A COMPUTATION MODEL FOR REAL-TIME SYSTEMS

NDZANA Benoît
Senior Lecturer, National Advanced School of Engineering,
University of Yaounde I, Cameroon

BIYA MOTTO
Frederic, Senior Lecturer, Faculty of Sciences,
University of Yaounde I, Cameroon

LEKINI NKODO Claude Bernard
P.H.D. Student; National Advanced School of Engineering,
University of Yaounde I, Cameroon

ABSTRACT

This paper examines a simplified computation model for real-time systems, TMRS (Transition Model for Real-time Systems). It investigates a simpler way to develop a model in order to highlight the main aspects and properties of it and mainly timing constraints problem. Such model takes into account the physical process and the controlling process (computer process); the physical process being either a discrete-event system such as a manufacturing process, or continuous system such as a chemical process. The controlling process is an active program being executed to control and monitor the physical process; the controlling process is often denoted by reactive systems. The whole environment is considered in this study. The difference of this model with some proposed models is discussed. Such model enables to discuss properties of real-time systems and mainly timing constraints as a safety property using a temporal logic based specification language are based on temporal logic and transition systems; the communication model between computation processes and physical processes is based on CSP formalism. The contents of the paper will be limited to the approach, the definition of the model and how to specify its invariant properties.

INTRODUCTION

Advances in the theory of programming and computation over the last three decades have enabled the introduction of an increasing amount of mathematical formality. The application of these formal methods leads to systems that are developed by means of mathematical and engineering disciplines. A formal method of system development can be decomposed in three main or major components as follows:

- Formal notation for describing specifications and designs in a mathematical precise manner.
- A collection of inference/proof system for demonstrating that implementations meet their specification. The specification meant here is based on the definition developed by Lamport [1].
- A methodological framework for deriving implementations from specifications.

Our concern is directed towards reactive systems with a particular emphasis on temporal properties: timing constraints. The paper is structured as follows: the first section is an introduction on real-time systems and the necessity to develop a computation model. The most fundamental concepts will be illustrated through some existing models. The use of temporal logic for defining temporal semantics will be outlined. The second models introduces a complexity in evaluating the timing constraints problem. Our approach reduces the indeterminism by taking into account the physical process dynamics (temporal aspects) into the model. The basis of the approach is to give a formal solution with a pragmatic view of the problem. Temporal logic is introduced in the third section. In the fourth section, the computation model will be defined. A specification on safety assertions based on temporal logic will be presented. The safety properties specification and mainly timing constraints will be discussed in the fifth section. An attempt to define formally the interaction between both processes will be done.

I. REAL-TIME SYSTEMS

Programming has been classified into three main categories [2]: sequential programming, parallel (multi) programming and real-time programming. Real-time programming is carried either some classical languages or specific ones called “real-time languages” [3].

Automatic control discipline is the basis of linking a controller (being actually implemented on a computer system) and a physical process. Control scientists were concerned first on analysis study; the main property being the stability property. Some verification and correction methods exist to guarantee the stability using linear algebra (for linear systems) [4]. The step to control a system is a systematic way of development of control and automation. The control/supervisor is now implemented on a computer system. The discipline in programming a control algorithm, real-time programming, has emerged since: the aim is to make control software as time efficient as it can be made entirely through hardware.

Real-time programming has been defined and introduced as a discipline for its specific requirements with respect to the timing efficiency. Hence, the main difference with other categories mentioned above is that real-time programming is the execution time constraints; this will, according to Wirth’s terminology, be the difficulty involved in the interleaved access to shared data is eliminated by augmenting parallel programming with monitor like constructs, which can be used to enforce mutually exclusive access to the shared data. The use of synchronization primitives allows the processes order their activities.

As an example: controlling a liquid level in a tank, the case where the liquid reach a high level (alarm), the physical process does not wait for the control program to read the sensor and invoke an emergency action to the process. Some point in the program must ensure that certain conditions are satisfied before the event occurs.
Timing constraints in real-time programming do depend on timing constants of the physical process. Taking the precedent example, there may be some reasoning about tank overflow duration as time constant system dependant.

