# A Process Algebra Interpretation of a Verification Oriented Overlanguage of VHDL

**C.Bayol, B.Soulas [(1)]; F.Corno, P.Prinetto [(2)]; D.Borrione [(3)]**

**(1) EDF/DER, 77250 Moret/Loing, France;**
**(2) Politecnico di Torino, Dpt Automatica e Informatica, Turin, Italy;**
**(3) J. Fourier University, IMAG/ARTEMIS Lab, Grenoble France.**

*Abstract*

*The VOVHDL language was defined as a verification oriented VHDL–based language in order to obtain a VHDL simulable specification at system level and to be able to verify this specification in a Process Algebra approach. This paper presents a formal semantic model for VOVHDL in terms of parallel composition of Labelled Transition Systems, and its implementation with the EVAL CCS–based verification tool.*

## 0 – Introduction

In the context of complex systems development, as VLSI components, it is now usually admitted to proceed by successive specification refinements. The quality of the final system is then ensured by a Verification process which allows to control the conformance of each specification step with the previous one [SGT, 92].

Thus, such a methodology comes up against the semantic gap between two specification levels : from a General Requirement File to a Detailed Design Specification, where technological constraints appear explicitly, there is necessarily one (or more) frontier hard to cross.

The System Specification file is, doubtless, the first of these borders. It creates a link, generally by contract, between customer and designer. According to the General Requirement File, it constitutes a more complete and non ambiguous translation of foreseen services, while according to design, it draws a first functional decomposition, introducing the dimensioning constraints of the chosen technology.

A typical example is the design of an ASIC component, which implements the functionality of a communication protocol. On the one hand, the General Requirement File, frequently given by a protocol standard, can be formally specified and validated according to a Process Algebra approach, on the other hand, detailed design will be supported by means of CAD tools using the standard description language VHDL. Then, in a formal verification context, which language are we going to use to describe the system level specification file, knowing that the link between VHDL and CCS is yet in research domain [DJS,93], [OC,93]?

To answer this question, an overlanguage of VHDL, VOVHDL (Verification Oriented VHDL) was defined [CCPBS, 93a] with the following properties:

– to offer more powerful communication features, dedicated to the elaboration of a specification at system level.

– to be compatible with the standard VHDL, thus enabling a traditional VHDL_based design at the lowest levels.

– to guarantee the translation of a VOVHDL specification system, into a Process Algebra model, with the view to formally verify its conformity with the General Requirement File.

The content of this paper relates the third point, showing how to express new semantic rules of VOVHDL language as behavioral models. This point illustrates more particularly how the gap between synchronous semantics can be bridged.

In a first part, we give a short summary of the VOVHDL semantics. Then, we explain its interpretation according to a CCS approach. Finally, we show how these concepts were put in concrete models, using the industrial tool EVAL, based on Labelled Predicates/Transition nets.

## I – VOVHDL Communication Semantics

### I.1 – Based on high level protocols ...

**I.1.1 – Channel Definition :** in VHDL, communication channels are not explicitly defined; the broadcasting of signals implements the links between VHDL processes.

In VOVHDL, a new specific statement allows to define a multiple channel with several senders and several receivers. The several senders of a channel must verify Composition Rules which can be : ALL (all senders have to be ready and transmit the same value at this simulation step), ONE (as soon as a sender is ready it transmits its value on the channel), SOME (in a non deterministic way, a subset of senders among those that are ready and that transmit the same value is chosen to realize the communication). In the same way, several receivers of a channel have to verify Dispatching rules such as : ALL (all receivers have to be ready to receive the message on the channel), ONE (in a non deterministic way only one receiver receives the message), SOME (all the receivers which are ready receive the value of the channel)

A channel is characterized by its name, its protocol, its transmitter set with composition rules, its receiver set with dispatching rules.

The protocol can be MULTI–RENDEZ–VOUS (MRV), CALL&RETURN (CALL), VSIG_value, VSIG_event (where VSIG means ”VHDL signal”).

**0.1.1 – Communication Protocols :** protocols play a key role in the VOVHDL communication model. We briefly introduce their semantics.

