Expression and Verification of Temporal Constraints for Real-Time Systems

D. Delfieu and A.E.K. Sahraoui*

Laboratoire d'Automatique et d'Analyse des Systèmes du C.N.R.S
7, avenue du colonel Roche, 31077 Toulouse Cedex, FRANCE.

Abstract

The specification of temporal properties covers two issues : the expression and the validation in a machine-independent way. The common sense associates Real-time to fast computing though Real-time contains two aspects : the problem of the coherence of Temporal Constraints between themselves and with performance constraints of the system. We propose a method that makes that distinction in order to decrease the complexity of Real-Time application and replaces each problem at the good step of the life cycle of the application. The originality of this specification method is to express temporal properties in a formal manner that handles the time in a qualitative and pseudo-quantitative way.

1 Introduction

In system design we distinguish two types of applications : transformational and reactive. Transformational ones behave like a black box which produce outputs from inputs and which can be defined by a function, it is the case of scientific calculator, information management...Reactive ones interact with their environment (man-machine interface, radar controller, embedded systems ...). In reactive systems we are interested in those which deal with time and particularly those with hard temporal constraints : reactive systems. We use the terminology of system because those problems are often solved by a combination of hard and soft. The notions of innocuousness, robustness are crucial notions for reactive systems because of the permanence and the repercussion which can be sometimes disastrous on the environment (system safety of nuclear plant). The specificity of reactive problems consists in taking into account a great number of events and temporal constraints (TC). If there exists a lot of specification methods, the expression and verification of TC for those systems stay today a matter of research. Some purposes for a new method should be to express all the TC of the requirement, to prove that all TC are compatible, to compare two temporal specifications. In the different works about this subject we can try to make a classification. The logic approach with TL (temporal logic) [4] , with Real-Time Logic (RTL) which is a first order logic incorporating "real-time operators" [9]. The system of transitions approach which allows to describe the dynamic behavior of the program in a very fine manner. In this family we can distinguish two parts : the visual formalism with Petri Nets, Statecharts and textual formalism like Esterel, Lustre, Signal...Our proposition can be classified in that last part.
In the first part we will survey the logic approach and try to compare the different logics used, in a second part we will mention the transition system approach, in which we will present our method.

*Also at Ecole Nationale d'Ingénieurs de Tarbes

2 A synthesis of different logic based approaches

In this part we compare the logic used, the expressive power of each for the expression of each approach and the verification of TC.

2.1 The Logics

Many logic based works use the framework of formal systems. A formal system is constructed on an inductive scheme. Induction is a philosophic, logic and scientific concept inseparable of deduction. The aim of formal systems is to mechanically encompass a reality in order to produce all its true propositions, the power of calculus of computers allowing us to profit of this mechanical aspect. Modeling an entity (program or environment) needs to express first all its true propositions (**B**) in a language based on logic and then the inferences rules (**R**). When B is infinite it must be described recursively by an induction scheme. R is the set of syntactic transformations that can be deducted from the entity, we have to verify their consistency

0-8186-4030-8/93 $03.00 © 1993 IEEE

(they must transmit the truth). This step concern the mechanization aspect.

2.1.1 Classical logic (CL)

A formal system based on CL should be composed of let be A, B, X atomic propositions ; $\vee, \wedge, \rightarrow, \neg$: usual connectors of boolean logic. **(B)**: $A \vee A \rightarrow A; A \rightarrow A \vee B; A \vee B \rightarrow B \vee A; (A \rightarrow B) \rightarrow ((C \rightarrow A) \rightarrow (C \rightarrow B))$ + specific axioms for the application. **(R)** contains the Modus Ponens and specific inference rules. It is easy to prove that a well formed formula is valid[1] or not but this system has a low power of expression we cannot express the notion of temporality, appurtenances, properties on object.

