# The COTRE Project: How to model and verify Real Time Architecture?

P Farail, P Gaufillet, J-M Farines, J-L Lambert, P Dissaux, H Hafidi, P. Michel, M Filali, J-P Bodeveix, François Vernadat, et al.

# The COTRE Project: How to model and verify Real Time Architecture?

**P. Farail & P. Gaufillet**
Airbus France, 316, route de Bayonne F-31060 Toulouse
Software Engineering Methods Experts
Tel : (0)5.61.93.66.28 - Fax : (0)5.61.93.03.54
pierre.gaufillet@airbus.com - patrick.farail@airbus.com

J-M. Farines
DAS-UFSC, Florianopolis, Brasil

J-L. Lambert & P. Dissaux & H. Hafidi
TNI-Valiosys, 120, rue René Descartes, F-29608 Brest

P. Michel
ONERA-DTIM, 2, avenue Edouard Belin, F-31400 Toulouse

M. Filali & J-P. Bodeveix
IRIT, 118, route de Narbonne, F-31062 Toulouse

F. Vernadat & B. Berthomieu & P.O. Ribet
CNRS-LAAS, 7, avenue du Colonel Roche, F-31077 Toulouse

**Abstract**

COTRE (``COmposant Temps Réel") is a two years project starting in January 2002, supported by Réseau
National des Technologies Logicielles (RNTL)

The COTRE Project aims at providing a design methodology and its associated software environment for the development of embedded real-time avionic systems. It contributes at bridging the gap between requirements of such systems, typically expressed in Architecture Description Languages, and formal development techniques, relying on modeling of systems and verification.

This paper will summarize the final status of the project:

- The graphical and textual description language used to model the architecture. We will also describe the work done with the AADL (Avionics Architecture Description Language) SAE standard sub-committee,
- The specification and verification of properties of COTRE components through the definition of a verification language and the use of formal proof techniques.

URL: http://www.LAAS.fr/COTRE

**Main author Biography**

**Patrick Farail** is Airbus Methods Development Engineer for Airborne Computer Software since 1989. He is the Team leader of experts on Software development activities including Specification, Design, Coding and Requirements Engineering.

He was involved earlier in Ada development (since 1983), he was a member of Ada Real-time groups ARTEWG and ExTRA and then participated to the definition of the HOOD method in 1987. Since these first experiments, he investigates, improves and promotes the use of software engineering methods in Airbus airborne computer software.

# 1   Introduction

The improvement and evolution of large and complex systems require more and more advanced development methodologies. Hardware and software components, their interactions and mappings, are some elements to consider in the development process of these systems. In real-time systems, as for example avionic systems, requirements are not only oriented towards functional behavior but also deal with non-functional aspects such as time, safety, reliability, and performance, constraints.

Criticism of avionic systems leads to improve system quality, through a rigorous development approach relying on verification. Due to their growing complexity, verification tools must be more and more powerful and efficient to guarantee this quality. A formal system description able to consider different levels of abstraction is an essential element to reach this goal. Other features commonly required for system development include re-usability, ability to reason about, and flexibility.

The COTRE Project research initiative provides a design methodology and a supporting software environment for the development of embedded systems. Its results should be extended to other application domains (e.g. automotive systems).

The COTRE project supplies the designer with a language that allows describing the system architecture and the behavior of components with their functional and non-functional requirements. The project also provides the designer with methods and tools to verify the expected properties of the system being developed, and derives a correct and robust system implementation.

Section 2 describes the software development process in avionics systems, from industrial and academic practices. The requirements, language and environment of the COTRE project are presented in Section 3. Finally, specification and verification of COTRE components are discussed in Sections 4 and 5.

# 2   Avionics software development

**Life cycle**

Software development in avionic systems is generally divided in four main phases, analysis, design, implementation and verifications

- During analysis phase, some detailed specifications are obtained in a formal language like SDL [CCITT88] or LUSTRE/SCADE [HCPR91, SCADE96], but software requirements are also defined informally through natural language and textual requirements.

