Electronic Notes in Theoretical Computer Science 55 No. 3 (2001)
URL: http://www.elsevier.nl/locate/entcs/volume55.html 12 pages

Automated validation of distributed software using the IF environment

Marius Bozga and Susanne Graf and Laurent Mounier [1]

VERIMAG, Centre Equation, 2 avenue de Vignate, F-38610 Gières

Abstract

This paper summarizes our experience with IF, an open validation environment for distributed software systems. Indeed, face to the increasing complexity of such systems, none of the existing tools can cover by itself the whole validation process. The IF environment was built upon an expressive intermediate language and allows to connect several validation tools, providing most of the advanced techniques currently available. The results obtained on several large case-studies, including telecommunication protocols and embedded software systems, confirm the practical interest of this approach.

1 Introduction

Automated validation of distributed software is a desirable objective to improve the industrial production of correct systems like communication protocols or embedded systems. In spite of the numerous ongoing researches and tool developments carried out in this area, this activity remains difficult in practice: on the one hand the initial software description is usually provided in a high-level formalism (either a programming language or a formal design notation like Lotos [23], Sdl [25] or Uml [30]), and, on the other hand, a wide range of tools are necessary to cover the whole development process, operating at different levels of program descriptions. Even if several interesting tools are currently available, either commercial or academic ones, none of them can fulfill in itself all the practical needs.

Commercial tools (such as ObjectGeode [32], Tau [1], StateMate [22], Rational Rose [31], *etc.*) provide several development facilities, like editing, code generation and testing. However, they are usually restricted to basic verification techniques (exhaustive simulation, deadlock detection, *etc*) and are "closed" in the sense that there are only limited possibilities to interface them with others. On the other hand, there exist numerous academic tools

[1] Email: {bozga, graf, mounier}@imag.fr

(like Smv [28], Hytech [19], Kronos [34], Uppaal [27], Spin [20], InVeSt [2], *etc.*) offering a broad spectrum of quite efficient verification facilities (symbolic verification, on-the-fly verification, abstraction techniques, *etc.*), but often supporting only low-level input languages. This may restrict their use at an industrial scale.

This situation motivated the development of IF, an intermediate representation for distributed software together with an open validation environment. This environment fulfills several requirements. First of all, it is able to support *different validation techniques*, from interactive simulation to automatic property checking, together with test case and executable code generation. Indeed, all these functionalities cannot be embodied in a single tool and only tool integration facilities can provide all of them. For a sake of efficiency, this environment supports *several levels of program representations*. For instance it is well-known that model-checking verification of real life case studies usually needs to combine different optimization techniques to overcome the state explosion problem. In particular, some of these techniques rely on a syntactic level representation (like static analysis and computations of abstractions) whereas others techniques operate on the underlying semantic level. Another important feature is to keep this environment *open* and *evolutive*. Therefore, tool connections are performed by sharing either input/output formats, or libraries of components. For this purpose several well-defined application programming interfaces (apis) are provided.

The IF validation environment is quite similar in its philosophy to the one proposed in the Bandera project [12], which also relies on a dedicated intermediate format to translate (abstract) Java source code into the input language of existing model-checkers (like Spin or Smv). However, currently we mainly address with IF distributed software validation from design formalisms (like Sdl or Uml) which are widely used in the application area we consider (communication protocols and embedded systems).

2 Architecture

The IF validation environment relies on three levels of program representation: the *specification level*, the IF *intermediate level*, and the Lts *semantic model level*. Figure 1 describes the overall architecture and the connections between the toolbox components.

The **specification level** is the initial program description, expressed for instance using an existing language. To be processed, this description is (automatically) translated into its IF representation. The main input specification formalism is Sdl, but connections with other languages such as Uml, Lotos and Promela are envisaged.

The **intermediate level** corresponds to the IF representation [8]. In IF, a system is expressed by a set of parallel processes communicating either asynchronously through a set of buffers, or synchronously through a set of

