IF: An Intermediate Representation and 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
Marius.Bozga@imag.fr
[2] LSR/IMAG, BP 82, F-38402 Saint Martin d'Hères Cedex
Jean-Claude.Fernandez@imag.fr

Abstract. Formal Description Techniques (FDT), such as LOTOS or SDL are at the base of a technology for the specification and the validation of telecommunication systems. Due to the availability of commercial tools, these formalisms are now being widely used in the industrial community. Alternatively, a number of quite efficient verification tools have been developed by the research community. But, most of these tools are based on simple adhoc formalisms and the gap between them and real FDT restricts their use at industrial scale.
This context motivated the development of an intermediate representation called IF which is presented in the paper. IF has a simple syntactic structure, but allows to express in a convenient way most useful concepts needed for the specification of timed asynchronous systems. The benefits of using IF are multiples. First, it is general enough to handle significant subsets of most FDT, and in particular a translation from SDL to IF is already implemented. Being built upon a mathematically sound model (extended timed automata) it allows to properly evaluate different semantics for FDT, in particular with respect to time considerations. Finally, IF can serve as a basis for interconnecting various tools into an unified validation framework. Several levels of IF program representation are already available via well defined API and allow to connect tools ranging from static analyzers to model-checkers.

keywords: asynchrony, timed systems, model-checking, static analysis, test generation

1 Introduction

Formal Description Techniques, such as LOTOS [ISO88] or SDL [IT94c] and related formalisms such as MSC and TTCN are at the base of a technology for the specification and the validation of telecommunication systems. Due to the availability of commercial tools, mainly for editing, code generation and testing, and the fact that these formalisms are promoted by ITU and other international standardization bodies, these formalisms are now being widely used in the community of telecommunication systems.

* Work partially supported by Région Rhône-Alpes, France
*** Verimag is Research Laboratory of CNRS, Université Joseph Fourier and Institut National Polytechnique of Grenoble

There are also increasing needs for description and validation tools covering as many aspects of system development as possible. This is the reason why the commercial editing tools contain also some verification facilities. Unfortunately, these verification facilities are often quite restricted and the tools are "closed" in the sense that there are only limited possibilities to interface them with others. On the other hand, a number of quite efficient verification tools have been developed by the research community, but they are in general based on adhoc input formalisms and the gap between them and real FDT restricts their use at an industrial scale. Even if these tools are in general less closed than commercial ones, they have rarely well-defined interfaces. For example, a lot of developments were made around the Spin verification tool [Hol91], but they are based on the availability of the source code and not on *a priori* defined interfaces.

A different approach was followed within CADP [FGK+96], a toolbox for the verification of LOTOS specifications. It was conceived right from the beginning as an open platform for interfacing different algorithms and provides several well-defined and documented interfaces. The initial motivation for the work presented here was the fact that SDL becomes a more and more popular formalism in the telecommunication community, and that we wanted to adapt CADP to deal also with SDL specifications. Since the intermediate program level formalisms used within CADP are not appropriated for SDL specifications, we had to investigate alternative representations. For example CADP is based on a synchronous communication model (rendez-vous), whereas SDL communications are fully asynchronous (via queues).

Another motivation concerns time modeling. Finding a "reasonable" notion of time is a central problem which admits many possible solutions depending on choices of semantic models. This is certainly a non trivial question and this is reflected by the variety of the existing proposals for existing FDT. For instance, SDL syntax defines a timer concept, but there is no consensus on its semantics for the moment and different SDL tools have adopted different choices. Similarly, in the original LOTOS definition there was no particular notion of time, whereas different timed extensions are currently being proposed [LL97,Que98]. Choosing an appropriate timed extension for an FDT should take into account not only technical considerations about the semantics of timed systems but also more pragmatic ones related to the appropriateness for use in a system engineering context. We believe that the different ideas about extensions of the language must be validated experimentally before being adopted to avoid phenomena of rejection by the users.

These problems motivated the development of IF, a new intermediate representation for timed asynchronous systems. The requirements on this formalism were the following:

- it must be sufficiently expressive to be used as an intermediate representation for the above mentioned specification formalisms, or at least for reasonably large subsets of them.
- it must have a formally defined operational semantics, but flexible enough to experiment different choices and extensions.
- it must be supported by a set of well defined API, at different levels of program representation, allowing either to interface existing validation tools or to experiment new ones.

The paper is organized as follows. First, we define the IF formalism, its main concepts and its operational semantics. We also discuss its expressiveness with respect to other models and specification formalisms, in particular regarding the timing aspects. Then, we present a set of tools interconnected within an open validation environment for IF specifications. We further detail two specific components, based on *static analysis* and *compositional generation*, working at the program level and experimented within this environment. Finally, we illustrate the use of IF on a small example, a distributed leader election algorithm on which different kinds of validation are performed.

2 Presentation of IF

In the following sections, we give a brief overview of the main features of IF, its operational semantics in terms of labeled transition systems. A more complete description of IF and of its semantics can be found in [BFG+98].

2.1 Syntax

An IF system is a set of processes communicating either asynchronously through a set of *buffers* or synchronously through a set of *gates*. The timed behavior of a system can be controlled through *clocks* (like in timed automata [ACD93,HNSY94]).

IF system definition: A system is a tuple $Sys = (glob\text{-}def, \text{PROCS}, \mathtt{S})$ where

- *glob-def* = (*type-def*, *sig-def*, *gate-def*, *var-def*, *buf-def*) is a list of global definitions, where *type-def* is a list of type definitions (enumerated types, arrays, records and also abstract data types[1]) *sig-def* defines a list of parameterized *signals* (as in SDL), *gate-def* defines a list of parameterized *gates* (as in LOTOS), *var-def* is a list of global variables, and finally, *buf-def* is a list of *buffers* through which the processes communicate by asynchronous signal exchange (as in Promela[Hol91]). Notice that we allow various types of buffers: FIFO queues, stacks or bags, which can chosen to be unbounded or bounded and reliable or lossy.
- PROCS defines a set of processes described in section 2.1.
- S is a synchronization expression, as in LOTOS or CSP, telling how the processes defined in PROCS synchronize. Such a synchronization expression is given by the following grammar where C is a (possible empty) set of gates:

  $\mathtt{S} ::= \mathtt{P}{\in}\text{PROCS} \quad | \; \mathtt{S}\ [\![\mathtt{C}]\!]\ \mathtt{S}$

  Thus, a system S is either a process P or a parallel composition of two subsystems $\mathtt{S}_1$ and $\mathtt{S}_2$ with rendez-vous synchronization on the set of gates C. In a system of the form $\mathtt{S}_1\ [\![\mathtt{C}]\!]\ \mathtt{S}_2$ transitions concerning a gate in C are executed synchronously in the two subsystems whereas all other transitions are interleaved.

