Verification in the Codesign process by means of Lotos based model-checking

Fabrice Baray and Pierre Wodey

ISIMA/LIMOS Laboratory,
Blaise Pascal University Clermont Ferrand II,
BP 10125 F63173 Aubière, France,
fabrice.baray@isima.fr,pierre.wodey@isima.fr

Abstract. When considering the design of complex systems, the designers use ever more synthesis tools transform formal specifications into an implementation of the system. Such tools are based on a given description of the system. The description is based on a model of computation including behaviour and communication mechanisms. The model of computation depends on the level of representation of the system and varies from the specification to the implementation. There exist generally verification tools associated with the specification level but, for a more implementation oriented model the verification is often inexistent. But according to the semantic transformations in the design process (mainly at the communication level), this verification is needed. As generally implementation model of computation are composed of communicating finite state machines with datapath (CFSMD) in which the communications are performed by mean of "hardware" signals (physical connections), we propose to allow model checking verification on this model of computation. This paper presents the translation between the CFSMD and an equivalent Lotos description in which the communication basic mechanism is the rendezvous. Based on process algebra a Lotos description is easily translated into a labelled transition system by existing model checkers such as the Cadp toolbox which we use in our experiment. We apply this technique on the Cosmos Codesign environment in which the deadlock free property has to be verified at each step of transformation in the design, the equivalence of communication semantics being not assured by the transformations. The deadlock free property is described by temporal logic formulas handled by the Xtl model checker included in the Cadp toolbox.

Keywords. Verification, Model-Checking, LOTOS, Communicating Finite State Machine, Codesign.

1 Introduction

For the design of complex systems the designers use ever more CAD tools working at the system level [GM93,Wol94,GV95,ELLSV97]. Such tools offer generally the following capabilities :

– formal or abstract specification of the system,

– verification at the specification level,
– architecture exploration linked with performance analysis,
– automatic synthesis of behaviour and communication,
– automatic code generation,
– simulation of the generated model.

Thus, such tools handle descriptions of the system based on model of computation. A model is composed of a behavioural (control, action) model of individual components and a communication model (communication mechanisms) among components [LSVS98].

The abstraction level of the specification formalism and its model of computation offer the ability to perform easily formal verification by a model checking technique for instance.

During the design, starting from the specification to the code generation, the description evolves together with the model of computation. Actually, the communication mechanisms at the specification level are quiet different from those at the implementation level.

Furthermore, at the communication point of view, the generated system does not implement a semantically equivalent mechanism as the one at the specification level (which is too abstract for implementation). This induces that the implemented system has not an equivalent global behaviour as the specified one. So, the properties verified at the specification level are no more guaranteed at the implementation level.

Thus, there is a need to be able to verify properties by applying model checking at the implementation level or at any level in the design process [WB99]. This needs to be able to generate a verifiable model from the model of computation at the implementation level. This is the purpose of our paper.

The considered model of computation is the communicating finite state machines with datapath (CFSMD) where the only communication mechanism is the "hardware" signal (connection net between components). This communication mechanism is at a very low level of abstraction.

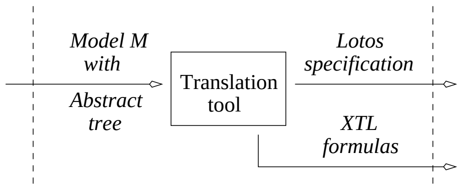

Fig. 1. Tool architecture

From such a description our tool generates (Fig. 1) :

– a semantically equivalent description in LOTOS language [ISO88,GLO91] in order to use model checking tools,

– temporal logic formulas allowing to verify the deadlock free property which is at this time the only one considered.

The choice of LOTOS is motivated by :

– LOTOS is an ISO standard [ISO88],
– LOTOS is based on process algebra and induces clearly a Label Transition System (LTS) needed for model checking,
– several verification tools accept LOTOS as entry.

The proposed translation has been applied considering the CADP toolbox developed at INRIA [GS90,Gar89,FGM+91,Gar98], on one hand, and on the Codesign environment COSMOS developed at TIMA Laboratory [IJ95,VCJ96], on another hand.

The CADP toolbox accepts LOTOS as entry and performs model checking on a generated LTS. It includes also logic formula checking described in the XTL language [SM98,Mat98].

The COSMOS tool is a good representative of a complete and realistic Codesign tool.

This paper is structured as follow :

– introduction of an example used to illustrate the different part of the paper and also pointing out a deadlock introduction in the COSMOS tool,
– formal presentation of the implementation oriented model of computation including behaviour and communications (CFSMD),
– the translation of the communications which are based on different mechanism in the implementation model and in the LOTOS model (rendezvous),
– the translation of the behaviour into LOTOS,
– XTL formulas automatically generated for deadlock free checking,
– the application on COSMOS Codesign tool.

2 Illustrating example

As example, we propose a system composed of three processes : two producers, and one consumer. The consumer accepts data from the two producers, but in some states, it limits to one specific producer. The structural representation of this simple example is given in figure 2.

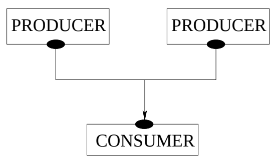

Fig. 2. Structural representation

The behavioural representation of the consumer is given in figure 3. The system is described in SDL [SDL88] specification language. Each component behaviour is described by a finite state machine. At this high level of specification, the consumer has three possible states. In one state, it waits independently a data from the two producers and in others states, it waits data only from one producer. In the communication point of view, in SDL, the components have gates and are communicating by asynchronous signal exchange through buffer. In the example, *Prod1_Cons_Value* are *Prod2_Cons_Value* two gates on which consumer read data. When a component send a data, it is not blocked and the data is placed in the buffer. When a component received a data, it is blocked until a data is in the buffer.

