An approach translating CoD specification to be checked by UPPAAL

Faïez CHARFI*, Leila JEMNI BEN AYED*, Samir BEN AHMED*, Sofiène TAHAR**
faiez.charfi@fst.rnu.tn , leila.jemni@fsegt.rnu.tn , samir.benahmed@fst.rnu.tn , sofien.tahar@cs.edu.ca

*Institut Supérieur d'Informatique. 2, Rue Abou Rayhan Bayrouni. 2080 Ariana - TUNISIE
Tel : (+216) 71 70 61 64 Fax : (+216) 71 70 66 98

**Concordia University. Dept. of ECE , Montreal Quebec CANADA

Abstract.

The paper presents a formal development approach borrowing features from a specification method based on duration calculus CoD and verification method using the model checker UPPAAL. In this approach we translate CoD specification and required properties to be checked by UPPAAL, by introducing rules translating CoD formula to UPPAAL model which composed by automata system. Two classes of rules are introduced. The first is related to required properties and the second to behavioral ones.

Keywords. *Formal Specification, Formal Verification, CoD, UPPAAL, Real-Time System, Temporal Logic, UPPAAL*

1. Introduction.

Real Time Systems (RTS) become more and more complexes. Roughly 20% of the product development cost of modern transportation devices such as high-speed trains and airplanes is devoted to information processing systems. About 50% of the total software projects costs are devoted to testing [10]. Similarly RTS software are used for the process control of safety-critical system such as nuclear and chemical power plants. It is clear that bugs of such system may have disastrous consequence. For example, the software bug of the Radiation therapy Therac-25 machine caused the death of 6 Cancer patients between 1985 and 1987. Thus, it is important to specify with precision the RTS and thereafter verify the required properties in order to ensure a construction and a reliable development of these systems. Several formal specification methods have been suggested in the literature such as B[1] and Z[20]. Some of them are based on temporal logics like CTL[8], RTL[8], TCTL[2], LTPI [9] IDL[16], CoD[17],etc. In particular, the Calculus of Duration CoD [17], based on the Duration Calculus [8], provides a requirement specification language that allows a concise expression of properties about such durations and a calculus to reason about them. CoD is a very expressive logic for the modeling of critical real-time system. Only expressivities is not sufficient to reach a necessary degree of reliability and safety. Thus, we also have to verify required properties. Tools like SMV[13], KRONOS[21], UPPAAL [4], or DCVALID[15] are based on the model-checking technique and are proved to be more suitable to the automatic verification of real-time systems behavior [18] [14].

In this paper we propose a specification and verification approach using CoD as an expressive logic and UPPAAL as an automatic verification tool based on model checking. In our approach we adapt UPPAAL to the verification of CoD by introducing rules translating CoD formula to UPPAAL syntax. This will provide an effective platform for verification of real-time specification by the use of the model-checking technique.

The paper is organized as follows : first we present an overview of the CoD logic and the model checker UPPAAL in section 2 and 3 respectively. In section 4, we introduce the proposed approach adapting UPPAAL to CoD and describe rules translating CoD specification to UPPAAL models. The proposed approach will be illustrated through an example of pedestrian light control system.

2. Overview of CoD.

CoD is a temporal logic based on duration calculus [8][6][17]. It uses dense time structures and needs operators dealing with duration over an interval. CoD is ground around the following approach :

1. Requirement are specified by constraints on a set of variables *X* representing the relevant physical states as a functions of real time.
2. A top-level design is given by a control law and some assumptions. The assumptions are invariants on the system state *X*. The control law consists of two parts : (a) A finite state machine describing how control progresses through a number of phases.
   (b) A set of approximation constraints that for each phase maintains certain relations among the system state variables.

The duration of a predicate over an interval forms the basic element of this logic. We note that lower case name are used to denote constant or static variables, upper case name are used to denote state variable and the symbol *tt* and *ff* to denote the Boolean constant, such as *true* and *false* are reserved to duration formula. Duration formula in CoD are constructed as follows :

State expression and State assertion

1. every constant, static variable, or a state variable is a state expression,
2. any correct expression op(S1,…,Sn) formed from an operator symbol op and state expressions S1,…,Sn is a state expression.

A state assertion is a boolean state expression.

