Compositional Verification of SDL descriptions

Matti Luukkainen[1], *Ari Ahtiainen*
Nokia Research Center
P.O.Box 407, FIN-00045 Nokia Group, Finland
tel: +358-9-437 66072, fax: +358-9-437 66856
email: Matti.Luukkainen@research.nokia.com

Abstract

Model-checking has shown to be an efficient and easy-to-use technique in verification of formally described programs. However, there is one major drawback in using exhaustive model-checking: the behavior models of real-life programs tend to be extremely large. In the article it is shown how the theories of behavioral equivalences with a compositional style of state space generation can alleviate the state space explosion in verifying the externally observable properties of SDL descriptions.

Keywords

SDL, behavioral equivalences, minimization, compositional verification

1 INTRODUCTION

Testing and debugging of concurrent and reactive programs is extremely tedious, partly due to the nondeterminism caused by the computation environment. If a program is described by using a formal language, such as SDL [1], behavior of the program can be defined by means of a mathematically defined structure, such as a *behavior graph*, which describes all the possible computation sequences of the program.
If the correctness requirements of a formally defined program are given in a mathematical notation, such as linear temporal logic [2], branching time temporal logic [3] or infinite state automaton [4], an algorithm called *model-checker* [5] can be used to check if the program respects its correctness requirements. The model-checker goes through every possible computation sequence of the program, thus it is said to be an *exhaustive* verification technique. Because all the possible program runs are went through, model-checking gives a total confidence of the programs correctness.

Model-checking has shown to be an efficient and easy-to-use technique in program verification. However, there is one major drawback in using exhaustive model-checking: behavior graphs of real-life programs, for example communication protocols, tend to be extremely large. In literature this problem is often referred as *state space explosion*. In this article we show how theories of behavioral equivalences with a compositional style of state space generation help us to analyze large program descriptions by alleviating the state space explosion.

The article is structured as follows: Section 2 motivates the use of labelled transition systems as behavioral model of SDL descriptions. Section 3 shows how the behavioral equivalence theories and compositional style of state space generation could be used in alleviating the state space explosion. In section 4, the compositional method is adapted to the context of SDL. Section 5 show results of applying the compositional method in verification of Inres protocol. Conclusions are finally drawn in section 6.

2 EXTERNALLY OBSERVABLE BEHAVIOR

The behavior of a SDL description can be defined by means of a behavior graph. A behavior graphs consists of nodes and transitions between the nodes. A node describes a state of the program (i.e. values of the program

[1]The corresponding author

variables, contents of message queues ...) at one moment of time and transitions describe the atomic actions (i.e. assignment, sending or receiving of a message ...) that change the state of the program. Transitions may be labeled with the corresponding action names. Intuitively, the behavior graph of a SDL description contains all the possible computation sequences of the described system. The behavior graph corresponding to a SDL description can be obtained implicitly by interpreting the description according to formal semantics of SDL, defined in Annex F.1 of the Z.100 standard [6].

When using formal techniques in developing reliable software, good tool support is extremely important. Commercial SDL tools, like SDT [7] and ObjectGEODE [8] offer some means to use model-checking in system verification. However, there is a serious drawbacks in model-checking facility of both the tools. When using exhaustive model-checking to verify a program, both of the tools save the complete behavior graph of the program to computers memory. Since SDL descriptions comprise usually lots of data variables, generation of the complete behavior graph is extremely memory consuming. Thus, the size of the programs that can be exhaustively model-checked is limited.

State space explosion is handled within the tools by using non-exhaustive model-checking methods, such as checking only some randomly selected computation paths or using the bit state hash technique described in [9]. Drawback of the non-exhaustive methods is that only part of the computations are checked and thus, they do not give full confidence about the correctness of a program.

We are sometimes interested only in the external behavior of a system, in another words, the activity of the system which is visible to an external observer, such as the communication actions between the system and its environment. For example if we specify a communication protocol, it is enough that the service it offers (i.e. the external behavior visible to the user of protocol) is consistent with the correctness requirements of the protocol. Thus, as far as protocol behaves as expected, we do not care how the behavior is achieved.

3 THE EQUIVALENCE THEORY

Externally visible actions of a system specified in SDL are the $input$ and $output$ statements it uses to communicate with the environment. Rest of the actions within the system and its internal state (e.g. values of different variables) are not interesting if only the external behavior is concerned. As a consequence, the external behavior of a SDL description can be captured by a simpler structure than behavior graph, namely a *labelled transition system*, or LTS in short.

