Published in:
Szczerbicka, H. (ed). 1999. *Modelling and simulation: A tool for the next millenium. 13th european simulation multiconference 1999.* Society for Computer Simulation International, Delft, Vol. I, 575-580.

NON-STANDARD GRAPHICAL SIMULATION TECHNIQUES FOR TEST SPECIFICATION DEVELOPMENT

Jan Peleska[1], Stefan Bisanz[1], Ingo Fiß[1] and Manfred Endreß[2]
[1]Universität Bremen FB3,
P.O. Box 330440
D-28334 Bremen, Germany
E-mail: {jp,bisanz,fiss}@informatik.uni-bremen.de
[2]Daimler Chrysler Aerospace Airbus, K.I.D.-Systeme
Lüneburger Schanze 30
D-21614 Buxtehude, Germany

KEYWORDS
Virtual Reality, Event-oriented, Real-time, Aerospace, Man-machine interfaces

ABSTRACT

In this article we present a graphical simulation-based method and tool for the development of formal Timed CSP (TCSP) test specifications. Typically, this method is used in the context of automated testing of embedded real-time systems with complex user interfaces (our first industrial application is the hardware-in-the-loop test of the Airbus Cabin Intercommunication Data System CIDS). While the problem of generating, executing and evaluating test executions from TCSP specifications is already solved and supported by appropriate tools, the creation of the formal specifications still represents a serious stepping stone, since it does not only require expertise about the application to be tested but also considerable skill in the field of Formal Methods. Our graphical test specification method aims at facilitating this task. The method is based on two types of graphical models: (1) A virtual reality (VR) representation of the system interface and its operational environment allowing to define timed traces of events and (2) a graphical meta language allowing to construct complete TCSP specifications from the specifications fragments developed in the VR representation. We use the term 'non-standard simulation', because in our context simulation is not performed to animate an existing model of system behaviour. Instead we start with an *incomplete* simulation model containing only basic interface descriptions and – using the interaction features of the VR representation and the graphical meta language – *create* a TCSP model which may be used for testing purposes.

INTRODUCTION

Testing embedded real-time systems is a complex and time-consuming task, since – in order to analyse the reactive behaviour of the *System Under Test (SUT)* – test executions consist of *sequences* of interleaved inputs and outputs, and correctness conditions are related to (1) the value of SUT outputs, (2) the sequencing and (3) the timing of inputs and outputs. Since timing requirements are often specified in units of milli seconds and below, manual test control is usually insufficient with respect to both the generation of appropriate executions and the evaluation of the SUT behaviour. As a consequence, tests are usually performed as *hardware-in-the-loop tests*, where executions are controlled by computers stimulating the SUT inputs and monitoring its outputs. Before starting a hardware-in-the-loop test, the (set of) executions has to be defined in a machine readable manner. For this purpose, two main techniques are applied: The conventional technique uses *test scripts* describing linear sequences of inputs to be sent to the SUT and associated expected SUT outputs. The scripting technique has some serious disadvantages: (1) The length of the script to be designed grows linearly with the length of the execution to be performed. (2) If the SUT exhibits non-deterministic behaviour – for example, due to internal scheduling decisions which are invisible at the system interface – the script execution will abort if the SUT reacts with an unexpected output which may be perfectly legal, but just not admitted as the next step of the script. Therefore *specification-based* testing has been investigated during the last decade, which has led to automated test generation, execution and evaluation methods based on *models* specifying the complete set of legal – possibly non-terminating – SUT executions. See (Brinksma 1989) for an early result.

