Object Nets for the Design and Verification of Distributed and Embedded Applications

Jürgen Nützel; Bernd Däne; Wolfgang Fengler
nuetzel;bdaene;wfengler@theoinf.tu-ilmenau.de
http://www.theoinf.tu-ilmenau.de/ra1/
Faculty for Informatics and Automation
Technical University of Ilmenau
D-98684 Ilmenau, Germany

Keywords: Object-orientation, Verification, Petri nets, Embedded systems

Abstract: In this paper we present an object-oriented method for the design, verification and implementation of embedded and distributed systems. The method is called Concurrent Object Net (CON). The CON method is based upon extended statecharts which use specific message links for communication. For simulation and verification corresponding Petri nets are used. A platform abstraction framework for CON accesses hardware in optimized manner. An application example from the automotive domain is used to show further CON details.

1 Introduction

The design and implementation of embedded and distributed real-time systems differs hardly from traditional software design known from office applications. Despite all known differences we found object-oriented paradigm suitable for embedded systems as well. But using the object-oriented design techniques for embedded systems means to deal with two contradictory intentions. On one side the object-orientation tries to assist the designer handling huge software systems by providing techniques like abstraction, inheritance and polymorphism. On the other side designing embedded applications stands for accessing hardware directly and using provided computing resources most effectively. This often conflicts with the first intention because the traditional usage of object-orientation doesn't provide the demanded code efficiency for distributed and embedded applications.

For this reason why designed a new object-oriented design method directly for the design and verification of distributed embedded applications. The method is called *Concurrent Object Nets*. The method provides a graphical semantics for classes and objects and their interaction. An internal hidden Petri net representation of the specification supplies the designer with simulation and verification facilities, especially for safety critical systems. A hardware abstraction framework was integrated to realize optimized hardware access in combination with portability and code efficiency.

In the next section a description of the object-oriented design model is given. Afterwards the underlying Petri nets and their simulation and verification possibilities are described. In section 4 the architecture of a visual CASE tool for *Concurrent Object Net* (CON) design is introduced. Section 5 is devoted to Mercedes-Benz's *Electronic Stability Programm* (ESP) which gave us a simplified application example.

2 Design Method for Distributed and Embedded Applications

In this section we introduce a new graphical and object-oriented design method for distributed and embedded real-time systems. The method is called *Concurrent Object*

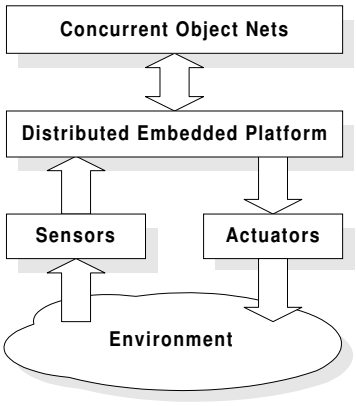

Fig. 1. The Elements of Distributed Embedded Real-Time Systems

Nets (CON) and additionally provides a framework for the abstraction of distributed embedded platforms including their actuators and sensors (see figure 1). The section starts with a discussion about different design properties which are originated from contradictory views on embedded and distributed systems.

2.1 Contradictory Views

If a software designer is directly involved with the logical view on the system to design, she normally expects three fundamental logical realization properties from a design method:

- *Concurrency*. A property coming from the environment of the real-time system. At each time different activities may run concurrently in the environment to which the system has to react simultaneously. *Concurrent Objects*: All objects (instances of Object Net classes) are working concurrently.
- *Reactivity*. A reactive system reacts continuously to signals from the environment. *Active Objects* (like actors) [SeGuWa 94]: All instances have their own thread of control (hierarchical ones have several threads). Each instance is able to react directly to its environment.
- *Guaranty for time and logical restrictions*: In case of safety and real-time requirements facilities will be needed, which ensure that some system states will reached and some other will never occur. Additionally strict time borders for some reactions are needed too. *Safety Objects*: For every Object Net class a set of such constraints may be given. CON design ensures the fulfilment of these constraints.

  If the designer (may be different to the one above) is involved with the physical view

(implementation) on the hardware, she appreciates different properties provided by the Concurrent Object Net platform abstraction framework (PAF).