II. THE APPROACH

Let give a brief summary of some proposed approaches in using temporal logic (or other model) to reason about time in programming. We intend to classify them into three categories:

i- Reasoning about time at the instruction level
ii- Reasoning at the specification level
iii- Reasoning at the model level

This bottom-up classification enables to see how the timing properties can tackled in complementary steps. Our concern in the present work will be focused on (ii) and (iii).

Reasoning about time at instruction level:

Timing properties such as deadlines, periodic execution of processes, and external event recognition are studied at the instruction level. The main contributions are due to Haase [5] and Shaw [6]. Such scheme has been extended to concurrent programming. From this, two main ideas have emerged; the first that upper and lower bounds on execution time for instructions can be derived, based on given bounds for primitive statements in the language. The second is the use of Dijkstra guarded commands (or Hoare logic for Haase approach) to include the effects of updating real-time. There was an attempt of idealisation of real-time as realized by a perfect global clock denoted by \( \exists g_{1870} \), the computer time by \( \exists g_{1872} \), and the relation between the two expressed as

\[
\exists g_{1872} = \exists g_{1872} + \exists g_{2012}, |\exists g_{2012}| \leq \exists g_{2012},
\]

where \( \exists g_{20} \) is the clock drift.

The major outstanding issue is whether or not useful best or worst case execution time bounds can be found for statements in contemporary high level languages. The main contribution of this type of approach is the technique that, in principle, permits the prediction of the timing as well as the logic behavior of programs.

Dealing with time at the specification level:

Most approaches for specifying real-time systems (or in more general program behavior) combine transition systems models (state machines (SM), extended SM (ESM), Petri net (PN)) for modelling a system (software or hardware components) and temporal logic for specifying system behavior in terms of properties (timing, fairness, termination, etc).

Limitation: Although the approach seems interesting for discrete event system (DES); we may overcome this by considering continuous systems (CS) and DES under a generalized mode by emphasizing more on time occurrences of stimulus.

Dealing with time through the model approach:

The approach is similar to the above if we associate a model to a language (specification language). However, in order to avoid any confusion let give some terminological definition. We see by a model a defined entity of a system expressed by some theories (functional analysis, logic, etc.). This definition is control system oriented. The main idea is by taking into account the critical nature of real-time system application and the extension of previous work on concurrent programs models [7,8,9,10]. The model should express the timing aspect: that will be the main extension. An execution model is developed and an associated proof system has been obtained in [11].
There is an attempt in the present work to define a computation model. The originality of this model is that it involves both dynamics of the components contained in the reactive system: the computer process and the physical process. The model consists of communicating processes; the communication model being CSP. Each process, computer process or physical process is modelled as a state transition system.

III. TEMPORAL LOGIC

The version of temporal logic we will use was developed by PNUELI [12] and further described by LAMPORT [7]. A discussion of its use in program verification can be found in many papers of the authors cited above. The two main applications of temporal logic are artificial intelligence and software engineering. The first in on a proper treatment of time, since an action takes place in time and requires an appreciation of the logic structure of temporal facts. The main temporal logics are the one based on first order logic [McDERMOTT-82] and Allen’s theory of time [ALLEN-83]. In software engineering, many steps are required, among them specification, synthesis and validation (testing and verification). The temporal logic of programs introduced by Pnueli [PNUELY-77] has been fundamental on the field for the last decade.

In order to get a flavor, let take an example. Having the following algorithm written in a structured pascal like program:

```
y1:=n;
y2:=m;
while y2 > 0
do
    y1:=y+1;
y2:=y2-1;
od.
```

Which calculates two numbers \( y_1 \) and \( y_2 \). Such sequential program when executed is an active program, called computer proves in our terminology. This can be represented by a state diagram shown in figure 1.

**FIGURE 1: State diagram of a computer process**
There is a condition and an assignment. The states denoted by $S_i$ can be linked through the modal formula:

$$S_1: \text{cond}(S_1, S_2) Y \rightarrow Y := f(S_1, S_2)(y): S_2$$

$y: (y_1, y_2, ..., y_n)$ represent the variables of the programs

Temporal structure

If we define computational sequence as $S_0 \rightarrow S_1 \rightarrow \ldots$

Where $S_i$ is a state whose structure is

$S_k = \langle I, C > and l \in \{i; i = 1, ..., n\}$

$C: c_1, c_2, ..., c_n$; values taken at state $S_i$ by variables $Y$.