2.1.2 Propositional logic (PL)

Among the different works reasoning with TC, a lot of them had based their formal system on this logic, the first Hoare in [9], Jahanian and Mok with RTL in [10], Razouk and Gorlick with RTIL in [13], ...
Hoare has introduced logic for program verification, he has developed an axiomatic basis for the execution of programs. Then some have tried to extend this methodology primitively made for sequential to parallel problems using Temporal Logic (TL). TL is well suited for specification of parallel program but RT problems can be considered like the parallel execution of an environment process and system , therefore TL seems then be adequate for our problem. TL is a special case of a more general framework : Modal Logic

2.1.3 Modal logic (ML)

ML is CL augmented with modal operators : $(\Box, \Diamond =_{def} \neg\Box\neg)$ These had came to express a philosophic concept the *strict implication* which means *whenever A is true B is true* which is different from classical implication *When A is true B is true* because it introduces the notion of temporality. It is expressed by the notation : $(A \Rightarrow B) =_{def} \neg\Diamond(A \wedge \neg B) \equiv \neg\Diamond\neg(\neg A \vee B) \equiv \Box(\neg A \vee B) \equiv \Box(A \rightarrow B)$ (see [1, chap. 1]) $\Diamond$ is the eventually operator so $\neg\Diamond(A \wedge \neg B)$ can be read by "it is impossible that to have A true and B false". There are several systems (S, T, ..., S4, S5) which are determined by specific sets of axioms.
Temporal logic Modal logic bounds the valuation of a formula to the semantic of the "possible worlds" (also in [1]). A world is an interpretation of a formula. Each world is accessed by a relation and the axioms of the different systems (cited above) of ML defines the characteristics of this relation. For TL this relation models "the passing of time" so some properties must be verified : it must be reflexive (if we consider that the present belongs to the past and the future), anti-symmetric (for every pair of dates, one precedes the other and not reciprocally) and transitive, linear and discrete. This set of properties defines then the axiomatic called DUX ([1, chap. 4] for basics definitions, [11, chap. 3] for properties of temporal operators).

[1] A formula is valid if it is true for every interpretation ; an interpretation is the set of valuations of all its atomic propositions

2.2 The different approaches

2.2.1 Real Time Logic (RTL)

In [10] the authors argued that TL is not appropriate when analyzing RT systems because this formalism is only concerned with relative order of events that why they use a propositional logic named RTL and a function "@" which captures time and allows to deal with time in quantitative way (it assigns time values to event occurrences). This function is not total (not all the events get a temporal stamp) and the dates are non negative integer. Proving a temporal property (TP) is proving that it does not exist an occurrence function that is consistent with the conjunction of the negation of the TP and the set of axioms.
We can notice that "@" maps events with date of type integer set and a temporal property could be consistent with real date but not with integer date. Their method consists in three steps, first specifies the requirements in a event-action model that is the analysis part, second is an automatic phase of translation in RTL and third is an inference mechanism which allows to prove temporal properties.

Event/action model You express behavior in term of sequence or parallel actions, the rule of the model is to capture data dependences and to make a temporal scheduling on executions that can be launched in responses of events. Events are temporal stamp : begin or end of action, change of state, external events. Two types of TC can expressed sporadic and periodic TC :

```
Periodic : While < state_predicate >
           execute < action > with
        period = < time >, deadline = < time >
```

This constraint assumes that some action will be periodically executed at fixed interval whenever some predicate is true.

Sporadic : Assumes a maximum rate of service of the event.

```
When < event >
   execute < action > with
```

deadline = < *time* >, separation = < *time* >

Deadline : means that the action must be completed before a date Period : interval defined by [$\uparrow$ *action*, $\downarrow$ *action*] Separation : ensures that the sporadic request constituted by the occurrence of the event will be serviced at a maximum rate (considering that the events can be bufferized) These TC are upon the system, < *event* > is an external event.

Axiomatic and inference rules The automatic phase of translation produces axioms for each entity.

- For the function @: $\forall i E, i) = t \rightarrow t \geq 0$ "The date of the ith occurrence of E is greater than zero: Time has an origin" $\forall i, \forall j[@(E,i) = t \wedge @(E,j) = t' \wedge i < j] \rightarrow t < t'$ "Every posterior occurrence is temporally posterior"
- Start/end events: $\forall i, \forall t[@(\downarrow A, i) = t \rightarrow @(\uparrow A, i) \leq t)$: "start of A precedes end of A."
- Transition Event: If $@((S := T), 1) = 0$ Then $\forall i, \forall t @((S := F), i) = t \rightarrow @((S := T), i) < t \ldots$

In most of event/action model like in the input/output automaton (Mealy machine) there is a choice operator which lacks here. The process of axiomatization is very heavy because one has to describe the characteristics for each event that the start precedes the stop, that the two occurrences of the same event are ordered...
Concerning efficiency the example presented in the paper although simple produces after the translation in RTL a great number of axioms, a technique to improve efficiency of the inference mechanism to reduce the set of axioms implied in the inference by determinating the minimal set of necessary axioms. But determining this set must be a new challenge !