**MRV** : the communication takes place when all transmitters (according to the specified composition rule) are ready to transmit AND all the receivers (according to the specified dispatching rule) are ready to receive. The communication is supported simultaneously by all the processes. Otherwise, ready processes are blocked, waiting for the other ones.

**CALL** : the transmitter sends a message and holds, awakening the receiver, that is waiting for a message. Only when the receiver acknowledges the end of its task, the sender resumes its activity. The call&return protocol is easily interpreted in terms of two MRV communications.

**VSIG** : the transmitters (according to the specified composition rule) transmit a message on the channel which can be seen as a one–message buffer holding the most recently written value. The two following reception modes may be defined :

♦value–driven reception (VSIG_value protocol) : receivers read at any time the most recently written message.

♦event–driven reception (VSIG_event protocol) : receivers wait until a new value is written, then resume with the new value read.

**0.1.2 – Send and Receive Statements :** in order to constrain the use of the VOVHDL protocols, the signal assignment and WAIT statement are forbidden. In VOVHDL, the information transfer between two processes is explicitly declared by the use of the two following communication primitives:

♦SEND (ch1, val_out) : a module transmits the value val_out on the channel ch1;

♦RECEIVE (ch1, val_in) : the module receives the value val_in on the channel ch1.

Thus, in a VOVHDL specification, all the communication events correspond to either a SEND or a RECEIVE statement invocation.

However, the information transfer between the sender and the receiver is realized if and only if the channel properties are satisfied (composition, dispatching, and protocol rules). For instance, on a channel with the MRV protocol, the ALL composition rule and the ONE dispatching rule, the communication is possible if and only if, at the same simulation time, all the senders of the channel are ready to transmit the same value and, at least one among the receivers of the channel is ready to receive. Otherwise, the module, either a sender or a receiver, has to wait until these rules are satisfied. Note that the VSIG_value protocol never leads to a blocking state, neither for a sender nor for a receiver.

## 0.2 – ... but Synchronous Executive Semantics.

In order to interpret a VOVHDL specification according to a VHDL–like simulation view, the communication semantics of VOVHDL needs to inherit the VHDL synchronous characteristic. So, a simulation step $\delta_{VOVHDL}$ is defined as the internal time which separates two VOVHDL communication events. It is like an event driven simulation.

Like a ”blocking sampler” functioning, the unrolling of a VOVHDL program can be decomposed into three simulation parts as follows :

1– At the beginning of the simulation step, the input values of all channels are sampled and blocked.

2– Then, each process resumes its processing until its next communication event. It can be a SEND or a RECEIVE event.

3– When all processes are stabilized, the availability of information transfer is evaluated for each channel, according to the composition, dispatching and protocol rules. In successful cases, the value of the channel is propagated to the receiver for the next simulation step.

In unsuccessful cases, the concerned modules are blocked until the third part of the next simulation step, in order to evaluate again the channel rules, and so on until their satisfaction. By this way, a process may be in a waiting state during several $\delta_{VOVHDL}$.

## I – Formal Semantic Model for VHDL in terms of Labelled Transition System (LTS)

### I.1 – Preserving Trace

In order to support the double objective of VDHL–based design and CCS–based verification, as described in the introduction, we have to consider two translations. On one side, we have to translate a VOVHDL specification into a VHDL script, on the other side we interpret it into CCS semantics.

However, to guarantee the validity of the verification itself, these translations have to be performed in such a way as to preserve the conformance between the VHDL specification and the CCS model. VOVHDL internal and external communication events must be observable in theVHDL and CCS translations.

VOVHDL and VHDL have both a synchronous executive semantics. Therefore, the package feature of VHDL tool facilitates the translation from one to the other. In fact, the translation leads to decompose each VOVHDL simulation step into several VHDL simulation steps. This introduces additional internal transitions. That is why the VOVHDL protocols are modelled in a VHDL way that respects each VOVHDL simulation step (VHDL simulation steps are adjusted to VOVHDL ones). The VOVHDL event traces are preserved by the VHDL translation.