A labelled transition system is a quadruple (S, Σ, Δ, s_0)*, where*

- S *is a set of states,*
- Σ *is the set of observable actions,*
- $\Delta \subseteq S \times \Sigma \cup \{\tau\} \times S$ *is the transition relation, (*τ *denotes an internal action i.e. an action invisible for the external observer) and*
- $s_0 \in S$ *is the initial state.*

External behavior of a program is easily modeled as a labelled transition system. Let us consider a simple example, a coffee vending machine. A labelled transition system describing the external behavior of the machine is in Figure 1. In the initial state `s0`, the machine is waiting for a coin. When a customer inserts a coin, the machine moves to another state (transition $\mathtt{s0} - \mathtt{coin} \rightarrow \mathtt{s1}$). In this state, the machine either serves the coffee to the customer (transition $\mathtt{s1} - \mathtt{coffee} \rightarrow \mathtt{s0}$) or breaks down. The breaking down is modeled as a internal action τ which takes place nondeterministically. Once the machine is broken, it just accepts more coins without doing anything else.

With the LTS representation of programs, we have possibility of comparing syntactically different programs with respect to their behavior. Thus, we can check if two programs P_1 and P_2 behave similarly, or if a program $Impl$ is a 'valid implementation' of another program $Spec$. We can even search algorithmically for a 'minimal'

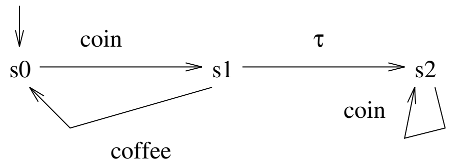

Figure 1: A simple vending machine

program which behaves similarly as a given program. Next we define formally what is intended by saying that two programs behave similarly, in another words the concept of behavioral equivalence is defined. There exists a vast amount of different equivalences in the literature, see for example [10]. In the following we use the CFFD equivalence [11, 12] as our notion of behavioral equivalence.

The following customary definitions are used:

- $P - a \rightarrow P'$ means that there is transition a from state P to the state P', in another words $(P, a, P') \in \Delta$.
- $P - a \rightarrow$ means that there is exists a state P', such that $P - a \rightarrow P'$.
- $\neg(P - a \rightarrow)$ means that it is not possible to find a state P', such that $P - a \rightarrow P'$.
- $P - a_1 a_2 \ldots a_n \rightarrow Q$ iff $\exists P_0, \ldots, P_n$ such that $P_0 = P, P_n = Q$ and $\forall i \in \{1 \ldots n\} : P_{i-1} - a_i \rightarrow P_i$.
- $P = b_1 b_2 \ldots b_n \Rightarrow Q$ iff $P - \tau^* b_1 \tau^* b_2 \tau^* \ldots \tau^* b_n \tau^* \rightarrow Q$, where τ^* denotes a finite or empty sequence of internal actions.

To define the CFFD equivalence, following concepts are also needed:

- $\sigma \in \Sigma^*$ is a *divergence trace* of a state P if $\exists Q : P = \sigma \Rightarrow Q \wedge Q - \tau^\infty \rightarrow$, where τ^∞ denotes an infinite sequence of internal actions.
- The set of divergence traces of a state P is denoted by $div(P)$. Divergence traces of a LTS corresponds to the divergence traces of its initial state.
- A state P is *stable*, if $\neg(P - \tau \rightarrow)$. Predicate $stable(P)$ denotes stability of the state P. A LTS is stable if and only if its initial state is stable.
- Pair (σ, A), where $\sigma \in \Sigma^*$ and $A \subseteq \Sigma$ is a *stable failure* of state P, if and only if $\exists Q : P = \sigma \Rightarrow Q \wedge stable(Q) \wedge \forall a \in A : \neg(Q - a \rightarrow)$.
- The set of stable failures of a state P is denoted by $sfail(P)$. Stable failures of a LTS corresponds to the stable failures of its initial state.

Finally we define the CFFD equivalence relation between two LTS's. *Labelled transition systems* $lts = (S, \Sigma, \Delta, s_0)$ *and* $lts' = (S', \Sigma', \Delta', s_0')$ *are CFFD-equivalent,* $lts_1 =_{CFFD} lts_2$*, iff*

- $stable(s_o) = stable(s_o')$,
- $sfail(s_0) = sfail(s_0')$,
- $div(s_0) = div(s_0')$.