2.2.2 Using temporal logic for proving temporal properties

In the last decade, Bernstein proposed in [4] a formal system based on TL, his model of execution is a system of transitions where each state contains the variable t, which is the date at which the state was entered : We need for that an observer that dates according to the same origin all the entries of state.

Axiomatic It holds the axioms of DUX and Bernstein increases the expressiveness by defining new operators :
- diadic version of $\square$: $A \square B = A \Rightarrow_{def} B \wedge \bigcirc(A \square B)$[2]
- path(P,Q,R) : which means "if execution reaches a state satisfying P, then if it is to reach a future state satisfying R, it must pass by a state satisfying Q". To express RT he defines assertions for describing the notion "no more than n unit of time" and "no less than n unit of time".

Inferences rules

- Relative progress rule : **If** it takes less than n units of time for B to become true starting in a state in which A is true **and** more than n units for D to become true from any state in which C is true, **then if** A and C are true simultaneously, **then** B must become true before D becomes true. $\dfrac{A \Rightarrow^{n} B, C \Rightarrow^{n} D}{A \wedge C \rightarrow (\neg B \square \neg (D \vee \bigcirc))}$ with $A \Rightarrow B \Leftrightarrow (A \rightarrow \Diamond B)$ This is a safety property that Bernstein calls "Total correctness",
- Additive rule : Rule of calculus of minimum time of path $\dfrac{P \Rightarrow^{>n} Q, Q \Rightarrow^{>m} R, path(P,Q,R)}{P \Rightarrow^{>(n+m)} Q}$ This is a lower bound on the occurrence of Q after P, this notation is equivalent at : $\forall t, \forall t'(t < t' < t + n \wedge (H(t,P) \Rightarrow H(t',P)))$, where H(t,P) means P holds at time t.
- Hindrance rule : Rule of delay induced by an intermediate state, like in the work of [9] the notion of occurrences implies axioms and inferences rules ad. hoc., this will complicate the deduction mechanism. You cannot express floating date (i.e. a date depending of what happened before). It is important to note that he defines two basics assertions ("no more", "no less") for expressing all kind of TC.

We mention also the quantification of time in TL made by Pnueli [11].
Wolper has shown in [14] that TL is not enough expressive : every property of the type : "For any integer m $\geq$ 2, p is true in every state S_i where i = k*m (k : positive integer)" **is not expressible in TL**, it misses when one wants to express periodic properties like "Every even date p is true". He has introduced the solution of "grammar operators" but this complicates the axiomatization and will reduce the efficiency of the decision procedure.

2.2.3 Real Time Interval Logic (RTIL)

In [12], the authors introduces a layer to the interval logic of Schwartz which include the notion of RT date (i.e. not corresponding to the occurrence of an event) this allow the concept of floating dates, constructors

[2] $\bigcirc P$ means that P will be true in the next state ([10, chap. 3, page 187])

of interval increasing the expressiveness, and the possibility to express properties on occurrences of events, these extensions could be removed in the case of formal verification of functional properties. Using an interval temporal logic instead of a first order logic simplify also the deduction of properties. It is a limited form of quantified temporal logic with a dating function like @ in [10] but returning an interval instead of a date.
The execution model is a transition system where each transition is time-stamped. An interval is defined by a sequence of state an a pair of dates. RT points doesn't correspond to observations, there are bound to an interval and are in the model "pseudo-transitions" inside states. Formulas are evaluated in the context of interval. The semantic of interval is powerful because it allows to express useful temporal notions like search interval : $I \Rightarrow J$ (I, J are interval) is the interval beginning at the last point of I, ending with the at the last point of J.
This logic of interval is quite powerful but needs a good ability to express complex TC. This approach wants to test if execution traces in real execution environment meet temporal requirements, it is expensive is they does not.

3 Transition systems

3.1 Petri nets (PN)

There exists two classes of extensions of PN that deals with time in a quantitative way : Temporal Petri Nets and Temporized (timed) P.N. In the first model we associate at each transition a pair of dates (tmin and tmax), tmin is the minimum duration of sensitization of the transition and tmax is the date before which the transition must be fired. In the second model, time is given by a duration associated with transition, places or arcs. We can notice that n the two models, tokens that have two states : available and unavailable (when the token has just been put in a temporized place or when it is involved in a temporized transition or arc) it is trivial that these two models are equivalent.
PN are very useful without extensions for simple problems; but when we add them, it makes the formal analysis more complex. It lacks modularity and if it exists extensions dealing with the last remark can be made.