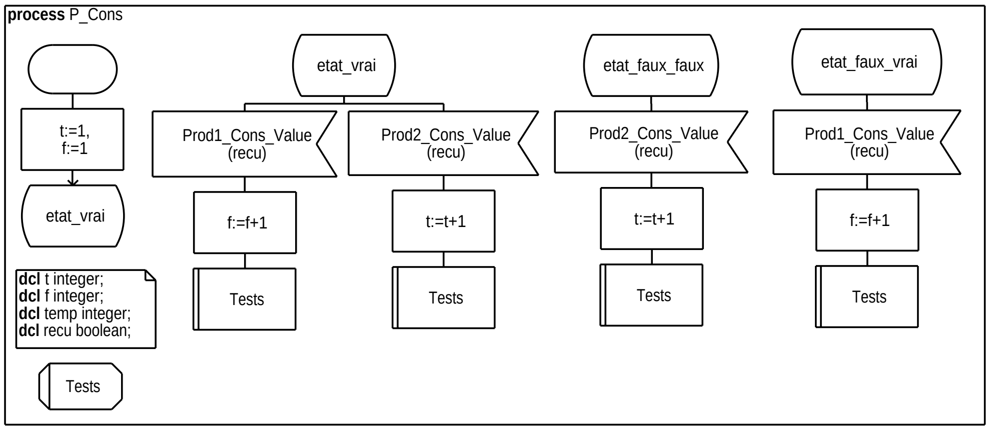

Fig. 3. Consumer behavioural representation

After one step of synthesis, a new description in a different model of computation is generated with a FIFO queue between the components (Fig. 4). There are three differences between the two models :

- The structural view grows with a new component and new connections between all components.
- The producers and consumer behaviours change in terms of communication protocol. The new communication principle implements hardware signals in the computational model which becomes more concrete.
- The initial and generated descriptions are semantically different in term of communication principle with the FIFO queue insertion.

3 Abstract model of communicating state machines

This section provides an abstract syntactic model definition of the considered CFSMD. This model is presented by inference rules which are described with

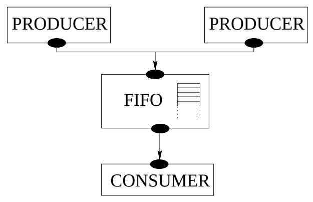

Fig. 4. Example after one step of communication synthesis

Backus-Naur syntax style. A system is a set of components communicating through signals. The behaviours of these components are finite state machines. Let $\mathcal{M} = \{M_0, M_1, \ldots, M_{n-1}\}$ be a model composed of $n = |\mathcal{M}|$ components. A signal e, used for the communication between components of the system, forms a part of the global signal set $\mathcal{E}$.

A signal communication is a non-blocking communication. Two components connected with a hardware signal are communicating by using only two actions :

- a new value can be written on the signal, which contains only one value at a time ;
- the current signal value can be read.

Hence to implement a more complicated communication protocol, many signals and many series of actions (read or write actions) are necessary. The blocking communication is implemented with loop on a state until the expected value is written by another component.

Each component M_i is a tuple $(\pi, V, E, S, \lambda, \delta)$ where :

- π is the component name (identifier) ;
- V is a local variable definition list. A variable, denoted by v (identifier), has an initial value vi ;
- E is a signal definition list. Each signal $e \in E$ is used for communication between M and other components of the system. Furthermore, we have $E \subset \mathcal{E}$;
- S is the set of states ($S = \{s_0, \ldots, s_{|S|-1}\}$). Let s be a state, and s_0 be the initial state ;
- λ is the "state-action" function ;
- δ is the transition function.

With each variable v (respectively signal e) is associated its type $t(v)$ (respectively $t(e)$). The two functions (state-action and transition function) are presented by an abstract syntax. The set of terminal symbols is composed of c for a constant value, v for a variable identifier and e for a signal identifier. And the set of nonterminal symbols is composed of ex for the expressions, ai for the internal actions (associated to a state), a for the action associated to a transition in the model, and δ_a for the transition function definition.

– **Expressions** (ex)

$$ex ::= c \mid v \mid e \mid uop\, ex_0 \mid ex_0\, bop\, ex_1 \tag{3.1}$$

The *uop* and *bop* are unary and binary operators.

– **Internal actions** (ai)

$$ai ::= \varepsilon \mid v := ex\ ;\ ai_0 \mid if\ ex\ then\ ai_0\ else\ ai_1 \tag{3.2}$$

- ε corresponds to no internal action ;
- ; is the sequential operator ;
- $v := ex$ is the assignment of a variable v with the value ex ;
- *if* is a conditional statement.

We denote by $\mathcal{A}i$ the set of all internal actions. The λ function of the machine M is defined by $\lambda : S \to \mathcal{A}i$.

– **Actions associated to transitions** (a)

$$a ::= \varepsilon \mid a_0\ ;\ a_1 \mid e := ex \mid if\ ex\ then\ a_0\ [else\ a_1] \tag{3.3}$$

- in action associated to transitions, assignment operation := is only applied to signals ;
- in the *if* statement, the expression ex is a boolean expression.

– **Transition expressions** (δ_a)

$$\delta_a ::= (s, a) \mid if\ ex\ then\ \delta_a^0\ [else\ \delta_a^1] \tag{3.4}$$

We denote by $\varDelta_a$ the set of all transition expressions. The function δ is defined by $\delta : S \to \varDelta_a$.

The figure 5 shows a part of consumer behaviour presented in figure 3 with the finite state machine model.

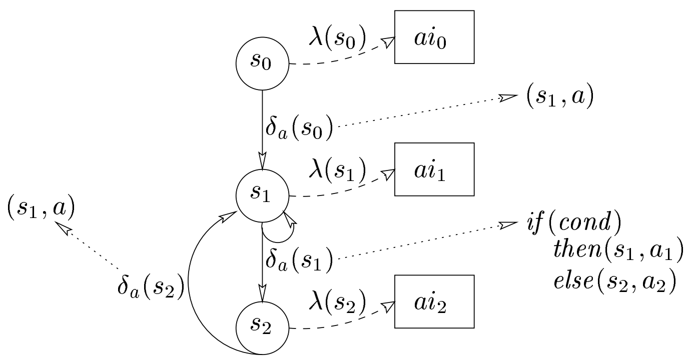

Fig. 5. Statemachine example on behaviour example

As example, one producer component (producer1) is defined in our syntactic model by $M = (producer1, V_{p1}, E_{p1}, S_{p1}, \lambda_{p1}, \delta_{p1})$ with $V_{p1} = \{\}$, $E_{p1} = \{bus\_req, wr\_req, wr\_ack, data\}$ and $S_{p1} = \{s_0, s_1, s_2, s_3, s_4, s_5\}$. In order to illustrate clearly the translation procedure, only two states s_1 and s_2 are detailed. For these two states, the functions λ_{p1} and δ_{p1} are defined in box example 1.