[1] where we suppose that the user provides also implementations of the introduced functions, otherwise expressions containing them are handled syntactically

IF process definition: Processes are defined by a set of local variables, a set of control states and a set of control transitions. A process P∈PROCS is a tuple P= (*var-def*, `Q`, cTRANS), where:

- *var-def* is a set of local variable definitions including also clocks[2] (as in timed automata)
- `Q` is a set of **control states** on which the following attributes are defined:
  - *stable*(q) and *init*(q) are boolean attributes, where the attribute *stable* can be used to control the level of atomicity: only stable states are visible on the semantic level.
  - the *tpc*(q) *time progress condition* attribute is a predicate depending on visible variable of the process (global variables and local ones) which defines when time can progress. Notice that, anyway, time cannot progress in non stable states.
  - the attributes *save*(`q`), *discard*(`q`) are sets of `filter`s of the form

    `signal-list` **in** `buf` **if** `cond`.

    which filter the buffers contents in this state. For example, *discard*(`q`) is used to eliminate silently unexpected signals: when consuming the next signal in the FIFO queue `buf`, all signals of `signal-list` preceding it are discarded in the same atomic step, if the boolean expression `cond` evaluates to `true`. These primitives are useful in practice and taken from SDL.
- cTRANS is a set of **control transitions**, between control states `q,q'`∈`Q`, which may be of the following types:
  - input transitions which are triggered by some signal read from one of the communication buffers (as in SDL):

$$\mathtt{q} \xrightarrow[(\mathtt{urg})]{\mathtt{g}\ \mapsto\ \mathtt{input}\ ;\ \mathtt{body}} \mathtt{q}'$$

  - synchronization transitions which are executed simultaneously with compatible ones in other processes of the system (as in LOTOS):

$$\mathtt{q} \xrightarrow[(\mathtt{urg})]{\mathtt{g}\ \mapsto\ \mathtt{sync}} \mathtt{q}'$$

  - internal transitions depending not on communications:

$$\mathtt{q} \xrightarrow[(\mathtt{urg})]{\mathtt{g}\ \mapsto\ \mathtt{body}} \mathtt{q}'$$

Where in all three cases:

- `g` is a boolean *guard* of the transition which may depend on visible variables in the process (including clocks) and predefined tests on buffers content (e.g., emptiness).

---
[2] one can also define timers (as in SDL) which can be set to any positive value, which decrease with progress of time and expire if they reach the value *zero*; however to simplify the description we don't mention them in this document

- urg$\in${**eager, delayable, lazy**} defines the *urgency type* of the transition. **eager** transitions have absolute priority over progress of time, **delayable** transitions may let time progress, but only as long as they remain enabled, whereas **lazy** transitions cannot prevent progress of time. These urgency types have been introduced in [BST98], which shows that the use of urgency predicates on transitions (instead of time progress conditions) facilitates the compositional specification of timed systems.
- `input` is an input of the form "**input** `sig`(*reference_list*) **from** `buf` **if** `cond`" where
  - `sig` is a signal,
  - *reference_list* the list of variables[3] (excluding clocks) in which the received parameters are stored,
  - `buf` is the name of the buffer from which the signal should be read
  - `cond` is a "post guard" defining the condition under which the received signal is accepted and it usually depends on received parameters.

  Intuitively, an input transition is enabled if its guard is true, the first signal to be consumed (according to the attributes *save*(`q`) and *discard*(`q`)) is of the form `sig`($v_1, ...v_k$) and the post guard holds (after assigning the values $v_1, ...v_k$ to the variables of the *reference_list*)
- `sync` is a synchronization of the form "**sync** `gate` *comm_list* **if** `cond`" where
  - `gate` is a synchronization gate defined at system level,
  - *comm_list* is a list of *communications offers*:
    - either an output communication offer of the form !`exp`, where the expression `exp` represents the sent value
    - or a input communication offer of the form ?`ref`, where `ref` is a local variable[3] in which the received value is stored.
  - `cond` is again a post guard usually depending on received values and which restricts the values that the process is willing to accept.

  The concept of synchronization is taken from LOTOS: the simultaneous execution of synchronization transitions concerning the same gate allows a instantaneous exchange of values between several processes. Notice that clock expressions cannot appear as communication offers.
- `body` is a sequence of atomic actions of the following types:
  - asynchronous *outputs* of the form "**output** `sig`(*par_list*) **to** `buf`" append a *signal* of the form "`sig`(*par_list*)" to the buffer `buf`.
  - usual *assignments* between discrete variables.
  - *resettings* of clocks, which have the effect to assign the value *zero* to the corresponding clock.

2.2 Semantics

The semantics of IF is based on concepts taken respectively from LOTOS, SDL and timed automata. We define it by translating IF sytems into Timed Automata with Urgency [BST98]. First, we show how to associate a timed automaton with a process,

[3] or "assignable" expressions such as elements of records or arrays

and then, how these timed automata can be composed into a single one[4]. The timed automata can then be interpreted either using discrete or dense time depending on the verification tools and properties considered. Notice that the discrete/dense interpretation of time does not influence the translation into a timed automaton.

Association of a Timed Automaton with a process: Let P= (*var-def*, Q, cTRANS) be a process definition in the system `Sys` and furthermore:

- Let BUF be a set of buffer environments $\mathcal{B}$, representing possible contents of the buffers of the system, on which — depending on the declared buffer type — all necessary primitives are defined: e.g. "get the first signal of a given buffer, taking into account the *save* and the *discard* attributes of the control state", "append a signal at the end of a buffer",...
- Let ENV be a set of environments $\mathcal{E}$ defining the set of valuations of all discrete variables defined in the system `Sys` (the local and the global ones)

The semantics of the process P is the timed automaton [P] = (Q×ENV×BUF,TRANS) where

- Q×ENV×BUF is the set of states, for which we extend the attributes of control states in a natural manner, e.g. *tpc*((q,($\mathcal{E}$,$\mathcal{B}$))) is the partial evaluation of *tpc*(q) in (($\mathcal{E}$,$\mathcal{B}$)). Notice that the set of data environments ENV can be split into $\text{ENV}_{loc} \times \text{ENV}_{glob}$ where ENV_{loc} concerns only local variables of the process and ENV_{glob} concerns the global variables of the system.
- TRANS is the set of transitions of the timed automaton obtained from control transitions by the following two rules:
  1. For any input transition (and for internal ones)
     $$\mathtt{q} \xrightarrow[\text{(urg)}]{\mathtt{g}\ \mapsto\ (\mathtt{sig}(x_1...x_n), \mathtt{buf}, \mathtt{cond})\ ;\ \mathtt{body}} \mathtt{q}' \in \text{cTRANS}$$
     and for any ($\mathcal{E}$,$\mathcal{B}$),($\mathcal{E}$',$\mathcal{B}$')∈VAL, there exists a transition
     $$(\mathtt{q}, (\mathcal{E}, \mathcal{B})) \xrightarrow[\text{(urg)}]{\ell\ :\ \mathtt{g}'\ \mapsto\ \mathtt{body}'} (\mathtt{q}', (\mathcal{E}', \mathcal{B}')) \in \text{TRANS} \qquad \text{if}$$
     - g' is the the partial evaluation of g in ($\mathcal{E}$,$\mathcal{B}$), which is an expression depending only on clocks.
     - let $\mathcal{B}''$ be the buffer environment obtained after consuming $\mathtt{sig}(v_1...v_n)$ in buffer `buf` (and after elimination of appropriate signals of the *discard*(q) attribute and saving of the signals of the *save*(q) attribute)
     - let $\mathcal{E}''=\mathcal{E}[v_1...v_n/x_1...x_n]$ is obtained by assigning v_i to x_i,
     - the post guard `cond` evaluates to `true` in the environment $(\mathcal{E}'', \mathcal{B}'')$
     - $(\mathcal{E}', \mathcal{B}')$ is obtained from $(\mathcal{E}'', \mathcal{B}'')$ by executing all the assignments of the `body`, and by appending all signals required by outputs in the `body`.
     - `body`' is the sequence of resettings of clocks which remain as such in the timed automaton,

[4] Notice that the semantics is compositional in the sense that, in order to associate a timed automaton with a system one can also first compose the system into a unique process and then associate a timed automaton with this process

– ℓ is an appropriate label used for tracing.