3.2 Statecharts (SC)

Harel has introduced SC like a visual formalism [8] that provides a way to represent state diagrams with notions like hierarchy, concurrency, broadcast communication and temporized state.. A SC can be seen like one or several automata where transitions are labeled by event[condition]/action, SC is said Synchronous because the system reacts to events by instantly updating its internal state and producing actions, the actions produced can trigger in the same instant other transitions, this is named chain reaction causing a set of transitions, the system is always in a waiting state until condition for a transition is true. An action is a fugitive event (start, stop or resume a task, update a variable, generate an event ...) .
Example of expressing a TC with SC (fig. 1) "Each

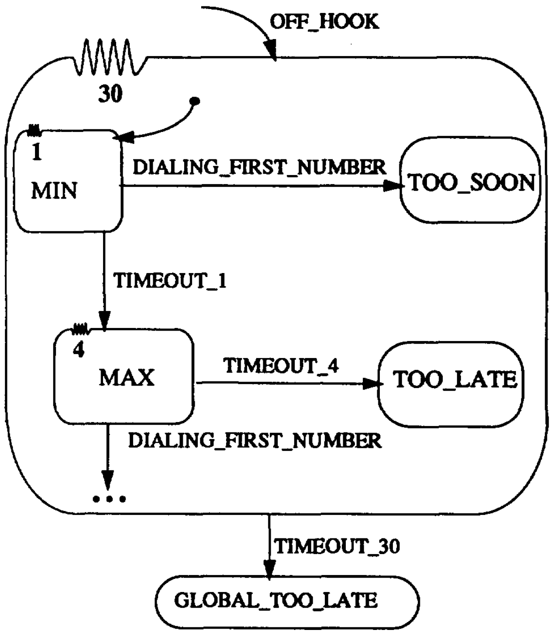

Figure 1: Timed SC for phone system

digit of a phone number must be dialed no sooner than one second and no later than five seconds and the entire phone number must be dialed within thirty seconds"

3.3 Esterel

Berry [5] and Harel [8] promote the very important notion of hypothesis of synchronicity (defined above) which will impose to the implementation a constraint of performance to . This hypothesis permits to have a standard style for programming building a skeleton of program which is in fact a deterministic RT scheduler of tasks. This method is better than using asynchronous parallel languages because process scheduling is made at the compilation step and parallel program are transformed in sequential automata producing better performances in the reaction, Berry argues also that this hypothesis improves the simplicity and

the safety of the style of programming considering that the performance constraint is not a crucial problem in the most of the problems (lift, underground ...).
Esterel is a synchronous language with standard constructs ";",":=", "if then else", "loop", ...and specific operators including parallel operators, statements scheduling of signals dealing with time in a multiform way : every signal can be seen as a clock. To illustrate some statements let's return to the example described in last subsection and represent it in Esterel code :

```
trap TOO_SOON, TOO_LATE in [
  do
    await FIRST_NUMBER
  watching SECOND
  timeout [ do
                 await FIRST_NUMBER
             watching 4 SECOND
             timeout exit TOO_LATE end
/* Code for the rest of the phone number */
                 ...] end;
  exit TOO_SOON ]
handle TOO_SOON ...
handle TOO_LATE ...
end trap;
||
loop
  every 1000 MILLISECOND emit SECOND
end
```

watching 4 SECOND means that the process wait for 4 occurrences of the event SECOND, itself emited each time 1000 MILLISECOND are received by the parallel composant.

4 The approach

There are two types of TC, constraints expressed on the system and constraints on the environment.
Our work is based from the categorization of Dasarathy in [6] which has clearly established the basis for analyzing and expressing the TC. A general confusion is to associate Real-Time to fast computing though there is in it two aspects : logic and performance, the first being relevant to specification step and the second in implementation step. Our view is to propose a method of specification of TC which separates these notions in order to be implemented independently and which allows to verify temporal properties. Performance aspect is relevant of code optimization, parallelization, power of computer, interrupt facility, preemptive scheduling ...There are some proposals and products to improve the performance, we focus our work on the logic and coherence aspects. For these aspects it must be defined a scheduling of events which permits to execute the specification and to test TC in regard to virtual scale of time.