$$\boxed{\begin{array}{ll}
\lambda_{p1}(s_1) = ai_1 & \text{with } ai_1 = \varepsilon \\
\lambda_{p1}(s_2) = ai_2 & \text{with } ai_2 = \varepsilon \\
\delta_{p1}(s_1) = \delta_{a_1} & \text{with } \begin{cases} \delta_{a_1} = (s_2, a_1) \\ \text{and } a_1 = bus\_req := true \end{cases} \\
\delta_{p1}(s_2) = \delta_{a_2} & \text{with } \begin{cases} \delta_{a_2} = if\ (wr\_ack = true)\ then\ \delta_{a'_2}\ else\ \delta_{a'_{2b}} \\ \text{and } \delta_{a'_2} = (s_2, \varepsilon) \\ \text{and } \delta_{a'_{2b}} = (s_3, data := 0; wr\_req := true) \end{cases}
\end{array}}$$

EXAMPLE **1.** CFSMD example

4 Translation rules

LOTOS is a high level specification language based on algebraic models CSP (Communicating Sequential Processes) and CCS (Calculus of Communicating Systems). A LOTOS model of a system is composed of interconnected processes via gates. Each process communicates through gates with rendezvous communication protocol. For instance, if we consider a gate G, a general rendezvous in LOTOS is written by $G\ O_0 \dots O_n$ where $O_0 \dots O_n$ are offers defined by $O ::= !V \ \mid ?X_0, \dots X_n : S$. One offer like $!V$ is the V value emission on gate G, and one offer like $?X_0, \dots X_n : S$ is $n+1$ receptions of values of type S on gate G.

The translated model of CFSMD is composed of interconnected processes. With each state machine is associated one LOTOS process. This section presents :

– the structure of the LOTOS generated model and the communication principles between these processes ;
– the translation of the internal behaviour of CFSMD into the LOTOS process behaviour ;

4.1 Structure and communication principles

In order to reproduce the semantic of hardware signal in our communicating finite state machine, one LOTOS process named *signal* is introduced between the LOTOS processes for each hardware signal. In figure 6, the structural view of the generated LOTOS model of our example is presented.

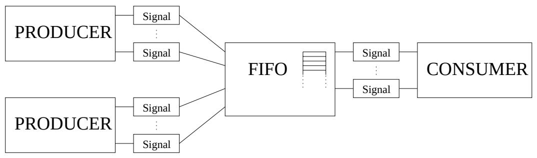

Fig. 6. Structural view of LOTOS generated model

The behaviour of the signal process, which is defined in a LOTOS library, is shown below in the example 2. It reproduces the semantic of the hardware signal communication. A signal is a physical link which take only one value at a time. A component can write a new value, or read the current value. The LOTOS process is based on a choice statement ([]). In the read signal action the current value is one offer of the synchronization. In the write signal action, the new value is received and memorized. Hence this is a description of a non-blocking communication with the rendezvous communication principle.

```
process signal[s](value : signaltype) : noexit :=
    (s!cread!value ; signal[s](value))
    []
    (s!cwrite?v : signaltype ; signal[s](v))
endproc (* signal *)
```

EXAMPLE **2.** LOTOS description of a signal process

4.2 Translation of internal behaviour of CFSMD

Each CFSMD is translated into one process. The state variable of the state machine is defined as a LOTOS process parameter. The transition between two states is reproduced with a final recursive call of the process, with the new state value in the state parameter.

The translation of the state machine behaviour is described by a set of inference rules. Only a commented subset of rules is presented in this paper. However, this section presents some significant rules, in order to give a precise idea of our translation method[1]. After three basic definitions, some notations and environments are defined. Then the global definition of the inference rules is described for different syntactic parts of the model.

4.2.1 Basic definitions

[1] It is possible to contact the authors to obtain the global set of rules

Definition 1 (Inference rule). *Let* $\frac{conditions}{a \to b}$ *be a rule where conditions* $= c_1, c_2, \ldots, c_n$ *must be satisfied to validate the transformation rule of a into b.*

$\square$

Definition 2 (Partially defined functions). *Let* D_1 *and* D_2 *be two discrete domains. We consider the function f defined on these domains :*

$$f : D_1 \to D_2$$

We define :

- $\perp$ *the undefined value.* $x \in D_1$; $f(x) = \perp$ *means that* f *is undefined for the value* x *;*
- *let* $x \in D_1$ *and* $y \in D_2$*, we denote by* $[y/x]$ *the function* f *defined only for the value* x *and such that* $[y/x](x) = y$ *:*

$$\forall x_i \in D_1 \;\; [y/x](x_i) = \begin{cases} y & \text{if } x_i = x \\ \perp & \text{otherwise} \end{cases}$$

$\square$

Definition 3 (function increase). *Let* f *and* g *be two functions defined on domains* D_1 *and* D_2*. We denote by* $f \triangleleft g$ *the function defined by :*

$$f \triangleleft g : D_1 \to D_2$$

$$\forall x \in D_1 \;\; (f \triangleleft g)(x) = \begin{cases} g(x) & \text{if } g(x) \neq \perp \\ f(x) & \text{otherwise} \end{cases}$$

$\square$

4.2.2 Environment definitions

Let LC be a LOTOS construction, and $LC = \perp$ the empty LOTOS construction. The set of all LOTOS constructions which can be written is denoted by $\mathcal{LC}$. Let Id be the set of all identifiers. Two environments are defined :

- $\alpha = (\pi, id_s, E, V)$ is a tuple constructed for each component of the system. Let π be the component name, id_s the state variable name of the component behaviour, E and V the signal and variable sets. The LOTOS description of one component consists of one recursive process, with parameters like the state variable name, component variables and gates for the signal communication. The environment α is used in order to translate the recursive call of this LOTOS process ;
- β is a second environment used to translate the signals contained in the expressions. The function β associates with each signal a local variable name to contain the signal value :

$$\begin{aligned} \beta : Id &\to Id \\ e &\to v \end{aligned}$$

Let Env_β be the set of all possible environments β. In order to translate the signal communications, we define the function L used to generate a LOTOS construct for all signals used in β.

$$\begin{array}{c} L : Env_\beta \to \mathcal{LC} \\ \beta \qquad \to LC \\ \text{with } \forall e_i \text{ such that } \beta(e_i) = v_i \neq \perp \;\; LC_i = e_i!cread?v_i : t_i \\ \text{and } LC = LC_0; LC_1; \ldots \end{array} \tag{4.1}$$

4.2.3 Global definition of transformation rules

Five rule types are necessary for the translation. They correspond to the expressions translation, internal actions translation, the action translation, the transition translation and finally one component and the whole model translation.