- The design phase of the software is achieved through a former phase. It includes static and dynamic modeling, verification of properties, and mapping/deployment of software functions on hardware components.

- The implementation phase consists of automatic or/and manual code generation for all software components. Production of code is supported by unit and integration verification, by procedure defined during design phase.

- The development process ends with a target verification phase which deals with simulations of the developed system on the computational model of the plane, ground tests and finally flight tests.

The aim of this paper is to present the COTRE approach with respect to the design phase of avionic systems, with a particular emphasis on behavioral properties such as safety and aliveness properties and especially timed constrained ones.

**Main requirements for the design phase**

The design process corresponds to well-known steps, but is often influenced by the designer practices and the experience in the company or more generally in the domain. In our case, the design phase includes two main steps: static and dynamic design. The static design defines the software structure with its modules (object, class and so on…), interfaces and interactions. The dynamic design defines the architecture of the system, taking into account hardware constraints and applications requirements. During this step, logical correctness but also time correctness (e.g. respecting deadlines), reliability, resource allocation (particularly for CPU) and performance are the subject of analysis and verification. The main properties to be verified are related to:

- Resource utilization like for example processor workload, buffers overflow,

- Process and message schedulability, like for example execution order constraints, dependency ordering, time scheduling analysis, sensitivity analysis for computing time,

- Time constraints like delay between events, time intervals for handling messages, response time above threshold, latency,

- Reliability and safety, like probability of failures with recovering, time and space partition independence for safety levels,

- Functional constraints like correctness and robustness.

Finally, from the validated software structure obtained, the implementation phase deals with generation of code skeleton, and the achievements of tests in order to guarantee the conformity between implementation and design.

# 3 The Cotre project

The above requirements for the design phase allow defining the features to be introduced in the description language for these systems and the characteristics of the required verification techniques and tools. The requirements, mainly considering the architectural point of view, lead us to design the Cotre language as an ADL (Architecture Description Language). The main concept underlying the use of an ADL is that requirement analysis must produce a structural decomposition of a system into components that will be developed independently. Most ADL languages provide the following basic modeling concepts [MT00]:

- **components**, which correspond to computation or data storage units;

- **connectors**, which represent component interactions and their rules;

- and **configurations**, which describe architectural structures from components and connectors.

So, the Cotre language has been defined to describe (in a textual way, and in a next future in a graphical way) the system architecture from hardware and software components. It allows also expressing the properties to be verified. The representation of each component concerns functional aspects as well as non-functional features, such as time, safety and performance characteristics. The definition of new type of components, ports and connectors can be obtained by extensions of existing ones, as in the ACME language [GMW97].

Moreover, unlike the majority of ADL languages, formal models are used to represent behaviors and specify their properties. The Cotre language allows also to describe some kinds of non-functional properties and to support different techniques for property analysis and verification.
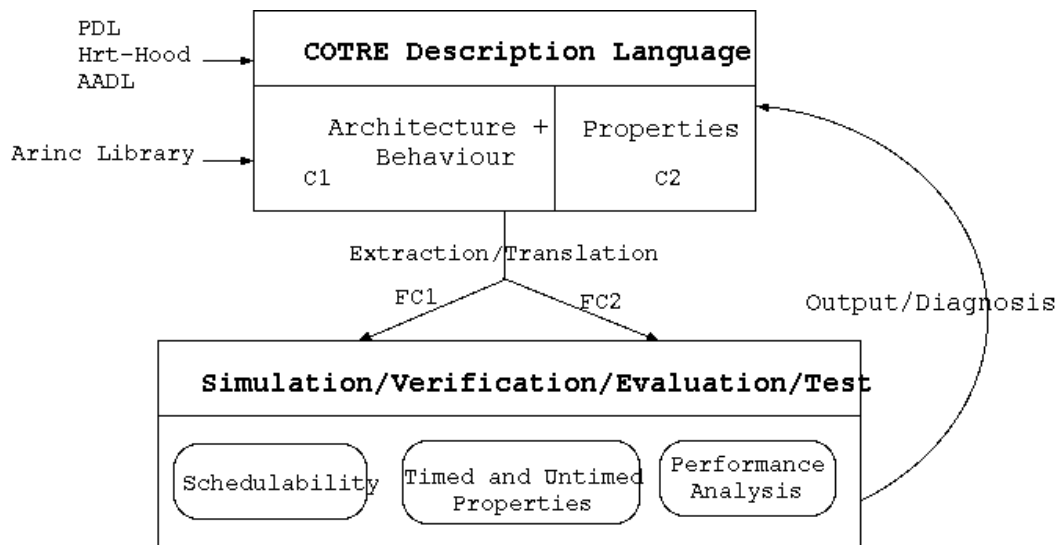