4.1 Semantic analysis of TC

We can represent RT problems like two types of processes in parallel : environment and application, therefore we differentiate signals of communication between processes. We call Stimulus (S) a signal emitted by the environment and Response (R) the observable event that the system makes to environment. The simple TC are those which involve an event and a date or an interval. It gives Maximum TC, minimum TC and durational TC.
ex1 (maximum constraint) : "When the caller has off hooked the dial-tone must come no later than 2 seconds".
ex2 (minimum constraint) : "When the caller has off hooked the dial-tone must come no sooner than 1 second".
These two constraints schedule a date with an event
ex3 (durational) : "To get his correspondent he must let ringing between 2 (minimum) and 300 seconds (maximum)". This constraint schedule an action between two dates. It is trivial to notice that every complex TC is expressible with combination with of the simple TC presented above. A date can be interpreted like the pair origin date and a duration but in most of the case dates are not all expressed regarding to the same origin, an origin can be materialized by the occurrence of an event a date is then perfectly defined by an event and a integer (representing a duration expressed in a chosen unit of time). Combining the types of events and constraints let's try to analyze the semantic of each type :
SS : This type of constraints is like a "if then ...else ..." branching, the system must measure the time between the occurrence of the events (according that the TC is verified or not) and will branch in consequence.
SR : The system must generate a response. If the response is the completion of a task R (taking time) this corresponds to the occurrence of the event "end R" ($\downarrow R$). In case of minimum constraint (SR_{min}) the system has to check that duration of R is not too short, adding eventually a temporization. For maximum constraint (SR_{max}) it is a performance constraints : PERF=$\frac{n}{T}$ where n is the number of instructions in task R and T is the duration given in the constraint.
RR = (R1;R2) then $RR \Leftrightarrow (\uparrow R1 \downarrow R2)$. For RR_{max} $PERF = \frac{n1+n2}{T}$ where n1 (resp. n2) is the number of instructions of task R1 (resp R2), the case of

$R1R2_{min}$ is similarly at SR_{min}, it deals with temporization knowing performance. If R1 and R2 are events that take no time for RR_{max} (resp RR_{min}) R2 has to be triggered before (resp after) the limit.
RS $\Leftrightarrow$ SR identic to the semantic of SS
DURATION : "Hold an action A during an interval of time [t1,t2]" If A is a fugitive event, like time is discrete, A will be executed t2-t1 ($=\delta$) unit of time. If A is a task that take time, A is scheduled in [t1,t2], and the events $\uparrow A, \downarrow A$ and "Stop A" can happen (eventually several times for $\uparrow$ A and $\downarrow$ A) according to the duration of A. In most of the case of durational constraint (DC) A is a task that hold indefinitely like "holding a high level on a line" and verify DC is verify that the system produce "Stop A" δ unit of time with a given precision after $\uparrow A$ the precision is equivalent at a pair of TC : RR_{min} and RR_{max}, because it gives an interval around the occurrence of "Stop A" which produces a maximum and a minimum constraint.

In conclusion, we can distinguish two types of basic temporal constraints (minimum and maximum) and we can abstract our specification from performance constraint considering that a minimal performance PERF will have to be respected at the implementation step. After analyzing all TC we can calculate : $PERF = max(perf_{CT_1}, perf_{CT_2}, \ldots, perf_{CT_n})$. Thus, a minimal performance constraint is defined, it will be respected at the implementatuion step.

4.2 Specific features

4.2.1 Qualitative/quantitative aspects of time

We consider two aspects in a TC : quantitative where a duration is expressed in a number of units of time and qualitative if we consider events like temporal markers. If we take a median approach we need intermediate markers which will define a scale of time.

4.2.2 Synchronous hypothesis

Hard Reactive systems are characterized by the fact they receive a great number of stimuli and data flows, in order to hold our approach consistent we consider that every response of the system can be completed before every occurrence of new events. This hypothesis implies on the implementation to respect the performance constraint PERF. Therefore the system will be continuously waiting for externals events. As we stated above, we need a temporal marker, we suppose the system gets an internal clock that produces an event h at every period defined by the precision required on the duration. The system being synchronous all the receptions of events are atomic.

4.3 Description language

We want a language that expresses the non determinism of the environment, that allows to analyze and to verify the TC according to a scale of time that corresponds to the precision required. If we suppose that the set of external events is finite, that time is discrete then the set of acceptable execution trace is finite and can be considered as a language. The operators of our language are given by the structure of "Time domain". The precedence (event A precedes event B) is needed because "Time" is a Total Order [2], alternative (A or B can occurs) will express the non determinism of the environment, the closure allow to express the notion of infinite of the time.