- let $\beta \vdash ex \longrightarrow \langle \beta', LC \rangle$ be the rule type for the expression ex translation. In environment β the expression ex is translated into LOTOS construction LC and returns a micro environment β' which contains the variable names associated to signals used in the expression ;
- let $(\alpha, \delta_a, \beta) \vdash ai \longrightarrow \langle \beta', LC \rangle$ be the rule type for the internal action ai translation. In the couple of environment α, β, and considering the δ_a transition expression, the internal action ai is translated into LOTOS construction LC and returns the micro environment β' ;
- let $\beta \vdash a \longrightarrow \langle \beta', LC \rangle$ be the rule type for the action a translation. In environment β, action a is translated into LOTOS construction LC and returns the micro environment β' ;
- let $(\alpha, \beta) \vdash \delta_a \longrightarrow \langle \beta', LC \rangle$ be the rule type for the δ_a transition translation. In the couple of environment α, β, the transition expression δ_a is translated into the LOTOS expression LC and returns a micro environment β' ;
- let $M \longrightarrow LC$ and $\mathcal{M} \longrightarrow LC$ be the rules for component and model translation.

4.2.4 Inference rules for the translation

Signal identifier in expressions ex **:** according to the environment β, the translation of a signal identifier e is defined with the help of two rules. If the signal has been used before, we just have to reuse its associated local variable. Else we assume that the function "newid" gives a new variable identifier for the signal e, and a new environment is constructed.

$$\frac{\beta(e) = ve}{\beta \vdash e \longrightarrow \langle \perp, ve \rangle} \tag{4.2a}$$

$$\frac{\beta(e) = \perp \; , \;\; ve = newid()}{\beta \vdash e \longrightarrow \langle [ve/e], ve \rangle} \tag{4.2b}$$

Binary operator in expressions : assuming that *bop* operator is declared for LOTOS language, a constructed environment for the binary operator in expressions with the addition of environments is defined as follows :

$$\frac{\beta \vdash ex_0 \longrightarrow \langle \beta', LC_0 \rangle, \quad \beta \triangleleft \beta' \vdash ex_1 \longrightarrow \langle \beta'', LC_1 \rangle}{\beta \vdash ex_0 \; bop \; ex_1 \longrightarrow \langle \beta' \triangleleft \beta'', LC_0 \; bop \; LC_1 \rangle} \tag{4.3}$$

Variable assignment for internal actions ai must be translated with the **let** LOTOS operator such that :

$$\frac{\beta \vdash ex \longrightarrow \langle \beta', LC_1 \rangle, \quad (\alpha, \delta_a, \beta \triangleleft \beta') \vdash ai_0 \longrightarrow \langle \beta'', LC_2 \rangle}{(\alpha, \delta_a, \beta) \vdash v := ex \; ; \; ai_0 \longrightarrow \langle \beta' \triangleleft \beta'', (\textbf{let } v : t = LC_1 \textbf{ in } (LC_2)\;) \rangle} \tag{4.4}$$

Signal assignment for actions a are translated into a LOTOS communication on gate denoted by e. This communication is prefixed by the word *cwrite* defined in the LOTOS model. It means that this is an assignment on the signal. The LOTOS communication is a rendezvous communication. In order to reproduce the signal semantic communication, this rendezvous is not implemented directly between the signal interconnected components in the model, but with a "signal LOTOS component" (see rule 4.9) :

$$\frac{\beta \vdash ex \longrightarrow \langle \beta', LC \rangle}{\beta \vdash e := ex \longrightarrow \langle \beta', e!cwrite!(LC) \rangle} \tag{4.5}$$

For instance, in example 1, the a_1 signal assignment on *bus_req* signal in δ_{a_1} action is translated with this rules and gives the LOTOS expression :
$bus\_req!cwrite!true$.

Conditional statement for transition function δ_a **:** three rules are required to translate the conditional statement. The first one has a restrictive condition such that it is applied when no else condition is present, and when the condition is dependent only on one signal. Then a LOTOS communication is derived with a predicate corresponding to the condition. The second rule is applied when the condition depends on more than one signal. In this case, it is not possible to create a LOTOS communication directly, thus a guarded LOTOS statement is used. The third rule is like the second one, with a else statement and the use of the L function defined in 4.1 to generate the LOTOS synchronization operator :

$$\frac{\beta \vdash ex \longrightarrow \langle \beta'', LC_1 \rangle \;,\; \beta'' = [v/s], \quad (\alpha, \beta \triangleleft \beta'') \vdash \delta_a^0 \longrightarrow \langle \beta', LC_0 \rangle}{(\alpha, \beta) \vdash if \; ex \; then \; \delta_a^0 \longrightarrow \langle \beta', s!cread?v : \tau[LC_1]; LC_0 \rangle} \tag{4.6a}$$

$$\frac{\beta \vdash ex \longrightarrow \langle \beta'', LC_1 \rangle \;,\; \beta'' \neq [v/s], \quad (\alpha, \beta \triangleleft \beta'') \vdash \delta_a^0 \longrightarrow \langle \beta', LC_0 \rangle}{(\alpha, \beta) \vdash if \; ex \; then \; \delta_a^0 \longrightarrow \langle \beta' \triangleleft \beta'', ([LC_1] \rightarrow (LC_0)) \rangle} \tag{4.6b}$$

$$\frac{\begin{array}{c}\beta \vdash ex \longrightarrow \langle \beta''', LC_2 \rangle, \\ (\alpha, \beta \triangleleft \beta''') \vdash \delta_a^0 \longrightarrow \langle \beta', LC_0 \rangle \ , \ LC_0' = L(\beta'), \\ (\alpha, \beta \triangleleft \beta''') \vdash \delta_a^1 \longrightarrow \langle \beta'', LC_1 \rangle \ , \ LC_1' = L(\beta'')\end{array}}{(\alpha, \beta) \vdash if\ ex\ then\ \delta_a^0\ else\ \delta_a^1 \longrightarrow \left\langle \beta''', \begin{array}{l}[LC_2] \rightarrow (LC_0' \ ; \ LC_0) \\ [\,]\,[\mathbf{not}(LC_2)] \rightarrow (LC_1' \ ; \ LC_1))\end{array} \right\rangle} \tag{4.6c}$$

For instance, the rule 4.6c is used to translate the condition statement of δ_{a_2} in the model example 1. In the LOTOS generated expression presented below, the value of *wr_ack* is saved in a LOTOS variable v defined with the 4.2b rule :

$$\begin{array}{ll}([v] \rightarrow & producer1[\ldots](s_2) \\ [\,]\,[not(v)] \rightarrow & (LC_1'; producer1[\ldots](s_3)))\end{array}$$

Next state, action in transition function δ_a **:** this rule generates a recursive call for the process π, with the new values for all variables and the next state of the component :

$$\frac{\begin{array}{c}\beta \vdash a \longrightarrow \langle \beta', LC \rangle \ , \ \alpha = (\pi, id_s, E, V), \\ E = \{e_0, \ldots, e_{|E|-1}\} \quad V = \{v_0, \ldots, v_{|V|-1}\}\end{array}}{(\alpha, \beta) \vdash (s, a) \longrightarrow \langle \beta', LC; \pi[e_0, \ldots, e_{|E|-1}](id_s, v_0, \ldots, v_{|V|-1}) \rangle} \tag{4.7}$$

For instance, this rule is used to generate the recursive call of LOTOS process in δ_{a_1}, $\delta_{a_2'}$ and $\delta_{a_{2b}'}$ actions in the model example 1.