In order to obtain the same result with the CCS translation, we have to consider two aspects :

♦the interpretation of the behavior of each process, in terms of LTS model, according to the VOVHDL communication events.

♦the interpretation of the VOVHDL communication semantics in term of a behavioral module described as a LTS.

Then, two VOVHDL processes, P1 and P2 composed according to VOVHDL communication are interpreted in CCS semantics by the parallel composition of (see Figure 1):

♦LTS representation of each process

♦The VOVHDL communication behavioral model $B_{VOVHDL\_COM.}$

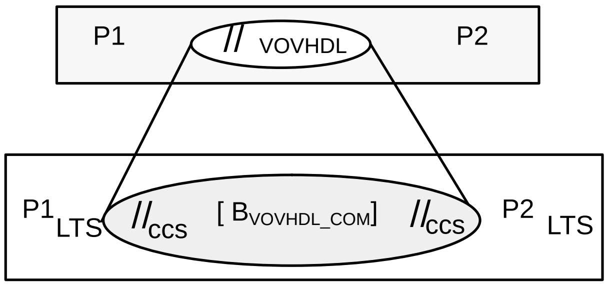

**Figure 1 : Vovhdl parallel composition into CCS**

## I.2 – LTS Interpretation of each process

**I.2.1 – A VOVHDL Process into LTS :** if we consider a VOVHDL process as a black box with input and output ports, we can observe VOVHDL communication events during the simulation of the global VOVHDL specification. Theoretically, it is possible to interpret the obtained trace in terms of a Labelled Transition System (LTS).

This LTS may be characterized by :

♦each state corresponds to the moment immediately before a communication statement,

♦each transition, labelled with the communication statement which follows the current state, corresponds to one VOVHDL simulation step.

In practice, the extraction of the LTS model from the VOVHDL specification comes up against the conditional branching structure allowed by the VHDL language. Thus, formal computing of all the possible transitions from one state to the other ones is made difficult.

That is why, a more direct solution is to use a tool such as EVAL (see below), whose description language allows to represent the conditional branching statement, and which supports an exhaustive simulator engine in order to obtain the reachable states graph.

In addition, as explained above, some VOVHDL protocols may lead to blocking situations. In order to represent them, the LTS model must be completed with a loop transition on each state from which a communication event occurs on channel with such a protocol.

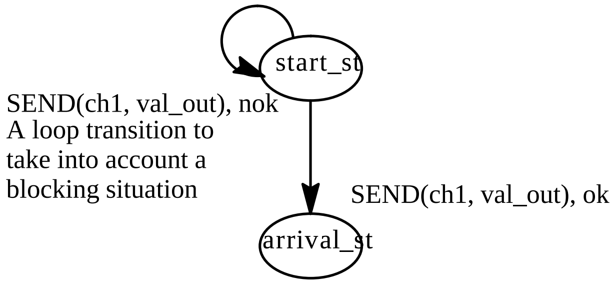

**Figure 2 : loop transition**

**I.2.2 – Synchronous Interpretation :** even without considering the communication exchange between the processes, the synchronous semantics of the VOVHDL enforce an implicit synchronization of them.

Indeed, the LTS models evolve in a synchronous way:

♦from their initial state at the same starting time,

♦by firing at the same time one transition.

If we consider the LTS model of one process separately from the global system, its behavior is non deterministic due to the loop transitions (see Figure 2). However, the modelling of VOVHDL communication semantics, and the parallel composition of all the processes should guarantee that, at each simulation step, only one of the two labelled transitions (the principal and loop ones) is enabled from each state. That is why we need to distinguish them by extending the label (for instance ok for not blocking, nok for blocking).

## I.3 – Communication Interpretation

**I.3.1 – A Fundamental property :** considering the LTS interpretation of the behavior of a VOVHDL process, a very simple but fundamental property may be brought into the fore

**At each moment, e.g. at each simulation step, each process is active on one and only one channel.**

The reason is that we consider one state before each communication event. However, it would be possible to extend the VOVHDL in order to take into account a multiple activity on several channels during one simulation step. But in this case, we must interpret this multiple exchange as a unique and atomic communication event.