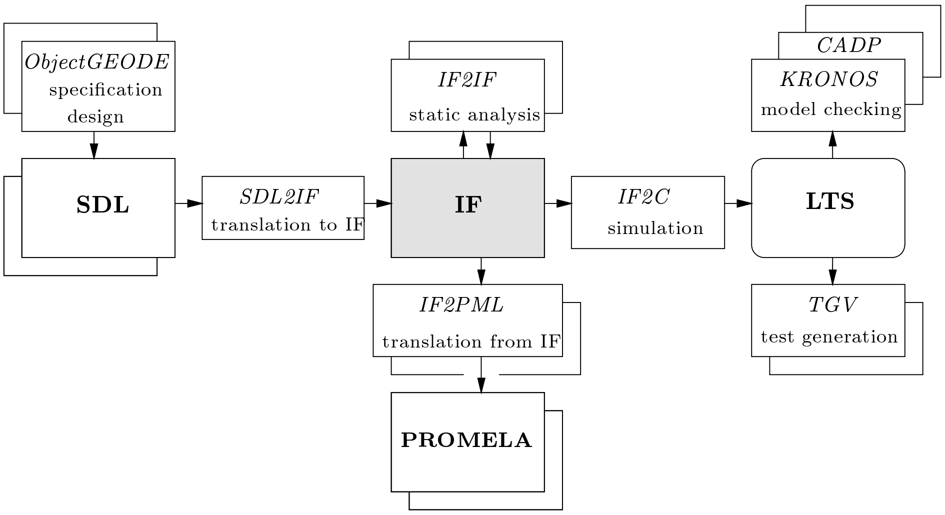

Fig. 1. An open validation environment for IF

gates. Processes are based on timed automata with deadlines [3], extended with discrete variables. Process transitions are guarded commands consisting of synchronous/asynchronous inputs and outputs, variable assignments, and clock settings. Buffers have various queuing policies (fifo, stack, bag, *etc.*), can be bounded or unbounded, and reliable or lossy.

A well-defined API allows to consult and modify the abstract tree of the IF representation. Since all the variables, clocks, buffers and the communication structure are still explicit, high-level transformations based on static analysis (such as *live variables* computation) or program abstraction can be applied. Moreover, this API is also well suited to implement translators from IF to other specification formalisms.

The **semantic model level** gives access to the LTS representing the behaviour of the IF program. Depending on the application considered, three kinds of API are proposed:

- The *implicit enumerative representation* consists in a set of C functions and data structures allowing to compute on demand the successors of a given state (following the OPEN/CÆSAR [16] philosophy). This piece of C code is generated by the IF2C compiler, and it can be linked with a "generic" exploration program performing *on-the-fly* analysis.
- In the *symbolic representation* sets of states and transitions of the LTS are expressed by their characteristic predicates over a set of finite variables. These predicates are implemented using decision diagrams (BDDS). Existing applications based on this API are symbolic model-checking and minimal model generation.
- Finally, the *explicit enumerative representation* simply consists in an LTS file with an associated access library. Although such an explicit representation is not suitable for handling large systems globally, it is very useful in practice to compute abstractions with respect to bisimulation based relations.

3 Components description

We briefly present here the main components of the environment, together with some external tools for which a strong connection exists.

The specification level components. ObjectGeode [32] is a commercial toolset developed by Telelogic supporting Sdl, Msc and Omt. In particular, this toolset provides an api to access the abstract tree generated from an Sdl specification. We have used this api to implement the sdl2if translator, which generates operationally equivalent IF specifications from Sdl ones. Given the static nature of the current version of IF, this translation does not cover yet the dynamical features of Sdl (e.g., process instances creation).

The intermediate level components. if2if [6] implements several algorithms based on static analysis to transform an IF specification. A first transformation concerns *dead variable resetting* (a variable is dead at some control point if its value is not used before being redefined). This optimisation can be also applied to buffer contents (a message parameter is dead if its value is not used when the message is consumed). Although very simple, such optimisation is particularly efficient for state space generation (reductions up to a factor 100 were frequently observed), while preserving the exact behaviour of the original specification. A second transformation is based on the *slicing* technique [33]. It allows to automatically abstract a given specification by eliminating some irrelevant parts w.r.t. a given property or test purpose [7].

if2pml [4] is a tool developed at Eindhoven TU to translate IF specifications into Promela.

The semantic model level components. Cadp [14] is a toolset for the verification of Lotos specifications. It is developed by the Vasy team of Inria Rhône-Alpes and Verimag. Two of its model-checkers are connected to the IF environment: Aldebaran (bisimulation based), and Evaluator (alternating-free μ-calculus). For both tools, diagnostic sequences are computed on the Lts level and they can be translated back into Msc to be observed at the specification level.

Kronos [34] is a model-checker for symbolic verification of Tctl formulae on communicating timed automata. The current connection with the IF environment is as follows: control states and discrete variables are expressed using the implicit enumerative representation, whereas clocks are expressed using a symbolic representation (particular polyhedra).

