Proceedings of Junior Researcher Workshop on Real-Time Computing

Liliana Cucu

Proceedings of the

Junior Researcher Workshop on Real-Time Computing 2007

JRWRTC'07

LORIA, Nancy, France

29-30 March 2007

http://srtsjr07.loria.fr

Table of contents

Junior Researcher Workshop on Real-Time Computing 2007

JRWRTC'07

Message from the conference chairs

JRWRTC'07

It is our great pleasure to welcome you to Junior Researcher Workshop on Real-Time Computing 2007, which is held conjointly with the 15th conference on Real-Time and Network Systems (RTNS'07). The first successful edition was held conjointly with the French Summer School on Real-Time Systems 2005 (http://etr05.loria.fr).

Its main purpose is to bring together junior researchers (Ph.D. students, postdoc, ...) working on real-time systems. This workshop is a good opportunity to present our works and share ideas with other junior researchers and not only, since we will present our work to the audience of the main conference.

In response to the call for papers, 14 papers were submitted and the international Program Committee provided detailed comments to improve these work-in-progress papers. We hope that our remarks will help the authors to submit improved long versions of theirs papers to the next edition of RTNS.

JRWRTC'07 would not be possible without the generous contribution of many volunteers and institutions which supported RTNS'07. First, we would like to express our sincere gratitude to our sponsors for their financial support : Conseil Général de Meuthe et Moselle, Conseil Régional de Lorraine, Communauté Urbaine du Grand Nancy, Université Henri Poincaré, Institut National Polytechnique de Lorraine and LORIA and INRIA Lorraine. We are thankful to Pascal Mary for authorizing us to use his nice picture of "place Stanislas" for the proceedings and web site (many others are available at www.laplusbelleplacedumonde.com). Finally, we are most grateful to the local organizing committee that helped to organize the conference. Let us hope for a bright future in the RTNS conference series !

Liliana CUCU, LORIA-INPL
Workshop chair

Program Committee

JRWRTC'07

Ben Hedia Belgacem, CITI Laboratory-INSA, France

Najet Boughami, LORIA-INPL, France

Bernard Chauvière, Université de Poitiers/ENSMA, France

Chung Shue Chen, LORIA-CNRS, France

Jean-François Deverge, IRISA, France

Pascal Fontaine, LORIA-Nancy 2, France

Nathan Fisher, University of North Carolina, USA

Mathieu Grenier, LORIA-INPL, France

Patrick Meumeu, INRIA Rocquencourt, France

Nuno Pereira, Polytechnic Institute of Porto, Portugal

Frédéric Ridouard, IRIT-ENSEEIHT, France

Louis-Marie Traonouez, Institut de Recherche en Communications

et en Cybernétique de Nantes, France

Online Testing of Real-Time systems

Noureddine ADJIR[1, 2]
[1] ENSICA, Université de Toulouse, 1 place Emile Blouin- 31056 Toulouse Cedex 5, France. {nadjir, pdss}@ensica.fr

Pierre de SAQUI-SANNES[1, 2]
[2] LAAS-CNRS, Université de Toulouse, 7 avenue du Colonel Roche- 31077 Toulouse Cedex 4, France.

Abstract

Labeled Time Petri nets with Stopwatches and Priorities (LPrSwTPN) are used for timed test sequences generation. LPrSwTPN enable modelling of system/environment interactions, time constraints and suspend/resume operations. Assuming the modelled systems are non deterministic and partly non observable, the paper proposes a test generation approach based on a relativized conformance relation named rswtioco. A test generation algorithm is presented. It implements an online testing policy.

1. Introduction

Model-based test generation of real-time systems requires a special approach because of reactivity, timeliness, and suspension/resumption actions. Papers on timed test sequence generation (conformance testing) have ignored the possibility for real-time systems to suspend their current behaviour and to resume later on. To overcome that limitation, this paper addresses timed test sequence generation from Labeled Time Petri Nets with Stopwatches and Priorities (*LPrSwTPN*). Among various contributions of the paper, a relativized conformance relation named *rswtioco* is defined and a test generation algorithm is presented. The algorithm proposed implements an online (on the fly) policy. The proposed approach is illustrated on an example.

2. Real-time systems modeling

Real-time systems are modelled using Labelled Stopwatch Time Petri Nets with priorities (*LPrSwTPN*). *SwTPN* [3] extend Merlin's TPN [6] by stopwatch arcs that control the progress of transitions to express suspension and resumption of actions. *TPN*'s extend *PN*'s by associating a temporal interval [Tmin, Tmax] with each transition. Tmin and Tmax respectively denote the earliest and latest firing times of the transition (after the latter was enabled). *PrSwTPN* increase the modelling power of *SwTPN* by adding a priority relation to the transitions. We also add a labelling function to transitions. Since we address reactive systems testing, we assume the existence of a set of actions *A* partitioned in two disjoints subsets: *inputs actions* A_{in} and *outputs actions* A_{out}. Inputs are the stimuli received from the outside environment. Outputs are the actions sent by the system to the environment. The internal operations (non observed behaviour) of a system are modelled by a specific action τ $(\tau \notin A)$.

A *LPrSwTPN* is a tuple $N = \langle P, T, \mathbf{Pre}, \mathbf{Post}, \boldsymbol{Sw}, Pr, m_0, Is, A_\tau, L \rangle$ where:

- $\langle P, T, \mathbf{Pre}, \mathbf{Post}, m_0 \rangle$ is a *Petri net* with *places P*, *transitions T*, *initial marking* $m_0 : P \to \mathbf{N}^+$ and *precondition* and *post-condition functions* **Pre, Post** $: T \to P \to \mathbf{N}$.

- $I_s : T \to \mathbf{I}^+$ is the *Static Interval Function* which associates a temporal interval with each transition in the net. The rational $\downarrow I_s(t)$ and $\uparrow I_s(t)$ denote the *static earliest firing time* and *latest firing time* of *t*, respectively.

- $Pr \subseteq T \times T$ is the *priority relation*. It is irreflexive, asymmetric and transitive.

- *A* is a finite set of *actions*, or *labels*, not containing the internal action τ.

- $L : T \to A_\tau$ is the *labelling* function.

- $\boldsymbol{Sw} : T \to P \to \mathbf{N}$ is the *stopwatch incidence function*. Values greater than 0 are represented by special arcs, called *stopwatch arcs*, possibly weighted, and characterized by square shaped arrows. Figure 1(a) depicts a *LPrSwTPN*. The arc from place p_2 to transition t_8 is a stopwatch arc of weight 1.

A *marking* is a function $m : P \to \mathbf{N}^+$. A *transition t* is *enabled* at *marking m* iff $m \geq \mathbf{Pre}(t)$. In addition, an enabled transition is "*active*" iff $m \geq \mathbf{Sw}(t)$. Otherwise, it is "*suspended*". The sets of enabled, active and suspended transitions at *m* are defined respectively by: $En(m) = \{t \mid \mathbf{Pre}(t) \leq m\}$, $Ac(m) = \{t \mid t \in En(m) \wedge m \geq Sw(t)\}$, $Su(m) = \{t \mid t \in En(m) \wedge m < Sw(t)\}$. The predicate specifying that a transition *t* is newly enabled by the firing of t' at *m* is defined by $\uparrow(t, m, t') = t \in En(m - \mathbf{Pre}(t') + \mathbf{Post}(t')) \wedge ((t \notin En(m - \mathbf{Pre}t(t))) \vee t = t')$. $(t_1, t_2) \in Pr$ is written $t_1 \succ t_2$ or $t_2 \prec t_1$ (t_1 has priority over t_2).

The semantics of an *LPrSwTPN N* is a timed input/output transition system (*TIOTS*) $\mathcal{Q}_N = (Q, (m_0, I_s[En(m_0)]), A_{in}, A_{out}, \rightarrow)$ where *Q* is the set of states. A

state is a pair $(m, I) \in \mathrm{N}^P \times \mathbf{I}^+$ in which m is a marking and $I : T \to \mathbf{I}^+$, the *interval function*, associates a temporal interval with every transition $t \in En(\mathbf{m})$. $(m_0, I_s[En(m_0)])$ is the initial state. $I_s[En(m_0)]$ is the restriction of I_s to transitions $En(m_0)$. The transition relation $\to$ corresponds to two kinds of transitions: discrete transitions are the result of firings transitions of the net and continuous transitions (or delay) are the result of elapsing of time. We have:

– $(m, I) \xrightarrow{L(t)} (m', I')$ iff $t \in T, L(t) \in A_\tau$ and

1. $t \in En(\mathbf{m}) \wedge t \in Ac(\mathbf{m})$
2. $0 \in I(t)$
3. $(\forall k \in T)(k \in En(m) \wedge k \in Ac(m) \wedge 0 \in I(k) \Rightarrow \neg(k \succ t))$
4. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
5. $(\forall k \in En(m'))(I'(k) = \mathbf{if}\ \uparrow En(k, m, t)\ \mathbf{then}\ I_s(k)\ \mathbf{else}\ I(k))$

– $(m, I) \xrightarrow{d} (m, I')$ **iff** $d \in R_{\geq 0}$ and

6. $(\forall k \in En(\mathbf{m}))(\mathbf{if}\ k \in Ac(m) \Rightarrow (\forall d' \leq d)(d' \leq \uparrow I(k)))$
7. $(\forall k \in En(m))\left(I'(k) = \mathbf{if}\ k \in Ac(m)\ \text{then}\ I(k) \dot{-} d\ \mathbf{else}\ I(k)\right)$

Transition t may fire from (m, I) if (1) it is enabled and active at m, (2) it is fireable instantly, and (3) no transition with higher priority satisfies these conditions. (4) is the standard marking transformation. From (5), the transitions not in conflict with t retain their firing intervals, whereas those newly enabled are assigned their static intervals. By (7), all firing domains of active transitions are synchronously shifted towards the origin as time elapse, and truncated to nonnegative values. (6) prevents time from elapsing as soon some enabled and active transition reaches its latest firing time.

The state space of a *LPrSwTPN* may be infinite. Finitely representing state spaces involves grouping some sets of states. We use the grouping method introduced in [1]. It groups some particular sets of states into state classes and preserve marking and traces. A state class is a pair (*m*, *D*) where *m* is a marking and *D* is a firing domain of $En(m)$. The domain *D* is described by a system of linear inequalities $W \underline{\phi} \leq \underline{\omega}$. The initial state is $c_\varepsilon = \left(m_0, \left\{\downarrow I_s[En(m_0)] \leq \underline{\phi}_t \leq \overline{\uparrow I_s[En(m_0)]}\right\}\right)$. The symbolic transition relation between state classes is:

– $(m, D) \xrightarrow{L(t)} (m', D')$ **iff** $t \in T \wedge L(t) \in A_\tau$ and

1. $t \in En(\mathbf{m}) \wedge t \in Ac(\mathbf{m})$
2. $0 \in \left\{\underline{\phi}_t\right\}\left(\left\{\underline{\phi}_t\right\}\text{is the set of solutions}\right)$
3. $(\forall k \in T)\left(k \in En(m) \wedge k \in Ac(m) \wedge 0 \in \left\{\underline{\phi}_k\right\} \Rightarrow \neg(k \succ t)\right)$
4. $m' = m - \mathbf{Pre}(t) + \mathbf{Post}(t)$
5. $(\forall k \in En(m'))\left(\underline{\phi'}_k = \mathbf{if}\ \uparrow(k, m, t)\ \mathbf{then}\ \underline{\phi'}_k \in I_s(k)\ \mathbf{else}\ \underline{\phi}_k\right)$
6. the variables $\underline{\phi}$ are eliminated

– $(m, D) \xrightarrow{d} (m, D')$ iff $d \in R_{\geq 0}$ and

7. $(\forall t \in En(\mathbf{m}))\left(\mathbf{if}\ t \in Ac(m) \Rightarrow (\forall d' \leq d)\left(d' \leq \mathbf{max}\left\{\underline{\phi}_t\right\}\right)\right)$
8. $(\forall t \in En(t)) \Rightarrow \underline{\phi'}_t = \mathbf{if}\ t \in Ac(m)\ \mathbf{then}\ \underline{\phi}_t - d\ \mathbf{else}\ \underline{\phi}_t$

The visible behaviour of $\mathscr{C}_N$ is described by the relation $\Rightarrow$ ($\Rightarrow \in (A \cup R_{\geq 0})^*$).

Let $a, a_0, ..., a_k \in A, \alpha_0, ..., \alpha_n \in A_\tau, \alpha \in A_\tau \cup R_{\geq 0}$ and $d_0, ..., d_{n+1} \in R_{\geq O}$. We have: $q \overset{a}{\Rightarrow} q'$ iff $q \xrightarrow{\tau} * \xrightarrow{a} \xrightarrow{\tau} * q'$ and $q \overset{d}{\Rightarrow} q'$ iff $q \xrightarrow{\tau} * \xrightarrow{d_0} \xrightarrow{\tau} * ... \xrightarrow{\tau} * \xrightarrow{d_n} \xrightarrow{\tau} * q'$ where $d = d_0 + d_1 + ... + d_n$. The relation $\Rightarrow$ is extended to sequences of delays and actions.

An observable timed trace is the timed word $\sigma \in (A \cup R_{\geq 0})^*$ *which is of the form* $\sigma = d_0 a_0 ... a_k d_{k+1}$. We write $q \xrightarrow{\alpha} \Rightarrow$ iff $q \xrightarrow{\alpha} q'$ and $\xrightarrow{\alpha} q$ iff $q' \xrightarrow{\alpha} q$ for some q'. We define the timed observable traces of a state q as:

$$\mathbf{TTr}(q) = \left\{\sigma \in (A \cup R_{\geq 0})^* \mid q \overset{\sigma}{\Rightarrow}\right\}$$

For a state q, and subset $\mathrm{Q}' \subseteq \mathrm{Q}$ and a timed trace σ, q **after** σ is the set of states which can be reached after σ:

$$q\ \mathbf{after}\ \sigma = \left\{q' \mid q \overset{\sigma}{\Rightarrow} q'\right\}, \quad \mathrm{Q}'\ \mathbf{after}\ \sigma = \bigcup_{q \in \mathrm{Q}'} q\ \mathbf{after}\ \sigma$$

We distinguish between two types of outputs. First, outputs in the common sense of the word that we call "active outputs". Second, special outputs that we call "indicators" or "suspended outputs". The latter are issued by the systems to give indications on suspended actions.