Considering the synchronous evolving of all processes, the communication model has to represent their parallel composition in terms of successful or unsuccessful rendez–vous of the different activity on each channel, according to its composition, dispatching and protocol rules.

For this reason, a first approach leads naturally to model each channel as a module linked to its sender processes and its receiver processes. But the analysis of this solution, as shown below, will lead to consider an other architecture.

**I.3.2 – A communication Module to each Channel :** this approach consists in replacing each channel of a VOVHDL specification with a communication module which reflects the behavior of the specified protocol on the channel, including CR and DR rules (see Figure 3).

This communication module is composed with its sender and receiver processes according to the CCS rendez–vous.

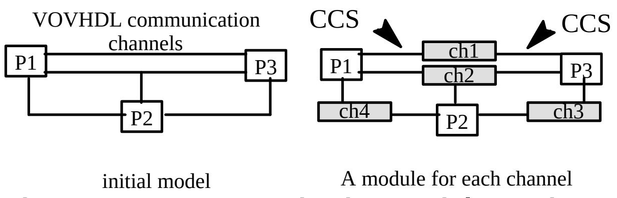

**Figure 3 : One communication module per channel**

This architecture has to be interpreted in terms of parallel composition of :

♦P1 || P3 || ch1; labels set [Send(ch1,_), Receive(ch1,_)]
♦P1 || P2 || P3 || ch2; labels set [Send(ch2,_), Receive(ch2,_)]
♦P2 || P3 || ch3; labels set [Send(ch3,_), Receive(ch3,_)]
♦P1 || P2 || ch4; labels set [Send(ch4,_), Receive(ch4,_)]

At each step, the communication module is in rendez–vous with a subset of its sender and receiver processes. According to the protocol rules, it has to manage so that processes evolve by firing their principal transition or their loop one. The channel module is in rendez–vous with an active process by means of one of its two transitions, with respect to the necessity to block it or not, at the same state.

At this point, a first problem appears due to the fact that there is at each step only a subset of active processes on a given channel. For example, if P3 is active on ch3, it can not be active on ch1, at the same time. For this reason, the behavioral model of the channel has to contain as many transitions as the number of possible subsets of the senders and receivers set, in order to avoid any deadlock.

Then a second problem appears : the communication module becomes by this way non–deterministic, due to the fact that two transitions relative to two subsets, one included in the other one, are concurrent. For instance, for ch2, it is necessary to have a transition for the case where P1 and P2 are active and an other one in case where only P1 is active : but this last transition is also enabled when P1 and P2 are active at the same time.

In order to ensure that only the transition corresponding exactly to the subset of active processes fires, it is necessary to label it by the non–activity of the other processes. This enforces that all processes are linked to each channel module. Thus each process broadcasts its communication event to all channel modules, and informs one channel of its inactivity on it by stating that it is active on an other one.

In conclusion, this analysis leads us to consider a new architecture as depicted below :

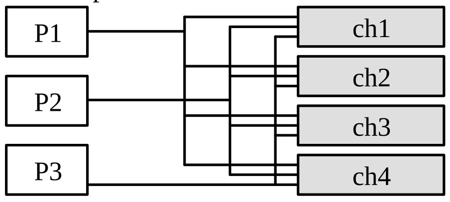

**Figure 4 : Broadcasting between processes and channels**

In this manner, one simulation step evolving is represented by a parallel composition which corresponds to the synchronization of one transition in each process and in each channel module. For example, if P1 and P2 are active on the channel ch4 and P3 on ch3, the evolution may be represented, as shown in Figure 5, by the synchronization of one transition in each process and the corresponding transition in each channel module labelled with the composition of the three communication events.

In fact we have to point out that it is the same transition in all channel module ; the single difference is that the computing of the protocol rules is distributed on each communication module supporting the characteristics of each channel.

This is the reason why we define a new communication model constituted by only ONE communication module connected to all channels.