*Figure 1: Cotre: language and platform*

Figure 1 helps to understand the objectives of the Cotre language and platform. The Cotre language allows to represent in a specification C1, the architecture of the system and the behavior of the associated components and also to express, in a specification C2, the expected properties of the component and its assumed environment.

The formal specifications FC1 and FC2, used for verification are extracted from the specifications C1 and C2, according to the properties (temporal, safety, etc...) to verify. Specifications can be improved from the results of simulation, verification or evaluation, either in the Cotre language as in C1 and C2 representations or in the FC1 and FC2 formal representations. After verification, specific tools for test generation could be used to help deriving test sequences for future implementation of the verified specification.

A library containing operating system specific components, including those for ARINC 653 standard [AEEC97], of common use in avionics domain, supplies the designer with reusable components. With the aim of reusing components, a description in Cotre language can also import components in HRT-HOOD [BW95] or AADL, standard language. For that, compatibility of the Cotre language with these standards must be guaranteed.

The Cotre language is still evolving. Two views anchored on different aims but which would exist together in the final version of the language are present in the current stage of the development: the ``user'' view and the ``verification'' view. This duality leads us to define in a first time two languages: a "Cotre for User" language (U-Cotre), closer to users and a "Cotre for Verification" language (V-Cotre), closer to verification formalisms. The latter one acts as an intermediate language between the U-Cotre and the various formalisms used for the

verification. Figure 2 summarize the current status of the Cotre project and shows the two views of the Cotre language.
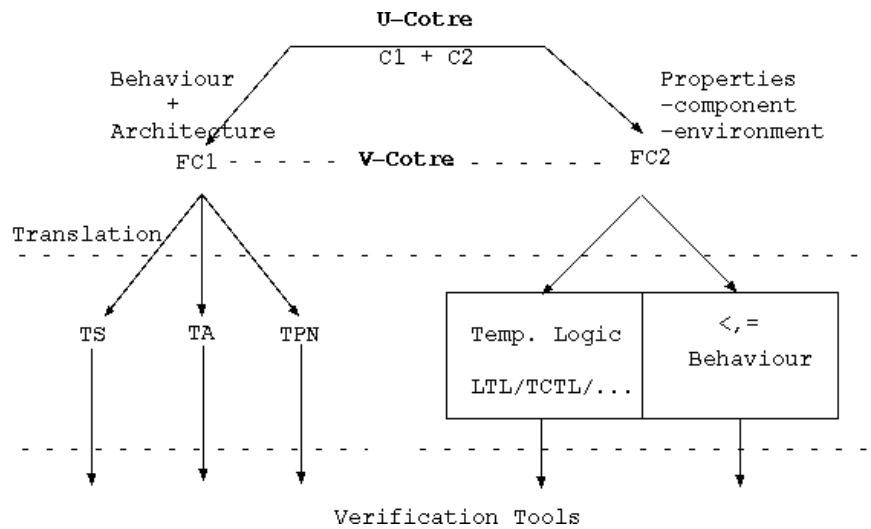


*Figure 2: Cotre: The point of view of the verification*

The understanding of these two points of view progressively leads us to a convergence between these two languages and to the definition of a unique Cotre language. But it is useful to maintain the two levels in order to keep some freedom in the definition of the U-Cotre user level.