If two programs, P and P' are CFFD equivalent, the intuitive meaning is that both have the same set of possible computation sequences and furthermore P can deadlock after a sequence of actions if and only if P' can also deadlock after the same sequence of actions. Also the computation sequences that lead to a divergence (infinite sequence of internal actions) are the same for the two processes.

In [13] it was proven that

> *CFFD is the weakest congruence relation for finite state systems, with respect to the operators in LOTOS [14], CSP [15] and CCS [16] process algebras, which preserves truth of nexttime-less linear temporal logic (or LTL' for short) [2].*

In practice the compositionality means that we can replace a LTS with a CFFD equivalent LTS in context of any of the LOTOS, CSP and CCS operators. It should be noted that this is not the case with all behavioral equivalences.

Because of compositionality, we can use the following method in building a minimized LTS describing the behavior of a given system. Assume that the correctness criteria of a system are given in REQ which is a set of LTL' formulae. Let us consider that our system consists of several concurrent entities. Naturally all the entities are modeled as separate LTS's, $lts_1, \ldots, lts_n$. Let us assume that $||$ is a parallel operator that enforces synchronization on all the common actions of the combined LTS's, similarly as CSP parallel operator. A LTS describing the behavior of the system is achieved by parallel composition of the separate components $sys = lts_1 \; || \; \ldots \; || \; lts_n$. After building the LTS describing systems behavior, a model-checker is used to test if the system respects the requirements given by the LTL' formulae in the set REQ.

A much more efficient and memory saving way to build the LTS of the system is to first minimize the separate system components with respect to CFFD-equivalence, and after that combine the minimized components ${lts_1}^{min}, \ldots, {lts_n}^{min}$ to form the LTS describing the behavior of system $sys^{min} = {lts_1}^{min} \; || \; \ldots \; || \; {lts_n}^{min}$. Because CFFD is a congruence relation, it is guaranteed that $sys =_{CFFD} sys^{min}$. Furthermore, because CFFD minimization preserves the truth of all LTL' formulae, the correctness of the system can be detected by model-checking the the minimized system description sys^{min} against the formulae in the set REQ. More formally, the following is true for all LTL' formulae ϕ: $sys \models \phi$ iff $sys^{min} \models \phi$.

The fact that CFFD is the *weakest* LTL' preserving congruence relation guarantees that the minimized LTS is optimally small and still contains all the information needed in verification.

The compositional method of building LTS's is not restricted to the use of CFFD and linear temporal logic only. We can also use other equivalence relations which are congruence at least with respect of parallel operator. Another equivalence relation we have used is the observational equivalence [16] which is defined next.

A binary relation $\mathcal{R}$ between states is a *weak bisimulation* if $(s_1, s_2) \in \mathcal{R}$ implies that for all actions a, the following is true:

- $s_1 - a \rightarrow s_1'$ implies $\exists s_2' : s_2 = a \Rightarrow s_2'$ and $(s_1', s_2') \in \mathcal{R}$
- $s_2 - a \rightarrow s_2'$ implies $\exists s_1' : s_1 = a \Rightarrow s_1'$ and $(s_1', s_2') \in \mathcal{R}$

Two labelled transition systems, $lts = (S, \Sigma, \Delta, s_0)$ and $lts' = (S', \Sigma', \Delta', s_0')$ are observational equivalent, $lts_1 \approx lts_2$, if there exists a weak bisimulation $\mathcal{R}$, such that $(s_0, s_0') \in \mathcal{R}$.

Observational equality between two LTS's means intuitively that each of them can 'simulate' the visible behavior of the other. Minimization with respect to observational equivalence preserves the visible behavior and branching structure, but information on τ-loops is lost. Because τ-loops are not preserved, observational equivalence does not preserve all the *liveness* properties. Liveness is usually considered in context of *fairness* requirements. A typical fairness requirement in context of communication protocols is to assume that an unreliable communication channel does not lose all the messages, more accurately: if infinite number of messages are sent to a channel, infinite number of messages is also received by the other end. If τ-loops of a LTS are considered as fair (for example if τ-loops are resulting of retransmissions to a unreliable channel), minimization with respect of observational equivalence preserves also the liveness properties. If still needed, the τ-loop preserving version of observational equivalence [17] can be used.

4 COMPOSITIONAL STATE SPACE GENERATION FROM SDL DESCRIPTIONS

Externally visible actions of a system specified in SDL are the *input* and *output* statements it uses to communicate with the environment. Rest of the actions within the system, and its internal state (e.g. values of different variables) are not interesting if only the external behavior is concerned.