A test automation system for embedded real-time systems is *RT Tester* (formerly called VVT-RT) which has been developed by Verified Systems International GmbH (Bremen) in cooperation with Bremen University. See (Peleska 1996), (Peleska 1997), (Peleska and Siegel 1997), (Peleska 1998) for the underlying theory. The formal language we use to specify tests is *Timed CSP (TCSP)* in the semantic interpretation of (Schneider 1995). A comprehensive introduction to CSP is given in (Hoare 1985), (Roscoe 1998), an overview of TCSP syntax and semantics is given in Section "Formal Specification Language Timed CSP". A TCSP test specification is interpreted in real-time by the RT Tester tool which generates SUT inputs, observes timing conditions and checks the correctness of the timed traces performed by the SUT. Since 1995 RT Tester has been used in several industrial applications in the field of train control, avionics and space applications. Presently, the tool is used in a hardware-in-the-loop test system designed by Daimler-Chrysler Aerospace Airbus, K.I.D.-Systeme in collaboration with Verified Systems International GmbH for the test of the Cabin Intercommunication Data System CIDS used in the Airbus family. CIDS controls all cabin-related communication aspects in the Airbus aircraft, for example, the passenger address system, the cabin interphone system, the No Smoking and Fasten Seat Belt signs and the cabin illumination. In addition to interfaces to other aircraft computers, CIDS communicates via several man-machine interfaces passing control events, visual information and sound to and from the passengers and cabin crew. Based on TCSP specifications, the test system stimulates all the CIDS input interfaces and checks proper reactive behaviour.

Since test specifications have to be interpreted by the computers driving the tests, they must be defined in a formal language. In contrast to test scripts which are usually fairly easy to construct because each step refers to explicit interfaces and data, the development of formal specifications requires considerable expertise. As a consequence, application experts possessing the knowledge about the required SUT behaviour do not necessarily have the skills to write a formal test specification. This serious disadvantage of specification-based test methods is addressed in the present article: We describe a test specification method which uses graphical VR simulations as a front-end and – guided by some rather simple commands defined in a graphical meta language – transforms the user's interactions during a test specification session into a TCSP specification.

Up to now, VR simulations have been widely used in the *animation* of specifications: In general, specification languages – whether based on textual syntax like TCSP or graphical syntax such as StateCharts – represent *abstractions* of the SUT and its operational environment, whereas VR simulations posses much closer a correspondence to the associated real-world entities. Therefore specifications are often easier to validate in a VR animation than in a symbolic execution of the specification language itself. The novelty of our approach is that VR simulations are now used to *create* the desired specifications from interface specifications in combination with user interactions on the virtual interface objects. While it is obvious how to construct test scripts in the VR simulation (you just select the interface objects in the sequence desired and define the values to be passed along these interfaces to/from the SUT), it is a non-trivial task to elaborate a method allowing to define arbitrary TCSP specifications including parameterized branching, recursion and potentially synchronized parallelism. Our solution to this task is presented in the main section "Test Specification By Graphical Simulation" of this article.

FORMAL SPECIFICATION LANGUAGE TIMED CSP

The TCSP core syntax is given in BNF notation as follows:

$$\begin{aligned} P \ ::= \ & P \underset{A}{\parallel} P \mid P \underset{A\ A}{\parallel} P \mid P \mid\mid\mid P \mid \\ & P \setminus A \mid f(P) \mid f^{-1}(P) \mid P \ \Box \ P \mid \\ & \Box_{i \in I} P_i \mid P \sqcap P \mid \textstyle\bigsqcap_{i \in I} P_i \mid \\ & X \mid P;\, P \mid \mathrm{STOP} \mid \mathit{SKIP} \mid \\ & \mathit{WAIT}[t_1, t_2] \mid a \to P \mid P \overset{t}{\rhd} P \end{aligned}$$