Durations and duration terms

For any assertion P, $\int P$ is a duration. A *duration term* is of type R and generated by

1. durations, real constants and real static variables are durations terms.
2. if op is an n-ary operator symbol of type R and r1,…,rn are duration terms then op(r1,…,rn) is a duration term.

The symbol ℓ is used as an abbreviation of the duration term $\int tt$.

Duration formulas

If *A* is any *n*-ary predicate symbol on R and $r_1,\ldots,r_n$ are duration terms then $A(r_1,\ldots,r_n)$ is an *atomic duration formula*. A duration formula is of type Bool and it is generated by :

1. atomic duration formula and the symbol *true* and *false* are duration formulas,
2. if D1 and D2 are duration formulas, so are (¬D1), (D1 ∨ D2), (D1 ; D2), and (∀x)D1 where x is a static variable.
3. an interval satisfies the formula D1; D2 if it can be divided in two sub intervals I_1 and I_2 where I_1 satisfies D1 and I_2 satisfies D2.
4. for any state assertion P, the duration formula $\lceil P \rceil$ is equivalent to $\int P = \ell \wedge \ell > 0$. It is true if P is true on all observation intervals different other that point interval.

Primitive duration formula are real arithmetic formula of $\int S$. In duration Calculus, the following abbreviations are used :

(a) $\lceil\ \rceil \triangleq (\ell = 0)$ and

(b) $\lceil S \rceil \triangleq (\int S = \ell) \wedge (\ell > 0)$.

For illustration purposes, we consider in the following examples specified in CoD :

- " The duration of an event E over an interval of length u doesn't exceed k time units"

$$\ell \leq u \Rightarrow \int E \leq k$$

- " Whenever p holds, it holds for at least x time units and until q holds "

$$\lceil p \rceil\ ;\ \lceil q \rceil \Rightarrow \ell \geq x\ ;\lceil q \rceil$$

3. The Model Checker UPPAAL

UPPAAL [4][3] is a tool for the design, validation and the verification of real time systems (Figure 1). It is appropriated to systems that can be modeled by temporized automatons or by linear hybrid automatons of the class $LHS_{\neq 0}$ [7]. UPPAAL uses finite state automata extended with clocks and data variables to describe real time system behavior. Transitions of temporized automata are labeled by : (a) guards on clock values and integer variables; (b) a synchronization action is performed when transitions are taken; and finally (c) assignment of integer variables. In addition, control nodes must

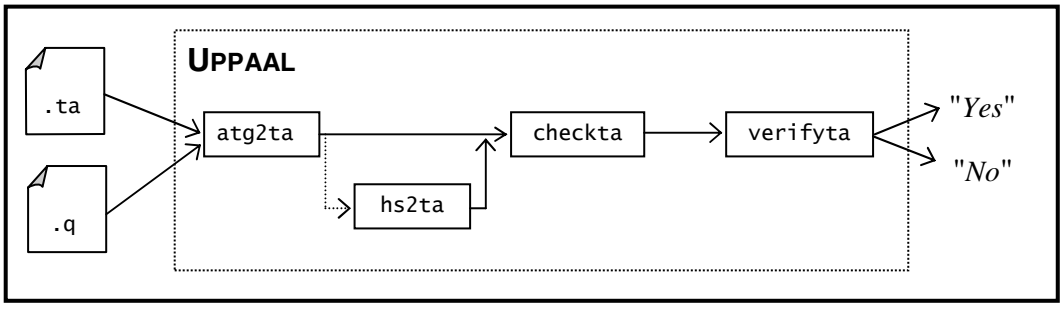

Figure 1. Overview of UPPAAL

be labeled with the invariant. UPPAAL uses a new data structure called *Clock Difference Diagram* CDD [11].UPPAAL is very appropriate for the verification of complex behavior of real-time systems because it provides a faster reachability algorithm [3]. It allows a graphic or a textual representation of models.

4. Translating CoD to UPPAAL

The proposed specification and validation approach using CoD and UPPAAL consists of following steps :

1) specify the system design behavior,
2) specify required user properties with CoD,
3) translating the behavior in (1) to UPPAAL model,
4) translating properties in (2) to UPPAAL properties,
5) proving that the model in (3) satisfies the specification of (1) using UPPAAL.

In this section we present an approach for translating CoD to UPPAAL beginning by the behavior properties and then by the required ones.

4.1 *Translating CoD specification to UPPAAL design (behavior properties)*

We present some general rules mapping CoD to UPPAAL. These rules regroup the most general used CoD formula.