As a matter of fact, one major requirement of COTRE is to provide designer (i.e. at U-Cotre level) a graphical formalism and behavior and a properties friendly language in order to model the architecture without knowing the underlying ADL and formal techniques. The graphical language may be HOOD or UML 2.0.

One main result of the COTRE project is a strong collaboration with SAE (Society of Automotive Engineers) committee in order to make possible the convergence between AADL (Avionics Architecture Description Language) standard [SAE02] and COTRE language. In the final first version of the AADL standard (planned in February 2004) some COTRE concept will already be included.

In the following section, we present the main characteristics of the intermediate language V-Cotre.

## 4   The V-Cotre language

The intermediate language V-Cotre appears as an architecture description language, but can also be seen as a common interface for different verification formalisms. The main characteristics of this language are the following:

- The static architecture hierarchy and described by components and connectors. The component interface is composed of input or output ports. Component interconnection is built with multi-senders and multi-receivers connectors. Unidirectional ports make them up.

- Dynamic behaviors are described by transition systems that are attached with basic components of the Cotre program. These transition systems are in fact communicating automata. Moreover, some elements of this dynamics allow to take into account real-time aspects (periodic wait, periodicity, ...).

- The specification of qualitative and quantitative properties is also allowed. In order to express the expertise, we have introduced a family of generic properties with their dedicated syntactic constructions.

- An important constraint to define the V-Cotre is easy translation to the "target formalisms" in the context of

the Cotre project: Transition System, Timed Automata [ACD90] and Time Petri Net [BV91]. Thus the existing software environments associated with these formalisms can be re-used.

In order to illustrate the V-Cotre representation and verification process, we present here a short example (called "deadlock verification") stemming from AIRBUS specification with the following specification. We consider two periodic processes `u1` (period = 1396) and `u2` (period = 1726) which can use two semaphores `s1` and `s2`, either taken in the same order, or in inverted order (this case is shown in figure 3). The verification problem consists of an analysis of the deadlock possibility, in both cases. Let us remark, that the problem is not trivial because of the drift of activations due to the different periods.
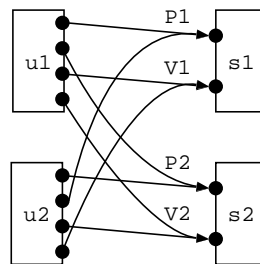


*Figure 3: Architecture of the example*

In the following, we present the characteristics of the language through the previous example.

### *4.1   Architecture description*

The static architecture is built from a hierarchical structure of components. At the more abstract level, a specification in V-Cotre is expressed as a list of parameterized components. We consider two kinds of parameters: typed communication ports and typed input data. The interface of each component can contain classical types and also its properties in the form of an agreement to be respected. The `agreement` part will be presented in section 4.3.

The implementation is described either as a composition of subcomponents linked by connectors or as a process. Connectors allow describing in an explicit way a topology and a communication protocol among various sender or receiver components. Cotre connectors are characterized by the fact that they do not hide buffers. Sending can be blocking or not; Receiving is always blocking. We have considered the following protocols: broadcast to ready receivers and synchronous communication between one receiver and one among n senders.

For the "deadlock verification", illustrated by figure 3, we present the main component where the connectors `P1,V1,P2,V2` and connect the subcomponents s1 and s2, instances of the `semaphore` component, and `u1` and `u2`, instances of the `process` component.

```
component main
  connector P1[2->1], V1[2->1], P2[2->1], V2[2->1]
    subcomponent s1: semaphore(1, P1, V1)
    subcomponent s2: semaphore(1, P2, V2)
    subcomponent u1: process(0, 1396, 1396, 1,
                    P1, P2, V1, V2,
                    1, 190, 10, 250, 150)
    subcomponent u2: process(0, 1726, 1726, 2,
                    P2, P1, V2, V1,
                    2, 190, 10, 250, 150)
end main
```

### *4.2   Process description*

The basic component is the process. To avoid explicit handling of clocks, we have distinguished constructors for periodic and for sporadic processes. Each process type is characterized by real time attributes (release time, period, deadline, ...) which will be taken into account by the "target formalisms".