A real-time system consists of several processes:

- Computer processes
- Physical processes

The possible states can be contained into a set $\Sigma = \{s_0, s_1, s_2, \ldots\}$ and a reachability relation defined by $\{s_i, s_j\}$ such that $i \leq j$.

The temporal specification language: The common temporal operators added on the ones used in propositional logic are $\Box$ and $\Diamond$ and defined as:

$\Box A$ is true in state $s_i$ if $\forall s_j$ such that $(s_i, s_j) \in R$ and $A$ is true in $s_j$.

$\Diamond A$ is true in state $s_i$ if $\exists s_j$ such that $(s_i, s_j) \in R$ and $A$ is true in $s_j$.

Programming with temporal formula:

If $V = \{v_1, ..., v_n\}$ $n$ variables different from those appearing in the program; the associated formula will be:

$$((at I_1 \land \text{cond}((I_1 I_2)(Y) \land Y = V) \rightarrow \Diamond(at I_2 \land Y = f(I_1 I_2)))$$

This means that at each instant during execution; if the control is in $I_1$ and $\text{cond}(I_1 I_2)$ is verified, then at that instant or later the control will be in $I_2$ and the assignment $Y = f(V)$ will be executed.

Concerning the above program, we will apply this to it:

a- $$at I_1 \land y_1 = v_1 \land y_2 = v_2 \rightarrow \Diamond(at I_2 \land y_1 = n \land y_2 = m)$$

Such formula indicates that if control is at initial state $(I_1)$ then there exists a reachable state where $y_1 = n$ and $y_2 = m$ and the control will be at $I_2$.

b- $$((at I_1 \land y_2 > 0 \land y_1 = v_1 \land y_2 = v_2) \rightarrow \Diamond(at I_n \land y_1 = v_1 - 1 \land y_2 = v_2 - 1)$$

Formula in (b) states that at any instant during execution, if control is at initial state $(I_1)$ and condition $y_2 > 0$ is checked, there exists a future state where $y_1 = v_1 - 1$ and $y_2 = v_2 - 1$ have been executed and control is at $I_n$.

c- Lastly for the other transition $(I_2 \rightarrow I_3)$ we will have the following formula:

$$((at I_1 \land y_2 = 0 \land y_1 = v_1 \land y_2 = v_2) \rightarrow \Diamond(at I_3 \land y_1 = v_1 \land y_2 = v_2))$$

Some invariant properties can be checked in the following way:

$P_1 \rightarrow P_2$ If $P_1$ is true then $P_2$ will be true now and in the future

$P_1 \rightarrow \Diamond P_2$ If $P_1$ is true then $P_2$ will eventually be true in the future (there exists a future where $P$ will be true).
Partial correctness: It is specified by \((at \ I_1 \land P) \rightarrow (at \ I_n \land Q)\)

P is at initial state and if the program terminates then Q must be true.

Total correctness: It is specified by \((at \ I_1 \land P) \rightarrow \Diamond (at \ I_n \land Q)\)

P is true at the initial state, then termination is guaranteed and Q is verified. Going back to the same program example; the partial correctness is expressed as

\[ at \ I_1 \rightarrow (at \ I_3 \rightarrow y_2 = n + m) \]

If terminal state is reached then \(y = n + m\)

And the total correctness as:

\[ at \ I_1 \rightarrow \Diamond (at \ I_3 \land y_2 = n + m) \] (The terminal state will be reached with \(y = n + m\))

**IV. THE COMPUTATION MODEL**

The model is defined as \(<PP, CP, CC>\) where

- \(PP = \{pp_1, pp_2, ..., pp_n\}\), set of physical processes
- \(CP = \{cp_1, cp_2, ..., cp_m\}\), set of computer processes
- \(CC = \{cc_1, cc_2, ..., cc_k\}\), set of communicating channels

Transition system plays an important role for describing and analyzing processes and systems of communicating processes. We make use of it for state description of processes. The TMRS, transition model for real-time systems, differs from common model encountered in the literature in one particular aspect: the insertion of physical processes in the model.