Component M **:** a LOTOS process construction is defined for one component. The process gates are derived from the signal set E such that one signal corresponds to one gate. A variable in the component implies a parameter in the LOTOS process. For each state of component M, the following LOTOS construct is used in a choice statement based on the sate variable value of the process π. :

$$\frac{\begin{array}{c} V \longrightarrow LC_1 \ , \ E \longrightarrow LC_2, \\ \forall s_i \in S \quad ai^i = \lambda(s_i) \ , \ \delta_a^i = \delta(s_i) \ , \ ((\pi_{tr}, id_s, E, V), \delta_a^i, \bot) \vdash ai^i \longrightarrow \langle \beta_i, LC_i' \rangle \\ \forall \beta_i \ LC_i'' = L(\beta_i) \end{array}}{(\pi, V, E, S, \lambda, \delta) \longrightarrow \begin{array}{l} \mathbf{process}\ \pi\, [LC_2](id_s : state, LC_1) : \mathbf{noexit} := \\ \quad [id_s\ eq\ s_0] \rightarrow LC_0''\ ;\ LC_0' \\ \quad \ldots \\ \quad [\,]\,[id_s\ eq\ s_{|S|-1}] \rightarrow LC_{|S|-1}''\ ;\ LC_{|S|-1}' \\ \mathbf{endproc} \end{array}} \tag{4.8}$$

This rule can be used to generate the whole process statement, with the gate parameters derived from E_{p1}, the parameters obtained from the state variable and V_{p1}, and with all the LOTOS constructions for all the states in S_{p1}. The global LOTOS statement for this component is :

```
process $producer1\ [bus\_req, wr\_req, wr\_ack, data]$
                    $(id_s : state)$ : noexit :=
  $[id_s\ eq\ s_0] \rightarrow \ldots$
  $[]\ [id_s\ eq\ s_1] \rightarrow ($ $bus\_req!cwrite!true$ ;
                    $producer1[bus\_req, \ldots, data](s_2))$
  $[]\ [id_s\ eq\ s_2] \rightarrow ($ $wr\_ack!cread?v : bool$ ;
                    $([v] \rightarrow$        $producer1[\ldots](s_2)$
                    $[]\ [not(v)] \rightarrow (data!cwrite!0$ of $int$ ; $wr\_req!cwrite!true$ ;
                                   $producer1[\ldots](s_3))))$
  ...
  $[]\ [id_s\ eq\ s_5] \rightarrow \ldots$
endproc
```

Example **3.** Lotos description of producer process

System $\mathcal{M}$: whole the system is described in a Lotos specification and a library (signallib) which contains signal process definition. The specification is made up of all the instantiations of the processes associated to the components and a Lotos synchronization to the signal process instantiation. *initvalue* is a function which associates an initial value at each variable.

$$\frac{\begin{array}{l}\mathcal{M} = \{M_i | 0 \le i < n = |\mathcal{M}|\},\\ \forall i \in [0 \ldots n-1] \begin{cases} M_i = (\pi_i, V_i, E_i, S_i, \lambda_i, \delta_i),\\ M_i \longrightarrow LC_i,\\ V_i = \{v^i_j\} \quad \forall j \in [0 \ldots |V_i| - 1]\ initvalue(v^i_j) = vi^i_j,\\ E_i = \{e^i_0, \ldots, e^i_{|E_i|-1}\}\end{cases}\\ \mathcal{E} = \cup_{i=0}^{i=|\mathcal{E}|-1} E_i\ ,\ \mathcal{E} = \{e_0, \ldots, e_{|\mathcal{E}|-1}\}\end{array}}{\begin{array}{ll} & \textbf{specification}\ \pi_s\ \left[e_0, \ldots, e_{|\mathcal{E}|-1}\right]\ :\ \textbf{noexit}\\ & \quad \textbf{library}\ \textit{signallib}\ \textbf{endlib}\\ & \quad \textbf{behaviour}\\ & \qquad (\pi_0 \left[e^0_0, \ldots, e^0_{|E_0|-1}\right] (s^0_0, vi^0_0, \ldots, vi^0_{|V_0|-1})\ |||\\ & \qquad \ldots\\ \mathcal{M} \longrightarrow & \qquad \pi_{n-1} \left[e^{n-1}_0, \ldots, e^{n-1}_{|E_{n-1}|-1}\right] (s^{n-1}_0, vi^{n-1}_0, \ldots, vi^{n-1}_{|V_{n-1}|-1}))\\ & \qquad |[e_0, \ldots, e_{|\mathcal{E}|-1}]|\\ & \qquad (signal[e_0](Z)\ |||\\ & \qquad \ldots\\ & \qquad signal[e_{|\mathcal{E}|}](Z))\\ & \quad \textbf{where}\\ & \qquad LC_0\ LC_1\ \ldots\ LC_{n-1}\\ & \textbf{endspec}\end{array}} \tag{4.9}$$

