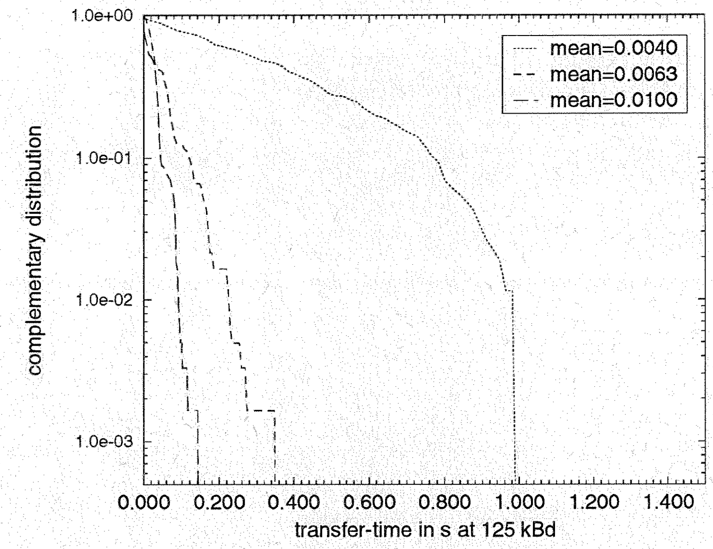

Figure 4.2 *Message Id 40 with increasing disturbance rate. Deadline is at 1.0s. The values for the arrival of disturbing messages are given as the mean values of a negative exponential distribution. It can be seen that an error source with a mean value of 4 ms causes Id 40 to reach the value of the deadline.*

Simulation of the whole scenario shows another interesting aspect: We first give a 3-D plot of the whole system with the influence of an error generator. Figure 4.3 shows the loss probability over all message Ids at various inter error times (1ms .. 100ms mean value of neg. exp. distribution).

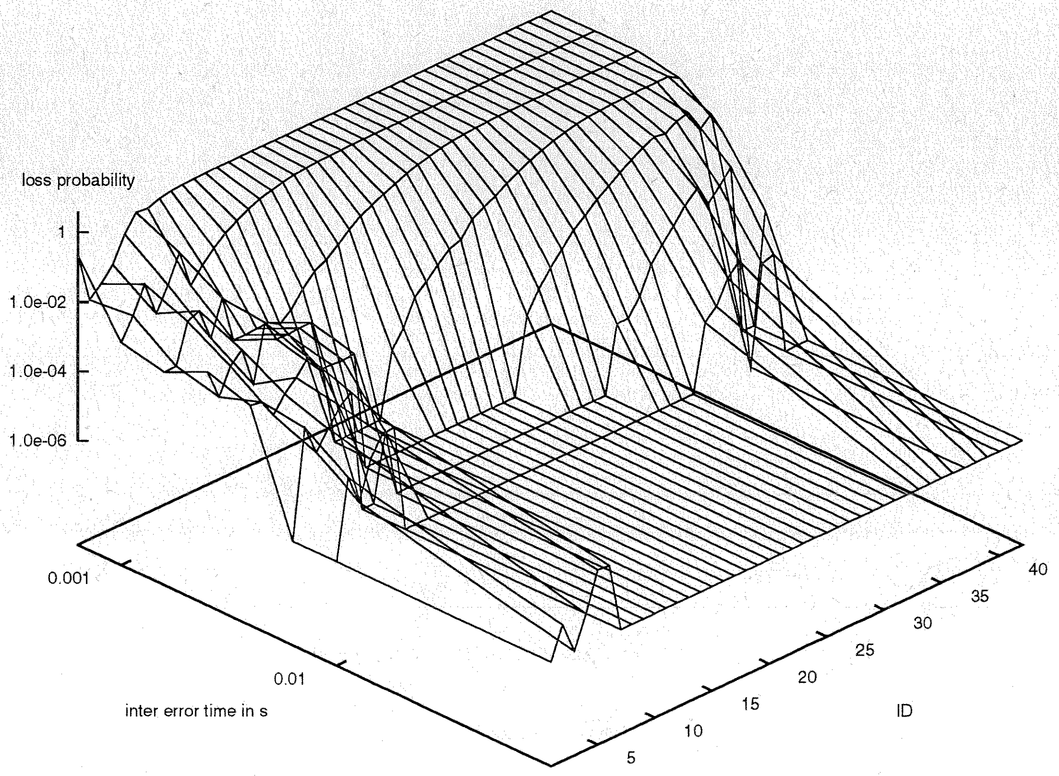

Figure 4.3: *Simulation of the whole scenario with error source. Results less than 10e-6 are equal to 0, i.e. represent no loss.*

As expected, loss probability increases with shorter inter error times. More interesting is the fact that also high priority messages show a certain loss probability even at low disturbance rates. The following picture gives a 2-D plot of this effect.