State variables

In CoD, a state variable (ve) can be Boolean expressing the active state of a component of a subsystem (SS), or it can take a value from a set of phases expressing states of the control automaton. By convention, the inactive state of a SS is expressed in CoD by the variable (¬ve).

According to the type of ve, two types translating toward UPPAAL are possible:

Rule(1)(a) : ve : Bool → State Nve,ve ;

Rule(1)(b) : ve : {…, ph*j*, …}→ State …, phj, … ;

For state variables ve*i* ($1 \le i \le n$) of type Boolean, we added Nvei that expresses the inactive state of the i^{th} component of the system. By default it corresponds to the initial state of this component. As then two states exists, we can deduce transitions between them, because they are not specified in CoD.

A state variable ve*m* of type set, it will hold in one of phases ph*j*. We can deduce neither transitions nor the initial state.

Initial state

Let Phj ($1 \le j \le m$) be a state variable such that ve*i* := ph*j*. It corresponds to the initial state of the control subsystem and it is followed by any other state.

Rule(2) : $(\lceil Phj \rceil; true) \wedge \lceil\ \rceil \rightarrow$ init Phj

Transition:

We distinguish in CoD between two types of transitions. Those that are independent of the state of a SS and that appear in the declarative part of CoD specification, and those that are dependent of another SS and that appear during the specification.

Rule(3) :

$\forall i,j \in \{1..n\text{-}1\}. \lceil Ph_i \rceil \mapsto \lceil Phj \rceil \rightarrow$ trans Ph_i -> Ph_j .

Where n is the number of states.

Temporized guards and invariants

The variable ℓ is used in CoD to compute the duration time of states. In UPPAAL we propose to declare for every process its own clock that can be reset to zero to compute the time on the different states of the subsystem. z being a positive value.

Rule(4) : $\lceil vei \rceil \models l \le z \rightarrow$ hPVEi $\le$ z

Rule(5)(a) :

$$\left.\begin{array}{l} \lceil \neg vei \rceil \wedge \lceil vei \rceil \wedge \lceil \neg vei \rceil \Rightarrow l > z \\ \lceil \neg vei \rceil \wedge \lceil vei \rceil \Rightarrow l > z \end{array}\right\} \rightarrow \text{hPVEi} \le z$$

Rule(5)(b) :

$\lceil \neg Ai \rceil; \lceil Ai \rceil; \lceil Ak \rceil \Rightarrow \ell > \alpha \rightarrow$ hPVEi $\le$ z

Discreet guard and assignment

We add a label to the transition where it use xj $\in \{0,1\}$ global variable. It enables the synchronization between process. By default xj has the value of 0.

Rule(6):

$$\left.\begin{array}{l} - (\lceil Ai \rceil \wedge \lceil Bk \rceil); \ell \le \varepsilon \Rightarrow \lceil Ai \rceil \\ - (\lceil Ai \rceil \wedge \lceil Bk \rceil); \ell \le \varepsilon \Rightarrow \lceil \neg Ai{+}1 \rceil \end{array}\right\} \rightarrow \begin{array}{l} \text{guard xj==1} \\ \text{assign xj:= 1} \end{array}$$

Action and Event

Actions and events depend on the behavior held in a state of one subsystem which result holds at another state of another subsystem. Otherwise, it can be considered as the conjunction of guard and assignment viewed differently when we consider action or event.

Rule(7):

$$(\lceil Ai \rceil \wedge \lceil Bk \rceil) \Rightarrow \ell \le \varepsilon \quad \rightarrow \quad \begin{array}{l} \text{guard xj==1} \\ \text{assign xj:= 1} \end{array}$$

Rule(8) :

$$\lceil Ai \rceil \Rightarrow \ell \le \varepsilon \vee (\ell \le \varepsilon; \lceil Bk \rceil) \rightarrow \begin{array}{l} \text{guard xj:=1} \\ \text{assign xj== 1} \end{array}$$

Parallel State

In the same observation interval, system can hold at two different states, which are considered parallel states. We use also guard and assign labels to state that the parallel states do hold.

Rule(9):

$\lceil Ai \rceil \Rightarrow \lceil Bk \rceil$ → guard xj:=1
assign xj== 1

To illustrate the above rules, we consider the following case study example.