As it was stated in previous sections, labelled transition system is a convenient mathematical structure to describe the external behavior of a SDL system. LTS describing the behavior of a SDL system is easily obtained from the systems behavior graph in the following way.

- For every node in the behavior graph there is a state in the LTS.
- If there is a transition between two nodes in the behavior graph, there should also be a transition between the corresponding states in the LTS.
- If a transition in the behavior graph is caused by an input or an output action, the corresponding transition in the LTS should be labelled with the action name.
- Other transitions in the LTS should be labelled with τ, thus they are invisible to the external observer.
- The state in LTS corresponding to the root node of behavior graph is the initial state of the LTS.

If we would like to verify the external behavior of a system specified in SDL, we can generate the corresponding LTS and use a suitable minimization to make the model-checking less memory and time consuming. Most transitions in a LTS describing the external behavior of a system are internal ones, thus it is evident that the minimized LTS is much smaller than the original one.

The same analogy as above can be used to define the external behavior of an arbitrary component (a process or a block) in a SDL specified system. Externally visible actions of a component are the $input$ and $output$ statements it uses to communicate with the rest of the system and the environment. Thus, the external behavior of an arbitrary component can also be described by means of a LTS. Note that a channel in a SDL description can also be modeled as a LTS.

We can now use the following procedure to build compositionally a minimized LTS describing the behavior of a complete SDL specified system.

1. Produce the behavior graphs ($bg_1 \ldots bg_n$) from the separate system components.
2. Translate $bg_1 \ldots bg_n$ to corresponding labelled transition systems $lts_1 \ldots lts_n$.
3. Minimize the LTS's with respect to a suitable equivalence relation.
4. Model the channels combining separate system components by LTS's. Note that a limited channel length has to be assumed to keep the system finite stated.
5. Combine the minimized LTS's and channels by using a synchronous parallel operator, and minimize the result.

The resulting LTS describes actually more than merely the external behavior: it contains also systems internal signalling. The LTS which describes only the external behavior of the system is obtained from the LTS resulting the above procedure by transforming the internal signalling actions to internal actions. This transformation can be done by using *hide*-operator of LOTOS process algebra. The operation just renames the corresponding labels with τ's. After hiding, the LTS is minimized again. The resulting LTS describes the external behavior of the original SDL specified system. If CFFD equivalence is used in minimization, all the safety and liveness properties of the original system can be checked from the generated LTS. In case of observational equivalence, the visible behavior and branching structure is still preserved in the minimized model.

If the internal signalling of the system or parts of it are of interest, hiding of internal signals can be skipped (or done only for those signals that are not needed). Thus, the compositional verification method is not limited to external behavior only.

4 CASE STUDY: THE INRES PROTOCOL

In this section results of applying the compositional method in verifying the *Inres protocol* [18] are presented. The Inres protocol implements a reliable, connection-oriented data transfer service, *the Inres service*, between two users. The Inres service is not symmetrical: it offers only one way data transition from an initiating process to a responding process. The protocol itself operates above a medium which offers a connection-less unreliable data transfer service. Basic architecture of the Inres system is shown in Figure 2.

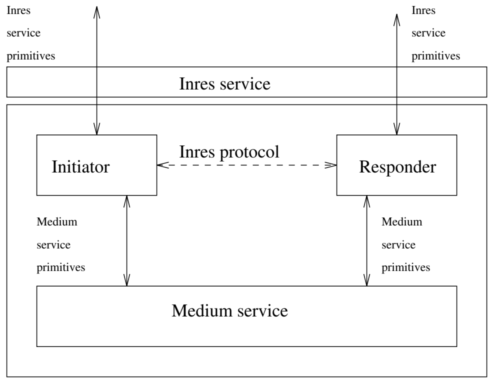

Figure 2: Basic architecture of the Inres system

SDL description of the Inres protocol is presented in [19]. The description consists of four processes, two at the both ends of the protocol (see the Figure 3). Two of the processes $Initiator$ and $Responder$ implement the service by exchanging protocol data units between each other. Two other processes, $coder\_ini$ and $coder\_res$ are used to hide the interface to the medium. The protocol implementation details (i.e. the SDL code) are not shown here, so please consult the referred book for a detailed description.