**I.3.3 – Only ONE Communication Module for all Channels :** in this approach, we collect all channels into a unique communication module.

Each process is connected to the communication module, and transmits to it which channel it is active on (see Figure 6).

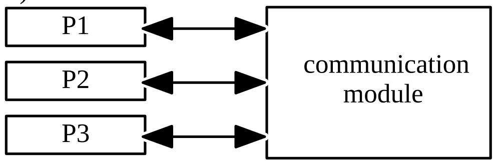

**Figure 6 : one communication module**

The communication module contains the definition of all channels, i.e. the composition rule, dispatching rules and protocol of each one. Formally in terms of the LTS model, we have to consider two points : the synchronous aspect of

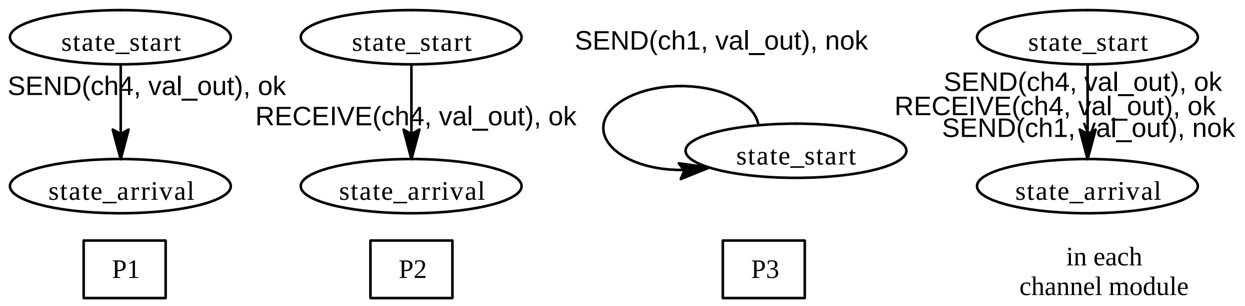

**Figure 5 : Simulation step as Transition comparison**

VOVHDL and the behavioral model of the different protocols allowed by the VOVHDL language

The first point is solved by modelling the communication module with only one state. In addition, this choice guarantees that the LTS model does not introduce extra internal transition.

The second point may be represented, for each protocol, by a set of transitions, as a loop on the unique state : each one has to represent each situation which can be met, concerning the activity or not of processes, and the corresponding blocking condition : for instance for MRV on ch1 between two modules P1 and P2. Process P1 can be ready to send a value val_out on channel ch1 (i.e. to be in a ok state). At the same time, process P2 can be ready to receive the value val_in on ch1. On the same schema, there are two kinds of possible transitions: one corresponding to the fact that P1 is in a blocking state and process P2 can be in a sending or receiving mode on a channel different from ch1; conversely, another transition corresponds to the fact that P2 is in a blocking state but P1 can be sender or receiver on another channel.

$\{$ SEND(ch1,val_out), ok || RECEIVE(ch1,val_in) ok $\} \cup$
$\{$ SEND(ch1,val_out), nok || EvtSt(chi,val_in) X | i≠1 $\} \cup$
$\{$ EvtSt(chi,val_out), X || RECEIVE(ch1,val_in) nok | i≠1 $\}$

X takes the value ’ok’ (there is no blocking) or ’nok’ ( there is a blocking). Val_in and val_out are the received and sent values on a channel. EvtSt can take the value SEND or RECEIVE.