- *Portability* and *distributability*. The functionality of the specification should be validated (by simulation) before hardware details become visible. The real access function to actuator/sensor hardware should be added after the validation and verification. If an abstract actuator/sensor access exists which hides platform details, automatic software-hardware-mapping algorithms could be applied. Such algorithms generate an optimized and distributed implementation based upon the time restrictions of the specification. The platform abstraction framework (PAF) provides the designer with a class system to wrap the functionality of typical embedded controllers.
- Being *close to the hardware*. Despites the request for hardware wrapping and abstraction the implementation has to access the controller as directly as possible. The lowest classes of PAF allow method implementation in platform's native code (also in assembler).

2.2 Abstract and Refined Object Nets

To make designer's live easier when handling concurrency, safety properties and hardware access we added object-oriented features. These features allow the designer to abstract from code internals, to reuse and refine a design easily.

As already told Object Nets (ON) are concurrent working instances of ON classes which may be also distributed over several platform target nodes. An ON class is a kind of graphical template for the creation of Object Nets instances with certain interface and behaviour. The interface of an ON class consists of a set of so called ports. Message links connect ports of different ON instances with each other. ON instances communicate via these message links (asynchronous or synchronous) by sending simple control messages or messages with user defined data structure. Beside its port interface every ON class has an internal behaviour (beside the abstract ones). Three different meta classes can be distinguished by their internal behaviour:

- An abstract ON class (AONC) has no specific behaviour. AONCs like all other ON classes may have a port interface and an optional list of time and logical constraints. During the design flow all instances of AONCs have to be substituted by instances with refined behaviour.
- A hierarchical ON class (HONC) encapsulates an Object Net which consists of several ON instances connected by a number of message links. These aggregated instances may also be created either from AONC, HONC or elementary ON classes. Selected ports from the interface of the aggregated instances can be exported into the interface of the surrounding HONC.
- An elementary ON class (EONC) encapsulates a hierarchical extended state machine [Harel 87] with additional time delays and time constraints. The EONC is the fundamental building block. HONCs are only used for a better design structure. Hierarchy can be removed without change in operating semantic. The peripheral interface ports are assigned to the internal state changes. Actions (within min./max. duration times for verification) with program code (which can be simulated because

of the coupling with an interpreter language) are assigned to the state changes. Within the actions access to actuators and sensors is possible. This is done through abstract actuator/sensor classes which couple with the ON specification of the environment. Additional local attributes (variables) extend the state space of the EONC.

Within the CON method certain inheritance rules have been defined. A subclass always refines the superclass. An AONC can be refined to either a HONC or EONC. Refining HONCs means (simplified speaking) to add new ports or ON instances (ONI). Each inherited ONI can be overwritten by an ONI which is more refined than the inherited one. EONCs can also be refined by extending their port interface. The inherited state machine can also be refined by adding new parts (like states, state changes, actions, attributes) or by refining the inherited states. The use of all inheritance rules is restricted by the CON design flow.

2.3 Platform Abstraction Framework (PAF)

Within the EONCs actions allow to access peripheral actuators and sensors through objects from special abstract actuator/sensor classes (ASC). These classes wrap the functionality of the hardware of actuators and sensors which are connected to the distributed embedded controller nodes. Abstract ASCs are platform independent. Their methods correspond to the complementary ASO of the Object Net environment specification which is used to create test patterns. In the case of implementation the CON platform abstraction framework (PAF) provides a technology to derive concrete ASCs from the abstract ones by inheritance. These concrete ASCs include the code for the different target controllers.

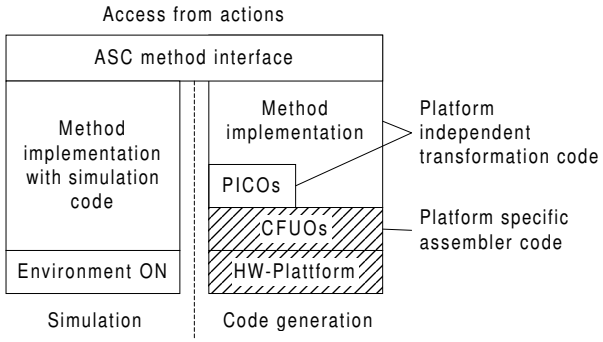

Fig. 2. Actuator/Sensor Classes for Simulation/Verification and Code Generation

As the right half of figure 2 shows the implementation of concrete ASC methods uses two further meta classes from the PAF. *Core function unit classes* (CFUC) encapsulate the controller core specific code which is needed to access the different function units of the core. The PAF provides a selection of abstract CFUCs which reflect typical functional units like ports, watchdogs or counters. These CFUCs will be used to design platform independent *peripheral interface coupler classes* (PICC). PICCs hide the process interface around the controller core. Different DACs and ADCs are typically represented

by PICCs. After the mapping of the ASOs on a specific platform CFUOs from abstract classes will be replaced by objects from concrete classes which are coded using target assembly language in an optimized manner.