The set of observable active outputs or delays that may occur in $q \in \mathrm{Q}' \subseteq \mathrm{Q}$ is defined as:

$$\mathbf{Out}_{aord}(q) = \left\{a \in A_{out} \cup R_{\geq 0} \mid q \overset{a}{\Rightarrow}\right\},$$
$$\mathbf{Out}_{aord}(\mathrm{Q}') = \bigcup_{q \in \mathrm{Q}'} \mathbf{out}_{aord}(q)$$

The set of suspended outputs that can occur in $q \in \mathrm{Q}' \subseteq \mathrm{Q}$ is defined by (*su* is extended to states):

$$\mathbf{Out}_{su}(q) = \left\{a \in su(q) \mid \overset{\alpha}{\Rightarrow} q \wedge \alpha \in A_{out} \cup R_{\geq 0}\right\},$$
$$\mathbf{Out}_{su}(\mathrm{Q}') = \bigcup_{q \in \mathrm{Q}'} \mathbf{out}_{su}(q)$$

3. Online testing

Online testing [5, 7] combines test generation and execution. It determinizes the specification, implicitly, on the fly. Unlike offline testing, where the complete test cases and their verdicts are computed a priori and before their execution, online testing enables to address models with full expressiveness. It indeed enables working with non deterministic specifications. Online testing dramatically lowers the state explosion risk, since only a subset of the states needs to be stored at any point of time. The testing may run for several hours or days, and consequently it may exhibit complex and long test sequences.

4. Relativized conformance relation

The paper considers a Relativized Stopwatch Timed Input/Output Conformance relation (*rswtioco*) which extends the *rtioco* relation [7], itself relying on *tioco* [5]

and Tretman's ioco relations [8]. The motivation behind introduction of *rswtioco*, is to test real-time systems and to take into account their suspend/resume operations. The relation's name includes "sw" by reference to Stopwatch TPN. Unlike papers that limit discussion to Merlin's TPN [6], this paper addresses *LPrSwTPN*. The conformance addressed by the paper is said to be "relativized" since results are obtained for one specific environment. Given a system under test, the test approach does not consider all possible environments. It considers one real operating environment. Furthermore, the environment's constraints are separated from the specification's one (the environment assumptions are taken explicitly into account and separately modelled from the system's constraints). So, modelling the environment explicitly and separately from the system makes it possible to synthesize only those scenarios which are relevant and realistic for the given type of environment. This in turn reduces the number of tests and improves the quality of the test suite [7]. Therefore, conformance between an implementation and its specification is heavily dependent on the environment. Test verdicts obtained for a specific environment remain valid for more restrictive environments.

The *rswtioco* relation does not allow either of usual outputs and indicators to be emitted in advance or on late by the system. Also, this relation brings more information about the non-conformance of a system. So, when the system emits an indicator or an erroneous output that was not expected at that time, the following question may be asked: if that indicator (resp. output) had not been issued at date d, what would have happened at d instead of an indicator (resp. an output) emission: nothing or an output (resp. an indicator)? The proposed *rswtioco* relation makes it possible to answer another question: "does some action *a* resume at the expected date?

The parallel composition of the input enabled and input complete *TIOTS*'s $\mathcal{G}$ and $\mathcal{E}$ that describe the semantics of the specification and an environment forms a closed system $\mathcal{G} \| \mathcal{E}$ in which observable behaviour is defined by a *TIOTS* where the transition relation $\rightarrow$ is defined as:

$$\frac{q \xrightarrow{a} q' \; e \xrightarrow{a} e'}{(q,e) \xrightarrow{a} (q',e')} \qquad \frac{q \xrightarrow{d} q' \; e \xrightarrow{d} e'}{(q,e) \xrightarrow{d} (q',e')}$$

$$\frac{q \xrightarrow{\tau} q'}{(q,e) \xrightarrow{\tau} (q',e')} \qquad \frac{e \xrightarrow{\tau} e'}{(q,e) \xrightarrow{\tau} (q',e')}$$

Given an environment *e*, the *e*-relativized *rswtioco* relation is defined as:

$$q \; rswtioco_e \; t \quad \text{ssi} \quad \forall \sigma \in \mathrm{TTr}(e)$$
$$\mathbf{Out}_{aord}((q,e) \; \mathbf{after} \; \sigma) \subseteq \mathbf{Out}_{aord}((t,e) \; \mathbf{after} \; \sigma) \quad \wedge$$
$$\mathbf{Out}_{su}((q,e) \mathbf{after} (\mathbf{Out}_{aord}((q,e) \; \mathbf{after} \; \sigma)))$$
$$\subseteq \mathbf{Out}_{su}((t,e) \; \mathbf{after} (\mathbf{Out}_{aord}((t,e) \mathbf{after} \; \sigma)))$$

We say that *q* is a correct implementation of the specification *t* under the environment constraints expressed by *e*.

Figure 1(a) depicts a *LPrSwTPN* model for a coffee machine which delivers light and strong coffee, respectively. After he/she inserted coins, the user has to push on a "prepare coffee" button. If he/she pushes the button for less than 30 time units (resp. more than 50 time units) he/she will get a light (resp. strong) coffee. If the button is pushed between 30 and 50 time units, the specification model allows for a non-deterministic choice between light and strong coffee (we assume an implementation will solve that non determinism). The user requesting for strong coffee can take his/her coffee at any time during its preparation and can again put back the cup to resume what remains in the machine, on the condition to not exceed 3 time units. The machine makes internal actions to be reset or to resume the preparation of strong coffee. This service is not allowed for the user requesting light coffee. The right *LPrSwTPN* of Fig. 1 models potential (nice) users of the machine that pay before requesting coffee and take his coffee after its preparation.

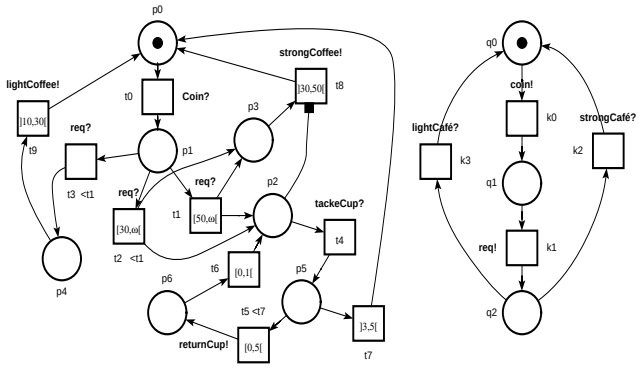


Fig. 1. the specification coffee machine $\mathcal{G}_C\|$ an environment $\mathcal{E}_C$

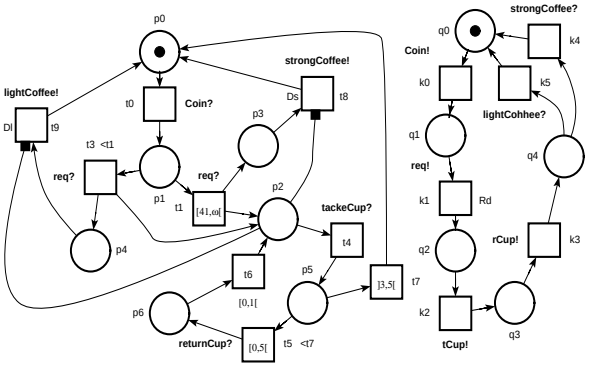


Fig. 2. (a) IUT([Ds, Ds], [Dl, Dl]) ‖ (b) an environment $\mathcal{E}_1$[Rd]

The (deterministic) implementation IUT([Ds, Ds], [Dl, Dl]) in Fig. 2(b) produces light (resp. strong) coffee if the button is pushed less than 40 time units (resp. more than 41 time units). The brewing time of Dl (Ds) time units is to be added to the 40 or more time units. The machine allows all users requesting coffee to take it during its preparation (including those requesting light

coffee). We have IUT([40, 40], [20, 20]) $rswtioco_{\mathcal{E}_C}$ $\mathcal{C}_C$ because $\mathcal{E}_C$ never takes up his cup while the machine is preparing coffee. By contrast, IUT([70, 70], [5, 5]) ~~$rswtioco_{\mathcal{E}_C}$~~ $\mathcal{C}_C$ for two reasons: 1) The IUT has the following timed trace: coin. 30 . req . 5 . lightCoffee whereas $\mathcal{C}_C$ has not, i.e. the IUT may produce light coffee to quickly (no time to insert a cup); 2) the IUT has a trace: coin . 50 . req . 70 not in $\mathcal{C}_C$ meaning that it produces strong coffee too slowly. We have IUT([40, 40], [20, 20]) ~~$rswtioco_{\mathcal{E}_1}$~~ $\mathcal{C}_C$ because it has the timed trace coin . 30 . req . 10 (tackeCup, lightCoffee) . 2 . (returnCup,lightCoffee) . 5 . lightCoffee impossible with $\mathcal{C}_C$. (tackeCup, lightCoffee) means that tackeCup is an active action and lightCoffee is a suspended one. By contrast, IUT([40, 40], [20, 20]) $rswtioco_{\mathcal{E}_1}$ $\mathcal{C}_C$ if $Rd = [60, \infty[$ because $\mathcal{E}_1$ (60) never requests weak coffee.

5. Test generation and execution algorithm

Test generation and execution algorithm
GenExeTest (Q, E, IUT, N). $C := \{c_{\mathcal{E}} = (m_0, D_0)\}$
while $C \neq \phi \wedge iterations \leq N$ **do**
RandomlyChoose(*Action*, *Delay*, *Restart*)
Action : // offer an input to the *IUT*
if $EnvOutput(C) \neq \phi$ **then**
a := *ChooseAction*($EnvOutput(C)$)
send a to the IUT
C := *After*(C, a)
Delay : // wait for an output of the *IUT*
δ := *ChooseDelay*(C)
// Wait δ unit of time for an output o
if $active(o)$ appears at $\delta' \leq \delta$ **then**
C := *After*(C, δ')
if $active(o) \notin ImpOutput(C)$ **then**
return *fail*
if $active(o) \in ImpSuspended(C)$ **then** a is suspended in Q
else C := *After*(C, $active(o)$)
if $suspend(o) \notin ImpSusp(C)$ **then**
return *fail*
for all $a \in suspend(o) - ImpSusp(C)$ **if** $a \in ImpOutput(C)$ **then** a is active in Q
else C := *After*(C,δ) // no output during δ
restart : // reset and restart.
$C := \{c_{\varepsilon} = (m_0, D_0)\}$
Reset *IUT*
if $C=\varnothing$ **then return** *fail* **else return** *pass*

The inputs to the test generation and execution algorithm are two *TIOTS*'s $\mathcal{C} \| \mathcal{E}$ describing the semantics of two *LPrSwTPN*'s, respectively modelling the *IUT* and an environment. This algorithm is based on maintaining the current reachable symbolic state $C \subset Q \times E$ representing all the states that the specification and environment models can possibly occupy after the timed trace observed so far. From this set, one can choose the appropriate test primitive and validate the *IUT* outputs. C initially contains the symbolic state $c_{\mathcal{E}}$. The tester can perform three basic actions: either send an input (an enabled environment output) to the *IUT*, or wait for an output after a delay or still reset the *IUT* and restart. If an output or a delay is observed, the tester verifies if this conforms to the specification. Any illegal occurrence or absence of an output is detected if the set C becomes empty, which happens when the observed trace is not in the specification. The illegal occurrence of a suspended action is detected if it does not belong to *ImpSusp(C).* The function *After* computes the set of states after the execution of a test event from the current states C by using the symbolic technique implemented in TINA [2] adapted to the needs of testing.

6. Conclusion

The paper discusses testing of real-time systems modelled using *LPrSwTPN*. The latter have been selected for their capacity to model suspend/resume operations and for the conciseness of the models. Using an online testing approach makes is possible to handle non determinism and partly observable systems.

The paper introduces *rswtioco*, a new conformance relation which differs from *tioco* because it addresses the constraints captured by the system separately from the ones inherent to the environment. Also, *rswtioco* differs from both *tioco* and *rtioco* because the latter were not defined for suspend/resume operations (i.e. operations where the system's context has to be stored and restored later on). The algorithm proposed in the paper will be implemented in TINA [2].

References

1. Berthomieu B., M. Diaz, "modelling and verification of time dependent systems using time Petri nets", IEEE trans. on software Engineering, 17(3), 1991.
2. Berthomieu B., Ribet P. O., Vernadat F., "The tool TINA -- Construction of Abstract State Spaces for Petri Nets and Time Petri Nets", *I JPR, 42(14)*, July 2004.
3. Berthomieu B., Lime D., Roux O. H., Vernadat F., "Reachability Problems and Abstract State Space for Timed Petri Nets with Stopwatches", To appear 2007.
4. Lin J. C., Ho I., "Generating Real-Time Software Test Cases by Time Petri Nets", *IJCA (EI journal), ACTA Press, U.S.A. 22(3):151-158*, Sept. 2000.
5. Krichen M., Tripakis S., "An Expressive and Implementable Formal Framework for Testing Real-Time Systems", *17th IFIP Intl. TestCom'05*, 2005.
6. Merlin P. M., Farber J., "Recoverability of communication protocols: Implications of a theoretical study", *IEEE Trans. Com., 24(9):1036-1043*, Sept. 1976.
7. Mikucionis M., K. G. Larsen, Brian Nielsen, "T-UPPAAL: Online Model-based Testing of Real-time Systems", *19th IEEE IC ASE, 396-397. Linz, Austria*, Sept. 24, 2004.
8. Tretmans J., "Testing concurrent systems: A formal approach", *IJCM Baeten and S. Mauw, editors, CONCUR'99 – 10th ICCT, 1664: 46–65 of LNC,1999.*

Real-Time Scheduling in a Virtual Machine Environment

Christophe Augier
VirtualLogix
78180 Montigny-le-Bretonneux, France
{*christophe.augier*}@*virtuallogix.com*

Abstract

In a virtualized environment, a real-time operating system (RTOS) can run in parallel with one or more commodity operating systems (OS) with negligible overhead. Real-time properties of the RTOS are guaranteed by giving it and its tasks the highest priority over CPU utilization which can lead to high-latency for the other OS tasks. This simplistic approach to processor resource sharing is not flexible enough when multimedia application are deployed on the commodity OSes. In this paper we present a design that enables a finer grained processor resource management by moving tasks scheduling from the OS level to the virtual machine monitor level.
Keywords: *virtualization, real-time, tasks scheduling*

1. Introduction

Today, consumer electronics are becoming more common and more complex, unifying features of many legacy devices into a single platform. The demand for higher levels of functionality has been steadily increasing while time-to-market has been decreasing. At the OS level, real-time operating systems (RTOS) were initially used on these devices for their small memory footprint and the timing guarantees required by communication protocols. RTOSes were not initially designed to provide rich environments such as mailers or web browsers, and have to evolve in different ways to match today's demands.