Operator $\underset{A}{\parallel}$ describes parallel composition with synchronization over events in set A; $\mid\mid\mid$ describes unsynchronized parallel composition. Operator $\setminus A$ defines abstraction from events contained in set A; $f(P)$ denotes renaming of P-events according to the event mapping f. Operator $\to$ is used to specify sequences of event occurrences. Operator $\Box$ denotes external choice which can be controlled by the environment while $\sqcap$ denotes internal choice not to be controlled from the outside, operator ; denotes sequential composition. Mutually recursive process equations may be defined by using free process variables in CSP terms and associate these variables with other process terms. Process $STOP$ blocks everything and never engages into any event, $SKIP$ denotes successful termination. There are two timing operators: $WAIT[t_1, t_2]$ denotes a nondeterministic delay $d \in [t_1, t_2]$ (t_1 may be equal to t_2); the timeout operator $P \overset{t}{\rhd} Q$ offers P but not Q up to time t and – if no P-event occurred – offers Q but no longer P after time t. The full set of TCSP constructs is described in (Schneider 1995).

TEST SPECIFICATION BY GRAPHICAL SIMULATION

Test Specification Sessions

In a test specification session, users navigate through a VR model showing the target system interfaces as embedded into their operational environment and interact with interface objects. Input interfaces to the target system are accessed by the user to specify input data to be passed to the target system during test execution. Output interfaces are accessed to specify expected reactions of the target system. The graphical meta language is used to combine specification fragments created by interactions in the VR simulation into complete test specifications.

Virtual Reality Interaction Concept

Virtual Periphery

The collection of objects modelling target system interfaces as embedded into their operational environment is called *virtual periphery*. Man-machine interfaces (e. g., switches, buttons and display panels) can be represented by a close approximation of their appearance in the real operational environment. However, it has to be taken into account that the target system may also interact with other sub-systems that would normally not be visible to users in the operational environment but have to be considered in the test specification process (e. g., sensors delivering data to the target system). Such sub-systems are represented as “black-boxes” with abstract symbols representing their interfaces. The virtual periphery offers two basic modes of interaction: (1) Visit a specific location in the VR-model, (2) get a zoomed presentation of a specific interface to facilitate interactions on this interface.

Example 1
In the present version of our specification system we use VRML (VRML Consortium 1997) to create a model of the virtual periphery as embedded into its operational environment. To give an example, Figure 1 shows a background view of an aircraft cockpit with a zoomed view of the cockpit switches controlling Fasten Seat Belts and No Smoking Signs. The pop-up menu at the left-hand side of the switches gives a list of the locations which may be visited in the VR-model; for example, to move from the cockpit into the cabin.

Interaction Methods For Interface Objects
When creating the VR simulation model, each interface object in the virtual periphery is associated with a class name which is used for all interfaces of the same type and a set of *attributes* that may be referenced in a test specification session in order to specify the objects to be accessed in a convenient way. For example, each interface object in the CIDS VR model is associated with a location in the airbus cabin or the cockpit. At present, we only use attributes remaining constant during the test specification session, but in principle it would be possible to change object attributes during the session.

In addition to its attributes, each interface object exports *interaction methods* which may be activated during a test specification session by acting upon the interface object with a pointing device. These methods can be classified as follows: (1) *Data and event definition methods:* During a test specification session, users will specify the data to be passed along input interfaces to the target system or the data to be expected on output interfaces. Depending on the type of data, different specification methods will be appropriate. For example, the “input data” associated with the Fasten Seat Belt switch shown in Figure 1 consists of events indicating the changes ON $\rightarrow$AUTO $\rightarrow$OFF $\rightarrow$AUTO $\rightarrow$ON of the switch position. An appropriate method to stimulate such an event is to click on the upper or lower part of the switch. For the loudspeakers of passenger service units as shown in Figure 2 it may be useful to consider a collection of pre-defined acoustic signals such as chimes or prerecorded announcements as “output data” which may be selected as the expected target system output in the test specification session. The inputs or outputs to be passed along interfaces with large data ranges may be specified by means of pop-up menus allowing data selection from lists of values or by means of predicate specifications. (2) *Methods changing the dynamic appearance:* The selection of interface data may change the appearance of an object. For example, the above mentioned switch changes to and remains in the middle position after stimulation of the $ON \rightarrow AUTO$ event. When selecting a data value such as “50%” as expected output on the indication module shown in Figure 3, this output interface object should change its appearance by moving the associated highlighted bar to the 50%-position. (3) *Interface selection methods:* An object may be selected in two different ways which may be toggled by invoking the interface selection method: In explicit mode, data and event definition methods are only applied to the one object which has been selected. In implicit mode, the selected object acts as a place holder for all interfaces of the same type whose attributes satisfy an attribute specification defined using the meta language. In implicit mode, all objects satisfying the same attribute specifications are marked by identical colours. These selection modes will be illustrated in the examples below. (4) *Time Specification Methods:* To specify that a target system output should occur within a time interval $[t_1, t_2]$ after the preceeding communication event, each output interface object allows to define such an interval. For input interface objects the time interval may also be defined; in the test execution this will lead to waiting for a random time $t \in [t_1, t_2]$ before producing the input.