Tgv [15] is a test sequence generator for conformance testing of distributed systems (joint work between Verimag and the Pampa project of Irisa). Test cases are computed during the exploration of the model and they are selected by means of *test purposes*.

4 Case studies

The IF environment was used in several case studies, including as well telecommunication protocols and embedded software. The most relevant ones, from the complexity point of view, and the results obtained are summarized below.

4.1 SSCOP Protocol

The Sscop (*Service Specific Connection Oriented*) protocol is standardized under reference ITU-T Q2110 [24]. Originally, it was conceived to reliably transfer data between two high bandwidth network entities. Although its design makes it ready to treat significant volumes of data, currently its use is confined in ones of the underlayers of the AAL layer (Atm Adaptation Layer). The services it provides are connection control (establishment, flow-control, release), data transfer, and error detection.

The Sscop standardization document contains an Sdl description of the protocol. This description has been coded by France Telecom R&D using ObjectGeode. It consists in approximately 2000 lines of Sdl textual code which describes the protocol as one single process with 10 control states, 134 variables, and 4 timers. The description was centered on signaling and some simplifications have been made according to Sscop implementations available in France Telecom R&D. Our main goals were the formal validation of the specification and, in addition, automatic test-case generation starting from it.

Clearly, the size and complexity of this specification made any brute force validation approach not applicable. In particular the data part was very large, and each state of the underlying model could not be stored in less than 2kB. Therefore only a small part of the state space could be explored from this initial specification, not sufficient to verify interesting properties.

Consequently, we adopted a more incremental verification strategy. A first step was to apply a very rough abstraction by (automatically) eliminating all the variables in the specification. Thus, it was possible to compare this very abstract specification with the one supplied by the standard to model the interactions between adjacent layers of Sscop, to check if the abstract specification provides *at least* the expected behaviour. This comparison was performed using Aldebaran, with respect to the so-called *safety preorder* [5]. Some subtle errors, such as omission of timers setting, were found using this method.

After this debugging phase, the second step was to "prepare" the initial Sdl specification for a more accurate state space analysis. It consisted in basic static analysis techniques like dead code elimination and live variable detection using IF2IF. The benefits were really spectacular on this example, and, in particular, the amount of memory required to store a model state fell to 0.2 kB.

Finally, these optimisations made possible the use of exhaustive simulation techniques. More precisely we considered a system consisting in a pair of entities, communicating through a bounded fifo channel, and we concentrated our validation effort to a set of representative distinct scenarios (connection establishment, disconnection, data transfer, ...). Using specific slicing criteria, it was therefore possible to (automatically) simplify even more the specification, depending on the property under verification or the test purpose. The underlying models obtained were about 20 000 states large, and errors were found in the data transfer phase of the specification. The complete experiment is reported in [9].

4.2 Mascara Protocol

The Mascara(*Mobile Access Scheme based on Contention And Reservation for* Atm) protocol is a special medium access control protocol designed for wireless Atm communication and developed by the wand(Wireless Atm Network Demonstrator) consortium [13]. A wireless Atm network extends transparently services to *mobile terminals* (mt) via a number of geographically distributed *access points* (ap). The task of the Mascara protocol is to mediate between aps and mts via a wireless link. The protocol has a layered structure, where we consider only the highest layer, the Mascara control layer.

The overall description of the Mascara protocol which we got is *300* pages of Sdl textual code. We concentrate on the verification of the Mascara control layer, for which the Sdl description could be made reasonably complete. Here we briefly present the verification of the dynamic control. For complete information, we refer the reader to [17] which reports the complete experiment on the dynamic part. In addition, another verification experiment has been carried out on static control[4].

Verification should be carried out under a general environment with realistic restrictions. As we have not obtained information on the Mascara upper layer, we considered initially an open system with an unconstrained upper layer, which would allow us to obtain the *most general* verification results. But communication via unbounded channels, leads to infinitely growing channel contents and thus an infinite state model in case that the environment sends requests too often. This is typically the case in *reactive* systems always ready to treat requests from the environment. The approach we have chosen to provide a more restricted, but still realistic environment consists in limiting the number of requests it can make per time unit. We assume that within one time unit, no more than N requests can be sent by the environment. The system has never to deal with more than N requests simultaneously which leads, in the Mascara protocol, to bounded channel contents. The success of the method depends on the use of a realistic bound. We use $N{=}4$.