One approach to address such demands has been to gradually embed all the needed functionalities (e.g., network stacks, graphical libraries) into the RTOS. However, this approach requires a careful reengineering and lengthy revalidation phases that slow down the development process. Another approach is to adapt existing commodity operating systems to real-time requirements. In this case, the legacy Linux kernel is run above a small real-time executive as a fully preemptable task [3, 4]. Hard real-time tasks can then benefit from any legacy Linux application, with communication being done through Inter-Process Communication (IPC) calls. This approach can be further generalized to a multi-OS platform by means of virtual machine monitors (also called hypervisor) like Xen [7] and VirtualLogix VLX [17] offering their services to guest OSes. In this setting, a legacy real-time OS may run in parallel with one or more General Purpose OSes (GPOS) such as Linux, thus offering seamless colocation to the applications of both worlds. Typically, the scheduling in such an architecture insures that the tasks of the real-time OS always get a higher priority than those of the GPOS instances.

In the embedded and real-time world, applications running on the RTOS and on GPOS instances cooperate to implement the service expected by the device end user. Hence, interaction between embedded applications running in the various OSes is tighter than in traditional virtualization environments. This further raises requirements on the scheduling of the tasks. Rather than having all tasks of the RTOS be executed before tasks of the first GPOS instance themselves executed before those of a second GPOS instance, interleaving of tasks execution is often required. Therefore, such embedded virtualized systems must base the processor resource sharing on applications timing constraints and on data streams flows rather than on the whole OS.

This paper presents a new design to provide a finer grain management of the processor resource in a real-time virtualized environment. We move scheduling decisions out of the guest OSes into a new virtual machine monitor component called Scheduling Virtual Device (SVD). The SVD is in charge of scheduling all the threads regardless of their operating system. SVD can implement any per-thread scheduling policy. Furthermore, the SVD provides an hierarchy of schedulers as promoted by frameworks such as HLS [16] and Bossa [14]. Threads can dynamically be attached to any scheduler based on application requirements, thus providing high flexibility.

The rest of the paper is organized as follows. Section 5 reviews related work. In Section 2 we present our solution based on a SVD and in Section 3 we discuss of the implementation issues. In Section 4 we present preliminary results and we conclude on the future work in Section 6.

2. A virtualized Approach to Task Scheduling

To share resources between several guest OSes, para-virtualization takes a different approach than full-

virtualization. para-virtualization systems such as Denali[20], Xen [7] or VirtualLogix VLX [17] provides no specialized access to devices on the physical system. Instead they exposes virtual devices with a generic interface. For instance, even though there is only one single network card in the hardware system, a driver at the hypervisor level (the virtual driver) will enable the multiplexing of the network resource between the OSes by exposing the same generic interface in each virtual machine (the virtual network card). Each OS will be provided a generic driver allowing itself to interface with the virtual driver. The virtual driver deals with the multiplexing by using different MAC addresses for each OS. Similarly we create a new interface to exchange the appropriate scheduling data between the OSes and the hypervisor.

We propose to virtualize the task schedulers in each OS, by introducing a *scheduler virtual device* (SVD) in the hypervisor. The SVD centralizes the scheduling data and performs all scheduling computations in place of the guest OSes scheduler. Then, each guest OS communicate with the SVD through a generic driver called a *scheduler interface* (SI.) The communication consists in updating the SVD with scheduling events: task creation, blocking, unblocking, destruction, etc. By doing so, the SVD has a global vision of the system scheduling needs.

Scheduler Virtual Device: The SVD implements the scheduling algorithms and provide an OS independent interface for the management of thread scheduling. The SVD is implemented as a virtual device in the hypervisor and thus must run safe code. One bug in the virtual device could lead to the crash of the complete system. As scheduling algorithms manipulate a lot of lists and similar data structures, which is known to be error-prone, we use the Bossa language [14] to write our schedulers.

Scheduler Interface: The SVD defines a generic interface to deal with scheduling events but some OS schedulers may implement finer scheduling events or more complex states transitions. The SI implements the glue to match the OS scheduling events to the SVD generic events. Besides dealing with scheduling events at the OS level, the role of the SI is to manage the scheduling policy's interaction with the applications. For instance an application can ask the SI to be scheduled by a specific scheduling policy or to invoke an interface function defined by the application's scheduling policy. In term of reengineering, the legacy scheduler must be removed and each call to the scheduler must be trapped and redirected to the scheduler driver.

3. Implementation Issues

Our design faces two design issues bond to the use of virtualization.

3.1. Interrupt Queueing

Interrupts are generated by hardware components to notify the software that something happened in the system. For instance a network card issues an interrupt to inform the OS that a packet arrived and is ready to process. In a virtualized environment, interrupts still occur yet it is more complicated to deal with them as there is more than one OS running. Basically an interrupt can be generated even if it is not destined to the OS currently running on the processor. The hypervisor needs to delay the interrupt until the correct OS is executed. This is called *interrupts queueing*.

In a real-time environment interrupts destined for GPOS may occur while the RTOS is running. As the RTOS is given the highest priority, these interrupts will be processed only when the RTOS is idle and a GPOS is selected to run. Depending on the load of the RTOS, it may take some time before an interrupt is processed. Any task waiting for this interrupt can then be woken up and the latency (the time between the interrupt and the task awakening) is actually function of the RTOS load. High latencies are problematic when running time constrained applications on the GPOS.

To decrease the latency, interrupts should be processed as soon as possible and, similarly, the GPOS should be quickly given the CPU. This can be done by preempting the running OS and switching to the one the interrupt is destined for but it is not possible if one wants to guarantee the timing requirements of the RTOS and its tasks.

However, our approach provides a global vision of the RTOS tasks at the hypervisor level. It is then possible to add a task that abstracts the GPOS interrupt servicing as a schedulable entity. This task is waked up by the interrupts destined to the corresponding OS and when the scheduler selects this special task to run, the hypervisor switches to the OS. When the OS finishes to process the interrupt it jumps back to the hypervisor through the SI.

3.2. Scheduling Model Coherency

The scheduling state of a thread can only be blocked, ready or running. By moving task scheduling from the OSes to the virtual machine monitor, task preemption becomes more complicated. Basically preemption can now occur at two levels: at the task level and at the OS level. Therefore two tasks can be stated as running at the same time on two different OSes, but only one will be executed on the CPU. This situation is incoherent regarding the scheduling model which ensures that only one task can be in the state running in the case of a uniprocessor system. This phenomenon breaks the integrity of the scheduling state and should be avoided.

To prevent this situation we change the state of the task running on the preempted OS to ready. We only change the scheduling state of the thread. From a scheduler point of view, the thread is ready to run while for the OS it is running. The new preemption of the thread may cause changes in the scheduling and it may be possible that the OS is given back the cpu to run another thread. We need a way to inform the OS that it needs to preempt the current thread and run the newly elected thread. We generate

a virtual interrupt to inform that an OS must preempt its current running task. The counterpart of this is the introduction of a new interrupt handler in the scheduler device driver. This handler will preempt the running task and call the scheduler virtual device driver to get the next task to run.

4. Preliminary Results

We implemented SVD in VirtualLogix VLX, a virtualization solution for embedded systems, and used a typical configuration as a proof-of-concept. The VLX hypervisor enables to run a real-time operating system along with one or more general purpose operating systems. In our case we set up one instance of Nucleus [11] as the real-time operating system and one of Linux [1]. We implemented support for SVD in both OSes and replaced their scheduler by a single fixed priority scheduler in SVD. The scheduler uses a round-robin algorithm to guarantee fair-sharing of the processor and Nucleus tasks are given more priority than Linux tasks. In this section we evaluate the overhead of virtualizing task scheduling in this configuration.

Virtualization slightly decreases performance and in the case of para-virtualization the overhead is below 5% [7]. Virtualization isolates guest OSes and the hypervisor. This isolation causes an overhead when an hosted operating system and the hypervisor needs to exchange data. When using SVD, all scheduling events require a data exchange. Therefore to measure the overhead of SVD we need to measure the overhead of the scheduling operations. Amongst these operations some are more frequent and thus easier to measure. This is the case during tasks switches. In fact, tasks switches are caused whether by a task which blocks or by a tasks which is unblocked. By measuring the context-switch time from the user tasks it is possible to evaluate the overhead of SVD.

We have written a benchmark program in Nucleus to measure context-switch time. This program creates several threads which by passing a token around them can trigger context-switches. Before sending the token and thus being put in a waiting state, the thread measures the current time. The thread receiving the token – being woken up – measures the time again. Context-switch time is the difference between both measures. Table 1 shows the average time of a context-switch when nucleus is running under VLX with SVD and without. We can see that using SVD increases context-switch time by 5 microseconds which represents an overhead of 83%. While context-switch time is almost doubled, it is still sensibly low.

Next, we measure the context-switch time for Linux thanks to the `lat_ctx` program from the LMBench test suite [2]. This program is very similar in its implementation to the program we developed for nucleus. The results can be found in Table 1 and the overhead of SVD support represents 3.22% of the initial value.

We notice that the overhead times are slightly the same, approximately 5 microseconds but the overhead represents 3% in Linux and 83% in Nucleus. The difference between the two results can be explained by the difference between the two environments. In our configuration, Nucleus does not support memory protection, therefore tasks share the same address space. Under Linux, each tasks (processes) have its own address space and when switching from a process to another, Linux needs to switch memory contexts which means flushing the TLB and the cache. This operation costs a lot of memory cycles and increases context-switch time.

	VLX	VLX w/ SVD	overhead
Linux	145.33	150.02	3.22%
Nucleus	6	11	83.33%

Table 1. Context-switch time measured in μs

Overall the impact on the system performance is small. Actually a system where a context-switch is done every millisecond only adds about 0.5% overhead (after one second, the overhead sums up to 5 milliseconds).

5 Related Work

Our work focuses on the management of the CPU resource in a virtualized environment. We can find different approaches to CPU sharing in the literature. For instance the project Xen [7] chooses a standard way to deal with the processor multiplexing between virtual machines. In Xen, the virtual machine monitor uses a scheduling algorithm originally developed for multimedia task scheduling [8] to schedule virtual machines. A more recent work by Zhang et al. [21] presents an original solution providing a distributed approach to control CPU congestion instead of a centralized one. In this work each guest operating systems adapt its load through a feedback-control model. All these approaches multiplex the processor resource between virtual machines with no consideration of the importance of the applications hosted by the guest operating systems. The finer grain such solutions offer is to set different priorities to each virtual machine. While that may be enough in a desktop or server environment, a finer grain scheduling is needed in real-time embedded systems.

Complete virtualization of hardware systems is not widespread still virtualization initiatives already exist with RTLinux [4] and TimeSys Linux/GPL [5]. Both of these projects make possible the cohabitation of hard real-time tasks and Linux processes through a small executive which provides an environment to execute tasks with hard real-time requirements. Linux is run as the task with least priority and thus Linux processes get CPU time only when all the real-time tasks are finished. The task scheduler may be modified to run Linux at a higher priority, but all Linux processes are still seen by the executive scheduler as a single entity and thus are considered to be of equal priority. It is possible to implement synchronization between the real-time tasks and the Linux processes through lock-

ing mechanisms but this solution would involve a more complex analysis of task scheduling. The easiest way to ensure fine-grained scheduling would be to enable the executive scheduler to communicate with Linux scheduler yet, we do not know of any existing work allowing the co-scheduling of hard real-time tasks and Linux processes.

Other research have shown that implementing specific services in the hypervisor can increase global system performance or enhance some aspects of the system. Chen and Noble were the first to claim that the advantage of virtualization was to enable the implementation of services which would then be perceived by guest operating systems as hardware features [6]. Moreover as these services are implemented in software, they are easier to create and maintain. These services allows to enhance security [13, 10] or resources management [18, 12].

There is a plethora of scheduling frameworks [9, 14, 15, 16, 19] in the literature and we cannot cite them all. The scheduling framework in RED-Linux [19] identifies three paradigms of tasks scheduling in a single framework: priority-driven, time-driven and share-driven scheduling. With a fixed number of scheduling attributes, RED-Linux provides flexible scheduling while keeping the implementation of scheduling policies simple. On the other hand, Bossa [14] offers a framework to develop specific schedulers. Its advantages range from its Domain Specific Language which facilitates the writing of scheduling policies and enables static verifications of the policies to its use of a scheduling hierarchy *à la* HLS [16].

6 Future work and Conclusion

Work in the field of virtualization have shown that virtualization could lead to scheduling issues or CPU misuse. We argued that these issues were globally caused by the lack of feedback from the operating system in virtual machines scheduling. Our approach tries to solves this problem by exposing all the scheduling objects at the same level, independently from their virtual machine. We have presented an approach for breaking the data isolation between virtual machines by migrating the scheduling from the operating systems to the virtual machine monitor.

We implemented a prototype of SVD in VLX and the preliminary results show that the overhead of SVD is negligible. To further prove the flexibility of our approach, we will present a real case study where the use of SVD and virtualization enables the development of a complex application consisting of programs from both the real-time operating system and the general purpose operating system which cooperate to provide a service.

References

[1] Linux. Online at http://www.kernel.org.

[2] LMBench. Online at http://www.bitmover.com/lmbench/.

[3] Rtai. Online at http://www.rtai.org.

[4] Rtlinux. Online at http://www.fsmlabs.com.

[5] TimeSys Linux/RT. Online at http://www.timesys.com.

[6] P. M. Chen and B. D. Noble. When Virtual Is Better Than Real. *hotos*, 00:0133, 2001.

[7] B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, I. Pratt, A. Warfield, P. Barham, and R. Neugebauer. Xen and the Art of Virtualization. In *Proceedings of the ACM Symposium on Operating Systems Principles*, October 2003.

[8] K. J. Duda and D. R. Cheriton. Borrowed-virtual-time (BVT) scheduling: supporting latency-sensitive threads in a general-purpose schedular. In *Symposium on Operating Systems Principles*, pages 261–276, 1999.

[9] A. Daz, I. Ripoll, A. Crespo, and P. Balbastre. A New Application-Defined Scheduling Implementation in RTLinux. In *Sixth Real-Time Linux Workshop*, pages 175–181. Peter Wurmsdobler, 2004.

[10] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection based Architecture for Intrusion Detection. In *Proceedings of the Network and Distributed Systems Security Symposium*, Feb. 2003.