2. For each synchronization transition of the form

$$\mathtt{q} \xrightarrow[\mathtt{(urg)}]{\mathtt{g}\ \mapsto\ \mathtt{c}\ :\ !\mathtt{exp}_1\ ?\mathtt{y}_2\ \ldots\ :\ \mathtt{cond}} \mathtt{q}' \in \text{cTRANS}$$

and for any $(\mathcal{E},\mathcal{B}),(\mathcal{E}',\mathcal{B}')\in$VAL, there exists a transition

$$(\mathtt{q},(\mathcal{E},\mathcal{B})) \xrightarrow[\mathtt{(urg)}]{\ell:\ \mathtt{g}'\ \mapsto\ \text{skip}} (\mathtt{q}',(\mathcal{E}',\mathcal{B}')) \in \text{TRANS} \qquad \text{if}$$

– g' is the the partial evaluation of g in $(\mathcal{E},\mathcal{B})$,
– the expression $\mathtt{exp}_i$ evaluates to the value v_i in $(\mathcal{E},\mathcal{B})$,
– $\mathcal{E}'=\mathcal{E}[v_2 \ldots v_j \ldots / y_2 \ldots y_j \ldots]$ for some v_j belonging to the domain of y_j,
– the post guard `cond` evaluates to `true` in the environment $\mathcal{E}'$,
– the label ℓ is equal to c $!\mathtt{v}_1$ $!\mathtt{v}_2$ ...

Composition of models: The timed automaton associated with a system of the form *Sys* = (*glob-def*,PROCS,S) is obtained by composing the timed automata of processes according to the composition expression S. The composition rules correspond to the *and*-parallel composition described in [BST98].

Let $[\mathtt{P}_i] = (\mathtt{Q}_i \times \text{VAL}, \text{TRANS}_i)_{i=1,2}$ be the timed automata associated with processes or subsystems of *Sys* — where VAL concerns only all global variables and is of the form $\text{ENV}_{glob} \times \text{BUF}$ and the valuations of local variables are integrated into the set of control states — and C a set of gates.

Then, $[\mathtt{P}_1[\![\mathtt{C}]\!]\mathtt{P}_2] = [\mathtt{P}_1]\ [\![\mathtt{C}]\!]\ [\mathtt{P}_2] = (\mathtt{Q}\times\text{VAL},\text{TRANS})$ where

- Q= $\mathtt{Q}_1 \times \mathtt{Q}_2$ where
$$\begin{aligned} init((\mathtt{q}_1,\mathtt{q}_2)) &= init(\mathtt{q}_1) \wedge init(\mathtt{q}_2)\\ stable((\mathtt{q}_1,\mathtt{q}_2)) &= stable(\mathtt{q}_1) \wedge stable(\mathtt{q}_2)\\ tpc((\mathtt{q}_1,\mathtt{q}_2)) &= tpc(\mathtt{q}_1) \wedge tpc(\mathtt{q}_2)\end{aligned}$$
- TRANS is the smallest set of transitions obtained by the following two rules: the first one applies to all transitions of TRANS_1 which are not synchronizations on gates in C and there is also a symmetrical rule for transitions of TRANS_2.

$$\boxed{\dfrac{(\mathtt{q}_1,\mathcal{V}) \xrightarrow[\mathtt{(urg)}]{\ell:\ \mathtt{g}\ \mapsto\ act} (\mathtt{q}_1',\mathcal{V}') \in \text{TRANS}_1 \ \text{ and } \ \neg stable(\mathtt{q}_1) \vee stable(\mathtt{q}_2)}{((\mathtt{q}_1,\mathtt{q}_2),\mathcal{V}) \xrightarrow[\mathtt{(urg)}]{\ell:\ \mathtt{g}\ \mapsto\ \mathtt{body}} ((\mathtt{q}_1',\mathtt{q}_2),\mathcal{V}') \in \text{TRANS}}}$$

The requirement on stableness implies that there is no interleaving in non stable states; they are transient states, such that a finite sequence of transitions between to stable states can be considered as *one* atomic transition.
The second rule concerns the synchronizations on gates in c∈C

$$\boxed{\dfrac{\begin{array}{l}(\mathtt{q}_1,\mathcal{V}) \xrightarrow[\mathtt{(u1)}]{\ell:\ \mathtt{g}_1\ \mapsto\ \text{skip}} (\mathtt{q}_1',\mathcal{V}') \in \text{TRANS}_1 \ \text{ and}\\ (\mathtt{q}_2,\mathcal{V}) \xrightarrow[\mathtt{(u2)}]{\ell:\ \mathtt{g}_2\ \mapsto\ \text{skip}} (\mathtt{q}_2',\mathcal{V}') \in \text{TRANS}_2\end{array}}{((\mathtt{q}_1,\mathtt{q}_2),\mathcal{V}) \xrightarrow[\mathtt{(urg)}]{\ell:\ \mathtt{g}_1 \wedge \mathtt{g}_2\ \mapsto\ \text{skip}} ((\mathtt{q}_1',\mathtt{q}_2'),\mathcal{V}') \in \text{TRANS}}}$$

In this rule, the synchronization of two transitions with the same urgency attribute result in a transition with the same attribute, the composition of an **eager** transition with any other transition results in an **eager** one, an in order to compose a **lazy** with a **delayable** transition, one needs to decompose the delayable one into two transitions, an **eager** and a **lazy** one, which under a reasonable restriction is always possible [BST98].

The semantics of Timed Automata The model of time of IF is that of Communicating Timed Automata with urgency introduced in [BST98]. Each process has a number of clocks which increase with progress of time (either in a discrete or continuous way). Clocks can be "tested" in the guards and "reset" in the bodies of the transitions. In this model, time is considered global, that is, it progresses synchronously in all processes of the system. The main problem is "when can time progress?". In timed automata [ACD93], time progress is defined by means of "invariants" associated with each state, such that time is allowed to progress as long the invariant expression evaluates to true. The problem with this model is that the composition of timed automata leads to time deadlocks. A time model avoiding this problem is obtained by associating with every transition a *deadline* (a predicate implying the guard), meaning that, whenever the deadline predicate evaluates to true, the transition has priority over progress of time. In [BST98], it has been shown that a much simpler model using three possible *urgency* attributes instead of deadlines is sufficient: *eager* transitions have always priority over time, *delayable* transitions may let time progress, but only as long as they remain enabled, and *lazy* transitions cannot prevent time from progressing. In IF the time progress condition in each state is defined depending on the urgency of enabled transitions and, in order to include the model of timed automata, one can associate an explicit *tpc* attribute with control states, with the potential risk of introducing time deadlocks by composition.