Graphical Design Of Timed CSP Specifications

Following the structure of TCSP syntax, we will now describe how TCSP test specifications may be generated from interactions with the virtual periphery in combination with additional commands in the graphical meta language.

Prefixing And Delay
Input/output sequences $a_1 \rightarrow a_2 \rightarrow \dots$ are created using the virtual periphery by selecting each associated interface (explicit selection mode) in the desired order and choosing the data value/event for input or expected output, respectively. Each interaction will create a CSP event *iname.atts.data* where *iname* is the name associated with the interface class, *atts* is the list of attribute values associated with *iname* and *data* is the selected data value or event. The interaction with the virtual periphery is accompanied by displaying the CSP fragment resulting from the interactions so far in a specification display window.

Example 2
Figure 2 shows the Passenger Call button on the upper part of a Passenger Service Unit. The button is associated with the single event *press* corresponding to pressing the button. An output interface object modelling the indicator light (not visible in Figure 2) is associated with each button object. To specify the test

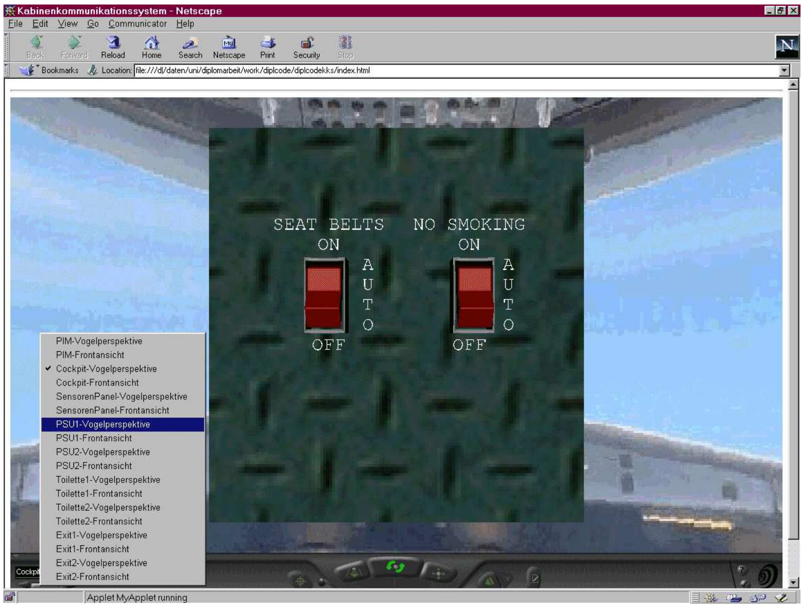

Fig. 1. VRML-generated Simulation view of the Fasten Seat Belts and No Smoking cockpit switches.