### 4.2.1  Behavior description

The semantics for the behavior in V-Cotre is based on labeled transition systems. The process behavior is described as an infinite loop with a non-deterministic choice over guarded transitions. The transition guard consists of a boolean expression and a communication (emission `exp_port!` or reception `exp_port?`). A time-out may be used to bound the waiting time for a transition.

The action part of transition can update local variables and wait between *m* and *n* time units or wait for the next period (ARINC 653 `periodic_wait` system call).

### 4.2.2  Basic components of the deadlock example

In ARINC 653 specification, we have various categories of components. In this paper, we consider two of them: communication and synchronization objects. In the "deadlock verification"' example, we will use the semaphore component described in V-Cotre as follows  (in ARINC 653, the traditional semaphore invariant "*#P - #V + cpt = n*" is not satisfied due to `t3`. *#P* is the number of achieved *P* operations).

```
component semaphore(n: int; P, V: in port)
  var cpt: int
  initially cpt = n

  sporadic {
      t1: from cpt > 0 when P? -> cpt := cpt-1
  [] t2: from cpt < n when V? -> cpt := cpt+1
  [] t3: from cpt = n when V? -> skip
  }
end semaphore
```

In order to complete the example, we present also the process model which use the Wait_Semaphore and Signal_Semaphore services.

```
component process(delta, period, deadline: thetatype; prior: int;
          Pa, Pb, Va, Vb: out port;
          ty:int; d1, d2, d3, d4: thetatype)
  type PC = {PeriodicWait, Action1, WaitSemaphore1, Action2,
  WaitSemaphore2, Action3, ReleaseSemaphore1, Action4,
  ReleaseSemaphore2}
  var pc : PC
  initially  (ty = 1 => pc = PeriodicWait)
      & (ty = 2 => pc = Action1)

  periodic(delta, period, deadline, prior)
  {
     t1: from pc = PeriodicWait -> periodic_wait; pc := Action1
  [] t2: from pc = Action1 -> delay(d1,d1); pc := WaitSemaphore1
  [] t3: from pc = WaitSemaphore1 when Pa! -> pc := Action2
  [] t4: from pc = Action2 -> delay(d2,d2); pc := WaitSemaphore2
  [] t5: from pc = WaitSemaphore2 when Pb! -> pc := Action3
  [] t6: from pc = Action3 -> delay(d3,d3); pc := ReleaseSemaphore2
  [] t7: from pc = ReleaseSemaphore2 when Va! -> pc := Action4
  [] t8: from pc = Action4 -> delay(d4,d4); pc := ReleaseSemaphore1
  [] t9: from pc = ReleaseSemaphore1 when Vb! -> pc := PeriodicWait
  }
end process
```

We can note the initial value is specified by a predicate, thus it can be non-deterministic.


### *4.3   Compositional specification*

In addition to the declaration of typed ports that allows the communication with other components, the component interface consists of an `agreement` to be abided, with the intention to allow its compositional validation. This `agreement` includes not only the properties to be satisfied by the component, but also the assumptions on the environment.

```
agreement ::= requires id_component ['('exp,*')']
             | property language : ident : formula
             | property ident: prop
```

In the context of the Cotre project, the environment specification is made in an operational way, e.g. in terms of another component (clause **requires**).

Properties to be verified (clause **property**) can be expressed either as expert-oriented pre-defined properties (e.g. deadlock...), or as tool-oriented properties directly expressed using the assertion language of the target tool (TINA, UPPAAL).