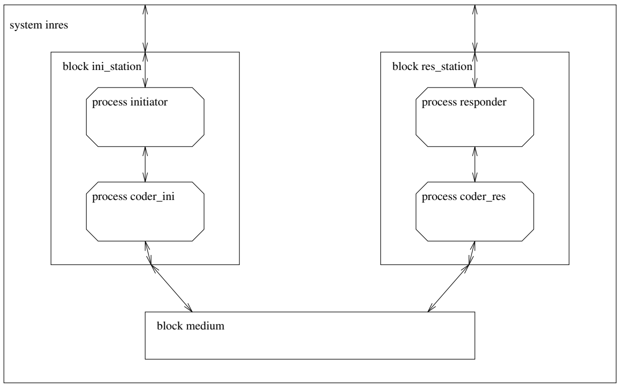

Figure 3: SDL skeleton of Inres protocol

The compositional method described in previous sections has been tested with the Inres protocol. A LTS corresponding to the external behavior of the protocol (i.e. the service it provides) was generated from the original SDL descriptions. The generated LTS was used in verifying the offered service. SDT Validator [20] was used to generate the behavior graphs of the processes in Figure 3. They were algorithmically translated to LTS's according to guidelines given in section 4. The labelled transition systems representing the channels and process input queues were generated with ARA toolset [21]. Cæesar/Aldebaran [22] was used in composition and minimization of LTS's. The minimization was carried out with respect to observational equivalence. Due to lack of a adequate tool support, we were unfortunately not able to use CFFD minimization in the case study.

Generation of the LTS corresponding to the external behavior of Inres protocol proceeded as follows:

1. LTS's corresponding the four SDL processes in Inres protocol description were generated and minimized.
2. The LTS's $initiator$ and $coder\_ini$ were combined by parallel operator. Internal communication between the processes was hidden and the result ($ini\_station$) was minimized. The input queue with size of one was used.
3. Analogously $responder$ and $coder\_res$ were combined by parallel operator. Internal communication was

hidden and the result ($res\_station$) was minimized.

4. The resulting LTS's were combined by parallel operator with an unreliable channel, with length of one. Communication over the channel was hidden and the resulting LTS, $Inres$ was minimized.

Size of the LTS's generated during the process can be seen from the Table 1. As it can be noticed, the resulting minimized LTS which describes the service provided by Inres protocol, has only 28 states and 71 transitions, so it is almost visually verifiable. The largest LTS generated during the process (i.e. the one which had the greatest memory requirements) had 3217 states and 13614 transitions. Because the Cæsar/Aldebaran tool is capable of handling LTS's with over a million states, the compositional style could be used with a far larger systems than Inres protocol.

As a comparison we also tried to generate the state space of Inres protocol with SDT validator. We selected the input queue length of one and channels without capacity, so the same configuration as in our compositional model. The protocol turned to be too large for exhaustive state space generation, thus we used bit state hash technique to compute a lower bound of the state space. With a low hash collision percentage, SDT validator generated 388408 unique system states and over 1880000 transitions. So, a system which was impossible to exhaustively verify with SDT validator was easily handled with our compositional method.

component	without minimization states, transitions	minimized states, transitions
$Initiator$	131, 189	30, 75
$Coder\_ini$	92, 161	10, 18
$Responder$	35, 44	14, 21
$Coder\_resp$	101, 177	10, 19
$ini\_station$	3217, 13614	1424, 6771
$res\_station$	947, 3859	391, 1845
$inres$	135, 264	28, 71
State space generated by SDT	388408, 1880000	

Table 1: Sizes of labelled transition systems generated in verifying the Inres protocol

5 CONCLUSIONS

In the text we pointed out the inability of current commercial SDL tools to cope with state space explosion in exhaustive model-checking. We proposed a labelled transition system based framework to compositionally build the description of a systems external behavior. The key step of our framework was the minimization of LTS's with respect to a suitable equivalence relation. One such is the CFFD equivalence which preserves the truth of nexttimeless linear temporal logics (LTL'), thus the minimization with respect to CFFD equivalence does not change the outcome of LTL' model-checking. Another equivalence that we have used is the observational equivalence, which preserves all the observable computation sequences and also the branching information. Even though observational equivalence abstracts away the information on infinite internal computation sequences, it can be argued to bee a good choice if internal loops are considered as fair (i.e even if there is a τ loop in a labelled transition system, a computation leading to an infinite τ-sequence is not performed).

We also presented results of a case study where the compositional method was used to generate a LTS describing the external behavior of Inres protocol. The initial results were extremely promising, and it can be concluded that our compositional method is in particular suitable for verification of external or signalling behavior of SDL specified systems.