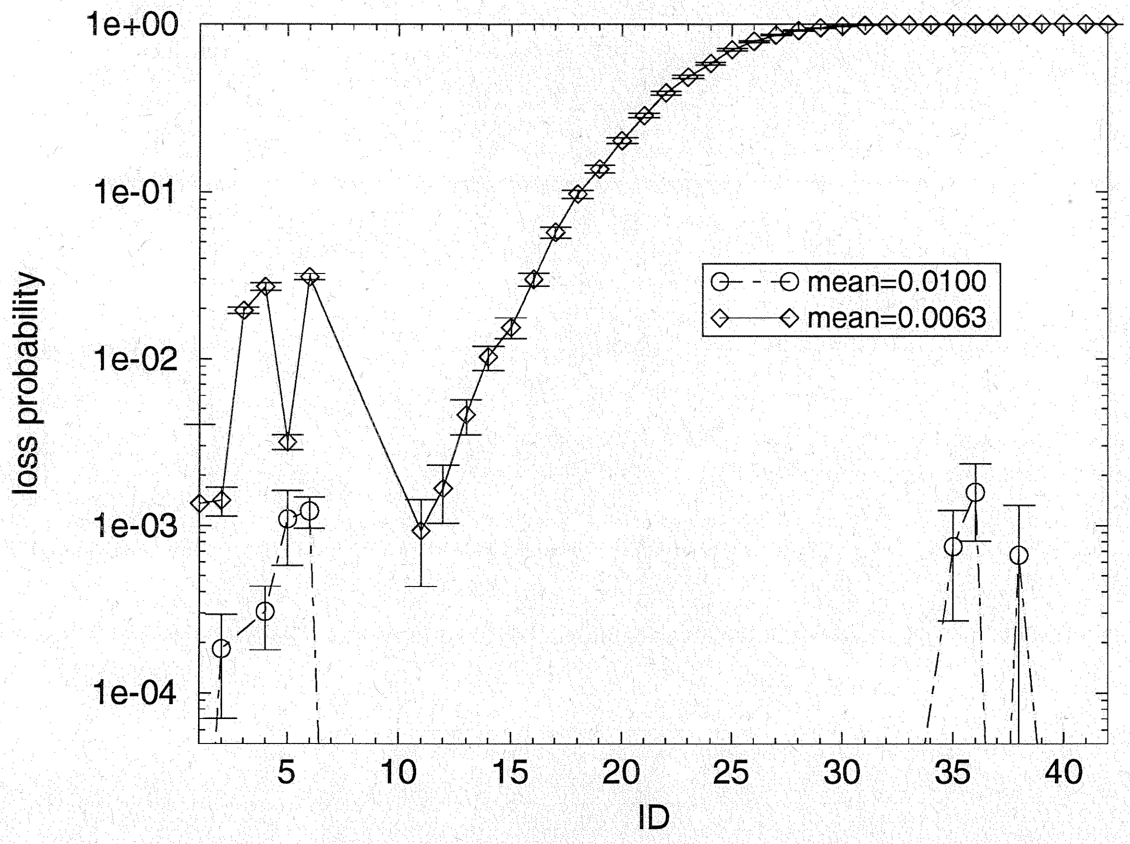

Figure 4.4 *Loss probability over all identifiers at two mean values, 10 ms and 6.3 ms, of the neg. exp. distribution of the error source. Bus rate is 125 kBd. (This plot gives simulation values together with their 95% confidence intervals.)*

The problem in the presented case is that too many requests of even high priority messages arrive in the system that cannot be transmitted due to errors on the bus. This results in a certain loss probability for high and low priority messages whereas messages with intermediate priority are only affected at higher error rates. What can be learned from this is that priority assignment according to deadlines is not necessarily the best choice. A better way might be assignment according to arrival rates (rate monotonic assignment).

5 Summary

In this paper, we presented a flexible object-oriented framework for the simulation of CAN systems. With its help it is easy to build complex and heterogeneous systems for gaining desired information about the performance of the designed systems without the need of having the real system. The whole design and simulation process is supported by the description language ODL.

Our results showed that worst-case analysis is sometimes too hard for real systems. Also priority assignment is not in every case the best choice. Further simulations in this field have to propose better solutions.

Other simulation results showed, as expected for priority systems, that under no circumstances loss of messages can be accepted. This will always result in the loss of almost all further messages.

6 Acknowledgments

The authors would like to thank Oliver Friedrichsohn and Klaus Pantleon, who did a contributing work [2, 7] to this paper.

7 References

[1] Booch, G.: „*Object Oriented Analysis and Design with Applications*“, 2nd Edition, Benjamin Cummings, Redwood City, CA, 1994

[2] Friedrichsohn, O.: „*Modellierung und Simulation von CAN-Systemen mit Hilfe einer objektorientierten Simulationsbibliothek*“, Diploma-Thesis, Institute of Communications Switching and Data Technics, University of Stuttgart, Germany, 1994

[3] Intel Data Sheet. „*82527 Serial Communications Controller*“, October 1993

[4] Kocher, H.: „*Design and Implementation of a Simulation Library Using Object-Oriented Methods*“, Dissertation, Institute of Communications Switching and Data Technics, University of Stuttgart, Germany, 1993. [In German]

[5] Lang, M., Stümpfle, M., Kocher, H.: „*Building a Hierarchical CAN-Simulator Using an Object-Oriented Environment*“, MBB Tools Conference, Heidelberg, 1995

[6] Motorola Data Sheet. „*Microcontroller MC68HC705X16*“, August 1991

[7] Pantleon, K.: „*Simulation von CAN-Systemen*“, Semester Thesis, Institute of Communications Switching and Data Technics, University of Stuttgart, Germany, 1995

[8] Philips Data Sheet. *PCA82C200 Stand-alone CAN-controller*, October 1990

[9] Robert Bosch GmbH: *CAN Specification 2.0*, Stuttgart, 1991

[10] SAE: „*Class C Application Requirement Considerations*“, SAE Technical Report J2056/1, June 1993

[11] Siemens Data Sheet. Stand-alone Full CAN Controller SAE 81C90, Mai 1994

[12] Siemens Data Sheet. „*The On-Chip CAN-Module C167C*“, Mai 1993

[13] Stümpfle, M.: „*Beschreibung von Kommunikationsszenarien in heterogenen automotiven Systemen*“; Proceedings of „Kommunikation in verteilten Systemen“, Chemnitz, 1995

[14] Tindell, K., Burns, A.: „*Guaranteeing Message Latencies on Controler Area Network*“, Proceedings of 1. International CAN Conference, Mainz, 1994