The simple system shown in figure 4 has been translated in Lotos by applying this rule. The generated Lotos description has about 400 lines. It contains some processes : two producers, one consumer, the FIFO queue and some signals components. The structural view of the LOTOS description is given in figure 6. A part of the global LOTOS statement for this system is in example 4

```
specification ProdCons[rd_req, rd_ack, bus_req1, wr_req1, wr_ack1,
        bus_req2, wr_req2, wr_ack2, data] : noexit
  ...
behaviour
  (
    FIFO2[rd_req, rd_ack, bus_req1, wr_req1, wr_ack1,
        bus_req2, wr_req2, wr_ack2, data](q0, nil, 2ofint) |||
    P1[bus_req1, wr_req1, wr_ack1, data](q0) |||
    P2[bus_req2, wr_req2, wr_ack2, data](q0) |||
    C[rd_req, rd_ack, data](q0, 0ofint, 0ofint, 0ofint)
  )
  | [rd_req, rd_ack, bus_req1, wr_req1, wr_ack1,
        bus_req2, wr_req2, wr_ack2, data] |
  (
    signal[rd_req](zvalue) |||
    signal[rd_ack](zvalue) |||
    signal[bus_req1](zvalue) |||
    signal[wr_req1](zvalue) |||
    signal[wr_ack1](zvalue) |||
    signal[bus_req2](zvalue) |||
    signal[wr_req2](zvalue) |||
    signal[wr_ack2](zvalue) |||
    signal_int[data](0ofint)
  )
  where
    ...
endspec
```

EXAMPLE **4.** LOTOS description of producer process

5 Deadlock free property verification

The CADP toolbox is used for deadlock free property verification. According to the operational semantics of LOTOS, the LOTOS system specification is translated into a (possibly infinite) Labelled Transition System (LTS for short), which encodes all its possible execution sequences [SM98]. Only finite LTS can be generated with the CADP tool. An LTS is formally defined by :

Definition 4 (LTS). *Let* $L = \langle Q, A, T, q_{init} \rangle$ *be a* LTS *such that :*

– *$\mathcal{Q}$ is the set of states of the program ;*
– *A is a set of actions performed by the program. An action $a \in A$ is a tuple $G\,V_1, \ldots V_n$ where G is a gate, and $V_1, \ldots V_n$ are the values exchanged (i.e., sent or received) during the rendezvous at G ;*
– *$T \subseteq \mathcal{Q} \times A \times \mathcal{Q}$ is the transition relation. A transition $\langle q_1, a, q_2\rangle \in T$ (written also $q_1 \xrightarrow{a} q_2$) means that the program can move from state q_1 to state q_2 by performing action a;*
– *$q_{init} \in \mathcal{Q}$ is the initial state of the program.*

□

For each state $q \in \mathcal{Q}$, we denote by $Pathd(q)$ the set of all distinct paths $q(= q_0) \xrightarrow{a_1} q_1 \xrightarrow{a_1} q_2 \ldots$ issued from q (such that $\forall i, j\; q_i \neq q_j$).

Typically, when an expert designs a LOTOS specification, the graph is analyzed by searching deadlocks which appear as states q in the LTS such that $\nexists\langle q, a, q'\rangle$. No more communication can be done in the whole system if the behaviour reaches this sink state q. This technique is efficient when two conditions are satisfied :

– the specification is written assuming this search of deadlocks. In other words, it contains “true” blocking communications with the rendezvous semantics ;
– the deadlocks found are global in the system, meaning that no more communication can be done in the **whole** system. With this technique, *local deadlocks* in some processes are not detected. In a state, if one or more processes never have communication, it is possible that they are waiting for specific signal values. However other processes in the system continue their communications. This is our local deadlock definition.

In our LTS, a transition corresponds to one signal utilization. A signal utilization can be a reading (labelled cread) or writing (labelled cwrite) task. The signal processes introduced in the translated specification are designed in order to respect the signal semantics. Hence, sink states do not appear in our LTS. Furthermore, local deadlocks detection is an important issue in the context of systems derived from Codesign design.

Considering these aspects, correctness properties can be expressed with formulas inspired from ACTL temporal logic, and verified on the LTS model using the XTL model-checker [Mat98]. First, some notations (described in [SM98]) are presented, and then our deadlock correctness property are discussed.

5.1 Preliminary notations

Definition 5 extracted from [SM98] is presented for comprehension.

Definition 5 (Action Formulas). *Let α be an action formula as specified by the following context-free grammar :*

$$\begin{array}{rl} \alpha ::= & true \\ | & \{G\,V_1, \ldots V_n\} \\ | & \neg\alpha \\ | & \alpha \wedge \alpha' \end{array}$$

where $\{G\,V_1, \ldots V_n\}$ denotes an "action pattern", G a gate and all the values V_i match with the corresponding values exchanged when the action is performed.

*An action formula α is interpreted over an action $a \in A$. α satisfaction by an action a of the model (*LTS*) L, written with $a \models_L \alpha$ (or simply $a \models \alpha$ when the model L is understood), is defined by :*

$$\begin{array}{ll} a \models \mathit{true} & \mathit{always}; \\ a \models \{G\,V_1, \ldots V_n\} & \mathit{iff}\, a = G\,V_1, \ldots V_n; \\ a \models \neg\alpha & \mathit{iff}\ a \not\models \alpha; \\ a \models \alpha \wedge \alpha' & \mathit{iff}\ a \models \alpha\ \mathit{and}\ a \models \alpha'. \end{array}$$

□

The satisfaction of a formula φ by a state $q \in \mathcal{Q}$ of a LTS L is written with $q \models_L \varphi$ (or simply $q \models \varphi$ when the model L is understood).

5.2 The deadlock free property

In order to clearly present our verification, we introduce some formulas and their semantics. First, consider a process which is waiting for a specific value of a signal. The signal is read until it takes the expected value. This classical behaviour induces in the generated LTS some state like q in figure 7.

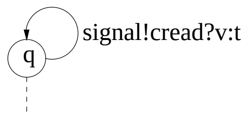

Fig. 7. One loop on a state

The φ formula EB_α is defined by the equation 5.1. It detects the loop on a state, with a specific label α. The global deadlocks in our system do not appear as sink states, the AB_α formula (equation 5.2) can be used to characterize a global deadlock on a state by evaluating $q \models AB_{true}$. This formula is almost the same as EB_α, with a *forall* quantifier.

$$q \models EB_\alpha \text{ iff } \exists q \xrightarrow{a} q' \text{ such that } q' = q \text{ and } a \models \alpha \qquad (5.1)$$

$$q \models AB_\alpha \text{ iff } \forall q \xrightarrow{a} q',\ q' = q \text{ and } a \models \alpha \qquad (5.2)$$

The XTL implementation of the EB_α formula is given as follows :

```
def EB(LS:labelset) : stateset =
    { S : state where
        exists T : edge among out(S) in
            ((target(T)=S) and (label(T) among LS))
        end_exists
    }
end_def
```

In order to detect local deadlocks, we define a φ formula F_α by the equation 5.3. A state q satisfies F_α if and only if all the reachable states from q satisfy EB_α. The second condition in 5.3 verifies that the transitions between two distinct reachable states have actions not satisfying α. This is not useful in our translated model because this is a deterministic model, and this condition is always true for a state q of a deterministic model which satisfy the first part of F_α. Figure 8 illustrates the equation 5.3

$$\begin{gathered} q \models F_\alpha \text{ iff } \forall P = (q \xrightarrow{a_0} q_1 \xrightarrow{a_1} \dots \xrightarrow{a_{k-1}} q_k) \in Pathd(q), \\ \forall i \in [0;k]\, q_i \models EB_\alpha \text{ and } \forall i \in [0;k-1]\, a_i \not\models \alpha \end{gathered} \tag{5.3}$$

Hence, if $\nexists q \in \mathcal{Q}$ such that $\exists \alpha \in A$ such that $q \models F_\alpha$, then the model does not contain any local deadlock.