Beside the ASC design PAF also supports the design of controller's communication interfaces (CI). CI classes (CIC) also use PICOs and CFUOs for their implementation. Unlike the ASCs the CIC are not visible from the Object Net specification. Their purpose is to implement the message link communication on the top of different types of protocols (e.g. CAN-Bus) and communication hardware.

2.4 The Design Flow with Object Nets

The design flow in the Object Net method is based on the principle of refinement through inheritance. The software designer starts to discuss the problem with her customer. As a result of this discussion the designer creates a first Object Net specification. This specification formalizes the informal requirement of the customer. It includes instances of AONCs. During the refinement (through inheritance) process the designer overwrites the instances from the first specification level with instances which are more specific.

The mechanism of refinement through inheritance is very restrictive in the CON method. Two fundamental inheritance rules define the allowed refinements. The *interface inheritance rule* is based upon the principle that the environment of a subclass can not distinguish between a subclass and its superclass if the subclass has replaced the superclass. The *constraint inheritance rule* forces a subclass to fulfil the constraints of the superclass as well.

If designer's refinements use these inheritance rules the properties of the first specification will be preserved. An automatic (formalized) check detects whether the designer has violated the rules.

At the end of the refinement process a specification with all details will be found. This specification can be graphically simulated. The last step in the design flow transforms the fully refined Object Net specification into a specification which can be implemented on the distributed target platform. The final mapping of the ONIs and ASOs onto the controller nodes is restricted by the node's performance parameters and the delays of the communication network between the nodes. The time constraints from the specification hold further restrictions for the mapping algorithm.

3 Simulation and Verification based upon High-Level Petri Nets

For simulation and implementation every hierarchical ON will be automatically transformed into a flat ON including only EON instances coupled by message links. If an Object Net environment specification already exists it will be integrated. In the second step these flat ONs will be transformed into high-level timed Petri nets using further information from the platform (performance and delay parameters) and mapping specification. After this conversion process the resulted Petri net will be executed/checked in the attached back-end simulator/checker. The conversion principles are shown within the application example at the end of the paper.

The Petri net class of the simulator/checker provides several high-level features which extend traditional Petri nets. Tokens are able to carry data of specific structure. Special queue places are used to model communication buffers directly. Different time extensions are available to describe time consumption of communication and software actions. These times may vary within given min/max limits.

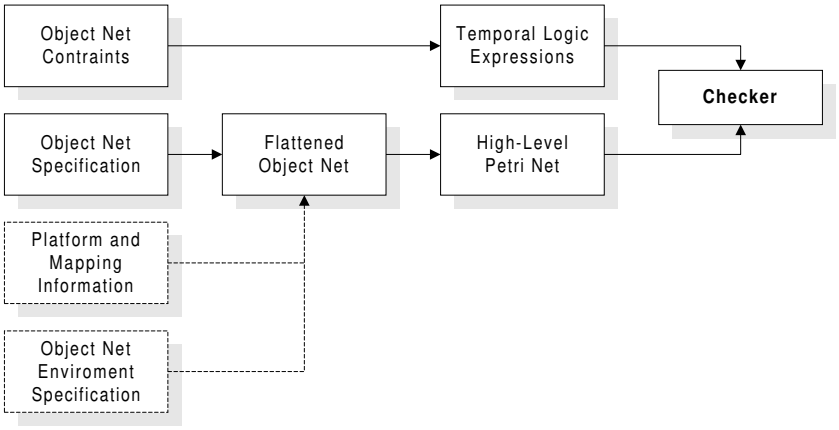

Fig. 3. Formal Verification of Specification Constraints

To check if the corresponding Petri net fulfils the timed and logical constraints of the specification a conversion is needed. After the conversion a set of temporal logic expression will result (see figure 3). These expressions in combinations with the Petri net form the input for the checker. The checker informs the designer whether the specification fulfils the constraints.