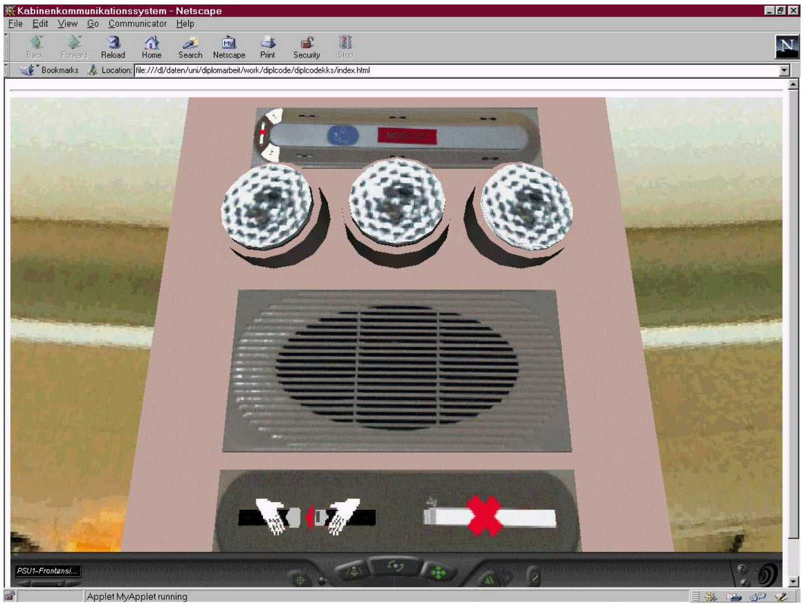

Fig. 2. VRML-generated Simulation view of a Passenger Service Unit showing Fasten Seat Belt Signs and No Smoking Signs, Reading Lights and Speakers.

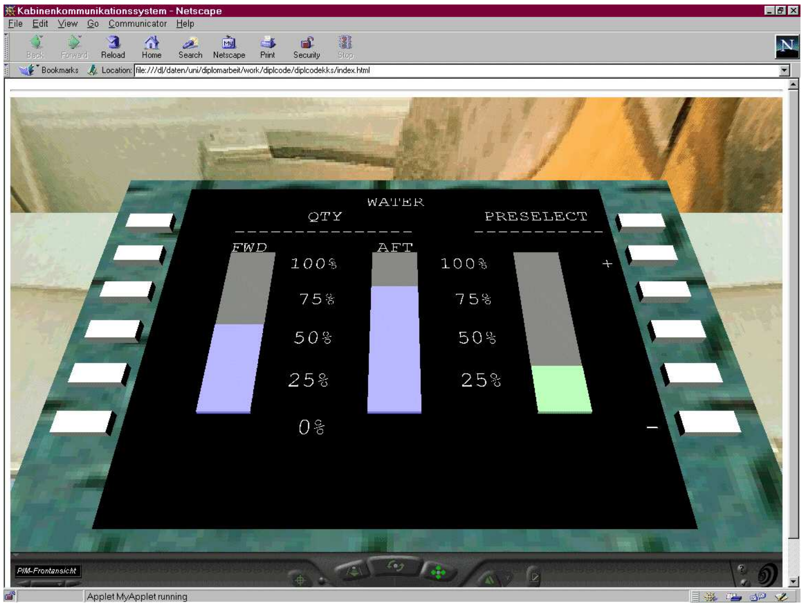

Fig. 3. VRML-generated Simulation view of the Airbus CIDS Programming and Indication Module.

sequence "after pressing the Passenger Call button the corresponding indicator will be switched on within 1 sec", we start by creating a new named specification fragment *SPEC* using the graphical meta language. Next we enter a mouse click on the selected Passenger Call button object in the virtual periphery. Since only one event is specified for this button, this will immediately lead to CSP event $pax\_call.\langle seatrow\rangle.press$ on the specification display window. Here $\langle seatrow\rangle$ is an attribute of the button interface object located at seat row number $\langle seatrow\rangle$ in the virtual periphery. Next, we point to the indicator object at this seat row, choose the value *on* and enter the timing specification $t \in [0,1]$. The specification display window will now show the fragment

$$
\begin{aligned}
&SPEC = \\
&\quad pax\_call.\langle seatrow\rangle.press \\
&\quad \rightarrow WAIT[0,1]; \\
&\quad pax\_call\_indicator.\langle seatrow\rangle.on \\
&\quad \rightarrow
\end{aligned}
$$