The "expert" properties are temporal properties, extended with explicit time representation. They are introduced with the help of keywords. The aim is to spare the user the trouble of handling formulae of temporal logic. The other aim is to identify patterns of formulae for which efficient verification algorithms can be implemented by the tools.

The semantics of the "expert" properties can be expressed in TCTL [ACD90] in the following way.

| expert property | temporal logic |
|---|---|
| is_alive | AG EF$_C$ true |
| reachable e1 from e2 | AG (e2 $\Rightarrow$ EF e1) |
| reachable  e1  from  e2 within d | **AG (e2 $\Rightarrow$ EF$_{\leq d}$ e1)** |
| stable e | **AG e $\Rightarrow$ AX e** |
| invariant e | **AG e** |
| Resettable_pot | AG EF init |
| resettable within d | **AG AF $_{\leq d}$ init** |
| e1 leadsto e2 | **AG e1 $\Rightarrow$ AF e2** |
| e1 leadsto e2 within d | **AG e1 $\Rightarrow$ AF $_{\leq d}$ e2** |

We use an extended version of TCTL where temporal operators can be indexed by component actions. We can notice the following points:

- The detection of the partial deadlock cannot be expressed in this description level, because process identification is necessary.

- The verification of a component property is realized after composition with the required environment has been made; therefore, verification is made on close systems.

- `init` is a predicate which identifies the initial states.

## 5   Verification

In this paper, we present the verification aspects of the Cotre language and platform. V-Cotre is essentially a common description for the underlying formalisms (transition system, timed automata, time Petri net). It is used as an intermediate representation between the high-level user description (U-Cotre) and these ones. Moreover, the Cotre platform includes a set of methods and tools associated with these formalisms. We present here the verification techniques and tools.

**Verification techniques and tools**

Qualitative and quantitative (timed) properties, invariance and accessibility, abstraction and bi-simulation are some of the kinds of properties that the software development platform in Cotre project will allow to verify. Expressing and checking these properties may require different models, methods and tools, each kind of model typically comes with its tool suite.

The verification techniques implemented in the Cotre platform, are based on formula satisfaction or on model comparison:

- When the expected properties of the system are expressed as an abstract behavior, verification of the system turns out to check equivalence (or inclusion for a specific behavioral pre-order) between the concrete system and one of its abstractions. Different equivalences or pre-orders may be considered in accordance with the property class (language equivalence, refusal or acceptance semantics, bi-simulations, ...). Available tools like Aldebaran allow such verifications. For timed systems, tools like Tina or Minim (XTL) allow to derive ``Time-abstracting bi-simulations''.

- When the expected properties are expressed by means of temporal logic formulae, verification of the system turns out to check that the system is a model of a specific set of formulae. Different temporal logics have

been considered: qualitative one like LTL or CTL, or quantitative, as their timed extensions TLTL or TCTL. Different model-checkers are considered for such formulae, including SMV [BCMD90] for untimed properties, and Uppaal or Kronos for timed properties.

In many cases, state space explosion will be a major problem, as most of these tools rely on some enumerative technique. Some tools, like Tina, include reduction techniques, based on partial orders. Another alternative is given by LPV tool [DL99], which relies on linear programming techniques instead and avoid the problems due to the enumerative techniques. Compositional verification may be also considered since the specification of a Cotre component define an "agreement" which captures not only its expected behavior but also its environment. In this context, we intend mainly to use the verification techniques based on pre-order and behavior equivalences.

## 6   Conclusion

The COTRE Project, of which members are research centers, software developers and avionic companies, is aimed at supplying to software designers a language, an environment and a methodology to describe, verify and implement embedded avionic systems.

The goal of the COTRE language is to describe real time architectures, behaviors and properties, taking into account functional and non-functional requirements such as timing constraints, reliability and performance.