[11] M. Graphics. Nucleus. Online at http://www.mentor.com.

[12] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Antfarm: Tracking Processes in a Virtual Machines Environment. In *Proceedings of the USENIX 2006 Annual Technical Conference*, June 2006.

[13] A. Joshi, S. T. King, G. W. Dunlap, and P. Chen. Detecting past and present intrusions through vulnerability-specific predicates. In *Proceedings of the 20th ACM Symposium on Operating Systems Principles (SOSP '05)*, pages 91–104, Brighton, United Kingdom, Oct. 2005.

[14] G. Muller, J. L. Lawall, and H. Duchesne. A Framework for Simplifying the Development of Kernel Schedulers: Design and Performance Evaluation. In *HASE 2005 - High Assurance Systems Engineering Conference*, Heidelberg, Germany, Oct. 2005.

[15] S. Oikawa and R. Rajkumar. Portable RK: A Portable Resource Kernel for Guaranteed and Enforced Timing Behavior. In *RTAS '99: Proceedings of the Fifth IEEE Real-Time Technology and Applications Symposium*, page 111, Washington, DC, USA, 1999. IEEE Computer Society.

[16] J. Regehr and J. A. Stankovic. HLS: A Framework for Composing Soft Real-Time Schedulers. In *Proceedings of the 22nd IEEE Real-Time Systems Symposium (RTSS 2001)*, pages 3–14, London, UK, Dec. 2001.

[17] VirtualLogix Inc. VirtualLogix VLX. Online at http://www.virtuallogix.com.

[18] VMWare Inc. VMWare VMPlayer. Online at http://www.vmware.com.

[19] Y.-C. Wang and K.-J. Lin. Implementing a General Real-Time Scheduling Framework in the RED-Linux Real-Time Kernel. In *IEEE Real-Time Systems Symposium*, pages 246–255, 1999.

[20] A. Whitaker, M. Shaw, and S. Gribble. Scale and Performance in the Denali Isolation Kernel. In *Proceedings of the Fifth Symposium on Operating System Design and Implementation*, December 2002.

[21] Y. Zhang, A. Bestavros, M. Guirguis, I. Matta, and R. West. Friendly Virtual Machines: leveraging a feedback-control model for application adaptation. In *VEE '05: Proceedings of the 1st ACM/USENIX international conference on Virtual execution environments*, 2005.

WCET Computation on Software Components by Partial Static Analysis

*C. Ballabriga, H. Cassé, P. Sainrat**

{ballabri, casse, sainrat}@irit.fr

IRIT - Université de Toulouse – France

* HiPEAC European Network of Excellence

Abstract: *The current Worst Case Execution Time (WCET) computation methods are designed to be used on a whole program. This approach has scalability limitations and can not be applied to programs made up of multiple components: a method to perform partial analysis on components is needed. In this paper, we present a general method to perform partial analysis on components and to compose these partial results to compute the overall WCET. To check the approach, we have implemented and experimented the partial analysis on the behavior prediction of direct-mapped instruction cache. The experimentation shows big speedup in analysis computation time with an almost negligible growth of pessimism.*

Keywords: Real-time, instruction cache analysis, WCET, partial static analysis, components, abstract interpretation.

1. Introduction

Hard real-time systems are composed of tasks that must imperatively meet deadlines constraints. To check this, scheduling analysis, based on tasks WCET, is used. Computation of the WCET by static analysis provides proven WCET estimation. This approach may be decomposed in two phases:

- the control flow of the task (program path analysis),
- the hardware which the task will run on (architecture model)

Each phase requires a bunch of computation to handle accurately the execution flow of the program and the hardware model. As real-time programs becomes bigger and bigger, two problems arises: (1) WCET computation time will increase exponentially, (2) the use of software components will make ineffective the current techniques.

In this paper, we propose to perform partial analyses on the program (1) to reduce the computation time by factorizing the performed analyses and (2) to provide functions summarizing the content of components for the WCET computation. We apply the approach to the case of direct-mapped instructions caches.

The next section explores in details the partial analysis problem that is applied in the third section to direct-mapped instruction cache. In the fourth section, we show experimentation of our method. Fifth section presents the related works and we conclude in the last section.

2. Problem definition

Currently, the WCET computation methods are designed to be used on a whole program. However, there is some drawbacks to this approach. First, the analyses used for WCET computation usually run in exponential time with respect to program size, and embedded real-time programs are getting bigger and bigger. Second, there is a problem when the program to analyze depends on external components developed by third-parties (for example, programs using libraries) whose sources are not available: it is not possible to compute the WCET of the whole program because of lack of information on the components. This problem will become more and more important as embedded and real-time industry will use more and more Component Off The Shelf (COTS).

Both problems suggest to change the current analysis practice: it is no longer possible to do the analysis all over the program. We need to find a way to do partial WCET analysis on parts of the program, and to compose the partial results into a global WCET for the whole program.

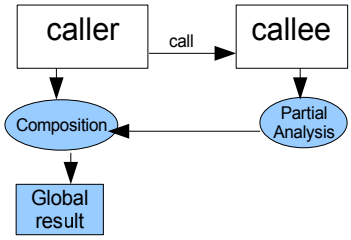


Figure 1: overview of the partial analysis

Figure 1 represents the partial analysis and composition of a program consisting of a function calling another function located in an external component. The partial analyzer takes the code of `callee` and produces a partial result. The composer takes the partial result, and the code of `caller`, to produce a global result, without accessing `callee` code.

In this paper we will consider the case described by the figure: a main program using an external component consisting of only one function. In the case of the instruction cache analysis, more complex cases can be processed similarly, and will be addressed in future work.

We need to define, for each analysis participating in the WCET computation, information needed in the partial results, the method to produce this data, and the method for integrating the partial results into the analysis of the whole program.

There are two main issues when considering the partial analysis:

(1) how to influence the analysis of the main program with the content of the component?
(2) how to make the analysis result of the component dependent on its call context?

To address (1), we describe a *transfer* function, which represents the effect of the component code on the analysis of the whole program. To address (2), we describe a *summary* function, which represents the analysis results for the component, according to the calling context (state before the function call). The method to compute these functions is specific to the particular analysis we want to do.

Once the transfer and summary functions associated with the component are available, we integrate them into the analysis of the whole program. This is done in two steps:

- Step 1: with the transfer function available, we can do the analysis on the main program without having to access to the component code. During the analysis, the component is represented by the transfer function.
- Step 2: once the analysis on the main program is done, we have access to the calling context of the component.

We can use this context, together with the summary function, to get the actual analysis results for the component.

Although we have not experimented nested function call, this approach may be easily adapted. First, the transfer functions may composed in a straight forward way. Yet, some additional computations need to be performed for the summary function to get a sound context before each nested function call.

3. Instruction cache analysis

This section shows how to apply the general approach described previously to the instruction cache analysis. The cache is a fast and small memory used to store a partial copy of the main memory, which speeds up data accesses. When a memory access is performed, either the data is present in the cache, resulting in a fast access called a *hit*, or it must be retrieved from memory, resulting in a slow access called a *miss*. The cache is divided into fixed-size *lines*, and the main memory is divided into *cache blocks* of the same size than cache lines. Each cache block from memory matches a single line. In case of miss, the whole cache block containing the target location is loaded into the matching line. In the direct-mapped cache, each line contains only one block, while in the A-way associative cache, each line contains A blocks.

3.1. Presentation of the analysis of a single component

To predict instruction cache behavior, we have used the 3 analyses (*Must*, *May*, and *Persistence*) from [2, 8], restricted to direct-mapped cache, by considering that a direct-mapped cache is a A-way associative cache where A = 1.

These analyses are based on the techniques of abstract interpretation [5], and are performed on a Control Flow Graph (CFG), composed of Basic Blocks (bb). They work by computing *abstract cache states* (ACS) before and after each basic block using two functions:

- the Update function computes the output ACS from the input ACS of a basic block, that is, the effect of the basic block on the ACS;
- the Join function merges the input ACS of a basic block that has several predecessors in the CFG.

We call $ACS^{\text{in}}_{may}(bb)$, $ACS^{\text{in}}_{must}(bb)$ and $ACS^{\text{in}}_{pers}(bb)$ the input ACS for, respectively, the *May*, *Must* and *Persistence* analyses. The syntax $ACS(bb)(l)$ allows to get the content of a specific line of the ACS.

The *May* ACS gives the set of blocks which may be in the cache. The *Must* ACS gives the set of blocks which must definitely be in the cache. The *Persistence* ACS maps two sets ($ACS^{\text{in}}_{pers,0}(bb)(l)$ and $ACS^{\text{in}}_{pers,1}(bb)(l)$) to each cache line. Both sets contains blocks that may have been loaded in the cache. While the blocks of the latter set may have been wiped out, the blocks of the former must be still in the cache.

The ACS resulting from the analyses will be used to determine basic block categories. For the sake of simplicity, we consider a CFG projected on the cache blocks (i.e. each basic block is split according to cache block boundaries). Now, since a basic block of the projected CFG cannot span multiple cache blocks, we can assign a single category to each basic block.

Let *cb* be the cache block containing the basic block *bb*, and *l* its cache line. The following array shows the process to assign a category to each basic block:

Condition to test	*Category*
$cb \in ACS^{\text{in}}_{must}(bb)(l)$	Always-Hit (AH)
$cb \in ACS^{\text{in}}_{may}(bb)(l) \wedge cb \notin ACS^{\text{in}}_{pers,1}(bb)(l)$	First-Miss (FM)
$cb \notin ACS^{\text{in}}_{may}(bb)(l)$	Always-Miss (AM)
else	Non-Classified (NC)[1]

Table 1: Computation of the category

3.2. The transfer function

The transfer function describes the effect of `callee` on the analysis of the main program. In the case of the cache analysis, the `callee` transfer function takes the ACS from the caller, and computes the ACS at `callee` exit. We call ACS^{entry}_{xxx} and ACS^{exit}_{xxx} the ACS at `callee` entry and exit (where *xxx* stands for one of *may*, *must*, or *pers*).

Independently of the particular cache analysis, there are two effects that must be taken into account to build the transfer function. First, cache blocks of `callee` may appear in the ACS at `callee` exit, independently of the ACS at `callee` entry. This effect is handled by executing the analysis on `callee` with an empty entry ACS, and the result is retrieved from ACS^{exit}_{xxx}.

Second, some cache blocks that were present in the ACS at `callee` entry may be wiped out by `callee`. This effect is handled by the *damage update* function. This function, of type $ACS \rightarrow ACS$, takes an ACS, and returns the *damaged* ACS, according to the cache blocks which may have been replaced by `callee` cache blocks. It is different from the transfer function: the damage update function does not take into account the cache blocks belonging to `callee` which appear on `callee` exit ACS.

Whatever the performed analysis, the transfer function can be expressed by the following formula:

$$transfer(ACS) = Join(DamageUpdate(ACS), ACS^{exit}_{xxx})$$

In the following, we will see how to apply this general method to the transfer functions of the *May*, *Must*, and *Persistence* analyses.

In the case of the *May* analysis, the damage update is deduced from a cache damage analysis. This analysis builds damage information that represents, for each line, the list of cache blocks which would definitely be replaced if they were present in the cache at the beginning of `callee`. For example, if $cb \in damage^{\text{in}}_{may}(bb)$, all the paths from `callee` entry to basic block *bb* remove *cb* from the cache.

We call $line_{cb}$ the cache line of the cache block *cb*. The Update function of this analysis is defined as follows:

$$\forall l \neq line_{cb}, Update^{damage}_{may}(damage, cb)(l) = damage(l)$$
$$Update^{damage}_{may}(damage, cb)(line_{cb}) = damage(line_{cb}) \cup \{cb' / line_{cb} = l\} - \{cb\}$$

The Join function is defined like this:

$$Join^{damage}_{may}(damage1, damage2) = damage' \;/\; \forall l, damage'(l) = damage1(l) \cap damage2(l)$$

Let call $damage^{\text{exit}}_{may}$ the damage information at `callee` exit. Using the result of the analysis, the following damage update is defined:

$$\forall l, DamageUpdate_{may}(ACS)(l) = ACS(l) - damage^{exit}_{may}(l)$$

The transfer function is then created as explained previously.

The *Must* transfer function works in the same way as the

1 We have conservatively considered NC category as an Always-Miss.

May transfer function, by doing a *Must* version of the cache damage analysis, and of the damage update function. The only difference is that, in the $Join_{must}^{damage}$ function, a union of damages is done, instead of the intersection.

The transfer function for the *Persistence* analysis is built a bit differently. It uses the results from the *Must* and *May* damage analysis, and is defined below:

$$DamageUpdate_{pers}(ACS_{pers}) = ACS'_{pers} \;/\; \forall l$$
$$ACS'_{pers,0}(l) = ACS_{pers,0}(l) - damage_{must}^{exit}(l);$$
$$ACS'_{pers,1}(l) = ACS_{pers,1}(l) \cup (ACS_{pers,0}(l) \cap damage_{must}^{exit}(l))$$

3.3. The summary function

The summary function will give a category to each basic block, according to the entry states of `callee`. This function is defined on $ACS_{must} \times ACS_{pers} \rightarrow CATEGORY$ and is associated to each basic block of `callee`.

The results from the *Must* and *Persistence* analysis are required to create the summary function.

Let *cb* be the container cache block of the basic block *bb*, and *l* its cache line. The following table represents the different possible types of summary functions of the basic block *bb*:

Condition to test	summary function
$cb \in ACS_{must}^{in}(bb)(l)$	$\lambda\, ACS_{must}^{entry}\, ACS_{pers}^{entry}\, .\, AH$
$cb \notin damage_{must}^{in}(bb)(l)$	$\lambda\, ACS_{must}^{entry}\, ACS_{pers}^{entry}\, .\, if\ (cb \in ACS_{must}^{entry}(bb)(l))$ $then\ AH\ else\ if\ (cb \notin ACS_{pers,1}^{entry}(l))$ $then\ FM\ else\ AM$
$cb \notin ACS_{pers,1}^{in}(bb)(l)$	$\lambda\, ACS_{must}^{entry}\, ACS_{pers}^{entry}\, .\, if$ $(cb \notin ACS_{pers,0..1}^{entry}(bb)(l))\ then\ FM\ else\ AM$
else	$\lambda\, acs_{must}^{ENTRY}\, acs_{pers}^{ENTRY}\, .\, AM$

Table 2: Computation of the summary function

3.4. Composition of the partial analysis

We explain now how to use the partial analysis results (the transfer functions, and the summary function) into the analysis of the whole program.