Case study : Pedestrian Light

Light can be Red or Green. The pedestrian request must take place by pushing on a button. If the button is pushed and Light is not green, then a request is recorded. Whenever light becomes Red it still at this state for at least 5 seconds before changing to Green. A request is canceled when light changes from green to red. Light changes from green to red automatically after 10 seconds. Initially, there is not a recorded request, the button is not pushed and light is red..

We show at first the behavior of Pedestrian Light specified using CoD followed by it's specification using UPPAAL

- CoD Specification

// Variable declaration
// req : Request ; bot : Button, ligh: Light ;
req, but : Bool
ligh : {red, green}
Red $\triangleq$ ligh = red
Green $\triangleq$ ligh = green

// Control automata Declaration
phases $\triangleq$ init $\wedge$ trans
init $\triangleq$ $(\lceil Red \rceil; \text{true}) \wedge \lceil \ \rceil$
trans $\triangleq$ $\Box((\lceil Red \rceil \rightarrow \lceil Green \rceil) \wedge (\lceil Green \rceil \rightarrow \lceil Red \rceil))$
// Constraint Declaration
contr $\triangleq$ $\Box$ (cntrRed $\wedge$ cntrGreen $\wedge$ cntGreen)
cntrRed $\triangleq$ ($\lceil Red \rceil \wedge \lceil Req \rceil \Rightarrow \lceil Green \rceil$
$\wedge \lceil Red \rceil \wedge \lceil \neg Req \rceil \Rightarrow \lceil Red \rceil$
$\wedge \lceil But \rceil \wedge \int Red < 5 \Rightarrow \lceil Req \rceil$)
cntrGreen $\triangleq$ $\lceil Green \rceil \Rightarrow \ell \leq 10)$
$\wedge (\lceil \neg Green \rceil \wedge \lceil Green \rceil \wedge \lceil \neg Green \rceil \Rightarrow \ell > 10)$
cntrReq $\triangleq$ $\lceil But \rceil \Rightarrow \lceil Req \rceil$

- UPPAAL Specification

```
//Global
int vreq, vbut, vligh ;
clock h ;
process PLIGH{
clock hligh,
state Red, Green{hligh <=10};
init Red;
trans Red -> Green{guard vreq==1;
        assign vligh:=1,hligh:=0;},
 Red -> Red{ guard vreq == 0, },
 Green -> Red{
      assign vlight:=0,hlight:=0;
              }; }
process PBUT{
clock hBUT;
state NBut, But;
init NBut;
trans NBut -> But{
    assign vbut:=1,hbut:=0     },
  But -> NBut{
    assign vbut:=0,hbut:=0;}; }
process PREC{
state NReq, Req;
init NReq;
trans NReq -> Req{
   guard vbut==1, assign vreq:=0;},
   NReq -> NReq{ guard vbut==1,
                vligh==1,hligh<5 },
   Req -> NReq{ assign vreq:=0; }; }
// System definition.
system PLIGH,PRQ,PBUT;
```

4.2. Translating CoD required properties to UPPAAL properties

Usually, several required properties are distinguished (Safety, Liveness, Reachability, etc,… [19] [18] [12]). We have chosen some of them to be translated in UPPAAL properties corresponding to the CoD formula, with the following classification. In Figure 2 φ is a state assertion and ψ a duration formula, such as $\lceil \varphi \rceil \equiv \psi$, where ψ can be the conjunction or the disjunction of duration formula.

(a) Reachability property	$\Diamond\psi \Leftrightarrow \Diamond\lceil\varphi\rceil$
(b) Safety property	$\Box\neg\psi \Leftrightarrow \Box\neg\lceil\varphi\rceil \Leftarrow \Box\lceil\neg\varphi\rceil$ $\Box(\psi_1 \Rightarrow \psi_2)$
(c) Liveness properties	$\Diamond\psi \ \wedge\Diamond\neg\psi$ $\Box((\psi_1; \text{true}) \Rightarrow \Diamond(\psi_1; \psi_2)$
(d) Invariance property	$\Box\psi$
(e) bounded liveness properties	$\Box(\ell \leq \alpha; \psi)$

Figure 2. Properties classification

Properties from (a) to (d) can be extended by adding duration value for a duration formula, such as $(\lceil\varphi\rceil \wedge \ell \leq \alpha)$. They will be temporized

properties.
For the general elementary CoD properties we identified the corresponding UPPAAL properties as follow (Figure 3).