4 Visual Design Toolset

In order to assist all the features of the CON method the modular design toolset OSSI (*Object System Specification Inventory*) is in development (see figure 4). It supports the complete design and implementation cycle within a software workgroup. OSSI provides a class/object inventory which is based on a client-server SQL database. The complete class inheritance mechanism is controlled by that inventory. It holds the complete ON class tree and all ON instances designed by a workgroup using CONs. The Object Net class browser and the platform abstraction browser allow the designer to control its project data stored in the inventory. The object mapping interface wraps the physical representation of inventory and makes the tool architecture independent from the selected database technology.

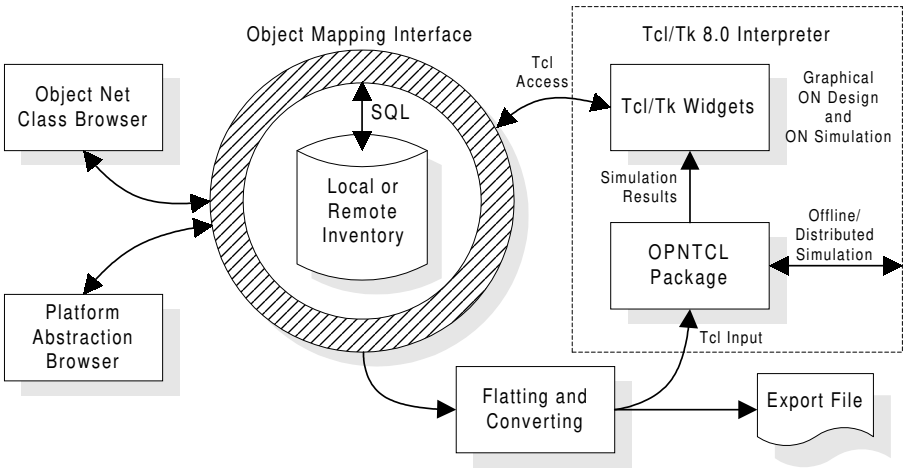

Fig. 4. The Tool Architecture of OSSI

Beside the inventory OSSI integrates the full simulation and verification capability for CON specifications. Within the Delphi designed OSSI a Tcl/Tk (Tool command language/Tool kit) interpreter (version 8.0) is embedded [Ousterhout 94]. The graphical design and simulation interface uses Tcl/Tk widgets for realization. To give the interpreter access to the inventory OSSI provides additional Tcl commands written in Object Pascal. For simulation and checking the OPNTCL [OPNTCL 98],[UnDäNü 98] package is loaded into the interpreter. OPNTCL is a back-end Petri net simulator/checker extension for Tcl which also allows offline and distributed execution using socket communication (e.g. in a UNIX workstation cluster). The package was written in C++ and accepts Tcl-like textual command sequence as input. This command sequence has been generated by the flatting and converting module. It is possible to write the command sequences to file for the use by external code generation modules.

5 Example: Electronic Car Stability System

We use for demonstration an application example which comes from a typical real-time domain - the electronic safety equipment for road vehicles. Our example is a simplified car stability system similar to the Electronic Stability Program (ESP) from Mercedes-Benz, a system which assists the driver in critical driving situations. With ESP, a computer continually monitors car's handling, comparing the data it receives from various sensors with pre-programmed ideal data. The moment the car deviates from its ideal line, ESP takes over control to bring the car back on course.

ESP works in two situations: If the driver enters a left-hand curve too quickly, the car oversteers and threatens to go into a spin. ESP immediately springs into action, applying the brakes to the right front wheel. The car is back on track (figure 5, left). If the car understeers in the same curve and threatens to plow on straight ahead ESP applies the brakes to the left rear wheel, bringing the car back on track (figure 5, right).

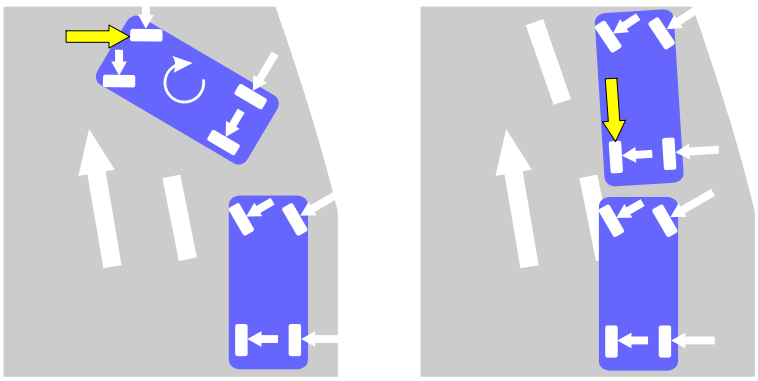

Fig. 5. ESP in Situation of Oversteering and Understeering [MERCEDES 98]