Unfortunately, even with such a restricted environment, it was impossible to generate the state graph of the global system as a whole. However, we have applied two different types of *compositional verification*: the first one is based on property decomposition [26], and the second one is based on compositional generation of a state graph minimized with respect to a behavioral equivalence [18]. In particular, using in addition both *live analysis* and *partial order reduction* for the generation of the subsystems, we were able to compositionally generate a reduced model of the global system using compositional generation.

generation strategy	AP model size	AP time	MT model size	MT time	AP + MT model size
- live reduction - partial order	7308400 st. 30689244 tr.	207′35″	4388765 st. 12811961 tr.	171′58″	
+ live reduction - partial order	351202 st. 1536699 tr.	12′22″	63628 st. 325312 tr.	1′03″	
+ live reduction + partial order	28069 st. 52983 tr.	1′53″	6580 st. 20913 tr.	7″	218130 st. 1142215 tr.
+ live reduction + partial order + slicing	1630 st. 2885 tr.	9″	977 st. 2845 tr.	3″	

Table 1
Mascara verification results

Table 1 gives an overview of a subset of the models we have generated using different reduction techniques and allows to compare their sizes and generation times. Finally, several properties ranging for generic ones such as deadlocks and livelocks to more specific such as association establishment, connection, disconnection, were verified on the generated models.

4.3 Ariane-5 Flight Program

The work on this experiment was initiated by Eads Launch Vehicles to better evaluate the applicability of formal validation techniques on an existing software, the Ariane-5 [2] Flight Program. This is the embedded software which solely controls the Ariane-5 launcher during its flight, from the ground, through the atmosphere and up to the final orbit.

The verification experiment is reported in [11]. First, this software has been formally specified in Sdl by reverse engineering from the existing code. Then, following a set of general methodological guidelines, the specification

[2] Ariane-5 is an European Space Agency Project delegated to CNES France.

has been continuously improved and all the initial requirements were verified on the final version. In particular, the combination of different optimisation techniques, operating either at the *source level* (like static analysis or slicing) or at the *semantic level* (like partial-order reductions) happened to be particularly useful in order to deal with large size state spaces. For example, the initial Sdl version of the flight program used no less than 130 timers. Using our static analysis tool we were able to reduce them to only 55 timers, functionally independent ones. Afterward, the whole specification was rewritten taking into account the redundancies discovered by the analyzer.

The main difficulty of this case-study comes from the combination of various kind of time constraints. On one hand, the functionality of the flight program strongly depends on an absolute time: coordination dates are frequently exchanged between components in order to synchronise their behaviour during the whole flight. On the other hand, this system has to be verified within a partially constrained environment, reacting with some degree of temporal uncertainty. In this experiment, this expressivity problem was solved at the IF level thanks to explicit urgency attributes. Clearly, such features should be made available at specification level. In particular, ongoing work address the introduction of high-level time and performance annotations in Sdl [10].

		time deterministic	time non-deterministic
model generation	- live reduction - partial order	na	na
	+ live reduction - partial order	2201760 st. 18706871 tr.	na
	+ live reduction + partial order	1604 st. 1642 tr.	195718 st. 278263 tr.
model verification	model minimisation	$\sim$ 1″	$\sim$ 20″
	model checking	$\sim$ 15″	$\sim$ 2′00″

Table 2
Ariane-5 Flight Program verification results.

In practice, we have considered two different situations regarding the environment. The first one is *time-deterministic*, which means that all environment actions (in particular the control part) take place at precise moments in time. The second one is *time-nondeterministic* which means that environment actions take place with some degree of time uncertainty (within a predefined time interval). From the environment point of view, the later situation corresponds to a whole set of scenarios, whereas the former situation focus only on a single one. Table 2 presents the sizes of both models generated according to different generation strategies. It gives also the average time required for verifying each kind of property (by temporal logic model checking and model minimisation respectively).

5 Conclusion and Perspectives

The IF environment has already been used to analyze some representative Sdl specifications such as Sscop, an Atm signalisation protocol, Mascara, an Atm wireless transport protocol and Ariane-5 flight program, a part of the embedded software of Ariane-5 launchers. It is currently used in several on going industrial case-studies, including respectively real-time multicast protocols Pgm and Rmtp-ii, and the session initiation protocol Sip. The benefits of combining several techniques, working at different program level, were clearly demonstrated. In particular, traditional model-checking techniques were not sufficient to complete on these large size examples.