Generally a process is a set of states and that an action or an event makes the current state of the process to change. Thus the possible behaviours of a process are represented by transitions; each transition contains the current state of the process, the new state it enters and the name of action or event which caused the change. An application on a sequential program has been presented in the previous section.

Let consider the following control program:

**Program control is**

\[ X, y: \text{address}; \]
\[ S: \text{sensor_value}; \]
\[ A: \text{actuator value} \]

**While true do**

- Read_port(s, x);
- Calculate error;
- Calculate control;
- Write_port(s, y);

**Od.**

This is the simplest form of a control program; there can be two possible enhancements to it; the first is to set a timer to each loop by using a timing primitive at the while statement, such enhancement is synchronous timing; another possible construct is to set predicate the state change at the s port; very often we make use of the first in normal execution and of second in case of critical action to be invoked by using interrupt constructs. The main assumption is that the time response of the program is less than the one of the system being controlled: this main idea is exploited in the development of specific control languages (synchronous languages).

In real-time systems, program interacts continuously with the external environment. An important issue is how to take real-time constraints into considerations. It is shown [2] that we cannot rely on timing information obtained from the static program text since we assume that the processor may be shared among the various processes; obviously, we need to know something about
the strategy by which it serves individual processes and, in particular, whether or not certain processes are served with priority.

Our approach is as follows:
- Defining the computation model
- Specifying timing properties using temporal logic
- Develop a model checker to get diagnosis about timing properties of the system.

The present work is limited to the two first steps. The model checking may be carried by using an extended version of existing ones (such as EMC, MEC, XESAR).

A fundamental assumption is the controllability/observability (for continuous systems) of the physical process. For discrete event systems, the assumption is implicitly done in the specification step when using Petri nets for example. Such properties can be handled using state space theory for continuous systems and Petri nets theory for discrete event systems. Almost all previous work were concerned with computer processes only in their models. The physical process is being considered for two reasons:

i. A global computation model for real-time systems cannot be completely expressive without one main component as the external environment; this is a common argument.

ii. There are many information to be gained when considered the external environment. The time constants corresponding to the dynamic modes are as fundamental as the time execution of a program we saw in dealing with time at the instruction level.

iii. Processes interact through messages; such interaction is formalized using CSP constructs (message passing with null size buffer). In this type of communication, two processes must be both at their respective synchronization points. For two communicating computer processes we have: \( cp_i \rightarrow cc_1 \) and \( cp_j \rightarrow cc_1 \), this means that processes \( i \) (sender), \( j \) (receiver) communicate on channel 1. We can also communication between a physical process and computer process expressed as \( cp_i \rightarrow cc_2 \)and \( cp_j \rightarrow cc_2 \).

The model of real-time message passing is a standard one [16]:

\[
M: TIME \rightarrow COM \times WTS \times WTR \quad (COM: \text{Communicating channels}, \ WTS: \text{Wait to send}, \ WTR: \text{Wait to receive})
\]

\( M .com \): set of communicating channels \( M .wtr \): set of channels waiting to receive

\( M .wts \): set of channels waiting to send

The critical point is when a \( pp_i \) is waiting to send or waiting to receive. These two cases have to be avoided. In order to get around the problem, we categorise the timing interaction into four cases, approach proposed in [17]. The timing parameters, denoted as \( t_{pp}, t_{cp}, t_{pc}, t_{cc} \) are maximum time allowed if the four following cases:

\( t_{pp}: \) two stimulus (from pp to cp in two successive steps): Data acquisition process
\( t_{cp}: \) Between an action and corresponding stimulus (From cp to pp and pp to cp)
\( t_{pc}: \) between a stimulus and corresponding action (from pp to cp and cp to pp)
\( t_{cc}: \) Between two actions (from p to pp in two successive steps): \( cp_i \rightarrow cc_1 \) and \( cp_j \rightarrow cc_1 \)

It appears that the most important parameter related directly to the timing constraints (or has more weight on it) is: \( t_{pc} \)