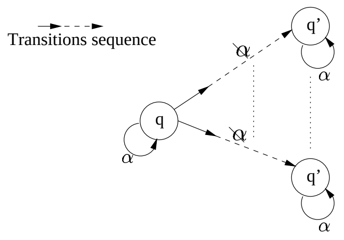

Fig. 8. Temporal formula illustration

Let function *succset* be a transitive closure of the successor relation, which can be achieved with a least fixed point function. Assuming that we have implemented the *succset* function in XTL, the φ formula F_α is defined with :

```
def F(LS:labelset) : stateset =
    let Bs : stateset = EB(LS) in
        { S : state among Bs where Bs includes succset(S) }
    end_let
end_def
```

This formula must be evaluated with all labels contained in A. These labels can be automatically obtained with the analysis of the communication in the first model. In the translation, only the labels used in the XTL formulas are generated. The verification with this technique is thus a push button like function.

6 Application on COSMOS Codesign tool

6.1 COSMOS presentation

This work has been applied in the scope of the Codesign domain. In our study we consider the COSMOS tool developed at TIMA laboratory [IJ95]. The main characteristics of the COSMOS method and tool are the following :

– the specification of the system is independent of the implementation technology of the different parts of the system. This high level of abstraction description is written in SDL (Specification and Description Language [SDL88]) ;
– the use of an intermediate format SOLAR, describing the system and the communication channels among processes (CFSMD like);
– the implementation of processes in hardware or software and the choices of communications implementation (for instance a choice of a communication protocol between two or more components) are performed by several iterations of refinement steps decided manually by the designer ;
– automatic generation of the C-VHDL virtual prototype from the completely refined SOLAR description of the system ;
– cosimulation environment of the virtual prototype.

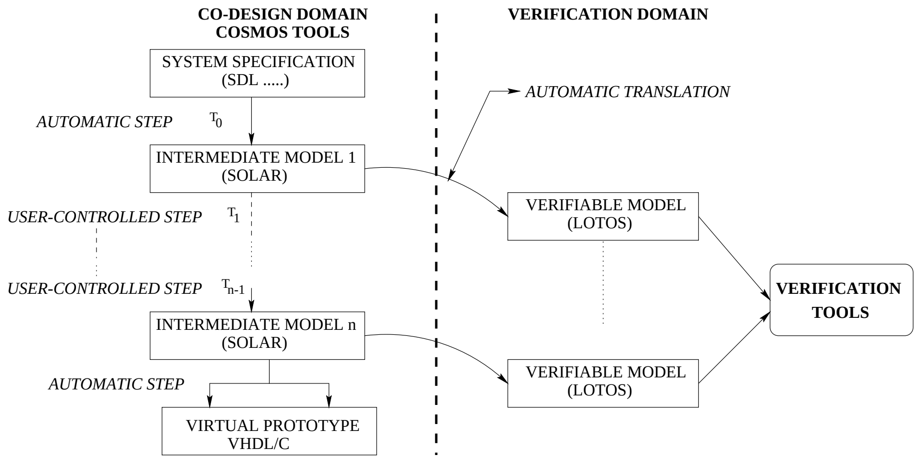

Fig. 9. Linking Codesign and Verification Domains

The design flow of the COSMOS tool is a sequence of refinement steps. At each step, a decision is taken by the designer, and the tool automatically integrates this decision by transforming the SOLAR description of the system. Typically this decision can be a communication synthesis among two components.

In such tools, the verification process is performed either by formal verification on the entry specification level or by cosimulation at the virtual prototype level [VCR+95,LNV+97]. But, as the refinement is decided by the designer, and as the choice concerns communication synthesis, deadlocks can be introduced inadvertently in the system at each refinement step. The detection of such deadlocks is performed at the virtual prototype level. However this task is difficult and uncertain because :

– deadlocks generally induce active loops in the generated model,
– the link between the generated code and the initial description is not easy to establish,
– there can be several errors in the design at the virtual prototype level, yielding several decisions to modify, but these are difficult to identify,
– the virtual prototype describes the system at a low level of abstraction, the description is thus complex.

Our work on the verification of CFSMD can be used to verify the SOLAR description at each step of Codesign (Fig. 9). Our tool architecture is completed with a front end analyzer of SOLAR description like shown in figure 10.

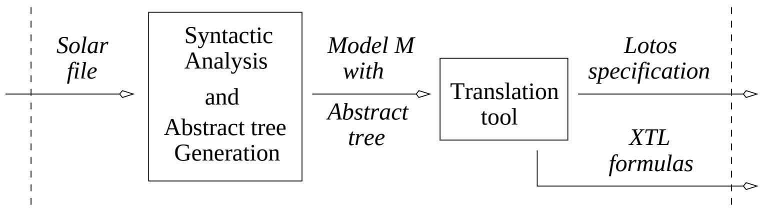
Solar file
Syntactic Analysis and Abstract tree Generation
Model M with Abstract tree
Translation tool
Lotos specification
XTL formulas

Fig. 10. Tool architecture

6.2 Example results

Considering the consumer behaviour and the protocol communication choiced, a deadlock appears in the system. This occurs when a non expected data is in front of the FIFO queue. Table 5 presents the results of the verification tools on the example. The graph generated by the CADP tool is minimized modulo strong bisimulation. The time to compute the graph is calculated on a SUN ULTRA 30. The last column presents the number of states satisfying the XTL formula F_α, with $\alpha = (wr\_ack2!cread!false)$. The signal wr_ack2 is an acknowledgement for the communication between the FIFO queue and the second producer. The F_α property verifies the correctness of the protocol in all execution cases.