Let be $\sum$ a set of letters corresponding to event occurrences "+ alternative", ". precedence", "* Kleene closure" [3]. Let be X = $\sum$,+,.,* the set of all the words formed on $\sum$ with the operators +,. and * ; a language L is a part of X^*. ex : $L = (a+b)^* =_{def} \bigcup_{n \geq 0} (a+b)^n = \{\epsilon, a, b, ab, \ldots\}$ set of strings containing eventually some a, b.

To express TC we use these operators on the alphabet $\sum$ constituted by all the events implied in TC and h. We can notice that this notation allow to express notion of floating dates or RT point [13] because these expressions define a language which is the set of all possible execution traces. The different forms of h (ex event 10h represents 10 occurrences of event h, h10 represents an a "divisor of h", 10 occurrences of h10 are equivalent to the event h) allow to adjust the precision required on a date ; the passing of time in a sequence will be represented by the increasing number of h in it.
example : ***A must phone to B before eight 'o clock, the phone number of B is composed of two digits, after hooking up a caller must dial each digit no later than 3 seconds and no sooner than 1 second, Considering that the system has been started at zero 'o clock***

We have a real date expressed relatively to the origin date of start of the program (8) and floating dates (no later than ...) for dialing the first digit expressed in regard of the event "off hook" for the dialing of the second digit it is expressed relatively to the event "first event dialed" We have seen above that a date is defined by an event and a duration therefore a TC is defined by two events and a duration. When analyzing the requirement we can find three TC, one concerning the date 8 and the event zero'o clock (CT1) and two similar TC on the digits :

CT1 : Maximum constraint (MC) on the ring (real date 8'o clock) CT2, CT3 : Minimum and Maximum constraint on digits $\sum$ = df, Ds, h, of, ri ; L is the set of words formed on $\sum$ with +, ., * of for Off hook, df for dialing first digit, ds for dialing second digit, h for event emitted every second, ri for ring LI = 8*60*60 it is the limit date before the telephone of B must ring, considering that h worths one second. For the first digit we have the expression :

$$CT_2 = of.h.df + of.h.h.df + of.h.h.h.df = of.\sum_{i=1}^{3} h^i.df$$

After the event OFF_HOOK (of) the caller wait one second (of.h.df) or two (of.h.h.df) or three but no more before dialing the first digit (df). For the second digit we have :

$$CT_3 = df.h.ds + df.h.h.ds + df.h.h.h.ds = df.\sum_{i=1}^{3} H^i.ds$$

For the limit date we have :

$CT_1 = \sum_{i=1}^{LI-2} h^i.of.w.ri$ with $\|w\|_h \geq (LI - i)$ w represent what is happening before the ring (ri), it is a word formed over $\sum - \{O, R\}$, $\|w\|_h$ is the number of occurrence of h in w), It means that if the caller spends i seconds (h^i = i occurrences of event h = i seconds) before he starts phoning (event OFF_HOOK: of) he will have only LI - i seconds to dial the phone number, because the number of occurrences of h in w measures the duration spent in the word w.

$$CT_{gl} = \sum_{i=1}^{LI-2} h^i.of.(\sum_{j=1}^{3} h^j.df.(\sum_{k=1}^{3} h^k.ds)).ri$$

This example shows that we can compose the TC. A synchronous, compositional and graphical formalism like Statecharts could help to express the TC [14].

4.4 A closed approach

One approach is to make apart a temporal specification that can verify a full verification expressed in an other formalism, a constraint being that we need a timer that produce in the execution trace the event h at the frequence defined by PERF defined above. The present approach is best illustrated by the following figure. Zaman is a declarative language close to natural language that allows to express all types of TC in a simple and structured way (see [14]).

In (fig. 2)[3] we define a method which contains several steps. In a first step, the specification is done using a formal or semi-formal model like Petri-Nets, Statecharts, SA-RT, Temporal Logic, VDM, CCS etc...