Since the base address of the component file containing `callee` is unknown when the partial analysis is performed, we assume that the base address is 0. If we force the linker to allocate the component object file at a base address multiple of the cache block size, we keep the same structure for the mapping of the component cache blocks into lines: if we know the matching line of a location in the original mapping, we can find the new cache line with a simple addition, modulo the number of cache lines.

Therefore, using this simple transformation, we can adapt the transfer and summary functions for `callee`, and use them for the composition in the two steps described in section 3:

1. When the *May*, *Must*, and *Persistence* analyses on `caller` are performed, `callee` CFG is replaced by a *virtual node,* whose Update function is the Transfer function.
2. With the analyses from step 1 performed, we have the ACS at `callee` entry, which the summary function can be applied to, in order to determine `callee` basic block categories.

4. Experimentation

This section describes the environment of experimentation and presents the obtained results.

4.1. Mode of operation

Our method has been implemented using OTAWA [7], a framework dedicated to the development of static analyses for WCET computation. We have performed experimentation on the SNU-RT[2] benchmarks, a set of small programs widely used to test WCET computation and have been focused on the benchmark program containing function calls.

The partial analysis of each function induces a computation overhead due to the summary and transfer functions computation. Yet, this drawback may be balanced by the time gain at composition time if the program performs enough partially-analyzed function calls. To evaluate this issue, we have done 7 tests with different numbers of calls of the partially analyzed function, and different loop nesting depths of these calls.

For each test, we have compared the computation time and the obtained WCET between the partial and the non-partial analyses.

The architecture that the WCET computation is applied to is a simple non-pipelined processor with each instruction taking 5 clock cycles (since the goal of this paper was to show results for the cache analysis, we did not include other effects, like the pipeline), and with a direct-mapped instruction cache of 8 lines with 8-bytes blocks. Such a small cache allows exhibiting more block conflicts with the small programs of the SNU-RT benchmark.

4.2. Results

There are two interesting aspects to examine in the results:

- The comparison between the computed WCET on both analyses, to show that the partial analysis does not add too much pessimism.
- The comparison of the analysis time for both analyses, to show in which cases the partial analysis is faster.

The following tables show the relevant information (P.A. and N.A. stands, respectively, for Partial and Non-partial Analyses):

Bench	Function	N.A. (cycles)	P.A. (cycles)	P.A. / N.A.
fibcall	fib	1100	1100	1
matmul	matmul	18010	18010	1
fft1 (1)	fft1	33950	34420	1,01
fft1 (2)	sin	33950	33950	1
qurt	qurt	19105	19645	1,03
fir (1)	sin	219020	219020	1
fir (2)	gaussian	219020	219020	1

Table 3: WCET comparison

Bench	Function	P.A. / N.A.	Call sites	Max loop depth
fibcall	fib	2	1	0
matmul	matmul	2,25	1	0
fft1 (1)	fft1	1,01	2	0
fft1 (2)	sin	0,41	2	2
qurt	qurt	0,67	3	0
fir (1)	sin	0,75	2	1
fir (2)	gaussian	0,66	2	1

Table 4: call context sensitivity

The table 3 compares the computed WCET, while the table 4 shows the computation time ratio for each test in function of the call context.

The WCET results are coherent: the WCET found by the partial analysis is always equal or slightly greater than the non-partial analysis WCET. Furthermore, only few pessimism is added: the partial analysis adds on average a pessimism of 1% (P.A. / S.A. column)

2 Singapour National University - Real-Time

For the *fibcall* and *matmul* tests, the diagram shows that the partial analysis is slower. This can be explained because, for these tests, the function which was partially analyzed is called only once, so the partial result is only used once: the overhead for computing the partial result is not compensated.

For the *fft1 (1)* test, the function fft1 is called exactly two times, and the overhead is almost exactly compensated.

For the *qurt, fft1 (2)*, *fir (1)*, and *fir (2)* tests with a high number of calls to the function, the partial analysis is faster than the non-partial analysis. As shown by the *fft1 (2)* test, the loop nesting level of the call site has a great impact.

To summarize, although the computation of the transfer and the summary functions creates an overhead compared to a single non-partial analysis, the transfer function is much more fast to apply and the summary function is used only once for each call site. As shown in the experimentation, this overhead is compensated as soon as the partial analysis is applied twice. Such a situation arises very often in programs as (1) functions are usually defined to be called several times, and (2) in case of loops, the fix-point computation of static analyses requires at least two iterations.

5. Related work

The effect of the instruction caches on WCET computation has been extensively studied.

A widely used method for WCET computation is the Implicit Path Enumeration Technique (IPET) [6]. In IPET, the program structure and flow facts (such as loop bounds) are modeled by linear constraints. The WCET is then expressed by an objective function to maximize, and an Integer Linear Programming solver is used to compute the result.

In [1], F. Mueller defines a method to categorize instructions into three categories (Always-Hit, Always-Miss, and First-Miss). These categories describe the worst-case instruction cache behavior, and are included in the WCET computation. While Always-Miss and Always-Hit are self-explaining, First-Miss means that the first execution of an instruction in a loop may result in a miss, but subsequent executions will result in a hit.

C. Ferdinand [2, 8] describes a method to compute the categories using abstract interpretation. He uses three analyses that consist in computing an Abstract Cache State (ACS) for each basic block as explained in the section 3.1. The ACS resulting from this analysis are used to build the categories for each instruction memory access.

In [4], F. Mueller extends the method defined in [1] to a program made up of multiple modules. First, a *module-level* analysis, consisting of four analyses for different possible contexts (*scopes*), is done independently for each module, resulting in temporary categories. Next, a *compositional* analysis is performed on the whole program to adjust the categories according to the call context. This results in a merged context-insensitive category causing a lot of pessimism. Moreover, this approach requires also more analyses passes than our analysis that is performed in one pass followed by the fast instantiation of our functional categories.

In [3], a method is proposed to perform component-wise instruction cache behavior prediction. It adapts Ferdinand's cache analysis [2] (bound to *May* and *Must* analyses) to an executable linked from multiple components. This method addresses two main aspects of the problem.

First, the absolute base address of the component is unknown during the analysis, so it works with relative addresses (i.e. assuming that the base address is 0). To overcome this issue, it is proposed to force the linker to put the component into a memory location that is multiple of the cache size. This method ensures that the cache mapping is the same, whether the base address is absolute or relative. As shown in 3.4, we alleviate this constraint as we only require a cache block size alignment.

Next, when a component calls a function in another component, the called function has an influence on the cache state, and on the ACS of the caller. To handle this problem, the paper defines, for each called function, a "cache damage update" function that tells which cache lines are replaced (and how many times for associative caches) by the function. In this paper, we have augmented this analysis, as we also compute the persistence information, but we have also improved the accuracy of the results thanks to a context-sensitive definition of the categories.

6. Conclusion

We have presented a method to perform partial analysis on the instruction cache of a function in a component, and to compose the obtained partial result to compute the WCET of the whole program. Then, we have compared our method to the non-partial analysis on several benchmarks. The results show that, for the tested cases, the WCET produced using our method induced a very small pessimism.

Also, it seems that if the function being partially analyzed is called at least twice, or is called in a loop, the time needed to do the partial analysis and the composition is lower than the time needed to do the non-partial analysis. This means that the partial analysis is useful to speed up the analysis of large programs containing functions called many times.

Our future work will include the adaptation of this method to the A-way associative cache, and to other types of analyses for WCET computation (pipeline effects, etc...). Our goal is to apply partial analysis to the overall WCET computation.

7. References

[1] F. Mueller, *Fast instruction cache analysis by static cache simulation*, Journal of Real-Time Systems, 2000.

[2] M. Alt, C. Ferdinand, F. Martin and R. Wilhelm, *Cache behavior prediction by abstract interpretation*, Proceedings of Static Analysis Symposium, 1996.

[3] A. Rakib, O. Parshin, S. Thesing and R. Wilhelm, *Component-wise instruction-cache behavior prediction*, Proceedings of 2nd International Symposium on Automated Technology for Verification and Analysis, 2004.

[4] K. Patil, K. Seth and F. Mueller, *Compositional static instruction cache simulation*, ACM SIGPLAN Notices, 2004.

[5] P. Cousot and R. Cousot, *Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints*, Proceedings of 4th ACM Symposium on Principles of Programming Languages, 1977.

[6] Y. T. Li and S. Malik, *Performance analysis of embedded software using implicit path enumeration*, Proceedings of the 32nd ACM/IEEE conference on Design automation, 1995.

[7] H. Cassé and P. Sainrat, *OTAWA, a Framework for Experimenting WCET Computations*, 3rd European Congress on Embedded Real-Time Software, 2006.

[8] H. Theiling, C. Ferdinand and R. Wilhelm, *Fast and precise WCET prediction by separate cache and path analyses*, Journal of Real-Time Systems, 2000.

IMPROVEMENT OF ZIGBEE ROUTING PROTOCOL INCLUDING ENERGY AND DELAY CONSTRAINTS

Najet Boughanmi and YeQiong Song

LORIA – INPL
Campus scientifique, BP239
54506 Vandoeuvre-lès-Nancy, France
emails : {Najet.Boughanmi ; Ye-Qiong.Song }@loria.fr

Abstract*: Besides energy constraint, wireless sensor networks should also be able to provide bounded communication delay when they are used to support real-time applications. In this paper, we propose an improvement of Zigbee routing protocol integrating both energy and delay constraints. By mathematical analysis and simulations, we have shown the efficiency of this improvement.*
Keywords: Real-time, wireless sensor network, routing, energy, delay, lifetime, delivery rate.

1. INTRODUCTION

With recent technical and technological advances in WSN (Wireless Sensor Network), it becomes now possible to envisage not only simple non real-time data collect but also more complicated real-time applications. Thus, WSN can be extended to include actuator nodes, called by some researchers wireless sensor and actuator network [1]. Each sensor node is composed of one or more sensors, a processor and a radio transmission unit. All of them are supplied by an unchangeable battery. Sensor nodes collect data from the environment that they are supervising and send them to other nodes or a base station (sink). This station processes received data and sends appropriate action commands to the actuators [2, 3]. Actuator nodes are assumed less energy constraint than the sensor nodes.

It is worth pointing out that the main research efforts in developing WSNs have focused on how to extend the network lifetime with respect to limited battery energy. However, when real-time applications are deployed on them, extending the lifetime of the network should be done without jeopardizing real-time communications from sensor nodes to other nodes or to data sinks. For example, a surveillance system needs to alert authorities of an intruder within a few seconds of detection [4]. Unfortunately, there is little work on the real-time communication support for WSNs.

For energy saving, most of work focuses on the communication protocol design since in a WSN the radio communication unit is the major power consumer in the node (it consumes about one thousand CPU units) [5]. IEEE 802.15.4 Task Group together with Zigbee Alliance [6] have developed an entire communication protocol stack for Low-Rate Personal Area Networks. One of the potential applications of this standard is in WSNs. This standard represents the new generation of distributed embedded systems for pervasive computing. IEEE 802.15.4 standard deals with the energy optimization in the physical layer and the Medium Access Control (MAC) sub-layer. Energy saving is mainly achieved by defining a sleeping period (inactive period) in a superframe. The Zigbee specifications define the routing and the application layer. The Zigbee routing protocol is almost the same as AODV (Ad hoc On-Demand Distance Vector) with the exception of route maintenance. Even one may agree that AODV can always choose the route that minimizes the delay (or equivalently the number of hops), it does not take into account energy optimization. In this paper, we aim at improving the Zigbee routing protocol by including both energy and delay considerations.

Several energy-aware metrics have been proposed [7, 8, 9] to optimize the energy consumption during the routing process. However they omit the real-time aspect. [10] presents a routing approach which optimizes the network lifetime for real-time applications. However, it does not take into account the link's reliability. It should be noted that a route that chooses an unreliable link may experience longer delay because of the higher retransmission probability, which will in turn increase the energy consumption. The Real-time Power-Aware Routing (RPAR) protocol [11] reduces communications delays by adapting the transmission power to the workload. However, it does not optimize the network lifetime.

So, in this paper, we will focus on maximizing the sensor network lifetime while still taking into account the delay requirement of real-time communications. Our main idea is to find a new routing metric which is capable of including delay, energy, as well as link reliability factors. In our study, we used IEEE 802.15.4 protocol and Zigbee AODV. We are going to optimize the network lifetime under the delay constraint at the routing layer. Without loss of generality, the delay of a route is considered equivalent to the number of hops on the route and we assume that one can find the limit on the hop number for a given real-time communication constraint.

The rest of this paper is organized as follows. In Section 2 provides a mathematical analysis for packet forwarding. We will give a routing metric that trades off between maximizing the sensor network lifetime and satisfying the communication delay. By simulations, we will compare the performance of our

routing approach with the existing ones in Section 3. Finally, Section 4 gives conclusions and describes future directions.

2. PROPOSED ROUTING METRIC

2.1 Model

In this study, we adopt the model defined in [7]. This model captures the packet reception rate (PRR) between two nodes as follows. Nodes have full connectivity if they have a distance less than D_1. They are disconnected if they are separated by a distance greater than D_2. The expected PRR decreases smoothly in the transitive region between D_1 and D_2. The behavior is modeled by (1)

$$PRR = \begin{cases} 1 & d<D_1 \\ \left[\dfrac{D_2 - d}{D_2 - D_1} + X \right]_0^1 & D_1 \le d \le D_2 \\ 0 & d > D_2 \end{cases} \quad (1)$$

where $[.]_a^b = max\{a, min\{b, .\}\}$ and $X \sim N(0, \sigma^2)$ is a Gaussian variable with variance σ^2.

2.2 Metrics

The wireless sensor network is presented by a graph $G = (V, A)$, in which V is the set of nodes including the base station. The set of edges $A \subset V \times V$ such that $(i, j) \in A$ if nodes i and j can transmit to each other. To optimize the routing path, we assign each node the remaining energy and each vertex the delivery rate.

In the following, we are interested in the metric of the path efficiency. This metric considers the path energy efficiency and the delay experienced along this path. Here we are going to maximize energy efficiency while minimizing the delay together. Thus, we first define this path efficiency, E, to be the ratio of the path energy efficiency, E_{eff}, to the delay required to transmit a packet from the source to the destination. The energy efficiency represents a trade-off between delivery rate and energy consumption along this path. In order to maximize the path efficiency and minimize the energy consumption, the energy efficiency is quantified as the ratio of the delivery rate, E_r, to the total energy consumed to transfer the packet to the destination node E_e. Thus, this energy efficiency is expressible by

$$E_{eff} = \frac{E_r}{E_e} \quad (2)$$

[1] This equation is modified, in numerator, $d - D_1$ is replaced by $D_2 - d$ to find 1 when $d = D_1$.