The semantics of Timed Automata with Urgency is defined in [BST98]. Let $A = (\mathcal{Q}, \textsc{Trans})$ be a Timed Automaton. Let TIME be a set of environments for clocks, where $\mathcal{T} \in \text{TIME}$ defines for every clock a value in a time domain T (positive integers or reals). Resetting a clock affects $\mathcal{T}$ by changing the value of the reset clock to *zero*. Progress of time by an amount δ transforms the valuation $\mathcal{T}$ into the valuation $\mathcal{T} \boxplus \delta$ in which the values of all clocks are increased by δ.

The semantics of A is defined by the labeled transition system $(\mathcal{Q} \times \text{TIME}, \rightarrow)$ where the transition relation $\rightarrow$ consists of two types of transitions, discrete ones and time progress transitions:

- For any transition $\mathtt{q}_1 \xrightarrow[(\mathtt{urg})]{\ell :\ \mathtt{g} \ \mapsto \ \mathtt{body}} \mathtt{q}_2$ of TRANS and $\mathcal{T} \in \text{TIME}$, there exists a discrete transition of the form $(\mathtt{q}_1, \mathcal{T}) \stackrel{\ell}{\rightarrow} (\mathtt{q}_2, \mathcal{T}')$ if
  - the guard $\mathtt{g}$ evaluates to $\mathtt{true}$ in $\mathcal{T}$,
  - and $\mathcal{T}$' is obtained from $\mathcal{T}$ by executing all assignments to clocks (resettings) of `body`.
- in any state $(\mathtt{q},\mathcal{T})$, time can progress by the amount δ, that is

$$(\mathtt{q}, \mathcal{T}) \xrightarrow{\text{time: } \delta} (\mathtt{q}, \mathcal{T} \boxplus \delta)$$

if time can progress in the state ($\mathtt{q}$,$\mathcal{T}$) and continuously until $\mathcal{T} \boxplus \delta'$: whenever time has progressed by an amount δ' where $0 \leq \delta' < \delta$, time can still progress in the reached state ($\mathtt{q}$,$\mathcal{T} \boxplus \delta'$).

Time can progress in a state ($\mathtt{q}$,$\mathcal{T}$) if and only if the following conditions hold:

- *stable*($\mathtt{q}$), that means time can progress only in stable, never in transient states
- the time progress attribute *tpc*($\mathtt{q}$) evaluates to `true` in $\mathcal{T}$
- *no* **eager** transition is enabled in ($\mathtt{q}$,$\mathcal{T}$)
- for each **delayable** transition `tr` enabled in ($\mathtt{q}$,$\mathcal{T}$), there exists a positive amount of time ϵ, such that `tr` remains enabled when time progresses by ϵ. That means enabled delayable transitions allow time to progress, but only as long as they remain enabled.

3 IF and other formalisms

IF is a formalism for the description of asynchronous systems at a programming language level. However, it has not been designed with the aim to replace specification languages such as LOTOS, SDL and Promela. IF has been designed as an intermediate representation for SDL but it can also be used for other specification formalisms. Thus, the expressiveness of IF and its adaptedness as an intermediate representation from SDL, LOTOS and Promela are discussed below.

3.1 SDL

The definition of SDL (Specification and Description Language) started in 1974 and it has been standardized by CCITT in 1988 [IT94c]. SDL is based on extended finite state machines communicating asynchronously via queues. There exists a formal semantics of the language defined in [IT94a,IT94b], various authors have criticized it and proposed alternative ones [Bro91,BMU98,God91] to name only a few of them. Currently, SDL is widely accepted by the industrial community. This is due mainly to the fact that SDL development is supported by methodologies [OFMP$^+$94] and tools [Ver96,AB93] in all phases from requirement analysis, design and validation to implementation. However, the validation capabilities of the industrial tools are rather limited with respect to the ones existing in the academic community.

There is no standard semantics of time defined for SDL. For example, *Object*GEODE uses a very "synchronous" time concept in which time can only progress when the system is blocked (that means all transitions are eager), whereas others consider that time can always progress (that means that all transitions are lazy). This shows that the currently used notions of time in SDL are extreme ones — which is often considered as problem by the users — and many intermediate solutions are possible using IF as discussed in previous section.

We have identified a static subset of SDL which we are able to translate into IF. That is, with the exception of dynamic creation of processes and some mobility aspects of communication, we can define a syntactic level translation between these two formalisms. A prototype translator has been implemented using the SDL/API Interface provided by ObjectGeode [Ver96]. More detailed information about it can be found in [BFG+99].

3.2 LOTOS

LOTOS (Language Of Temporal Ordering Specifications) [BB88] has been developed and standardized by ISO in 1989. It is a process-algebra based on CCS[Mil80] and CSP[Hoa84]. In LOTOS, the communication is synchronous using rendez-vous. LOTOS has a well-defined operational semantics and there exist tools supporting it.

The right approach to model and validate LOTOS specifications is recognized to be the use of Petri nets, rather than communicating extended automata [GS90] as intermediate representation. However, our experience with LOTOS has shown that often the specifications have the form of a parallel composition of sequential components (processes). This observation motivated also the use of compositional generation methods, which gives good results for this kind of LOTOS specifications [KM97].

The timed extensions introduced in E-Lotos[Que98], ET-Lotos[LL97] and Lotos-NT[Sig99] are similar to that of IF, only that the urgency of an action is defined implicitly by its type: "exceptions" and internal actions are urgent, whereas observable actions are not. This is due to the fact, that they want to achieve a much stronger form of compositionality, where with each process can be associated a directly labeled transition system (and not a timed automaton) which then can be composed into a system model.

We plan to investigate the translation of decomposable LOTOS specifications into IF, as parallel composition with synchronization between processes can be handled in IF. Furthermore, a reasonably small Petri Net (corresponding to a non-decomposable LOTOS part) can be modeled by an IF process.

3.3 PROMELA

Another language we have considered is Promela, the native language of the Spin model-checker [Hol91]. It was designed as an intermediate representation language for protocols, mainly for validation purposes. It is based on extended finite-state machines communicating asynchronously or synchronously via queues.

Promela has not really been designed as a specification language but it has a relatively important visibility as well in the academic community as in the industrial one. Its success is due to the high availability of Spin, which provides powerful model-checking algorithms based on partial-order reductions.

There exist timed extensions of Promela. The one proposed in [CT96] is based on Timed Automata, whereas [BD98] is very similar to the notion of time used in *Object*GEODE: all set timers decrease synchronously until one of them expires; then time is blocked until the corresponding timeouts are consumed, where these timeout consumptions take place when no other transition is possible in the system.

A translator from IF to Promela is currently developed in the framework of the VIRES Esprit-LTR project at Eindhoven University. We plan to study also a backward translation from Promela to IF. However, as for SDL, there are some limitations due to dynamic process creation feature of Promela.

4 A validation environment based on IF

One of the main motivation for developing the IF intermediate representation is to provide an "open" validation environment, able to make heterogeneous tools cooperate within a single framework. Especially for SDL, solid industrial tools for editing and code generation have been built which are used by a large community of users. On the other hand, there exist many verification tools built upon diverse formalisms — such as the Spin tool [Hol91] based on Promela, the CADP tool[FGK+96] based on LOTOS, the SMV tool [Mac93] based on extended automata, tools for the verification of Timed systems such as KRONOS[Yov97] and Uppaal [LPY97] based on different representations of Timed Automata, to name only a few of them.