In a second step timing constraints are captured using a temporal language : the file obtained contains the timing assertions of the specification

In a third step, the specification is translated into au-

[3]The ellipse represents an object (file) and the box, the transformation

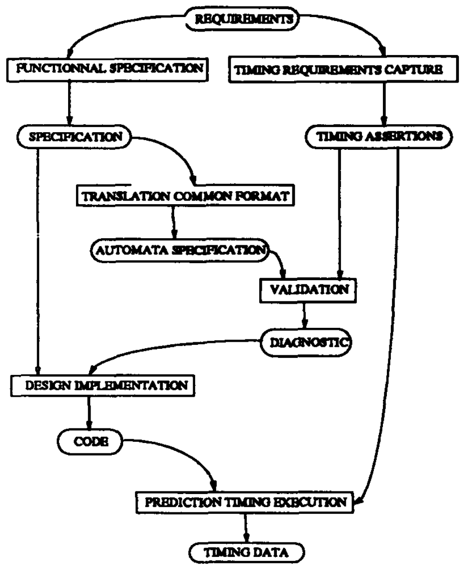

Figure 2: General approach

tomaton chosen as a common format code.

In a last step validation is done on this obtained automaton : had the specifier taken into account all timing assertions set at the requirement step?

If the result is not satisfactory the specification is modified, in the other case the design and implementation is carried out. The specification is entirely done in the left side of the figure. The right hand side is necessarily be redundant (heterogeneous redundancy) in terms of temporal assertions.

4.5 Verification of temporal aspects of a specification

An expression defined on the structure defined above represents a language, the set of all execution traces that are consistent with temporal requirements. In order to automatize the recognition of a trace (produced by the execution of a full specification) we build an acceptor of language.

For "regular temporal constraints" i.e. expressed with a regular expression [3], we are going to build a finite state automata. The method is based on successive derivations of the regular expression. See an example in annex A1 and note that at each derivation corresponds one state in the automaton. For every regular temporal constraints the theorem of Kleene shows that it exists a finite number of distinct derivations (then a finite number of states into the automaton) and it corresponds with a finite automaton.

If all the TC were expressible with regular expres-

sions it is simpler to give directly the automaton, let see TC that are not representable by automata. We call Context-free TC temporal constraints like $\{a^n.h^n, n \geq 0\}$ this corresponds to a non rational language and cannot be represented by a finite automaton. It is easy to show that this expression has an infinite set of distinct derivations. Consider the class of languages L_k=$\{a^k.h^{p+k}, p \geq 0\}$ which is infinite, we see that each L_k is a derivation of L : $L_k = (L')_{a^k}$ However there exists a way to represent with context free grammar (type 2 in Chomski classification) $G = [(a,h), S, S, (S \Longrightarrow aSh, S \Longrightarrow \epsilon)]$ the acceptor will be then a pushdown automaton.
We call calculable TC, temporal constraint of type $\{a^n.h, n = k^2, k \geq 0\}$ there is a notion of calculability that can be only expressed by a Turing machine that has the possibilities to recognize a language and to calculate.
When one adds a new temporal constraint (ntc) he could want to verify if this TC is not already included in the global expression. This new TC could have been be inducted by the previous already expressed. Let be G_{ntc} the grammar (see Annex A2 for definition of a grammar) generated by ntc, we have to find a TC from which the grammar G_{TC} = (X, N, P[{$\frac{x}{\epsilon}$, with x $\in N - N'$}], S) verify G_{ntc} is a sub-grammar of $G_{TC} : X \supset Y, N \supset N', P \supset P', S' \in N$ with G_{ntc} = (Y, N', P', S'). This method works for languages generated by grammar which are regular or context-free.
To prove the equality of languages in the case of regular languages the algorithm is quite simple : First find the minimal automaton then show the equivalence of automata (see definition in annex A3).
The main essence of our approcah is to look at Timing Constraints validation as a lexical and syntaxical analysis.

5 Conclusion

The contribution of this paper is to give a preliminary approach to deal with timing constraints. It shows how to analyze the notion of temporal constraint and to propose a simple way to express them using the notion of language. The aspect verification is played by the acceptor of those languages : to test the temporal aspects of a full verification we check an execution trace on the acceptor. This proposition allows to test full specification expressed in any formalism. A future work is to experiment the approach on existing software tools.