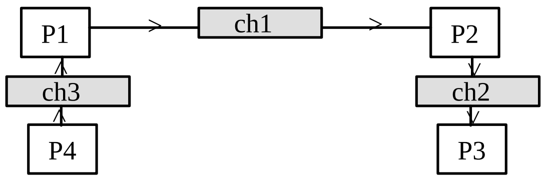

**Figure 7 : An example**

The transition is in fact labelled with the same number of communication events as the number of processes. Thus the Cartesian product of the set of transitions for each channel must be considered. In the previous example, if we add a third process P3 connected to P2 with the MRV channel ch2, and a fourth process P4 connected to P1 with the MRV channel ch3 (see Figure 7), the set of possible transitions (events in the order of the processes P1,P2,P3,P4) is the following:
{SEND(ch1,val_out) ok || RECEIVE(ch1,val_in) ok || RECEIVE(ch2,val_in) nok || SEND(ch3,val_out) nok } ∪
{SEND(ch1,val_out) nok || SEND(ch2,val_out) ok || RECEIVE(ch2,val_in) ok || SEND(ch3,val_out) nok } ∪
{RECEIVE(ch3,val_in) ok || RECEIVE(ch1,val_in) nok || RECEIVE(ch2,val_in) nok || SEND(ch3,val_out) ok } ∪
{RECEIVE(ch3,val_out) ok || SEND(ch2,val_out) ok || RECEIVE(ch2,val_in) ok || SEND(ch3,val_out) ok }

In this example, we can notice that two concurrent information transfers may occur on two different channels (the last case).

It is sure that in the case of more complex system, the manual computation of all information transfer cases is impossible. That is why a predicate–based approach is more appropriate to express,in a symbolic way, the logical link between the channels on which processes are active and the status (blocking / non blocking) for each process.

A last point must be considered by the communication module, concerning the computation of the value of each channel (val_in) which are transferred as input of processes, on the base of the values (val_out) which were transmitted by the SENDER processes of the channel. In fact, due to the delay introduced by the simulation step, it is necessary to consider a resource for a protocol such as VSIG ; the value val_in is thus computed from the current value of this resource, which itself evolves according to the transmitted values val_out.

We can notice that the MRV protocol does not need any resource. This is because the information transfer occurs in the same simulation step, if senders AND receivers are ready.

To compute this data flow, we also benefit from a predicate–based approach, implemented in the EVAL tool.

## II – A Predicate–Based Modelling

### II.1 – EVAL Tool Principles

EVAL is an industrial tool developed by EDF and VERILOG providing modeling and validation features for the analysis of system behavior. Thanks to Labelled Predicate/ Transition Nets, and a Prolog engine, the EVAL tool can be used for high abstract level modeling, particularly suitable for the case of global system and software [LlAV, 90].

A multi–level hierarchical modeling is performed by means of an efficient graphic editor.

The tool is based on Petri net formalism, with Predicate and Labelled extensions. A transition is characterized by :

♦preconditions : the set of required communication events (input or output), the set of required places, a PROVIDED condition,

♦postconditions : the set of replacing places.

The places are defined as predicates. The condition PROVIDED, expressed in Prolog, may specify typing, database, logical conditions or processing.

The semantics of communication between the sub–modules is based on a multiple RENDEZ_VOUS mode which is an extension of CCS [Mi, 89] and LOTOS [BR,88] communication. The tool also includes a communication mode by FIFO queues. The global Petri Net is computed using the parallel composition operator of the CCS algebra, which is expressed by transition merging [LlAV,90] of the models of all sub–modules, at the decomposition level considered.

Thanks to an exhaustive simulation engine, EVAL builds the reachable state graph, interpreted as LTS, on which standard model checking properties (deadlock, liveness, models that are not cyclic, dead code) are provided.

Finally, EVAL supports Process Algebra operators such as composition, reduction based on bisimulation [Mi, 89], trace equivalence, temporal logic formula checking...

## II.2 – EVAL Implementation

From section II, we define the requirements that the EVAL communication has to verify.

♦In a SYNCHRONOUS way, it receives the activity of each module, e.g. on which channel it communicates. From a SENDER, it receives also a value.

♦For each channel, it decides if protocol and rules on SENDERS (composition rules) and RECEIVERS (dispatching rules) are verified.

♦To each process, it sends the blocking conditions and, to a receiver, the current value of the channel.

♦For each channel, it updates the resource (containing the current value on the channel).

The communication module behavior is defined by means of only one transition, thanks to the predicate–base approach.

Each process $P_i$ is connected to the communication module by one interaction point. This link is a bi–directional link and allows to transmit in one step :

♦in one way : the channel on which the process is active and the value sent ( nil if receiver),

♦in the other way : the blocking result and the value received (nil if sender).