Several directions can be investigated to improve this environment. The first direction of improvement concerns the IF language. As currently defined, it allows only the description of static systems, were the number of components (processes and buffers) as well as their interactions are fixed throughout the execution. This strongly limits our ability to handle complex dynamic specifications. We work on a less restrictive definition, where both parameterized descriptions (containing some fixed number of replicated components) as well as general dynamic creation and destruction of components are allowed. Furthermore, some improvements will be made regarding the description of components itself, such as the possibility to express structured control using composed states (like in statecharts).

A second direction of improvement concerns the IF simulator, the core component allowing to construct and to explore the underlying semantic model of IF specifications. Currently, this model is labeled transition systems, and its construction and exploration are quite restricted: first, only pure asynchronous execution (by interleaving) is possible and second, no access is provided to the state of the system (e.g, current values of variables, current states of processes). We envisage to improve these points, by implementing a *flexible* simulator able, for instance, to deal with both synchronous and asynchronous components, or more generally, to take into account some scheduling policy over components during the simulation. In addition, this simulator will interact with running components through a well-defined abstract interface, thus allowing to integrate also external components (for example, directly expressed as executable code).

A third direction of improvement concerns the validation methods. Clearly, we will continue to adapt and to improve our static analysers as well as our model checkers to handle the extended IF descriptions. Also, some work must be done to reduce the manual overhead, yet important, needed by sophisticated techniques such as compositional verification. Finally, another important issue concerns the validation of non-functional requirements. In particular, performance evaluation becomes crucial for an important part of internet protocols (such as Pgm or Rmtp-ii) which are not necessarily designed to achieve full reliability, but only an *average* correct behaviour with respect to probabilistic assumptions on their execution environment (e.g, propagation delays, message loss, network elements speed, etc). At middle term, we plan to connect the IF environment to simulation environments like Opnet [29] and Ses/Workbench [21].

Acknowledgements We gratefully thank Guoping Jia and Lucian Ghirvu for their help on tools development and experimentations. The IF package can be downloaded at `http://www-verimag.imag.fr/DIST_SYS/IF`.

References

[1] Telelogic AB. *SDT Reference Manual.* `http://www.telelogic.se/`.

[2] S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In A. Hu and M. Vardi, editors, *Proceedings of CAV'98 (Vancouver, Canada)*, volume 1427 of *LNCS*, pages 319–331. Springer, June 1998.

[3] S. Bornot, J. Sifakis, and S. Tripakis. Modeling Urgency in Timed Systems. In *International Symposium: Compositionality - The Significant Difference (Holstein, Germany)*, volume 1536 of *LNCS*. Springer, September 1997.

[4] D. Bošnački, D. Dams, L. Holenderski, and N. Sidorova. Model Checking SDL with Spin. In S. Graf and M. Schwartzbach, editors, *Proceedings of TACAS'2000 (Berlin, Germany)*, volume 1785 of *LNCS*, pages 363–377. Springer, March 2000.

[5] A. Bouajjani, J.Cl. Fernandez, S. Graf, C. Rodriguez, and J. Sifakis. Safety for Branching Time Semantics. In *Proceedings of ICALP'91*, volume 510 of *LNCS*. Springer, July 1991.

[6] M. Bozga, J.Cl. Fernandez, and L. Ghirvu. State Space Reduction based on Live Variables Analysis. In A. Cortesi and G. Filé, editors, *Proceedings of SAS'99 (Venice, Italy)*, volume 1694 of *LNCS*, pages 164–178. Springer, September 1999.

[7] M. Bozga, J.Cl. Fernandez, and L. Ghirvu. Using Static Analysis to Improve Automatic Test Generation. In S. Graf and M. Schwartzbach, editors, *Proceedings of TACAS'00 (Berlin, Germany)*, LNCS, pages 235–250. Springer, March 2000.

[8] M. Bozga, J.Cl. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, and L. Mounier. IF: An Intermediate Representation and Validation Environment for Timed Asynchronous Systems. In J.M. Wing, J. Woodcock, and J. Davies, editors, *Proceedings of FM'99 (Toulouse, France)*, volume 1708 of *LNCS*, pages 307–327. Springer, September 1999.

[9] M. Bozga, J.Cl. Fernandez, L. Ghirvu, C. Jard, T. Jéron, A. Kerbrat, P. Morel, and L. Mounier. Verification and Test Generation for the SSCOP Protocol. *Journal of Science of Computer Programming, Special Isssue on Formal Methods in Industry*, 36(1):27–52, January 2000.