We have been mainly concerned, in this paper, by verification of temporal and time properties and presented a suitable development process for it. The semantics of the behavioral model is based on Time Transition Systems, this allow to use different modeling or abstraction techniques for COTRE components, such as Timed Automata, Time Petri Net, Timed Process Calculi, or LPV. The properties are expressed in logic-based formalisms like for example TCTL or in behavioral based formalisms. The verification tools for time-constrained components and systems are built from techniques based on formula verification on abstractions of the system behaviors or on model comparisons.

The COTRE language will integrate various conceptual aspects such as composition and hierarchy (and in a next version: refinement). It takes into account behaviors and qualitative as well as quantitative properties. Moreover, it allows performing various kinds of verification, from scheduling to safety, through performance evaluation. COTRE also provides a semantics framework, which allows integrating in a coherent manner these different aspects and proposes bridges between the languages of behavior or properties used by the existing verification tools.

The interpretation by the COTRE end-user of the results (diagnostic, counter-example, test sequences, ...) obtained in a formal level, during simulation, verification or test is also part of the critical tasks to be confronted in the COTRE Project. We are now working on that topic.

An expected benefit of the COTRE project is to bridge the gap between requirements, usually expressed in ADLs and whose scope extends way beyond the usual functional specifications, and the semantics approaches usually more concerned by theoretical issues like decidability and complexity of verification algorithms.

The COTRE approach consists in proposing a language that seems rich enough to express most of the properties expressible in ADLs and a semantics model that will allow us to reason about the different abstractions that will be made for verification purposes.

Furthermore, one major requirement of COTRE is to provide designer with a graphical formalism and also a behavior and a properties friendly language in order to model the architecture without knowing the underlying ADL and formal techniques. The graphical language may be HOOD or UML 2.0.

The COTRE approach consists in proposing a standard language that seems rich enough to express most of the properties expressible in ADLs, and a semantics model that will allow us to reason about the different abstractions

that will be made for verification purposes.

## References

[AEEC97] Airlines Electronic Engineering Committee, Avionics Application Software Standard Interface, ARINC Specification 653, 1997.

[ACD90] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In *Fifth Annual IEEE Symp. on LICS'1990*, 1990.

[BV91] B. Berthomieu and M. Diaz. Modeling and verification of time dependent systems using time Petri nets. *IEEE Tr. on Software Engineering*, 17 (3): 259-273, 1991.

[BW95] A. Burns and A. Wellings. HRT-HOOD a structured design method for hard real-time Ada Systems. Elsevier, 1995.

[DL99] S. Devulder, J.L. Lambert. A Comparative Study between Linear Programming Validation (LPV) and other Verification Methods. *IEEE Int. Conf ASE* 1999: 299-302

[GMW97] D. Garlan, R. T. Monroe, and D. Wile. Acme: An architecture description interchange language. In *Proceedings of CASCON'97,* Ontario, 1997.

[HCRP91] N. Halbwachs, P. Caspi, P. Raymond, and D. Pilaud. The synchronous data flow programming language LUSTRE. In *Proceedings of the IEEE, Another Look at Real-Time Programming*, volume 79(9), pages 1305-1320, 1991.

[CCITT88] ITU. CCITT Recommendation Z.100: Specification and Description Language SDL, volume VI.20 - VI.24.

[MT00] N. Medvidovic and R.N. Taylor. A classification and comparison framework for software architecture description languages. *IEEE Tr. on Software Engineering*, 26 (1): 70-93, 2000.

[SAE02] SAE. Aerospace information report. avionics architecture description language. Technical Report AS5506, SAE, 2002.

[SCADE96] Scade. Manuel de Référence SCADE-SAO+FD. Vérilog, 1996.

## URLs

CADP http://www.inrialpes.fr/vasy/cadp

TINA http://www.laas.fr/tina

Kronos http://www-verimag.imag.fr/TEMPORISE/kronos

XTL http://www.inrialpes.fr/vasy/cadp

UPPAAL http://www.docs.uu.se/docs/rtmv/uppaal

MetaH. http://www.htc.honeywell.com/metah/.