The end-to-end delivery rate for a path φ takes into account the delivery rate of each link in this path. So, this end-to-end delivery rate is the product of packet reception rate of each link in φ as shown by

$$E_r = \prod_{k \in \varphi, k \neq destination} prr_{k,k+1} \quad (3)$$

where $prr_{k,k+1}$ is the packet reception rate between node k and its forwarder $k+1$ as shown in Figure 1.

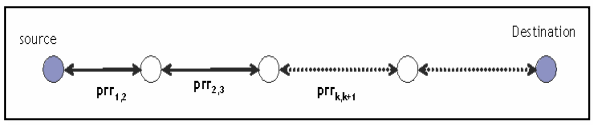


Fig. 1. Path

Taking into account the retransmission (R: number of allowed retransmission), the required energy for the packet delivery for the first transmission is calculated by

$$\hat{E}_e^1 = p_{i,i+1}(E_e^i + b) + a(b + \hat{E}_e^2) . \quad (4)$$

Using a recurrent calculation, the required energy for the packet delivery for the R^{th} retransmission is given by

$$\hat{E}_e^{R+1} = p_{i,i+1}(E_e^i + b) + ab . \quad (5)$$

Finally, the required energy for the packet delivery is

$$E_e = \frac{(prr_{i,i+1}(E_e^i + b) + ab)(1 - a^{R+1})}{1 - a} \quad (6)$$

where $prr_{i-1,i}$ is the packet reception rate for the forwarder i-1, E_e^i is its energy cost that refers to the energy consumption from the source to the node i. b is the packet processing energy (transmission and reception) and $a = 1 - prr_{i-1,i}$.

As we are using the IEEE 802.15.4, the number of allowed retransmission is fixed to 3. Therefore the required energy will be

$$E_e = \frac{(prr_{i,i+1}(E_e^i + b) + ab)(1 - a^4)}{1 - a}$$

If $prr_{i-1,i}$=0 (the link is broken), the consumed energy is equal to $(R+1)b$, in our case $4b$.
By replacing E_r and E_e in (2), the energy efficiency is given by

$$E_{eff} = \frac{\prod_{k \in \varphi, k \neq destination} prr_{k-1,k}}{(prr_{i,i+1}(E_e^i + b) + ab)(1 - a^4)} (1 - a) \quad (7)$$

As the routing approach has to respect the delay to guarantee the "deadline" for real-time communications, the path efficiency could further be represented by $E = E_{eff}/delay$.

$$E = \frac{\prod_{k \in \varphi, k \neq destination} prr_{k-1,k}}{delay(prr_{i,i+1}(E_e^i + b) + ab)(1 - a^4)}(1-a) \quad (8)$$

The routing approach presented by Coleri [10] guarantees the delay performance too. However, the corresponding delay is not included in the routing metric. In fact, in this approach only paths that offer delay less than the allowed delay are considered in the routing choice. Furthermore, the time is divided into time frames and at the beginning of each frame, the base station floods the network with a tree construction packet. Thus, there is significant energy consumption in the routing process. However, we use the AODV routing protocol with a modified routing metric as shown in (8). Hence, the route is setup according to the AODV request/response cycle. The delays are collected by route response message. Consequently, we have not increased the network load.

However, considering only the consumed energy is not sufficient to maximize the lifetime of the sensor network. We must include the remaining energy in the routing metric to balance the load of the network. Thus the lifetime efficiency E_{leff} is given by

$$E_{leff} = E \cdot e_i \quad (9)$$

where e_i is the remaining energy of the forwarding node i.

The new metric for the path efficiency which includes the delay, the path reliability and the lifetime efficiency, E_l, can be calculated from

$$E_l = \frac{\prod_{k \in \varphi, k \neq destination} prr_{k-1,k}}{delay(prr_{i,i+1}(E_e^i + b) + ab)(1 - a^4)}(1-a) \cdot e_i \quad (10)$$

Once we have defined our routing metric, we included it in the AODV routing protocol. Thus, our new version of AODV chooses the most efficient path to the destination node by considering both energy and delay constraints.

3. SIMULATION RESULTS AND DISCUSSION

In this section, the performance of the proposed routing metric is evaluated and compared with AODV routing protocol and Coleri routing metric. Furthermore, we use NS-2 simulators to implement the physical and MAC layers of IEEE 802.15.4. We have changed the existing implementation in NS-2 of AODV to integrate our metric. Thus, we have a new version of AODV, which we call Enhanced AODV.

The primary purpose of our simulation is to observe the network lifetime resulted by our routing optimization. Moreover, we consider the delivery rate as another performance metric.

The simulated networks consist of 11, 22 and 101 nodes respectively.

3.1 Assumptions

The following assumptions are made in this study.

1. We consider a wireless sensor network that consists of a base station and several sensor nodes. These sensor nodes generate data for transfer to the base station. Delay constraint is only imposed on this sensor to base station communication.
2. Sensor nodes have a low mobility that is the case for most of the sensor network applications.
3. The delay needed to transmit a packet from a source node to a destination node is equivalent to the number of hops counted between these two nodes.
4. The operational lifetime of the sensor network is defined as the time until the first 5% of nodes died as proved in [10].

3.2 Lifetime

We study here the sensor network lifetime. We observe in Figure 2 that at the beginning the three routing approaches have the same result. In fact, in the beginning of the network life, all nodes have a maximal amount of energy. Thus, the three routing approaches will have the same routing choices. Once the sensor energy decreases, the difference between these routing approaches appears. We observe that the Enhanced AODV routing approach let sensors be alive for a longer time than AODV routing protocol does.

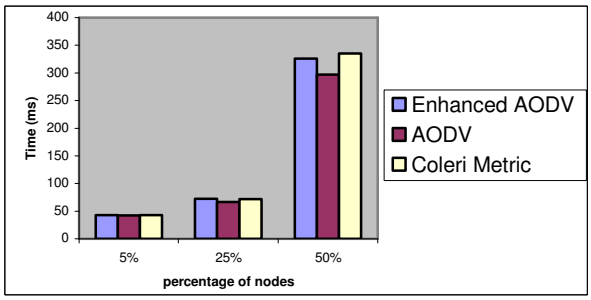


Fig. 2 Comparison of the time at which a specific percentage of the nodes are dead between Enhanced AODV, AODV and Coleri metric .

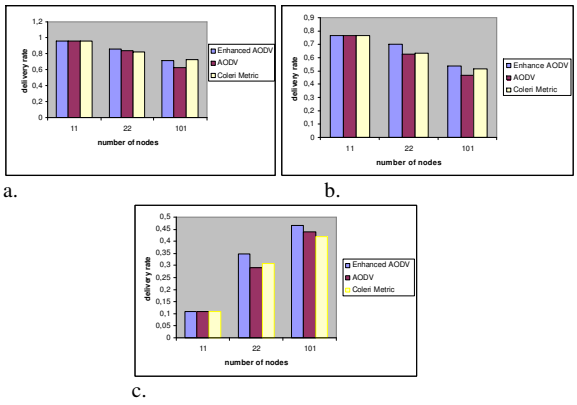
Moreover, both of Enhanced AODV and Coleri routing metric give almost the same time for the death of a specific percentage of nodes. This is an

expected result since both routing metrics aim to maximize sensor network lifetime.

3.3 Delivery rate

In this sub-section, we focus on the optimization of the network delivery rate. We define the network delivery rate as the ratio of the total received packets to the total sent packets in the sensor network. We compute this delivery rate at different times in the sensor network lifetime and compare the results among Enhanced AODV, AODV and Coleri metric.

Figure 3 a. gives the delivery rate before the death of 5% of nodes. We notice that for a sensor network of a small number of nodes, all of the studied routing approaches offer the same delivery rate. In fact, in small sensor network there is almost one path from the source to destination. Thus, all of the routing algorithms choose the same path. However, for a network with a larger number of nodes, the Enhanced AODV performs better than AODV does. Moreover, the Enhanced AODV and Coleri routing metric gives almost the same delivery rate.



a. b.

c.

Fig. 3 a. Delivery rate before the death of 5% of nodes. b. Delivery rate before the death of 25% of nodes. c. Delivery rate before the death of 50% of nodes

Figure 3 b. shows the delivery rate before the death of 25% of nodes. In the same way as mentioned before, for a small sensor networks, all of the studied routing approaches give the same delivery rate. However, the benefit due to the optimization of delivery rate by the Enhanced AODV is clear. In fact, these routing approaches give better delivery rate than AOV and Coleri metric. Thus, although the Enhanced AODV and the Coleri metric offer the same network lifetime, the former gives a better delivery rate.

From the results given by the Figure 3 c. we notice that the Enhanced AODV offers better delivery rate than AODV and Coleri routing approaches. Thus, for different moment of the network lifetime, the delivery rate is always better with the Enhanced AODV routing approach.

4. CONCLUSION

A successful deployment of real-time applications over WSNs needs to satisfy the required timing properties under energy consumption constraints. As Zigbee routing protocol does not address energy and delay issues together at the same time, we propose in this paper a new routing metric. The benefit of this metric has been shown by simulations when embedded into AODV protocol. Moreover, we started implementing this metric in MecaZ. As a future work, we plan to test this metric in a real WSN.

5. REFERENCES

[1] I. F. Akyildiz and I. H. Kasimoglu, "Wireless sensor and actor networks: research challenges", *Ad hoc networks*, 2 (2004), pp 351-367, May 2004.

[2] Y. Li, Z. Wang, and Y.Q. Song, "Wireless sensor network design for wildfire monitoring", 6th *World Congress on Intelligent Control and Automation (WCICA 2006)*, Dalian (China), 2006.

[3] C. Lu, G. Xing, O. Chipara, C.L. Fok, and S. Bhattacharya, "A spatiotemporal query service for mobile users in sensor networks", *ICDCS*, pp. 381-390, 2005.

[4] T. He, P. A. Vicaire, T. Yan, L. Luo, L. Gu, G. Zhou, R. Stoleru, Q. Cao, J. A. Stankovic and T. Abdelzaher, "Achieving real-time target tracking using wireless sensor networks ", *12th IEEE Real-Time and Embedded Technology and Applications Symposium*, pp. 37-48, April 2006.

[5] S. Coleri Ergen, and P. Varaiya, "PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks", *IEEE Transactions on Mobile Computing*, vol. 5, pp. 920-930, July 2006.

[6] Zigbee Specifications, 2004. http://www.zigbee.org.

[7] M. Busse, T. Haenselmann, and W. Effelsberg, "An Energy-Efficient Forwarding Scheme for Wireless Sensor Networks", *Technical report 13*, University of Mannheim, Dec. 2005.

[8] M. Busse, T. Haenselmann, and W. Effelsberg, "Poster-Abstract: A Lifetime-Efficient Forwarding Strategy for Wireless Sensor Networks", *EWSN*, 2006.

[9] Q. Cao, T. He, L. Fang, T. Abdelzaher, J. Stankovic, and S. Son, "Efficiency Centric Communication Model for Wireless Sensor Networks", *Infocom*, 2006.

[10] S. Coleri Ergen, and P. Varaiya, "Energy Efficient Routing with Delay Guarantee for Sensor Networks", *ACM Wireless Networks WINET*, 2006.

[11] O. Chipara, Z. He, G. Xing, Q. Chen, X. Wang, C. Lu, J. Stankovic, and T. Abdelzaher, "Real-time Power-Aware Routing in Sensor Networks", *Forteenth IEEE International Workshop on Quality of Service (IWQoS 2006)*, June 2006.

Negative Results on Idle Intervals and Periodicity for Multiprocessor Scheduling under EDF

Christelle Braun
ENSMN
54000 Nancy, France
braun67@mines.inpl-nancy.fr

Liliana Cucu
LORIA-INPL
54000 Villers les Nancy, France
liliana.cucu@loria.fr

Abstract

In this paper we present negative results for global scheduling of arbitrary deadlines periodic systems under EDF. We reconsider the definition of the idle intervals from the uniprocessor case to the multiprocessor case. Unfortunately, this new definition does not provide feasibility results for these systems. We provide counter-examples for the periodicity of EDF-feasible schedules in the multiprocessor case.

Keywords : scheduling, multiprocessor, global.

1 Introduction and existing results

Real-time systems are widely used in nowadays industry such as spacecraft guidance or mobile phone communications. The correctness of a real-time system not only relies on its logical result, but also on the time required to produce it.

In this paper, we consider scheduling of real-time systems on multiprocessor platforms. For economical and practical reasons, these systems are often preferable to powerful uniprocessor architectures. However, specific scheduling issues arise with multiprocessors, which need a better understanding and justify our present contribution.

Real-time scheduling has been much studied since Liu's seminal paper in 1973 [1]. However, the research in the field has mainly focused on *uniprocessor* platforms, while in comparison only few results are known for the *multiprocessor* case. See a complete presentation of these results in [2].

We consider the model of a *periodic task system* [1], where such a system is usually modelled as a set of elementary units called *jobs*. A *job* J_k ($k \in \mathbb{N}^*$) is defined by (r_k, e_k, d_k) where r_k is its *release time*, e_k its worst-case *execution time* and d_k its *deadline*. In this model, the job J_k, released at time r_k, must have finished execution before or at $r_k + d_k$. A job is *active* from its release time until its execution ends.

In this paper, the jobs satisfy the following assumptions:

(H1) **Jobs are independent**: the release time of a job does not depend on the execution of another job.

(H2) **Job preemption is permitted**: a job executing on a processor may be preempted prior to completing execution, and its execution may be resumed (without penalty) later.

(H3) **Job migration is permitted**: after its preemption, a job may resume execution on a processor different from the one upon which it had been executing prior to preemption. In this case we deal with *global* scheduling.

(H4) **Job parallelism is forbidden**: each job may execute on at most one processor at any given instant in time.

Given a task system, a *scheduling algorithm* determines at every time instant which job(s) should be executed.

We consider two set of jobs J and J' which are identically except for the execution times. The jobs belonging to J have execution times larger or equal to those of the jobs belonging to J'. A scheduling algorithm is predictable if when it schedules, separately, J and J', each job of J' finishes, always, before or at the same time as the corresponding job in J.