Therefore, an integrated validation environment should fulfill the following requirements:

- First of all, it is able to support several validation techniques, from symbolic interactive simulation to automatic property checking, together with test case generation 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.
- Moreover, for a sake of efficiency, this environment also has to support several level of program representations. For instance it is well-known that model-checking verification of real life case studies usually needs to combine several optimization techniques to overcome the state explosion problem.
  In particular, some of these techniques rely on a program level representation, like static analysis and computations of abstractions (for which it may be necessary to cooperate with decision procedures or theorem-prover). Other techniques operate on a representation of the underlying model, such as on-the-fly analysis, bisimulation based model reduction or model-checking. These representations can be either implicit, enumerative or symbolic and are explained below.
- Another important feature is to keep this environment open and evolutive. Therefore, tool connections are performed only by means of file sharing or program representation access. For this purpose several well-defined interfaces are offered.

In the remainder of the section we present the overall architecture of the existing environment and some of its related components. Then, we describe in a more detailed manner two specific modules concerning static analysis (section 4.2) and compositional generation (section 4.3).

4.1 Overall architecture

The IF validation environment is built upon two levels of program representation, each of them being accessed through well-defined API.

The syntactic level allows to consult and modify the abstract tree on an IF program. Since all the variables, timers, buffers and the communication structure are still explicit, high-level transformations based on static analysis (such as *live variable* computation, see below) or abstraction computations can be applied. Moreover, this API is also well suited to implement translators from IF to other specification formalisms (like Promela or INVEST).

The execution model level gives access to the underlying LTS of the IF program. In practice three distinct API are offered, depending on the representation used.

- The **implicit enumerative representation** is based on the OPEN-CAESAR [Gar98] philosophy. It consists in a set of C functions and data structures allowing to compute on demand the successors of a given state. This piece of C code is generated by the IF compiler, and it can be linked with a "generic" exploration program performing on the fly analysis (deadlock detection, model-checking, test-case generation, ...).
- In the **symbolic representation** (called SMI [Boz97]) set of states and transitions of the LTS are expressed by their characteristic functions over a set of finite variables. These functions are implemented in terms of decision diagrams(BDDs[Bry86] andMDDS). 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 format with the associated access library. Although this explicit representation is not suitable for handling a large system globally, it is still useful in practice to minimize some of its abstractions with respect to bisimulation based relations (like in *compositional generation*, see below).

Figure 4.1 describes the existing connections in this environment (plain arrows) and the planned ones (dashed arrows). Most of the tools mentioned are presented below:

- CADP [FGK$^+$96,BFKM97] is a tool set for the verification of LOTOS specifications. It has been developed and by VERIMAG and the VASY team of INRIA Rhône-Alpes. We briefly present here two verifiers integrated in this tool set which have already been connected to the new IF environment:
  - ALDEBARAN compares and minimizes finite LTSs with respect to various *simulation* or *bisimulation* relations. This allows the comparison of the observable behavior of a specification with its expected one, described at a more abstract level.
  - EVALUATOR is a "on-the-fly" model-checker for formulas of the alternating-free μ-calculus [Koz83].

  As for the IF environment, an important feature of CADP is to offer several LTS representations and in particular the OPEN-CAESAR API which is fully compatible with the IF implicit enumerative representation.
- TGV [FJJV97] is a test sequence generator built upon CADP jointly by VERIMAG and the PAMPA project of IRISA. TGV aims to automatically generate test cases for conformance testing of distributed systems. Test cases are computed during the exploration of the model and they are selected by means of *test purposes*. Test

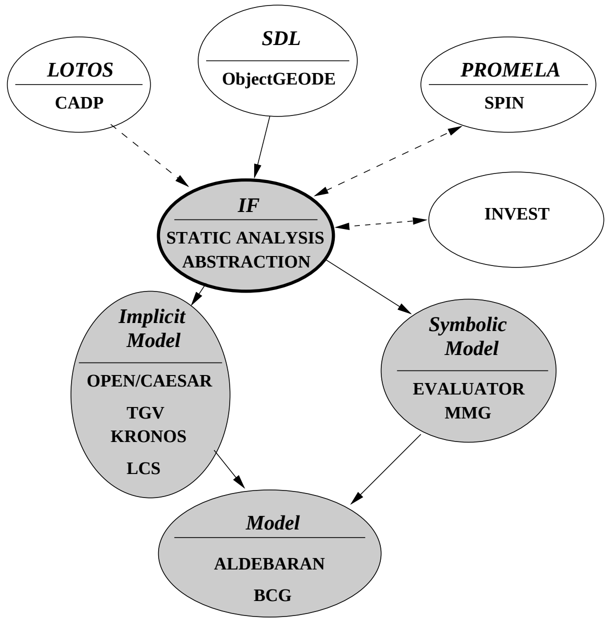

Fig. 1. An open validation environment for IF

purposes characterize some abstract properties that the system should have and one wants to test, given trees of labels, decorated with verdicts "ok" and "fail".

- MMG [FKM93] is a minimization tool based on a partition refinement algorithm combined with a reachable state space computation [BFH90]. This tools works on the symbolic SMI interface.
- *Object*GEODE is a tool set developed by VERILOG supporting SDL, MSC and OMT. It includes graphical editors and compilers for each of these formalisms. It also provides a C code generator and a simulator to help the user to interactively debug an SDL specification.
  *Object*GEODE also provides an API offering a set of functions and data structures to access the abstract tree generated from an SDL specification. A translation tool (SDL2IF) uses this abstract tree to generate an operationally equivalent IF specification.
- KRONOS [Yov97] is a TCTL model-checker for communicating timed automata. The current connection with the IF environment is as follows: control states and discrete variables are expressed using the IF implicit enumerative representation whereas clocks are expressed using an appropriate symbolic representation (particular polyhedra).
- INVEST [BLO98] is a tool computing abstractions and invariants on a set of guarded command processes communicating through shared variables. We are actually implementing the translation between this formalism and IF, which will allow to compute abstract systems.

4.2 Static analysis and abstraction

Considering the expressiveness of IF, static analysis and abstraction techniques can be applied in order to improve the verification and the test generation process. The expected benefits of such techniques are mainly the reduction of the state space and the reduction of the complexity of the IF program.

We have already experimented very simple analysis, used classically in the context of data-flow analysis which can be divided into two classes:

- *property independent* analysis: without regarding any particular property or test purpose (such as live variable analysis or constant propagation),
- *property dependent* analysis: takes into account some information of the property and propagate them over the static control structure of the program (such as *irrelevant variables* abstraction)

We briefly describe two techniques currently implemented for IF.

"Live variables" analysis: A variable is *live* in a control state if there is a path from this state along which its value can be used before it is redefined. An important reduction of the state space of the model can be obtained by taking into account in each state only the values of the live variables.