Rule	CoD	UPPAAL
(10)	$\Diamond\psi$	E<> φ
(11)	$\Box\neg\psi$	A[] not(φ)
(12)	$\Box(\psi_1 \Rightarrow \psi_2)$	A[](φ_1 imply φ_2)
(13)	$\Diamond\psi \wedge \Diamond\neg\psi$	A[] (φ OR $\neg\varphi$)
(14)	$\Box((\psi_1;true) \Rightarrow \Diamond(\psi_1;\psi_2))$	φ_1 --> φ_2
(15)	$\Box\psi$	A[] φ
(16)	$\Box(\ell \le \alpha ; \psi)$	A[](φ imply $\ell \le \alpha$)
(17)	$\Diamond(\psi \wedge \ell \le \alpha)$	E<>(φ and φ .h<= α)
(18)	$\Diamond(\psi \wedge \ell \le \alpha) \wedge \Diamond(\neg\psi \wedge \ell \le \beta)$	A[](((φ and φ .h1<= α) OR($\neg\varphi$ and φ .h2<= β)) *Where h1 and h2 are respectively the local clocks of states* φ *and* $\neg\varphi$.
(19)	$\Box(((\psi_1 \wedge \ell \le \alpha);true) \Rightarrow \Diamond(\psi_1;\psi_2))$	(φ_1 and φ_1 . $\ell \le \alpha$) --> φ_2
(20)	$\Box(\psi \wedge \ell \le \alpha)$	A[](φ and φ . $\ell \le \alpha$)

Figure 3. Mapping properties from CoD to UPPAAL

More complex properties can be derived from the above, but there are some properties that cannot be translated automatically, like $\ell \le \alpha \Rightarrow \int\varphi \le \beta$. These will be divided into other properties according to the design of the system.

5.3. Tool adapting CoD to UPPAAL

We are still developing at the first stage a tool called *DC-UPPAAL* which translates respectively CoD specification and properties to UPPAAL model and properties. This tool must at the first step validate a CoD system behavior and properties. Then it will translate the system behavior specification into UPPAAL model in the file *sysmod.ta* and translate CoD required properties into UPPAAL properties in the file *reqprop.q* (Figure 4).
We notice that we can refine the transition to UPPAAL by disabling declaring the appropriates local clocks, if these last ones are not immediately from CoD formula. This for reduce the number of clock variables.
Otherwise we update the intermediate translated model when we encounter the need of a local state clock in one's of the required properties. The intermediate structure respectively for properties and system behavior will be necessary because it is build step by step after analyzing the CoD specification. Therefore, our outputs which are the UPPAAL inputs will be obtained easily from the intermediates data structures. Using this idea, users can visualize the system behavior and properties using UPPAAL and check the required properties.

6. Conclusion

In this paper, a new approach for the specification and the verification of complex real-time systems using CoD and UPPAAL has been proposed. The most distinctive characteristic of our approach is the specification of critical behavior referring to quantitive time structure and an automatic and fast verification. In order to group these two methods we have proposed rules transforming CoD specification to UPPAAL design. Our approach adapting UPPAAL to CoD requests a long term effort and this paper represents an attempt in this direction. As perspective, we will thoroughly

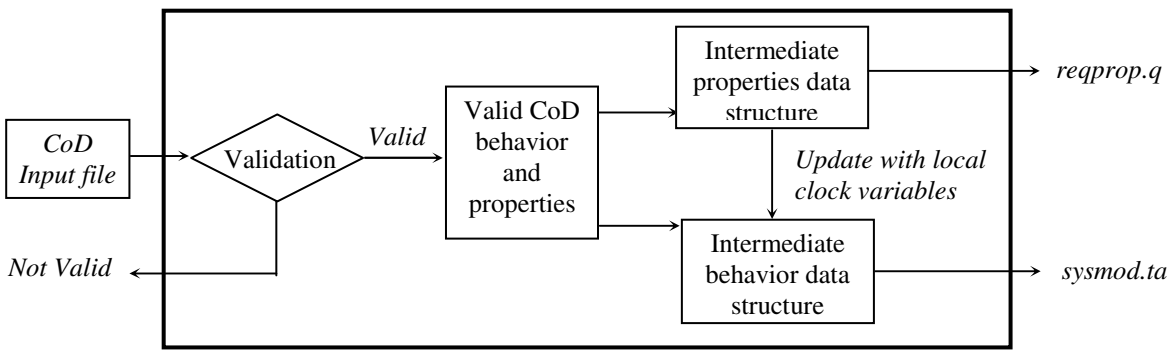

Figure 4. An overview of the tool

study the aspect of communicating process by channel to adapt it in our approach, and we will develop a tool DC-UPPAAL supporting the proposed transformation.