This work. The scheduling algorithm EDF is already proved predictable ([3]) in the multiprocessor case. It implies that the periodicity of an EDF-feasible schedule (if any) provides a feasibility interval[1]. To our best knowledge, there is no result on feasibility intervals for global multiprocessor scheduling under EDF.

In this paper, we reconsider the definition of the idle intervals from the uniprocessor case to the multiprocessor case. Unfortunately, this new definition does not provide feasibility results for these systems. We provide counter-examples for the periodicity of EDF-feasible schedules in the multiprocessor case.

In this paper, we consider *identical multiprocessors*, i.e., all processors have the same computing power. We consider, also, a discrete model, i.e., the characteristics of the tasks are integers. Moreover, we assume that the instants at which the scheduler makes decisions are equidistant.

The paper is organized as follows. In Section 2 we introduce the main definitions and notations. Section 3

[1] we understand by feasibility interval the finite interval such that if no deadline is missed while considering only requests within this interval then no deadline will ever be missed

presents EDF-scheduling and idle time related issues. In Section 4 we provide a conjecture on the periodicity of EDF-feasible schedules. We conclude in Section 5.

2 Definitions and notations

A *periodic task* $\tau_i = (O_i, C_i, D_i, T_i)$, $i \in \mathbb{N}^*$ is characterized by an *offset* O_i, a worst-case *execution time* C_i, a relative *deadline* D_i and a *period* T_i. Starting from O_i, a new job is released every T_i time units. In other words, the k'th job of the task τ_i ($k, i \in \mathbb{N}^*$) is released at time $O_i + (k-1)T_i$ and must execute C_i time units before its deadline occurring at time $O_i + (k-1)T_i + D_i$.

A periodic system is a finite system of periodic tasks $\tau = (\tau_1, \tau_2, \ldots, \tau_n)$, $\forall n \in \mathbb{N}^*$, where $\tau_i = (O_i, C_i, D_i, T_i), \forall i \in \{1, \ldots, n\}$. If there is a time instant at which jobs of all tasks are released synchronously, the system is said to be *synchronous*; otherwise the system is said to be *asynchronous*.

The *processor utilization* U of the system $\tau = (\tau_1, \tau_2, \ldots, \tau_n)$ is the sum $U \stackrel{\text{def}}{=} \sum_{i=1}^{n} U_i$ where U_i is the utilization of the task τ_i: $U_i \stackrel{\text{def}}{=} \frac{C_i}{T_i}, i \in \{1, \ldots, n\}$.

We are typically interested in feasibility and schedulability problems, where we understand:

Feasibility Determining whether the task system τ can be executed in such a manner that all jobs complete by their deadlines.

Schedulability Providing a scheduling algorithm which gives a feasible schedule for the task system τ.

In the following, we note $P \stackrel{\text{def}}{=} \text{lcm}\{T_1, T_2, \ldots, T_n\}$ and $O_{\max} \stackrel{\text{def}}{=} \max\{O_1, O_2, \ldots, O_n\}$.

The scheduling algorithms considered in this paper are *deterministic*, where a scheduling algorithm is said to be *deterministic* if it generates a unique schedule for any given set of jobs.

Furthermore, we shall distinguish between *implicit deadline* systems where $D_i = T_i, \forall i$; *constrained deadline* systems where $D_i \leq T_i, \forall i$ and *arbitrary deadline* systems where there is no constraint between the deadline and the period.

3 EDF scheduling algorithm

Definition 1 (Earliest deadline first algorithm (EDF)) *The* earliest deadline first algorithm *schedules, at every time instant, the active job with the earliest deadline.*

Definition 2 (EDF-schedulability) *A system is EDF-schedulable if and only if the schedule obtained using EDF is feasible.*

We present the main uniprocessor results on feasibility intervals under EDF.

Definition 3 (Uniprocessor idle time) *On a uniprocessor platform, the time instant t is an* idle time *in the schedule of a task system τ if all the jobs which were released prior to t have completed execution at or before t.*

Definition 4 (Idle interval) *A time interval $[t_1, t_2]$ $(t_1, t_2 \in \mathbb{N})$ in the schedule of a task system τ is an* idle interval *if it satisfies the following conditions:*

- $t_1 < t_2$ *(its length is strictly positive)*
- $\forall t \in [t_1, t_2) \cap \mathbb{N}$, *$t$ is an idle time.*

Theorem 1 (Idle time) *[4] When EDF is used to schedule a synchronous arbitrary deadline periodic task system, there is no idle time prior to a missed deadline.*

Theorem 1 is very useful in schedulability analysis, since it reduces the length of the time interval which has to be considered in order to decide EDF-schedulability on one processor:

Corollary 2 *In a uniprocessor platform, if a synchronous implicit deadline periodic task system τ is EDF-schedulable on $[0, t)$ where t is an idle time, then τ is EDF-schedulable.*

Idle time results in multiprocessor systems. Theorem 1 does not hold in a multiprocessor environment for a synchronous arbitrary deadline periodic task system [5]. For instance, let $\tau_A = (\tau_1, \tau_2, \tau_3)$ be the synchronous constrained task system with the parameters given in Table 1. We consider two processors $\{p_1, p_2\}$. The schedule of this system, obtained using EDF, is given in Figure 1, where gray filled squares correspond to p_1 and black filled squares to p_2. At time instant $t = 13$, the active job of the task τ_3 misses its deadline. However, the first idle time of this schedule occurs at time $t = 6$ and, therefore, there is an idle time prior to a missed deadline, which implies that Theorem 1 does not hold for synchronous arbitrary deadline periodic task systems, since we give a negative result for constrained deadline periodic task system.

	C_i	D_i	T_i	U_i
τ_1	3	6	6	0.5
τ_2	3	6	6	0.5
τ_3	5	5	8	$\frac{5}{8}$

Table 1: System parameters for τ_A

	C_i	D_i	T_i	U_i
τ_1	3	6	6	0.5
τ_2	3	6	6	0.5
τ_3	4	5	8	0.5

Table 2: System parameters for τ_B

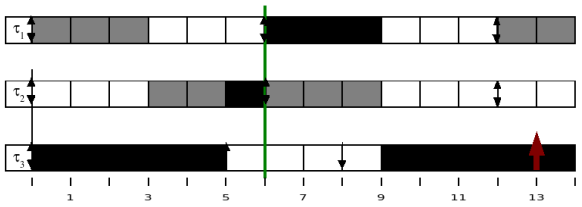


Figure 1: Counter-example for Theorem 1 in a multiprocessor system [5] (Table 1)

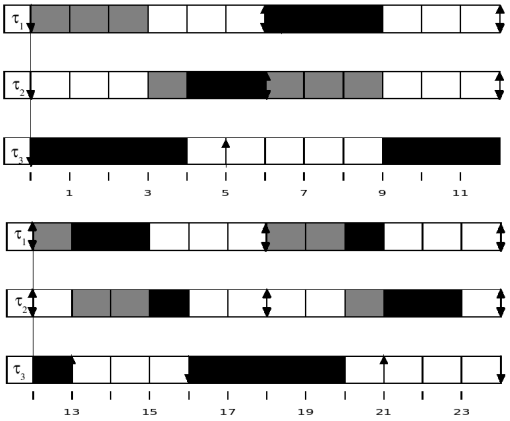


Figure 2: EDF-schedule for τ_B (Table 2)

All three tasks are *heavy* tasks in the sense that their density $\frac{C_i}{D_i}$ (≥ 0.5) is high. In particular, the task τ_3, which misses its deadline at $t = 13$, has a density of 1, meaning that any k'th job of this task must start its execution by its release time and it has to be executed non-preemptively.

On the other hand, Figure 2 shows that if the utilization of a task system τ_B is slightly smaller than the utilization of τ_A (by decreasing the execution time of τ_3 from 5 to 4), the missed deadline observed in Figure 1 may be avoided. In this case, τ_B is EDF-schedulable.

A direct consequence of the high utilization of τ_A on Figure 1 is that the idle time which occurs at $t = 6$ is not followed by an idle interval since new jobs are immediately released at this same time instant.

This leads to the question whether the notion of an idle time given in Theorem 1 remains actually relevant in a multiprocessor environment. A better definition of this instant could preserve the correctness of Theorem 1 in the multiprocessor case.

Definition 5 (Multiprocessor idle time) *In a multiprocessor environment, the time instant t is an* idle time *in the schedule of a task system τ if all the jobs which were released prior to t have completed execution at or before t, and no job is released at t.*

In contrast to the uniprocessor case, a multiprocessor idle time must be followed by an idle interval where Definition 4 is extended to the multiprocessor case. Does this refined definition of an idle time preserve the validity of Theorem 1 in multiprocessor environments? Unfortunately, the task system τ_C whose parameters are given in Table 3 shows that Theorem 1 does not hold. In τ_C, the tasks have a slightly smaller utilization than in task system τ_A (Table 1). The EDF-schedule (on two processors $\{p_1, p_2\}$) given in Figure 3 shows that time instant $t' = 6$ is an idle time, as defined in Definition 5. In Figure 3 gray filled squares correspond to p_1 and black filled squares to p_2. However, τ_3 misses its deadline at $t = 15 \in [0, O_{\max} + P)$ with $P = 63$. Therefore, Theorem 1 does not hold in the multiprocessor case, even if Definition 5 is used for the idle time.

If $T_1 = T_2$, the deadlines of the three jobs activated in the interval $[0, 6]$ coincide with time instant $t = 6$. This is a case of non-determinism in EDF, a problem considered in [6] and whose resolution may have an influence on the EDF-feasibility of the task system τ.

	C_i	D_i	T_i	U_i
τ_1	3	7	7	3/7
τ_2	3	7	7	3/7
τ_3	6	6	9	2/3

Table 3: System parameters for τ_C

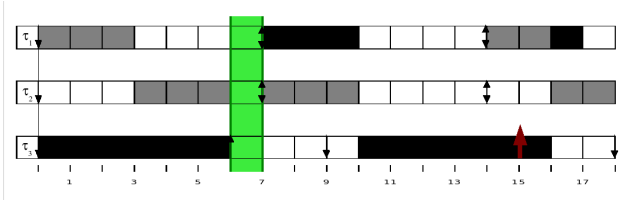


Figure 3: Counter-example for Theorem 1 after the redefinition of the idle time (Table 3)

4 Periodicity starting time of EDF-schedules revisited

In this section, we present negative results on the periodicity of EDF-feasible schedules of arbitrary deadlines periodic systems obtained.

Discussion on the periodicity In the multiprocessor case, the periodicity of a feasible EDF-schedule does not necessarily start at or before time instant $t = O_{\max} + P$. For instance, we consider asynchronous constrained deadline periodic task system τ_D whose parameters are given in Table 4. Figure 4 shows that τ_D is EDF-schedulable on 2 processors $\{p_1, p_2\}$, where gray filled squares correspond to p_1 and black filled squares to p_2. Moreover, starting from $O_{\max} + P + 2$, the schedule is periodic with period P. Since at time instants $O_{\max} + P + 1$ and

$O_{\max} + 2P + 1$ the tasks are not scheduled in the same manner (as highlighted on Figure 4), the periodicity does not hold at $O_{\max} + P$ for asynchronous arbitrary deadline periodic task systems, since we give a negative result for constrained deadline periodic task system.

In Figure 4, the difference in the schedules at time instants $t_1 = O_{\max} + P + 1$ and $t_2 = O_{\max} + 2P + 1$ comes from the non-execution of task τ_3 at t_1. Thus, the execution of τ_3 at t_2 shows that there must exist $t_{2'} < t_2$ with τ_3 executed at $t'_2 - P$ and τ_3 not executed at t'_2. In other words, the execution of τ_3 in time interval $[t'_2, t'_2 + 1]$ is *reported* to time interval $[t_2, t_2 + 1]$, with $t_2 > t'_2$. This phenomenon appears in Figure 4, where $t_1 = 17, t_2 = 29, t'_2 = 27$. The circles show the reported execution time slots and the related arrows the new time intervals of execution.

Figure 4 also shows that the report occurring a $t = 27$ actually results from task τ_2 preempting τ_3. By repeating the previous reasoning for τ_2, we obtain that this preemption is the direct consequence of the report of execution of τ_2 from $t = 24$ to $t = 27$. Stepping backwards in the schedule, a *chain of preemptions* appears, starting at time instant $t = 13$ where τ_2 preempts τ_3. This preemption could not occur one hyper-period before (at $t = 1$), since τ_2 had not been released yet. On the other hand, the preemption chain ends at $t = 29$, where the execution of τ_3 is reported from $t = 27$ but can be scheduled in an idle interval of the second processor.

	O_i	C_i	D_i	T_i	U_i
τ_1	0	2	3	3	2/3
τ_2	4	3	4	4	3/4
τ_3	1	3	6	6	1/2

Table 4: System parameters for τ_D

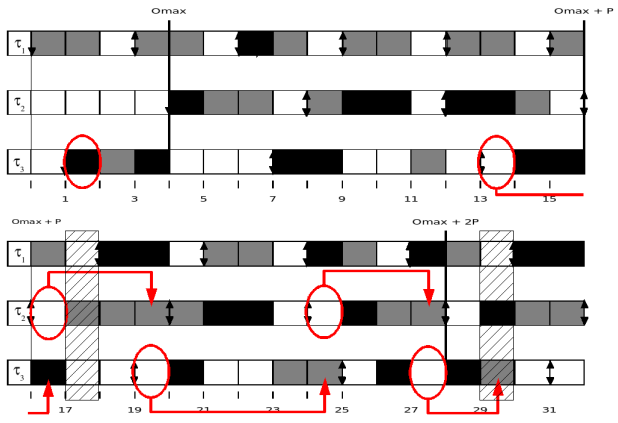


Figure 4: Counter-example for a periodicity of the EDF-schedule starting at time instant $O_{\max} + P$ in the multiprocessor case (Table 4)

From these observations, we **conjecture** that, in the multiprocessor case, the delay required by an EDF-schedule to become periodic corresponds to the length of the preemption chain which results from the specific disposition of the offsets of the tasks in the system.

5 Conclusion and future work

In this paper, we showed that feasibility intervals obtained in the uniprocessor case cannot be extended to the multiprocessor case. We also provided counter-examples highlighting differences with the uniprocessor case and conjectured a new result for the periodicity of EDF-feasible schedules. As future work, we plan to give feasibility intervals under EDF in the multiprocessor case, based on our present conjecture.

6 Acknowledgements

The authors would like to thank Joël Goossens for posing the feasibility problem and for his detailed comments on a previous version. The authors would like to thank, also, anonymous reviewers for their helpful comments.