The reduction considered is based on the relation $\sim_{live}$ defined on the state space of the model: two states are related if and only if all live variables have the same value. It can easily be proved that $\sim_{live}$ is a bisimulation on the model. This result can be exploited in several ways. Due to the local nature of $\sim_{live}$ it is possible to directly generate the quotient model w.r.t. $\sim_{live}$ instead of the whole model without any extra computation. Exactly the same reduction is obtained by "resetting"(that is assigning some predefined value) in the initial program systematically all non-live variables. The second approach is implemented for IF.

"Irrelevant variables" abstraction: Given a subset X of *irrelevant* variables (which in general depend on the considered property), an upper approximation for this program can be computed syntactically and iteratively as follows:

- the assignments of variables belonging to X are removed
- the expressions containing variables from X are replaced by a special `any` value (denoting any element of its domain)
- the guards which evaluate to `any` are removed (i.e., replaced by the `true` value)

Clearly the resulting program contains all executions of the initial one and does not depend anymore on variables in X. Any *safety* property valid on the resulting abstract program is also valid on the initial one, but the converse does not necessarily hold.

In practice, the set of irrelevant variables can be chosen in different manners. Either, the user can directly supply it based on his knowledge about the specification and the given verification context. Another possibility is to derive this set automatically from the considered property (or the test purpose). Finally, irrelevant variables can also be taken among the variables provided by the environment and the ones depending on them: such variables are *uncontrolled* since their values are nondeterministically chosen by the environment. This latter choice has been considered in [CGJ98].

We are also investigating more general abstraction techniques. For instance, through the connection with INVEST we will be able to compute abstract IF programs using more general and powerful abstraction techniques.

4.3 Compositional generation

As shown in the previous section, efficient reductions are obtained by replacing a model M by its quotient w.r.t an equivalence relation like $\sim_{live}$. Much weaker equivalences (that is smaller quotients) can be obtained by taking into account the properties under verification. In particular, it is interesting to consider a weaker equivalence R — which should be a congruence for parallel composition —, able to abstract away non observable actions. The main difficulty is to obtain (an approximation of) the quotient M/R without generating M as a whole.

A possible approach is based on the "divide and conquer" paradigm: splitting the program description into several pieces (i.e., processes or process sets), generate the model M_i associated with each of them, and then compose the quotients M_i/R. The hope is that the generated intermediate models can be kept small.

This compositional generation method has already been applied for specification formalisms based on *rendez-vous* communication between processes, and has been shown efficient in practice [GLS96,Val96,KM97]. To our knowledge it has not been investigated for systems based on communication via buffers, may be, because buffers raise several difficulties or due to the lack of suitable representations and tools.

The potential benefit of this compositional approach will be illustrated on an example in the next section.

5 An illustrating example

We present a simple example to illustrate the IF formalism and related verification tools. We consider a *token ring*, that is a system of n stations (processes) $S_1, \ldots\ S_n$, connected in a circular network, in which a station is allowed to access some shared resource R only when it "owns" a particular message, the *token*. If the network is unreliable it is necessary to recover from token loss. This can be done using a *leader election algorithm* [Lan77,CR79] to designate a station responsible for generating a new token.

Table 1 shows the global definitions of the IF specification corresponding to the particular protocol considered in [GM96]. The signals `open` and `close` denote the access and the release of the shared resource (here a part of the environment). The signals `token` and `claim` are the messages circulating on the ring.

All stations S_i are identical up to their identity and described by an IF process as the one of Figure 2. The timer `worried` is set when the station waits for the token and reset when it receives it. On expiration of the timer `worried` token loss is assumed and an election phaes is started. The "alternating bit" `round` is used to distinguish between valid claims (emitted during the current election phase) and old ones (cancelled by a

signal	
close;	**sync**
open;	S1 ||| S2 ||| S3 ||| S4
claim(pid, bool);	**end**;
token;	**process** S1;
buffer	**var**
Q1 : **queue** :lossy **of** claim, token;	worried : timer;
Q2 : **queue** :lossy **of** claim, token;	round, rnd: bool;
Q3 : **queue** :lossy **of** claim, token;	adr: pid;
Q4 : **queue** :lossy **of** claim, token;	...

Table 1. IF global definitions

token reception). In the `idle` state, a station may either receive the token from its neighbour (then it reaches the `critical` state and can access the resource) or receive the timer expiration signal (then it emits a claim stamped with its `address` and the current value of `round`) or receive a claim. A received claim is "filtered" if its associated `address` is smaller than its own address and transmitted unchanged if it is greater. If its own valid claim is received, this station becomes elected and generates a new token.

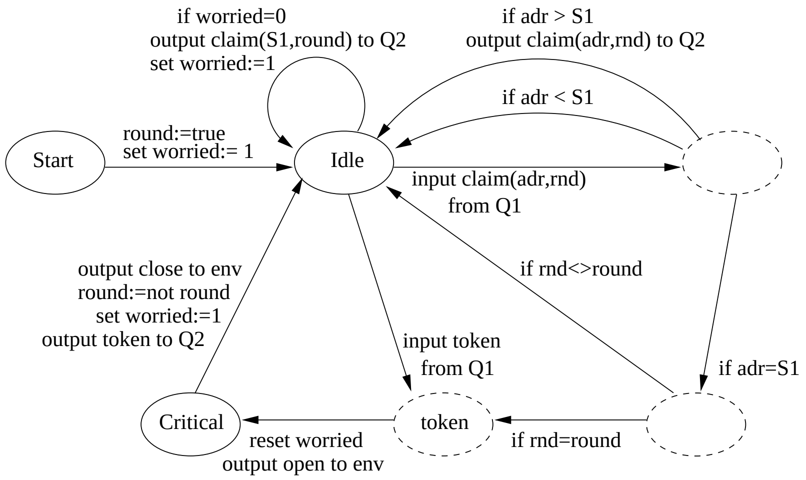

Fig. 2. The behavior of station S_1

Model generation: We summarize in Table 2 the size of the models obtained from the token-ring protocol using three generation methods: directly from the initial IF program (global generation), using the live variable reduction (global + live) and using a compositional generation strategy (compositional + live).

model	*generation method*	*states*	*transitions*
1.	global	537891	2298348
2.	global + live	4943	19664
3.	compositional + live	1184	4788

Table 2. Models obtained for the token ring example

The most spectacular reduction is obtained by the live reduction: the reduced model is about 100 times smaller than the one obtained by simultaneous generation, while preserving *all* properties (models 1 and 2 are strongly bisimilar). This is explained by the fact that only a few variables are live in each state: in the `idle` state the live variables are `round` and `worried`, in the `critical` state only `round` is live, while variables `adr` and `rnd` are never live.

More reduction is achieved by the following compositional generation strategy:

1. We split the IF description into two parts, the first one contains processes S_1 and S_2 and the second one processes S_3 and S_4. For each one of these descriptions, the internal buffer between the two processes is *a priori* bounded to two places. Note that, when a bounded buffer overflows during simulation, a special *overflow* transition occurs in the corresponding execution sequence.
2. The LTS associated with each of these two descriptions are generated considering the "most general" environment providing any potential input. As `claim` and `token` can be transmitted at any time, *overflow* transitions appear in the corresponding LTSs.
3. In each LTS the input and output transitions relative to the internal buffers (Q_2 and Q_4) are hidden (i.e., renamed to the special τ action); then they are reduced w.r.t an equivalence relation preserving the properties under verification. For the sake of efficiency we have chosen the branching bisimulation [vGW89] preserving all the safety properties (e.g. mutual exclusion).
4. Each reduced LTS is translated back into an IF process (without variables), and the resulting processes are combined into a single IF description, including the two remaining buffers (Q_1 and Q_3). It turns out that the LTS generated from this new description contains no *overflow* transitions (they have been cut off during the second composition, which confirms the hypothesis on the maximal size of the internal buffers).