The behavior graph of a SDL specified system is obtained implicitly by interpreting the specification according to formal semantics of SDL. However, the formal semantics of SDL is currently being revised [23] and the new semantical model will be based on transition systems. In the new formal semantics, behavior of a system component is defined compositionally from behavior of its subcomponents. Thus, the new semantical model gives an explicit operational interpretation to all the system components. With the new formal semantics, the translation from systems behavior graph to the labelled transition system describing its external behavior can be explicitly formalized.

6 ACKNOWLEDGEMENTS

Authors would like to thank Professor Martti Tienari and PhD Roope Kaivola from University of Helsinki for great instruction, inspiration and helpful comments and Sari Männynsalo from Nokia Research Center for reading and commenting the paper.

7 REFERENCES

[1] ITU-T. *Recommendation Z.100 - CCITT Specification and Description Language*. ITU, 1993.

[2] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems, Specification*. Springer-Verlag, 1991.

[3] E.M. Clarke and E.A. Emerson. Design and Synthetis of Synchronization Skeletons Using Branching Time Temporal Logics. In *Workshop on Logic on Programs*, number 131 in Lecture Notes in Computer Science. Springer-Verlag, 1981.

[4] W. Thomas. Automata on Infinita Objects. In J. van Leeuwen, editor, *Handbook of Theoretical Computer Science*, Volume B, Formal Models and Semantics, pages 133–191. Elsevier, 1990.

[5] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic Verification of Finite State Concurrent System Using Temporal Logic Specifications. In *ACM Transactions on Programming Languages and Systems*, volume 8, pages 244–263, 1986.

[6] ITU-T. *Recommendation Z.100 Annex F.1 - CCITT Specification and Description Language*. ITU, 1993.

[7] Telelogic Malmö AB. *SDT 3.1 Users Guide, SDT 3.1 Reference Manual* . Telelogic, 1997.

[8] Verilog. *GEODE - Technical Presentation*. Verilog, 1994.

[9] G. Holtzman. *Design and Validation of Computer Protocols*. Prentice-Hall, 1991.

[10] R. van Glabbeek. The Linear Time Branching Time Spectrum II: The Semantics of Sequential Systems with Silent Moves. In *CONCUR 93, Fourth International Conference on Concurrency Theory*, number 715 in Lecture Notes in Computer Science, pages 66–81. Springer-Verlag, 1993.

[11] A. Valmari and M. Tienari. An Improved Failures Equivalence for Finite-State Systems with a Reduction Algorithm. In *Protocol Specification, Testing and Verification XI*, pages 3–18. North-Holland, 1991.

[12] A. Valmari and M. Tienari. Compositional Failure-Based Semantic Models for Basic LOTOS. *Formal Aspects of Computing*, 7:440–468, 1995.

[13] R. Kaivola and A. Valmari. The Weakest Compositional Semantic Equivalence Preserving Nexttime-less Linear Temporal Logic. In *CONCUR 92, 3nd International Conference on Concurrency Theory*, number 527 in Lecture Notes in Computer Science, pages 207–221, 1992.

[14] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *Computer Networks and ISDN Systems*, 14:92–100, April 1987.

[15] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.

[16] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[17] J.A. Bergstra, J.W. Klop, and E.-R. Olderog. Failures Without Chaos: a New Process Semantics or Fair Abstraction. In *Formal Description of Programming Concepts - III*, pages 77–103. Elsevier Science Publishers, North-Holland, 1987.

[18] D. Hogrefe. OSI formal specification case study: the Inres protocol and service. Technical Report 91-012, University of Bern, 1991.

[19] J. Ellsberger and D. Hogrefe and A. Sarma. *SDL Formal Object-oriented Language for Communicating Systems*. Prentice Hall, 1997.

[20] A. Ek. Verifying Message Sequence Charts with the SDT validator. In *6th SDL forum - Using Objects*. North-Holland, 1993.

[21] A. Valmari and R. Savola. Verification of the Behaviour of Reactive Software with CFFD-semantics and ARA tools. In *Proceedings of ESA International Symposium On-Board Real Time Software*. ESTEC, Nordwijk, The Netherlands, 1995.

[22] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP (CÆSAR/ALDEBARAN Development Package): A Protocol Validation and Verification Toolbox. In *Proceedings of the 8th Conference on Computer-Aided Verification*, number 1102 in Lecture Notes in Computer Science, pages 437–440. Springer-Verlag, 1996.

[23] R. Gotzhein, B. Geppert, F. Rössler, and P. Schaible. Towards a New Formal SDL Semantics. In *SAM98, 1st Workshop of the SDL Forum Society on SDL and MSC*, 1998.