7. References

[1]. J.-R. Abrial, *AFADL: Approches Formelles dans l'Assistance au Développement de Logiciels*, ONERA-CERT, Toulouse, Mai, 1997.

[2]. R. Alur, C. Coucoubetis, and D. Dill, *Model-checking for real-time systems*, In Proc. 5th Symp. On logics in Computer Science, pages 414-425. IEEE Computer Society Press, 1990.

[3]. Gerd Behrmann, Johan Bengtsson, Alexandre David, Kim G. Larsen, Paul Pettersson, and Wang Yi. UPPAAL Implementation Secrets, Invited to 7th International Symposium on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'02), 2002.

[4]. Johan Bengtsson et Frederik Larsson. *UPPAAL – A Tool for Automatic Verification of Real-Time Systems*. DocS Technical Report N° 96/67. Université d'Uppsella. ISSN 0283-0574. Janvier 1996.

[5]. Z. Chaochen, C.A.R. Hoare, A.P. Raven. *A Calculus of Duration*. Information Processing Letters. Vol40, No5, p269-276, 1991.

[6]. Michael R. Hansen and Zhou Chaochen, *Semantic and Completeness of Duration Calculus*. Proc. Of Real-Time: Theory in Pratice, pp 209-226, LNCS600,92.

[7]. T. Henzinger. *The Theory of Hybrid Automata*. In Proceedings of the 11th Annual Sympnosium on Logics in Computer Science, pages 278-292. IEEE Computer Society Press, 1996.

[8]. F. Jahanian and A. K. Mok, *Safety analysis of timing properties in real time systems*, IEEE Transactions on software Engineering, SE-12, P 890-904, September 1986.

[9]. Leila JEMNI. *Une approche formelle pour la spécification et la vérification des systèmes temps-réel*. FST, Fevrier 2000.

[10]. Joost-Pieter KATOEN. *Principles of Models Checking*. Lecture Notes 2001/2002.

[11]. Kim G. Larsen, Justin Pearson, Carsten Weise, et Wang Yi. *Clock Difference Diagrams*, pages 271-298. Dans Nordic Journal of Computing, 6(3). 1999.

[12]. Zohar Manna and Amir Pnueli, *The Temporal Logic of Reactive and Concurrent Systems : Specification*, Springer-Verlag, 1992.

[13]. K. L. McMillan. The *SMV System for SMV version 2.5.4*. Novembre 2000.

[14]. Alfredo OLIVERO. *Modélisation et Analyse de Systèmes Temporisé et Hybrides*. VERIMAG-Institut National Polytechnique de Grenoble, Septembre 1994.

[15]. P.K. Pandaya. *Specifying and Deciding Quantified Discrete-Time Duration Calculus Formula using DCVALID*. Technical Report TCS-00-PKB-1. Mai 2000.

[16]. Paritosh K. Pandya, *Interval Duration Logic: Expressiveness and Decidability*. Electronic Notes in Theoretical Computer Science 65 No. 6(2002).

[17]. A.P. Ravaen, H. Rishel & K.M. Hansen, *Spécifying and Verifying Requirement of Real Time System*, IEEE Transaction on software Engineering, Vol19, No1, p41-55,1993.

[18]. Philippe Schnoebelen. *Vérification de Logiciels : Techniques et outils du model-checking*. Vuibert Informatique. 1999.

[19]. Jens Ulrik Skakkebaek, *Liveness and Fairness in Duration Calculus*. CONCUR'94: Concurrency Theory. Springer-Verlag, 1994(b). 283-298.

[20]. J. M. Spivey, *The Z Notation, A reference Manual*, Second Edition, Prentice Hall, 1992.

[21]. Sergio Yovine. *Kronos: A Verification Tool for Real-Time Systems*. Intenational Journal of Software Tools for Technology Transfer, Vol.1 , Nber. 1+2, p. 123-133, December 1997. Springer Verlag 1997.