The physical process: In order to illustrate the mapping of the physical system into a process, we make the assumption that the system is linear and hence can be described using state space theory as

\[
X = A.X + B.U \quad \text{and} \quad Y = C.X
\]

\( U \) being the control vector, \( X \) state vector.
A, B, C matrices corresponding to the physical process parameters  

The matrix A defines the dynamics of the system (eigenvalues of A correspond the time constants inverses of system dynamic modes). A supplementary and important assumption is that all system modes are observable and controllable. The system can be represented by a state transition system. The state being the $X$ vector; we consider this statement with great attention because of the state explosion problem; we may consider some finite intervals as to limit the number of possible values that can be taken of the elements of $X$. It was shown that both $p_{p_i}$ and $c_{p_i}$ can be modelled using a state system transition. The interaction is modelled using real-time message passing contract.

V. SPECIFYING TIMING PROPERTIES

**Timing constraints:** Let now give some basic definitions:

*Definition 1:* Computer process sends message through channels on a CSP based mechanism and a sensor synchronization is communication event between $c_{p_i}$ as a receiver and $p_{p_j}$ as a sender.

*Definition 2:* An actuator (action) synchronization is a communication event between $c_{p_i}$ as a sender and $p_{p_j}$ as a receiver.

*Definition 3:* A send/respond time constraint from precedent definition as the time imposed on the duration between sensor synchronization and actuator synchronization such that the actuator synchronization is triggered following the sensor synchronization.

This means that the sensor synchronization is an event causing a transition on the cp state, hence delivering an action causing an actuator synchronization.

**Timing properties and timing specification:** Our general interest is the real-time programs has the right properties. By decomposing each process into states, we make use of the available results to check:

- Fairness
- Deadlock freeness
- Timing properties

For similar approaches, see [18,19]; for a survey on general approaches see [20].

**Case study:** Let take a physical process consisting of two robots for manipulating manufactured parts and two spaces, one where the parts are stored, the other where the parts to be put for assembly.

Let define a set of activities $A = \{\text{wait}, \text{pick}, \text{put}, \text{move}\}$

$a_i$ being the variable, current state of robot $I$

Data variable $p_i$, the number of parts in space $i$

Communicating channels: output channels OR$i$ and OS$i$ to send $a_i$ and $d_i$ to the control process control (respectively from robot and space). Input channels $I_i$ to receive $s_t$ and as from the control process to enter the store or the assembly space. Using CSP, the communication semantics can be described as: OR$i$!ai means robot I is sending its state ai on channel OR$i$ and in the same way $I_i$?st means the robot is receiving an order $s_t$ on channel $I_i$ (to enter the store space). The dynamics of the physical process can described using Petri nets, let give a partial description as follows:
The control process can be described as a sequence of executions, each sequence is itself a sequence of states and events $\Sigma = s_0s_1s_2 \ldots (s_k, k^{th} \ sequence \ in \ \Sigma)$.

If we use temporal logic formula, ie if $w$ is a state formula, we denote by $\text{Sat } w$, that is the formula is satisfied in $s_0$. Having $w_1, w_2$ temporal formula we can express some temporal properties $w_1 \land t = T => \Diamond(w_2 \land t \leq T + n)$: If $w_1$ is true now and the time is $T$ and in $n$ time-units (clock pulses), the formula $w_2$ must become true. Such temporal formula will be used to make sure that some timing constraints are met.

Another interesting aspect in temporal logic is to set a safety formula for mutual exclusion in the case study:

$$a_1 \in \{\text{move, wait}\} \ or \ x_2 \in \{\text{move, wait}\};$$

this invariant property must be always true in order to avoid collision.

**CONCLUSION**

It was presented an approach to model real-time systems and how to deal with its properties. The main contribution is the taking into account both types of processes and that physical processes can be modelled using transition systems, a common model for computer and physical processes. Reasoning about time using temporal logic was outlined; an extension of logic to use physical time is possible.

**REFERENCES**

6. A.C. Shaw: Reasoning about time in higher-level language software. MASI rep No. 188, Univ Paris VI, July 87.
12. A. Pnueli: The temporal logic of programs. Proc. 18\textsuperscript{th} symposium foundations of comp science, Oct 77.