On the reduced graphs, the XTL AB_{true} property gives us one global deadlock. But, if the system is considered in an environment of other communications, this type of sink state disappears. In fact, no more communication is done between the producers, consumer and FIFO queue, but communications are made between the environment processes. This is a local deadlock. Hence the XTL F_α formula is performed on the LOTOS program to successfully find this local deadlock.

Description	Reduced graph		Time to compute	Number of states satisfying Ba_{true}	Number of states satisfying F_α
	states	trans.			
FIFO size 1	322	1288	07'	1	97
FIFO size 2	652	2608	13'	1	179
FIFO size 2 with environment	1304	6520	27'	0	358

EXAMPLE **5.** CADP graph generation - XTL verification

7 Conclusion

In this paper, we have proposed an approach of verification of communicating finite state machines with datapath (CFSMD). This abstract model of computation with a communication principle based on hardware signal has been translated into an equivalent LOTOS description in which the communication basic mechanism is the rendezvous. Model checking verification techniques are applied on the system in order to verify deadlock property.

Then, by using this translation, we propose an approach to link the Codesign tool COSMOS with the CADP validation/verification toolbox and the XTL model-checker. COSMOS is based on refinements of the system and verification is needed when the designer chooses the implementation of communications. We intend to implement the verification as a push button function of the system. The results show the usefulness and efficiency of our deadlock verification with temporal logical formulas.

In order to apply this work with other tools, the extension of the CFSMD model to different models of communication is to study. Future work will focus on some abstractions of the communications. The goal is to study the influences of abstractions on the size of the generated LTS, and on the deadlocks search. Furthermore, we will work on larger case studies, in order to examine the complexity limits of this approach.

The study of other kind of properties which could be verified on such system will lead to a more powerful tool. Perhaps it will interesting to generate XTL formula which depend on the step of communication synthesis.

A other aspect is the study of verification based on IF language currently developed at VERIMAG laboratory [BFG$^+$99]. This language integrates principles of communication which are different from the LOTOS rendezvous, and it is introduced in a complete open validation environment. Then we will work on the feasibility study of the system modeling with IF, and on the comparison between the verifications on a LOTOS generated model and the verifications on a generated IF model.

References

[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.

[ELLSV97] S. Edwards, L. Lavagno, E.A. Lee, and A. Sangiovanni-Vincentelli. Design of Embedded Systems: Formal Models, Validation, and Synthesis. In Giovanni De Micheli, editor, *Proceedings of the IEEE, Special issue on Hardware/Software Co-design*, volume 85, pages 366–390. The institute of electrical and electronics engineers, inc., March 1997.

[FGM+91] Jean-Claude Fernandez, Hubert Garavel, Laurent Mounier, Anne Rasse, Carlos Rodriguez, and Joseph Sifakis. Une boîte à outils pour la vérification de programme LOTOS. In *Actes du Colloque Francophone pour l'Ingénierie des Protocoles*, pages 479–500, September 1991.

[Gar89] H. Garavel. *Compilation et vérification de programmes LOTOS*. PhD thesis, Université Joseph Fourier, Grenoble, 1989.

[Gar98] Hubert Garavel. OPEN/CAESAR : An Open Software Architecture for Verification, Simulation and Testing. In *TACAS'98, Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science*, 1998.

[GLO91] S. Gallouzi, L. Logrippo, and A. Obaid. Le LOTOS, Théorie, Outils, Applications. In O. Rafiq, editor, *CFIP'91 - Ingénierie des Protocoles*, pages 385–404. Hermes, 1991.

[GM93] R.K. Gupta and G. de Micheli. Hardware-Software Cosynthesis for Digital Systems. *IEEE Design & Test of Computers*, 10(3):29–41, September 1993.

[GS90] Hubert Garavel and Joseph Sifakis. Compilation and Verification of LOTOS Specifications. In R.L. Probert L. Logrippo and H. Ural, editors, *10th International Symposium on Protocol Specification, Testing and Verification*, pages 379–394. IFIP, North-Holland, June 1990.

[GV95] D.D. Gajski and F. Vahid. Specification and Design of Embedded Hardware-Software Systems. *IEEE Design & Test of Computers*, 1995.

[IJ95] T.B. Ismail and A.A. Jerraya. Synthesis Steps and Design Models for Codesign. *IEEE Computer*, February 1995.

[ISO88] ISO-8807. LOTOS, a formal description technic based on the temporal ordering of observational behaviour. 1988.

[LNV+97] C. Liem, F. Nacabal, C. Valderrama, P. Paulin, and A. Jerraya. Co-simulation and Software Compilation Methodologies for the System-on-a-Chip in Multimedia. *IEEE Design & Test of Computers*, 1997. special issue on "Design, Test & ECAD in Europe".

[LSVS98] Luciano Lavagno, Alberto Sangiovanni-Vincentelli, and Ellen Sentovich. Models of computation for embedded system design. In A.A. Jerraya and J. Mermet, editors, *System-Level Synthesis*, chapter Models for system-level synthesis, pages 45–102. Kluwer Academic Publishers, 1998.

[Mat98] Radu Mateescu. *Vérification des propriétés temporelles des programmes parallèles*. PhD thesis, Institut National Politecnique de Grenoble, 1998.

[SDL88] *CCITT. Recommendation Z.100: Specification and Description Language, volume X.1-X.5*, 1988.

[SM98] M. Sighireanu and R. Mateescu. Verification of the Link Layer Protocol of the IEEE-1394 Serial Bus ("FireWire"): an Experiment with E-LOTOS. *Springer International Journal on Software Tools for Technology Transfer (STTT)*, 2(1), 1998.

[VCJ96] C.A. Valderrama, A. Changuel, and A.A. Jerraya. Virtual Prototyping For Modular And Flexible Hardware-Software Systems. *Journal of Design Automation for Embedded Systems*, 1996.

[VCR+95] C.A. Valderrama, A. Changuel, P.V. Raghavan, M. Abid, T. Ben Ismail, and A.A. Jerraya. A Unified Model for Co-simulation and Co-synhesis of Mixed Hardware/Software Systems. In *The European Design and Test Conference ED& TC'95, Paris (France)*, March 1995.

[WB99] Pierre Wodey and Fabrice Baray. Linking Codesign and verification by mean of E-LOTOS FDT. In Bob Werner, editor, *Euromicro 99, Digital Systems Design*, volume 1. IEEE Computer Society, September 1999.

[Wol94] W.H. Wolf. Hardware-Software Co-Design of Embedded Systems. *Proceedings of the IEEE*, 82(7), July 1994.