Termination And Recursion

To define a terminating test script, we simply terminate the specification fragment constructed so far by a *STOP* or a *SKIP* using the graphical meta language. Termination by *SKIP* indicates that the specification fragment $SPEC_1$ may be re-used in another context using sequential composition $SPEC_1;\ SPEC_2$. This type of re-use is not possible for fragments ending with the *STOP* process. However, the main advantage of the specification-based test approach in contrast to the script-based approach is that we can specify non-terminating executions using recursion. A recursive test specification is created by terminating a specification fragment *SPEC* in the graphical meta language with a process identifier $SPEC_2$ not equal to *SKIP* or *STOP*, but possibly equal to *SPEC* itself. This can be used to define a set of mutually recursive TCSP process equations.

Example 3

We extend the previous example by pressing the Passenger Call button again after some time $t \in [10,20]$ which should lead to switching off the indicator light within one second. By terminating the fragment with its own name *SPEC*, we create a recursive test specification

$$
\begin{aligned}
&SPEC = \\
&\quad pax\_call.\langle seatrow\rangle.press \\
&\quad \rightarrow WAIT[0,1]; \\
&\quad pax\_call\_indicator.\langle seatrow\rangle.on \\
&\quad \rightarrow WAIT[10,20]; \\
&\quad pax\_call.\langle seatrow\rangle.press \\
&\quad \rightarrow WAIT[0,1]; \\
&\quad pax\_call\_indicator.\langle seatrow\rangle.off \\
&\quad \rightarrow WAIT[10,20];\ SPEC
\end{aligned}
$$

This specification will lead to an infinite test execution checking the correctness of the indicator control at location $\langle seatrow\rangle$ and choosing different wait times in $[10,20]$ before generating the "press button" event again.

Choice And Quantification

So far we only have been capable of generating linear test specifications. The motivation for using choice operators is twofold: (1) A choice of inputs to the target system defines several possible continuations of a test execution in one test specification: On different executions, the test driver will select different branches of the choice construct according to the test strategy used. (2) A choice of possible target system outputs is useful when the target system may exhibit non-deterministic behaviour at the system interface: The test execution may always continue, as long as each observed output is in the specified choice of expected outputs. A choice operator is specified by activating the appropriate choice symbol in the graphical meta language. After that, each selection of an interface object in the virtual periphery will create a new branch using the specified choice operator. After having defined all the relevant branches, the user may select each branch in turn to specify the corresponding continuations. The explicit definition of choice is tedious if a large number of branches exists. To facilitate such a situation, we define attribute specifications in the meta language and access just one object as placeholder for all interfaces whose attributes match the specification. Similar techniques may be used to specify the choice ranging over a subset of data that may be passed along one interface.

Example 4
To specify a non-terminating test producing *press*-events for arbitrary Passenger Call buttons located in seat rows 5 to 10 and checking the corresponding indicator as in the example above, we first activate the attribute specification $5 \leq seatrow \wedge seatrow \leq 10$ in the meta language. This will highlight all interface objects associated with a seat row attribute of value in 5, ... , 10. Next we select the internal choice button in the meta language and select the Passenger Call button interface object in any seat row between 5 and 10, using implicit selection mode. This results in the specification fragment

$$SPEC = \sqcap_{seatrow:5..10}(pax\_call.seatrow.press \rightarrow$$

where *seatrow* is now an attribute parameter ranging over numbers 5 to 10. Still with implicit selection mode, an indicator light is accessed in this range of seat rows, the timing specification is entered as above and the *on*-data value is selected. We proceed as in the recursion example above, but keep the implicit selection mode. This leads to the specification