For the demonstration we make certain simplifications. Our example considers only understeering. Consequently we remove control of the front wheels. After this reductions the simplified system works as follows. One sensor measures the position of the steering wheel and two others measure the speed of each rear wheel. These values will be used in the electronic control unit to calculate the lateral acceleration the car should have. A lateral acceleration sensor measures the value the car really has. The measured and the calculated value will be compared. If the difference is too great the system will apply the brakes to one of the rear wheels until the car is back on track. Back on track means calculated and measured lateral acceleration are almost equal. The formula which calculates the brake pressure needed is rather complex and car type specific.

To keep the CON description short we skip the procedure of refinement through inheritance. We start with the fully refined Object Net specification. On this level the

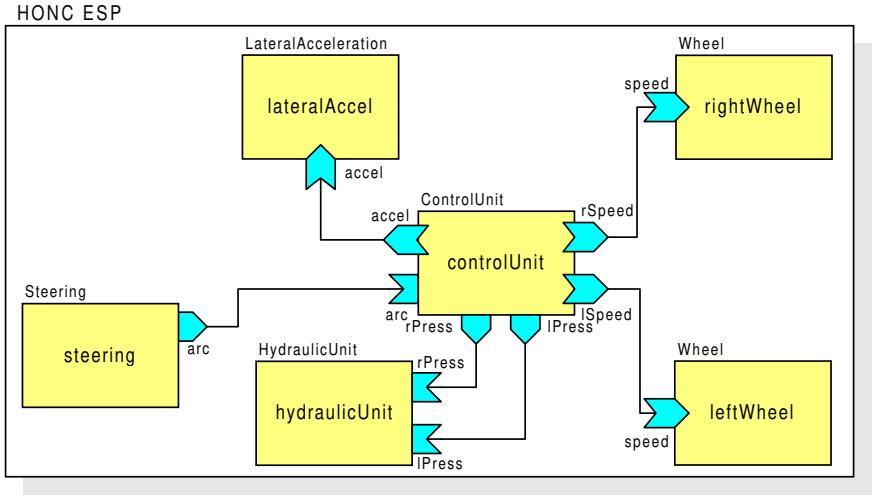

Fig. 6. The System Object Net Specification

HONC *ESP* appears (see figure 6) which aggregates Object Net Instances (ONI) representing the relevant parts of the stability system - the steering wheel (*steering*), the lateral acceleration sensor (*lateralAccel*), the electronic control unit (*controlUnit*), the hydraulic unit (*hydraulicUnit*) and both rear wheels (*rightWheel*, *leftWheel*). The elementary ONI (EONI) *steering* continuously sends the position of the steering wheel through its port *arc* (figure 7, left). The *controlUnit* receives this position value at its port *arc*. The message triggers *controlUnit* to request synchronously three further sensor values (from *lateralAccel*, *rightWheel* and *leftWheel*). Within the action *calcDiff* (figure 7, right) all received values will be combined to calculate a new value (*diff*) which will be used for the decision (by the guards *diffIsLow* and *diffIsHigh*) whether the brakes are to apply (enter state *nobrake*) or not (enter the state *brake*). After that decision new pressure values will be calculated and sent to *hydraulicUnit*.