[10] M. Bozga, S. Graf, L. Mounier, I. Ober, J.L. Roux, and D. Vincent. Timed Extensions for SDL. In *Proceedings of SDL FORUM'01*, LNCS, 2001. to appear.

[11] M. Bozga, D. Lesens, and L. Mounier. Model-Checking Ariane-5 Flight Program. In *Proceedings of FMICS'01*, 2001. to appear.

[12] J. Corbett, M. Dwyer, J. Hatcliff, C. Pasareanu, Robby, S. Laubach, and H. Zheng. Bandera : Extracting Finite-state Models from Java Source Code. In *Proceedings of the 22nd International Conference on Software Engineering*, June 2000.

[13] I. Dravapoulos, N. Pronios, and S. Denazis et al. *The Magic WAND, Deliverable 3D5, Wireless ATM MAC, Final Report*, August 1998.

[14] J.Cl. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In R. Alur and T.A. Henzinger, editors, *Proceedings of CAV'96 (New Brunswick, USA)*, volume 1102 of *LNCS*, pages 437–440. Springer, August 1996.

[15] J.Cl. Fernandez, C. Jard, T. Jéron, and C. Viho. An Experiment in Automatic Generation of Test Suites for Protocols with Verification Technology. *Science of Computer Programming*, 29, 1997.

[16] H. Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In B. Steffen, editor, *Proceedings of TACAS'98 (Lisbon, Portugal)*, volume 1384 of *LNCS*, pages 68–84. Springer, March 1998.

[17] S. Graf and G. Jia. Verification Experiments on the Mascara Protocol. In *Proceedings of the SPIN'01 Workshop*, 2001. to appear.

[18] S. Graf and B. Steffen. Compositional Minimisation of Finite State Processes. In E. Clarke and R. Kurshan, editors, *Proceedings of CAV'90 (Rutgers, USA)*, volume 3 of *DIMACS*, pages 57–74. AMS/ACM, 1990.

[19] T.H. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech : A Model Checker for Hybrid Systems. In O. Grumberg, editor, *Proceedings of CAV'97 (Haifa, Israel)*, volume 1254 of *LNCS*, pages 460–463. Springer, June 1997.

[20] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall Software Series, `http://cm.bell-labs.com/cm/cs/what/spin`, 1991.

[21] HyPerformix. *Ses/Workbench*. `http://www.hyperformix.com/`.

[22] I-Logix. *StateMate*. `http://www.ilogix.com/`.

[23] ISO/IEC. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Technical Report 8807, International Organization for Standardization — Information Processing Systems — Open Systems Interconnection, Genève, 1988.

[24] ITU-T. Recommendation Q.2110. ATM Adaptation Layer - Service Specific Connection Oriented Protocol (SSCOP). Technical Report Q-2110, International Telecommunication Union – Standardization Sector, Genève, 1994.

[25] ITU-T. Recommendation Z.100. Specification and Description Language (SDL). Technical Report Z-100, International Telecommunication Union – Standardization Sector, Genève, November 1999.

[26] R.P. Kurshan. *Computer-Aided Verification of Coordinating Processes: The Automata-Theoretic Approach*. Princeton University Press, Princeton, New Jersey, 1994.

[27] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL: Status & Developments. In O. Grumberg, editor, *Proceedings of CAV'97 (Haifa, Israel)*, volume 1254 of *LNCS*, pages 456–459. Springer, June 1997.

[28] K.L. McMillan. *Symbolic Model Checking: an Approach to the State Explosion Problem*. Kluwer Academic Publisher, `http://www.cs.cmu.edu/ modelcheck/smv.html`, 1993.

[29] Inc MIL3. *Optimized Network Engineering Tool OPNET*. `http://www.opnet.com/`.

[30] OMG. Unified Modeling Language Specification. Technical Report OMG UML v1.3 – ad/99-06-09, Object Management Group, June 1999.

[31] Rational. *Rational Rose*. `http://www.rational.com/`.

[32] Verilog. *ObjectGEODE Reference Manual*. `http://www.verilogusa.com/`.

[33] M. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, SE-10(4), July 1984.

[34] S. Yovine. KRONOS: A Verification Tool for Real-Time Systems. *Software Tools for Technology Transfer*, 1(1+2):123–133, December 1997.