The final LTS is branching bisimilar to the one obtained from the initial IF description.

Verification: We are interested in checking that the shared resource is accessed in mutual exclusion. For this, we consider as visible only the `open` and `close` actions.

Mutual exclusion property can be rephrased as follows: *every* $\texttt{open}_i$ *(station* i *access the resource) can only be followed by* $\texttt{close}_i$ *(station* i *releases the resource) possibly after some internal moves* τ. This property can be expressed in the μ-calculus (see below) and verified with EVALUATOR, on any of the generated models.

$$\bigwedge_{i=1}^{4} \nu X.\ ([\texttt{open}_i]\ \neg\mu Y.\ (\langle\overline{\{\texttt{close}_i, \tau\}}\rangle T\ \vee\ \langle\tau\rangle Y)\ \wedge\ [*]X)$$

Another approach to verify mutual exclusion is to compare the model of the specification with an abstract one expressing the desired behavior. For instance, all three models are branching bisimilar to the one shown in Figure 3. The reductions and comparisons are carried out using ALDEBARAN.

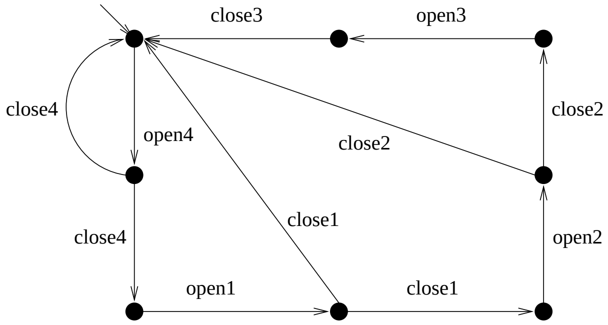

Fig. 3. The reduced behavior of the token ring.

Test Generation: We illustrate the use of the TGV to extract test cases for the token ring protocol. We want to test the property stating that a station filters a received claim with a smaller address than its own and transmits it unchanged otherwise. We chose a test purpose expressing that after S_4 has sent its claim, it will be transmitted unchanged by station S_1, then by S_2 and finally by S_3. The generated test case is shown in figure 4.

6 Conclusion and perspectives

We have presented the formalism IF which has been designed as an intermediate representation for SDL, but it can be used as a target language for other FDT as it contains most of the concepts used in these formalisms. The use of IF offers several advantages:

- IF has a formal semantics based on the framework of communicating timed automata. It has powerful concepts interesting for specification purposes, such as different urgency types of transitions, synchronous communication, asynchronous communication through various buffer types (bounded, unbounded, lossy, ...).

```
+-------------------------------------------------------------------------------+
|                         Test Case Dynamic Behaviour                           |
+-------------------------------------------------------------------------------+
| Test Case Name  : castest                                                     |
| Group           :                                                             |
| Purpose         :                                                             |
| Default         :                                                             |
| Comments        :                                                             |
+----+-------+------------------------------------+---------+-------------+---+
| Nr | Label | Behaviour Description              | Cts Ref | Verdict     | C |
+----+-------+------------------------------------+---------+-------------+---+
|  1 |       | s3? claim                          | claim3  | INCONC      |   |
|  2 |       | s2? claim                          | claim2  | INCONC      |   |
|  3 |       | s1? claim                          | claim1  | INCONC      |   |
|  4 |       | s4? claim                          | claim0  |             |   |
|  5 |       |   s1! claim, St tclaim             | claim0  |             |   |
|  6 |       |     s3? claim, Cl tclaim           | claim3  | INCONC      |   |
|  7 |       |     s2? claim, Cl tclaim           | claim2  | INCONC      |   |
|  8 |       |     s1? claim, Cl tclaim           | claim1  | INCONC      |   |
|  9 |       |     s1? claim, Cl tclaim           | claim4  |             |   |
| 10 |       |       s2! claim, St tclaim         | claim4  |             |   |
| 11 |       |         s3? claim, Cl tclaim       | claim3  | INCONC      |   |
| 12 |       |         s2? claim, Cl tclaim       | claim2  | INCONC      |   |
| 13 |       |         s1? claim, Cl tclaim       | claim1  | INCONC      |   |
| 14 |       |         s2? claim, Cl tclaim       | claim5  |             |   |
| 15 |       |           s3! claim, St tclaim     | claim5  |             |   |
| 16 |       |             s3? claim, Cl tclaim   | claim3  | INCONC      |   |
| 17 |       |             s2? claim, Cl tclaim   | claim2  | INCONC      |   |
| 18 |       |             s1? claim, Cl tclaim   | claim1  | INCONC      |   |
| 19 |       |             s3? claim, Cl tclaim   | claim6  | (PASS)      |   |
| 20 |       |             ? tclaim               |         | FAIL        |   |
| 21 |       |         ? tclaim                   |         | FAIL        |   |
| 22 |       |     ? tclaim                       |         | FAIL        |   |
+----+-------+------------------------------------+---------+-------------+---+
```

Fig. 4. TTCN test case

- IF programs can be accessed at different levels through a set of well defined APIs. These include not only several low-level model representations (symbolic, enumerative, ...) but also higher level program representation, where data and communication structures are still explicit. Using these API several tools have been already interconnected within an open environment able to cover a wide spectrum of validation methods.

The IF package is available at `http://www-verimag.imag.fr/DIST_SYS/IF`. In particular, a translation tool from SDL to IF has been implemented and allows both to experiment different semantics of time for SDL and to analyze real-life SDL specifications with CADP.

A concept which is not provided in IF is dynamic creation of new process instances of processes and parameterization of processes; this is due to the fact that in the framework of algorithmic verification, we consider only static (or dynamic bounded) configurations. However, it is foreseen in the future to handle some kinds of parameterized specifications.

The results obtained using the currently implemented static analysis and abstractions methods are very encouraging. For each type of analysis, it was possible to build a module which takes an IF specification as input and which generates an *reduced* one. This architecture allows to chain several modules to benefit from multiple reductions applied to the same initial specification. We envisage to experiment more sophisticated analysis, such as constraints propagation, and more general abstraction techniques. This will be achieved either by developing dedicated components or through the connections with tools like INVEST.

References

[AB93] Telelogic AB. *SDT Reference Manual*. http://www.telelogic.se/solution/tools/sdt.asp, 1993.

[ACD93] R. Alur, C. Courcoubetis, and D.L. Dill. Model Checking in Dense Real Time. *Information and Computation*, 104(1), 1993.

[BB88] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. *ISDN*, 14(1), January 1988.

[BD98] D. Bošnački and D. Dams. Integrating Real Time into Spin: A Prototype Implementation. In *Proceedings of the FORTE/PSTV XVIII Conference*, 1998.