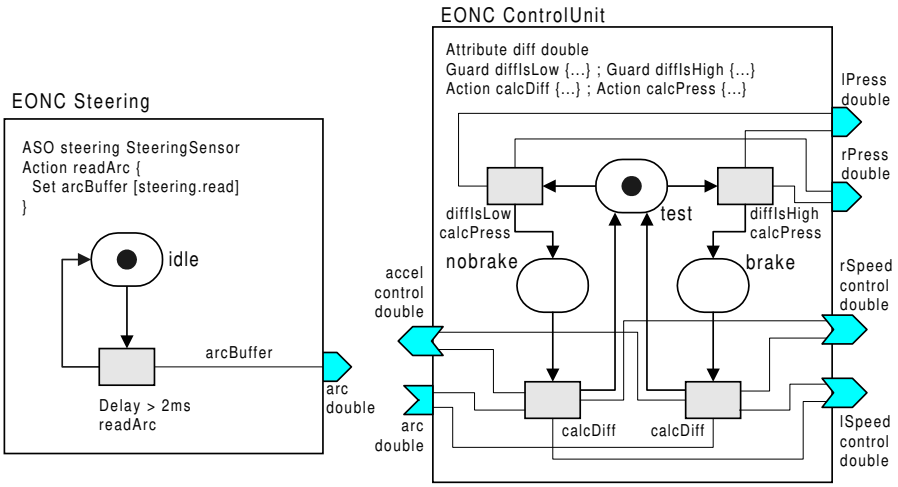

Fig. 7. Two EONCs from the hierarchical Object Net Specification

Before the designer starts to refine the specification by inheritance, she has to ask for the important safety properties of the system. The stability of the car depends on the time from the point of recognition of critical situation to the application of the brakes. In other words, the time between reading the acceleration sensor and changing the brake pressure has to be below a hard limit. A further safety property is the time between the first application of the brakes (state *brake*) and the reentering of normal state (state *nobrake*). All these constraints can be written with the *Object Net Constraint Language* (ONCL). Together with the corresponding Petri net (figure 8 shows a part from it) the verification can be started, as shown in figure 3.

To finish the description of the application we show a part of the corresponding high-level Petri net which is invisible to the designer. Figure 8 shows the EONI *steering*, a part of the EONI *controlUnit* and the message link between them. The right part of the net shows the port *accel* and its assignment to the state change. The white places may carry

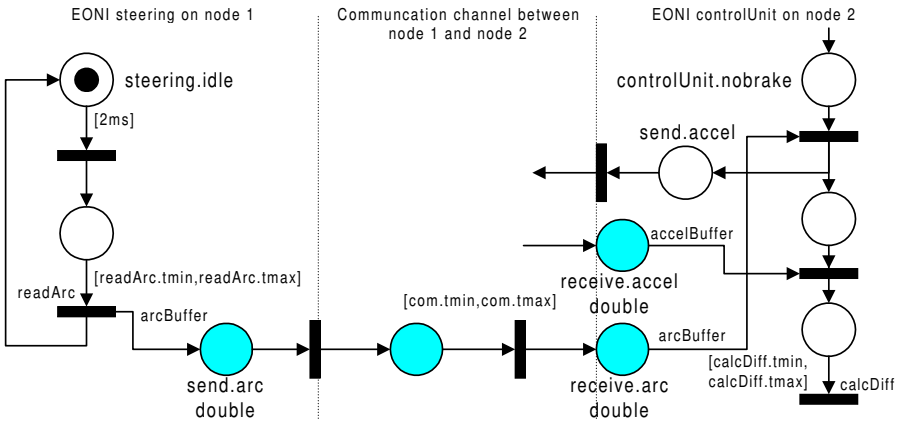

Fig. 8. A Part of the Corresponding Petri Net

only "black"control tokens. Gray places may carry data tokens describing the messages. Some transitions which activate actions or describe communication delays have additional time intervals. Within these intervals the transition fires.

6 Conclusion

The Concurrent Object Net (CON) method in combination with its platform abstraction framework (PAF) was designed for easy access to the domain of distributed embedded systems. Our intention was to address engineers who are familiar with hardware details but not willing to become a computer scientist. Beside easy programming of heterogenous platforms we focused on verification of safety critical systems. Our future goals will be to rise the acceptance of simulation and verification in the embedded system domain.

References

[Harel 87] Harel, David: Statecharts: A visual formalism for complex systems, Science of Computer Programming, Vol. 8, p. 231-274, 1987

[MERCEDES 98] Daimler-Benz: The Mercedes-Benz Homepage, 1998: http://www.mercedes-benz.com/

[OPNTCL 98] Nützel, Jürgen: Opntcl (Object Petri Nets based on Tcl) Homepage, 1998: http://www.theoinf.tu-ilmenau.de/opntcl/

[Ousterhout 94] Ousterhout, John: Tcl and the Tk Toolkit, Addison-Weseley, 1994

[SeGuWa 94] Selic, B.; Gullekson, G.; Ward, P. T.: ROOM - Real-Time Object-Oriented Modelling, John Wiley & Sons, 1994

[UnDäNü 98] Unger, H.; Bäne, B.; Nützel, J.: Experiences Simulating the Load Sharing System LYDIA with High Level PN, HPC'98, Boston, April 1998