$$
\begin{array}{l}
SPEC = \\
\quad \sqcap_{seatrow:5..10}(\\
\quad\quad pax\_call.seatrow.press \\
\quad\quad \rightarrow WAIT[0,1]; \\
\quad\quad pax\_call\_indicator.seatrow.on \\
\quad\quad \rightarrow WAIT[10,20]; \\
\quad\quad pax\_call.seatrow.press \\
\quad\quad \rightarrow WAIT[0,1]; \\
\quad\quad pax\_call\_indicator.seatrow.off \\
\quad\quad \rightarrow WAIT[10,20];\ SPEC \\
\quad)
\end{array}
$$

combining quantified choice and recursion. External choice is used analogously.

Other TCSP Operators
The other TCSP operators – parallel composition, abstraction, timeout, sequential composition and renaming – are produced by acting upon existing process specifications using the graphical meta language. Details are described in (Bisanz and Fiß 1999).

CONCLUSION

In this article, we have presented a method for developing TCSP specifications to be used for automated testing of reactive real-time systems. The development method is based on a VR simulation of the target system interfaces combined with a graphical meta language. The interaction concept described for the VR simulation and the meta language allows to define a fairly large set of TCSP specifications, especially specifications of the type which is useful to test systems with complex user interfaces. Examples were taken in simplified form from the test design for the Airbus CIDS communication system. At present, further graphical interaction paradigms are elaborated, with the objective to construct generic specifications and layered specifications connected by abstraction/refinement relations. The method has been implemented in a specification and simulation tool developed at Bremen University, see (Bisanz and Fiß 1999).

REFERENCES

Bisanz, S. and I. Fiß. 1999. "Grafischer Entwurf von CSP-Spezifikationen für den Test eingebetteter Echtzeitsysteme." Diploma Thesis, FB3, University of Bremen.

Brinksma, E. 1989. "A theory for the derivation of tests." In *The Formal Description Technique LOTOS*, P. H. J. van Eijk, C. A. Vissers and M. Diaz, eds. Elsevire Science Publishers B. V., North-Holland, 235-247.

Hoare, C. A. R. 1985. *Communicating sequential processes.* Prentice-Hall International, Englewood Cliffs.

Peleska, J. 1996. "Test Automation for Safety-Critical Systems: Industrial Application and Future Developments." (invited keynote). In *FME '96: Industrial Benefit and Advances in Formal Methods*, M.-C. Gaudel and J. Woodcock, eds. LNCS 1051, Springer-Verlag, Berlin Heidelberg New York, 39-59.

Peleska, J. 1997. "Formal Methods and the Development of Dependable Systems." Habilitationsschrift, Bericht Nr. 9612, Institut für Informatik und praktische Mathematik, Christian-Albrechts-Universität Kiel (Dec. 1996)

Peleska, J. 1998. "Testing Reactive Real-Time Systems." Tutorial, held at the FTRTFT '98, Denmark Technical University, Lyngby. New revision available as `http://www.informatik.uni-bremen.de/~jp/papers/ftrtft.ps`.

Peleska, J. and M. Siegel. 1996. "From Testing Theory to Test Driver Implementation." In *FME '96: Industrial Benefit and Advances in Formal Methods*, M.-C. Gaudel and J. Woodcock, eds. LNCS 1051, Springer-Verlag, Berlin Heidelberg New York, 538-556.

Peleska, J. and M. Siegel. 1997. "Test Automation of Safety-Critical Reactive Systems." In *South African Computer Jounal*, (1997)19:53-77.

Roscoe, A. W. 1998. *The Theory and Practice of Concurrency.* Prentice-Hall.

Schneider, S. 1995. "An Operational Semantics for Timed CSP." In *Information and Computation*, 116:193–213.

VRML Consortium. 1997. *The Virtual Reality Modeling Language.* International Standard ISO/IEC 14772-1:1997, Part 1 – 3. VRML Consortium Inc.