IF: A Validation Environment for Timed Asynchronous Systems*

Marius Bozga[1], Jean-Claude Fernandez[2], Lucian Ghirvu[1]**, Susanne Graf[1], Jean-Pierre Krimm[1], and Laurent Mounier[1]

[1] VERIMAG, Centre Equation, 2 avenue de Vignate, F-38610 Gières
[2] LSR/IMAG, BP 82, F-38402 Saint Martin d'Hères Cedex

1 Introduction

Formal validation of distributed systems relies on several specification formalisms (such as the international standards LOTOS [15] or SDL [16]), and it requires different kinds of tools to cover the whole development process. Presently, a wide range of tools are available, either commercial or academic ones, but none of them fulfills in itself all the practical needs.

Commercial tools (like *Object*GEODE [20], SDT [1], STATEMATE [14],*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 many academic tools (like SMV [19], HYTECH [12], KRONOS [22], UPPAAL [18], SPIN [13], 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 timed asynchronous systems 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

* partially supported by the PROUST RNRT project
** Work partially supported by Région Rhône-Alpes, France

formats, or libraries of components. For this purpose several well-defined application programming interfaces (APIs) are provided.

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.

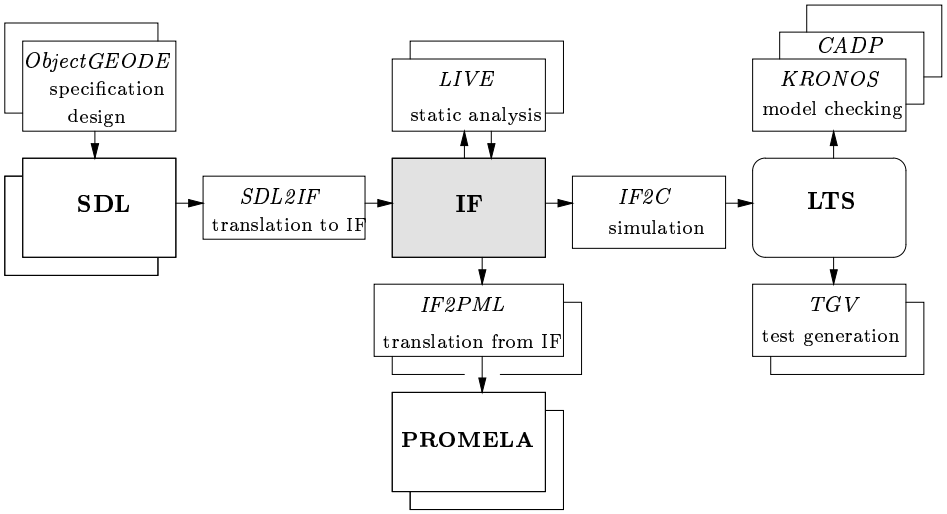

Fig. 1. An open validation environment for IF

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. Currently the main input specification formalism we consider is SDL, but connections with other languages such as LOTOS or PROMELA could also be possible.

The **intermediate level** corresponds to the IF representation [7]. 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 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-CAESAR [11] 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 still useful in practice to minimize some of its 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. *Object*GEODE [20] is a commercial toolset developed by TTT 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 IF, this translation does not cover the dynamical features of SDL (e.g., process instances creation).

The intermediate level components. LIVE [5] 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 [21]. It allows to automatically abstract a given specification by eliminating some irrelevant parts w.r.t. a given property or test purpose [6].

IF2PML [4] is a tool developed at Eindhoven TU to translate IF specifications into PROMELA.

The semantic model level components. CADP [9] 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 [22] 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 [10] 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 Results and perspectives

The IF environment has already been used to analyze some representative SDL specifications, like SSCOP, an ATM signalisation layer protocol [8], and MASCARA, an ATM wireless transport protocol. It is currently used in several on going industrial case-studies, including the real-time multicast protocol PGM, and the control part of the ARIANE 5 launcher flight sequencer. The benefits of combining several techniques, working at different program level, were clearly demonstrated. In particular, traditional model-checking techniques (as provided by *Object*GEODE) were not sufficient to complete on these large size examples.

Several directions can be investigated to improve this environment.

First of all, other formalisms than SDL could be connected to IF. In particular, the translation from a subset of UML is envisaged. To this purpose new features will be added to handle *dynamic process creation* and *parametrized network specifications*.

From the verification point of view, the results obtained using the currently implemented static analysis techniques are very encouraging. We now plan to experiment some more sophisticated algorithms implemented in the INVEST tool [2], such as *structural invariant generation* and general *abstraction computation techniques* for infinite space systems.

Another promising approach to verify large systems consists in generating their underlying model in a *compositional* way: each sub-system is generated in isolation, and the resulting LTSs are minimized before being composed with each other. The IF environment offers all the required components to experiment it in an asynchronous framework [17].

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. 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.
6. 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.
7. 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.
8. 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.
9. 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.
10. 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.
11. 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.
12. 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.
13. Gerard J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall Software Series, 1991.
14. I-Logix. *StateMate.* `http://www.ilogix.com/`.
15. 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.
16. ITU-T. Recommendation Z.100. Specification and Description Language (SDL). Technical Report Z-100, International Telecommunication Union – Standardization Sector, Genève, 1994.

17. J.P. Krimm and L. Mounier. Compositional State Space Generation with Partial Order Reductions for Asynchronous Communicating Systems. In S. Graf and M. Schwartzbach, editors, *Proceedings of TACAS'2000 (Berlin, Germany)*, volume 1785 of *LNCS*, pages 266–282. Springer, March 2000.
18. 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.
19. K.L. McMillan. *Symbolic Model Checking: an Approach to the State Explosion Problem.* Kluwer Academic Publisher, 1993.
20. Verilog. *ObjectGEODE Reference Manual.* `http://www.verilogusa.com/`.
21. M. Weiser. Program Slicing. *IEEE Transactions on Software Engineering*, SE-10(4), July 1984.
22. S. Yovine. KRONOS: A Verification Tool for Real-Time Systems. *Software Tools for Technology Transfer*, 1(1+2):123–133, December 1997.