[BFG+98] M. Bozga, J-C. Fernandez, L. Ghirvu, S. Graf, L. Mounier, J.P. Krimm, and J. Sifakis. The Intermediate Representation IF. Technical report, Vérimag, 1998.

[BFG+99] M. Bozga, J.C. Fernandez, L. Ghirvu, S. Graf, J.P. Krimm, L. Mounier, and J. Sifakis. If: An Intermediate Representation for SDL and its Applications. In *Proceedings of SDL-FORUM'99, Montreal, Canada*, June 1999.

[BFH90] A. Bouajjani, J.C. Fernandez, and N. Halbwachs. Minimal Model Generation. In *Proceedings of CAV'90, Rutgers, New Jersey*, DIMACS, June 1990.

[BFKM97] M. Bozga, J.-C. Fernandez, A. Kerbrat, and L. Mounier. Protocol Verification with the ALDEBARAN Toolset. *Software Tools for Technology Transfer*, 1, 1997.

[BLO98] S. Bensalem, Y. Lakhnech, and S. Owre. Computing Abstractions of Infinite State Systems Compositionally and Automatically. In *Proceedings of CAV'98, Vancouver, Canada*, volume 1427 of *LNCS*, June 1998.

[BMU98] J.A. Bergstra, C.A. Middelburg, and Y.S. Usenko. Discrete Time Process Algebra and the Semantics of SDL. Technical Report SEN-R9809, CWI, June 1998.

[Boz97] M. Bozga. SMI: An Open Toolbox for Symbolic Protocol Verification. Technical Report 97-10, Vérimag, September 1997.

[Bro91] M. Broy. Towards a Formal Foundation of the Specification and Description Language SDL. *Formal Aspects on Computing*, 1991.

[Bry86] R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Trans. on Computation*, 35 (8), 1986.

[BST98] S. Bornot, J. Sifakis, and S. Tripakis. Modeling Urgency in Timed Systems. In *International Symposium: Compositionality - The Significant Difference, Malente (Holstein, Germany)*, 1998. to appear in LNCS.

[CGJ98] C. Colby, P. Godefroid, and L.J. Jagadeesan. Automatically Closing Open Reactive Systems. In *Proceedings of ACM SIGPLAN on PLDI*, June 1998.

[CR79] E. Chang and R. Roberts. An Improved Algorithm for Decentralized Extrema-Finding in Circular Configurations of Processes. *Communications of ACM*, 22(5), 1979.

[CT96] C. Courcoubetis and S. Tripakis. Extending Promela and Spin for Real Time. In *Proceedings of TACAS'96*, volume 1055 of *LNCS*, 1996.

[FGK+96] J.-C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, and M. Sighireanu. CADP: A Protocol Validation and Verification Toolbox. In *Proceedings of CAV'96 (New Brunswick, USA)*, volume 1102 of *LNCS*, August 1996.

[FJJV97] J.-C. Fernandez, C. Jard, T. Jéron, and C. Viho. An Experiment in Automatic Generation of Test Suites for Protocoles with Verification Technology. *Science of Computer Programming*, 29, 1997.

[FKM93] J.C. Fernandez, A. Kerbrat, and L. Mounier. Symbolic Equivalence Checking. In *Proceedings of CAV'93, Heraklion, Greece*, volume 697 of *LNCS*, June 1993.

[Gar98] H. Garavel. OPEN/CÆSAR: An Open Software Architecture for Verification, Simulation, and Testing. In *Proceedings of TACAS'98*, volume 1384 of *LNCS*, March 1998.

[GLS96] S. Graf, G. Lüttgen, and B. Steffen. Compositional Minimisation of Finite State Systems using Interface Specifications. *Formal Aspects of Computation*, 3, 1996.

[GM96] H. Garavel and L. Mounier. Specification and Verification of Distributed Leader Election Algorithms for Unidirectional Ring Networks. *SCP*, 1996.

[God91] J.C. Godskesen. An Operational Semantic Model for Basic SDL. Technical Report TFL RR 1991-2, Tele Danmark Research, 1991.

[GS90] H. Garavel and J. Sifakis. Compilation and Verification of LOTOS Specifications. In *Proceedings of the 10th PSTV (Ottawa, Canada)*, June 1990.

[HNSY94] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic Model Checking for Real-Time Systems. *Information and Computation*, 111(2), 1994.

[Hoa84] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice Hall International, 1984.

[Hol91] Gerard J. Holzmann. *Design and Validation of Computer Protocols*. Prentice Hall Software Series, 1991.

[ISO88] 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, 1988.

[IT94a] ITU-T. *Annex F.2 to Recommendation Z-100. Specification and Description Language (SDL) - SDL Formal Definition: Static Semantics.* 1994.

[IT94b] ITU-T. *Annex F.3 to Recommendation Z-100. Specification and Description Language (SDL) - SDL Formal Definition: Dynamic Semantics.* 1994.

[IT94c] ITU-T. *Recommendation Z-100. Specification and Description Language (SDL).* 1994.

[KM97] J.P. Krimm and L. Mounier. Compositional State Space Generation from Lotos Programs. In *Proceedings of TACAS'97*, Enschede, The Netherlands, 1997.

[Koz83] D. Kozen. Results on the Propositional μ-Calculus. In *Theoretical Computer Science*. North-Holland, 1983.

[Lan77] G. Le Lann. Distributed Systems – Towards a Formal Approach. In *Information Processing 77*. IFIP, North Holland, 1977.

[LL97] L. Leonard and G. Leduc. An Introduction to ET-LOTOS for the Description of Time-Sensitive Systems. *Computer Networks and ISDN Systems*, (29), 1997.

[LPY97] K.G. Larsen, P. Petterson, and W. Yi. UPPAAL: Status & Developments. In *Proceedings of CAV'97, Haifa, Israel*, volume 1254 of *LNCS*, 1997.

[Mac93] K.L. MacMillan. *SYmbolic Model CHecking: an Approach to the State Explosion Problem*. Kluwer Academic Publisher, 1993.

[Mil80] R. Milner. A Calculus of Communication Systems. In *LNCS*, number 92. 1980.

[OFMP+94] A. Olsen, O. Færgemand, B. Møller-Pederson, R. Reed, and J.R.W. Smith. *Systems Engineering Using SDL-92*. North-Holland, 1994.

[Que98] J. Quemada. Final Comitee Draft on Enhancements to LOTOS. Technical report, ISO/IEC JTC1/SC33/WG9, April 1998.

[Sig99] M. Sighireanu. *Contribution at the Definition and Implementation of E-LOTOS*. PhD thesis, Universite Joseph Fourier, Grenoble, 1999.

[Val96] A. Valmari. Compositionality in State Space Verification. In *Application and Theory of Petri Nets*, volume 1091 of *LNCS*, 1996.

[Ver96] Verilog. Object*GEODE SDL Simulator - Reference Manual.* http://www.verilogusa.com/solution/pages/ogeode.htm, 1996.

[vGW89] R.J. van Glabbeek and W.P. Weijland. Branching-Time and Abstraction in Bisimulation Semantics. CS R8911, CWI, 1989.

[Yov97] S. Yovine. Kronos: A Verification Tool for Real-Time Systems. *Software Tools for Technology Transfer*, 1(1-2), December 1997.