Annex

- A1 : Example of getting an automaton from a regular expression
  $CT_2 = of.h.df + of.h.h.df + of.h.h.h.df = of.\sum_{i=1}^{3}.h^i.df$
  Definition : Let be L language over V,
  $L'_\alpha = \{\xi, a.\xi \in L\}$ is the derivation of L by $\alpha, (\alpha \in V^*)$ The process of derivation gives :
  $D_0 = L'_o = h.df + h^2.df + h^3.df$
  $D_1 = (D'_0)_h = df + h.df + h^2.df$
  $D_2 = (D'_1)_{df} = \epsilon$
  $D_3 = (D'_1)_h = df + h.df$
  $D_4 = (D'_3)_{df} = \epsilon$
  $D_5 = (D'_3)_h = df$
  $D_6 = (D'_5)_{df} = \epsilon$
  Then process stops when there is no new derivation. The tree of the derivation gives the automaton :

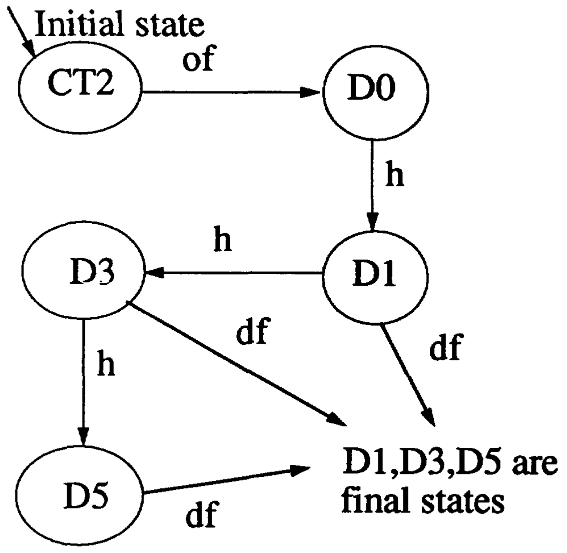

Figure 3:

- A2 : Definition of a Grammar
  A Grammar is a quadruplet G=(X,N,P,S) Where X Alphabet, N : non terminal symbols, P : set of the production rules, S : Axiom.
- A3 : definition of equivalence of automata
  Let P_0 and Q_0 two automata, $P_0 \cong Q_0$ iff $P_0 \equiv Q_0$ and $Q_0 \equiv P_0$, $P_0 \equiv Q_0$ iff for every action a we have : if $P_0 \overset{a}{\Rightarrow} P_0$' then $Q_0 \overset{a}{\Rightarrow} Q_0$' and $P'_0 \cong Q_0$'

REFERENCES

[1] E. Audurau, P. Enjalbert, L. Fariñas Del Cerro, "Logique temporelle sémantique et validation de programmes parallèles", ed. Masson, Chap. 1-4, 1990 [in French].

[2] H. Bestougeff, G. Ligozat, "Outils logiques pour le traitement du temps", ed. Masson, 1989, [in French].

[3] J.E. Hopcroft, J.D. Ullman, "Introduction to automata theory languages and computation", ed. Addison_wesley, 1979.

[4] A. Bernstein, P.K. Harter, "Proving real-time properties of programs with temporal logic", in Proc 8th Symp. Opera. Syst. Principles, ACM SIGOPS, pp. 1-11, 1981.

[5] Berry, P. Couronne, G. Gonthier, "Synchronous programming of reactive systems : an introduction to ESTEREL", INRIA report $N°$647, March 87.

[6] B. Dasarathy, "Timing constraints of real_time systems: construct for expressing them, methods of validating them", IEEE Trans Software Eng., Vol. SE-11, pp80-86, 1985.

[7] E. Harel, "On visual formalisms", Communication of ACM, vol. 31, pp. 514-530, 1988.

[8] C.A.R. Hoare, "An axiomatic basis for computer programming", Communication of ACM, vol. 12, pp. 576-580, 1969.

[9] F. Jahanian and A. K. Mok, "Safety analysis of timing properties in real-time systems", IEEE trans. software eng., vol SE-12, pp 890-904, 1986.

[10] Z. Manna and A. Pnueli, "The temporal logic of reactive and concurrent systems", ed. Springer-Verlag, book, 1992.

[11] A. Pnueli, E. Harel, "Applications of Temporal Logic to the specification of Real Time Systems", in Proc. FTRTFT Sys., Lecture Notes in Computer Science, pp 84-98, Warwick, Sept 1988.

[12] R. Razouk and M. Gorlick, "A real-time Interval logic for reasoning about executions of real-time programs", in Proc. 3th Symp. Soft. Test. An. Valid., ACM Sigsoft, pp10-19, 1989.

[13] A.E.K Sahraoui, D. Delfieu, "Zaman, a simple language for expressing timing constraints", 18th IFIP-IFAC workshop on real-time programming, Bruges, June 92.

[14] Wolper, "Temporal logic can be more expressive", Information and control 56, pp 72-99, 1983.