The EVAL model contains only one place ***list_fifo(L)***, which represents the list of channel resources.

All the simulation engine is defined by an unique predicate ***simul***. This one can be generic on the base of the formal model of the protocol. Prolog predicate ***simul*** computes output variables (blocking, val_out, new fifo) depending on input ones (channel, val_in, fifo) in the following manner.

For each channel :

♦it verifies if composition and dispatching rules are respected,

♦according to protocols of the channel, it computes the new values and the new processes blocking conditions

♦according to protocol, it updates the value contained in the channel.

For instance, let us consider a channel with the protocol RDV, Compositional Rule ALL and Dispatching Rule ONE.

If ALL transmitters of the channel are active and transmit the same value ’VAL, then CR is verified. If ONE receiver among ALL receivers of the channel is ready then DR is verified.

If CR and DR are verified then the communication is possible and the transmitters and receivers are unblocking. The receivers receives the value ’VAL.

If one of the two rules CR or DR is not verified then transmitters which were ready are blocked, waiting for the others.

Let us consider an other example: VSIG_val protocol for which no blocking conditions on senders and receivers are specified. So, it can be interpreted as a protocol with ONE as Composition Rule and SOME as Dispatching Rule. Among transmitters that are ready, one is selected in a non_deterministic way to put its value sent (VAL) into the buffer associated to the channel.

At the next simulation step, the value ’VAL contained into the buffer can be read by all receivers which are ready.

## III – Conclusion

This paper presented the LTS interpretation of the Verification Oriented VHDL language, and its implementation on CCS–based EVAL tool.

In order to meet this objective, we showed how the synchronous aspect of the VOVHDL language may be represented in terms of parallel composition of LTS models, concerning both the processes and the information transfer between them.

Thanks to a predicate–based approach, the communication model is reduced to a single generic transition, particularly significant of its synchronous characteristics.

Strengthened from this result, we now intend to study the VHDL semantics, according the same approach.

**Acknowledgement**

The authors would like to thank two anonymous reviewers for useful suggestions and comments.

**References**

**[BCG, 93]** M. Belhadj, R. MC Connell, P. Le Guernic ”a framework for Macro– and micro–time to model VHDL attributes” Proceeding of EURO–VHDL ’93, EURO–DAC ’93.

**[BR, 88]** E. Brinksma ”On the design of extended LOTOS” PhD Thesis. Twente University. Holland 1988

**[CCPBS, 93a]** P. Camurati, F. Corno, P. Prinetto, C. Bayol, B. Soulas ”VOVHDL : A verification–oriented dialect of VHDL” VHDL–FORUM 1993.

**[CCPBS, 93b]** P. Camurati, F. Corno, P. Prinetto, C. Bayol, B. Soulas ”Inter–process Communication for System–Level Design” International workshop on Computer Aided HW/SW Co–design 1993.

**[DJS, 93]** W. Damm, B. Josko, R. Schlör” A Net–Based Semantics for VHDL” proceeding of EURO DAC ’93, EURO VHDL ’93.

**[LRM, 88]** IEEE STANDARD VHDL Language Reference Manual IEEE Inc.,New_York NY, Mar 1988.

**[LlAV, 90]** JC. Lloret, P. Azema, F.Vernadat, ”Compositional design and verification with Labelled Predicates/Transition Nets” proceedings Computer Aided Verification 90, Rutgers University, New Jersey (1990).

**[Mi, 89]** R. Milner ”Communication and Concurrency” Prentice Hall, Englewood Cliffs, NY (USA), 1989.

**[OC, 93]** S. Olcoz, José Manuel Colom ”Toward a formal Semantics of IEEE Std. VHDL 1076” Proceeding of EURO–DAC ’93, EURO–VHDL ’93.

**[SGT, 92]** B. Soulas, JC. Geoffroy, Mme Tallec ”Toward an integrated approach to qualification of numerical systems” International symposium on Nuclear Power Plant, Instrumentation and control, Tokyo, may 1992.