References

[1] C.L. Liu and J.W. Layland. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM*, 20(1):46–61, 1973.

[2] J. Carpenter, S. Funk, P. Holman, A. Srinivasan, J. Anderson, and S. Baruah. A categorization of real-time multiprocessor scheduling problems and algorithms. *Handbook of Scheduling*, 2005.

[3] L. Cucu and J. Goossens. Feasibility intervals for fixed-priority real-time scheduling on uniform multiprocessors. In *Proceedings of the 11th IEEE International Conference on Emerging Technologies and Factory Automation*, pages 397–405, 2006.

[4] J. Goossens and R. Devillers. Feasibility intervals for the deadline driven scheduler with arbitrary deadlines. In *Proceedings of the Sixth IEEE International Conference on Real-time Computing Systems and Applications*, pages 54–61, 1999.

[5] J. Goossens, S. Funk, and S. Baruah. EDF scheduling on multiprocessors: some (perhaps) counterintuitive observations. In *Proceedings of the 8th International Conference on Real-Time Computing Systems and Applications*, pages 321–330, 2002.

[6] J. Goossens, R. Devillers, and S. Funk. Tie-breaking for edf on multiprocessor platforms. In *Proceedings of the 23rd IEEE Real-Time Systems Symposium*, 2002.

REAL-TIME SYSTEM FORMAL VERIFICATION BASED ON TIMING REQUIREMENT DESCRIPTION DIAGRAMS

B. Fontan*, **, P. de Saqui-Sannes*, **

*LAAS-CNRS, University of Toulouse, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 04, France
**ENSICA, University of Toulouse, 1 place Emile Blouin, 31056 Toulouse Cedex 05, France

Abstract. *The TURTLE UML profile particularly addresses the formal verification of real-time and distributed systems at their first development phases. TURTLE has recently been extended with requirement modeling capabilities based on SysML. Thus, system requirements may be described using a formal and graphical language offering temporal operators, and may later be automatically converted to observers, the role of which is to guide formal verification.*

Keywords: *UML, SysML, Real Time, Timing Requirements, Formal Verification.*

I Introduction

TURTLE (Timed UML and RT-LOTOS Environment) [APV 04] [APV 05] is a real-time UML profile [OMG 03] supported by TTool [TTOOL], a toolkit interfaced with RTL [RTL] and CADP [CADP]. Formal verification of TURTLE models relies on reachability analysis, minimization techniques [MIL 89], and observers [JAR 88]. So far, observers were designed by hand, from informal requirements that were not part of the TURTLE model.
The situation has changed when TURTLE was extended with SysML [SYS 06] requirement diagrams. The latter may include informal requirements expressed in natural language. They may further contain temporal requirements expressed in TRDD (Timing Requirement Description Diagram), a visual language based upon Timing Diagrams. An important result is that observers may be automatically synthesized from requirements expressed in TRDD and connected to the TURTLE model of the system under design.
The paper is organized as follows. Section II surveys related work. Section III overviews the TURTLE profile. Section IV introduces the TRDD diagrams. Section V sketches the principles of observer synthesis. Section VI discusses some limitations of the proposed approach. Section VII concludes the paper.

II Related Work

This section surveys various modelling techniques that might have been used to extend TURTLE with a requirement description language. It helps understanding the rationale behind the definition of the TRDD language.

In KAOS (Keep All Objective Satisfied [LAM 06]) requirements are expressed by means of logic formulas written in RT-LTL (Real Time Linear Temporal Logic). KAOS also includes a method for goal driven requirement elaboration. The KAOS tool Objectiver [OBJ] enables analysts to elicit and specify requirements in a systematic way and to achieve traceability from requirements to goals. The interest of the KAOS methodology is to formalize and trace functional and non-functional requirements (including security, safety, accuracy, cost, performance) throughout the design cycle. In this paper, we also link (temporal) requirements to a formalism and we integrate requirement capture and requirement traceability in a methodology [FON 07] not detailed here for space reasons.

Scenario based modelling techniques are also candidates for temporal requirement description. The verification process consists in matching [BRA 05] scenarios and the model of the system. For instance, Timed Uses Cases Maps [HAS 06] (see TUCM in table 1) describe Uses Cases Interactions including absolute time with a master clock and relative time constraint (Duration, Timer). Also, Visual Timed events Scenario [BRA 05] (see VTS in table 1) represent events interactions. An event represents an action which potentially occurs inside the system. VTS includes time representation. It may express partial orders and relative time constraints between events. Finally, Live Sequence Charts [DAM 01] (LSC in table 1) extend Messages Sequence Charts (MSC) to represent scenarios. LSC enable distinction between possible and necessary scenarios.

Name	TUCM	VTS	LSC
Reference	[HAS 06]	[BRA 05]	[DAM 01]
Formal Language	Clocked Transition Systems	Timed Computation Tree Logic	Bücchi Automata
Verification type	Model Checking	Model Checking (UPAAL/Kronos)	Model Checking

Tab. 1. Scenario-based visual languages with formal semantics

The scenario-based description languages discussed so far have a formal semantics, and so has TRDD. We defined observation points; the concept comes from VTS. Nevertheless, TRDD does not implement a scenario

paradigm because the latter seems us not appropriate for requirement capture and mostly geared towards the analysis phase.

To reduce the gap between requirement capture and formalization, temporal requirements might also be represented using Timing Diagrams. The latter make it possible to represent temporal requirements in an easy to read and formal way. [CHO 05] gives timing diagrams a formal semantics, based on Linear Time Logic. The author indicates that partial order is not represented in Timing Diagrams. The formalism used by the ICOS toolbox [FRA 01] is similar. Real Time Symbolic Timing Diagrams (RT-STD in table 2) are applied to SoC design. Regular Timing Diagrams [AML 99] (see RTD in table 2) improve the situation since they make it possible to represent partial order between diagrams.

Name	RT-STD	RTD	TRDD
Reference	[FRA 01]	[AML 99]	This paper
Formal Language	Bücchi Automata	Symbolic Values	RT-LOTOS
Type of verification	Model Checking	Model Checking	Observers

Tab. 2. Visual Languages based on Timing Diagrams

Overall, the timing diagram paradigm turned to be the one whose main concepts may be reused and adapted to express temporal requirements. Therefore, TRDD is based on timing diagrams.

III TURTLE

TURTLE (Timed UML and RT-LOTOS Environment) is a SysML/UML profile for real-time system analysis and design [APV 04] [APV 05]. The profile has a formal semantics expressed by translation to RT-LOTOS [COU 00]. It is implemented by TTool [TTOOL], an open source toolkit interfaced with two formal verification tools: RTL [RTL] and CADP [CADP]. RTL implements reachability analysis of the RT-LOTOS specifications generated by TTool. CADP minimizes the graphs generated by RTL.

Formal verification may be applied to the two groups of UML diagrams customized by TURTLE: (1) analysis diagrams (interaction overview and sequence diagrams), and (2) design diagrams (class and activity diagrams).

TURTLE diagrams are edited using TTool. As shown by Fig. 1, the latter translates all the diagrams into TIF: a TURTLE Intermediate Form expressed in native TURTLE [APV 05] which is made up of "basic" design diagrams. TIF serves as a starting point to generate either an RT-LOTOS specification or Java code. Java code generation is out of scope of the paper.

The purpose of the work presented in this paper is to reduce the gap between the requirement capture phase and formal verification. As suggested by Fig. 1, we automatically generate observers from requirement diagrams. Observers are translated into TIF and connected to the TIF form of relevant class and activity diagrams.

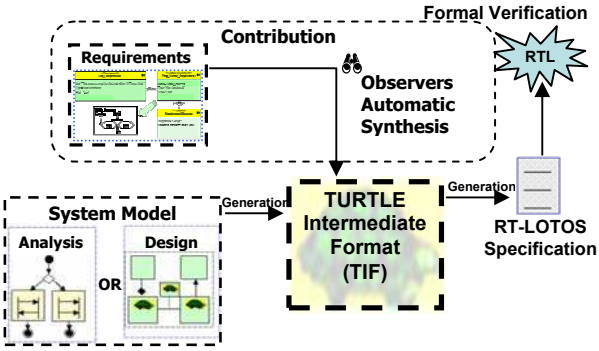


Fig.1. Main functions implemented by the TURTLE toolkit

IV Timing Requirement Description Diagrams

A SysML requirement is a test case [SYS 06] stereotyped by <<requirement>> and characterized by four attributes: (1) an *identifier*; (2) a *text* (an informal description of the requirement); (3) a *type*: "functional", "non-functional", or "performance"; (4) a *risk* level: "high" or "low" depending on whether the requirement is strong or weak, respectively.

The TURTLE requirement diagrams in Fig. 2 include an informal requirement and a formal one. Both address the same system constraint: "the process must be completed within 10 time units".

A Requirement Diagram also describes requirement refinement, derivation or verification. In Fig. 2, an informal requirement (stereotyped by <<Requirement>>) is derived (cf. the dependency relation stereotyped by <<derive>>) into a formal requirement (stereotyped by <<Formal Requirement>>). The latter will be verified using an observer (stereotyped by <<TObserver>>).Thus, the "formal requirement" serves as starting point for formal verification; the *text* in the informal requirement is replaced by a Timing Requirement Description Diagram (TRDD) in the formal requirement (fig. 2).

A TRDD describes one timing requirement. Again, the TRDD in Fig. 2 refers to a process which must complete within 10 time units. The process is defined by two actions "*Begin*" and "*End*" that we call "*observations points"*. The latter appears in the TRDD, which also depicts a temporal frontier (equal to ten in this example). The "temporal frontier" is introduced to distinguish between two time periods denoted by OK and KO that correspond to a requirement satisfaction and violation, respectively.
Formal requirements such as the one in Fig.2 serve as starting point to generate observers intended to guide

verification. As shown by Fig.2, an observer contains two attributes. First, *diagrams* states whether the observer is to be connected to the analysis or design diagrams of the TURTLE model of the system. In this paper, we restrict ourselves to design diagram verification and we ignore analysis ones. Second, *violated_action* specifies the label (identifier) to be used by the observer to denote the requirement's violation. The same label will be used in the reachability graph output by TTOOL and RTL, in such a way one may easily establish a correspondence.

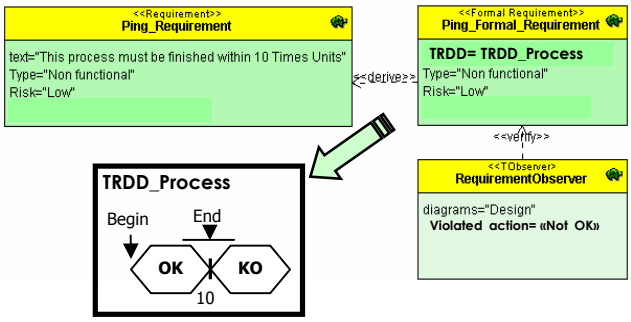


Fig.2. Requirement Diagrams in TURTLE including TRDD

V Observers synthesis

With observers, we extend the class diagram which defines the system's architecture and those activity diagrams which define the behavior of the objects to be observed. To every requirement defined by one TRDD corresponds one observer's class. We modify the behavior and the interfaces of the system's classes to be observed in order to collect observation data and, if one strong requirement is violated, to pre-empt the system's execution.

In TURTLE, objects communicate by rendezvous (synchronization) offers *à la* LOTOS. The observer and the observed objects also communicate by rendezvous. Of interest to us is the temporal operator which limits the amount of time that may be allocated to offering a rendezvous. That operator is named "time limited offer".

Fig. 3 sketches the translation process between the TRDD and the activity diagram of the observer associated with the temporal requirement defined in Fig. 2. The Observer behavior includes a *time limited offer* operator which bounds the amount of time allocated to a process to offer a rendezvous communication to its environment (see Step 2. in Fig. 3). The *time limited offer* starts just after the *"Begin"* action was executed. The observer expects *"End"* to occur before 10 time units (left path of *time limited offer*). After 10 time units, the observer executes the "*Not_OK"* action (right path of *time limited offer*).

Note: Observers synthesis algorithms and translation tables are detailed in [FON 07].

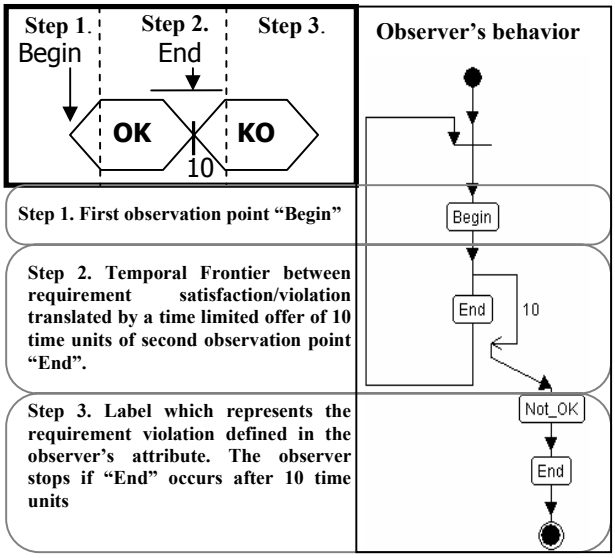


Fig.3. Observer synthesis: an example

VI Limitations

The approach discussed in the paper works under the following assumptions.

- *Observation points* by which observers and TURTLE objects may synchronize must not block the behavior of the TURTLE objects. In other words, observers must remain passive during the observation phase associated with the requirement addressed by the observer [JAR 88]. For instance, in Fig.3, if the observer's exiting action *"End"* occurs before entering action *"Begin"*, the observer should be able to accept rendezvous offers and therefore perform synchronization actions in this unexpected order.

- *Parallelism limitation*. If one observed action belongs to two parallel processes (inside the same object or not), the observer may deliver a wrong diagnosis. We decide to build one observer per observed object.

- *Preemption processes*. If a strong temporal requirement is violated, the observer must pre-empt all the system's objects. This increases the state space of the system's behavior.

- *Temporal indeterminism in the RTL tool*. This limitation comes from the RT-LOTOS semantics associated with the "time limited offer". The problem arises at the date Tmax which fixes the upper bound of the time limiter offer. At Tmax, the offer is possible but not mandatory. Therefore, the reachability graph contains two paths corresponding to a requirement satisfaction and violation, respectively.

VII Conclusions and Future Work

TURTLE is a real-time UML profile designed with formal verification in mind. It particularly applies to temporal requirement verification. The profile was recently extended with SysML requirement diagrams.

The paper shows how SysML requirement diagrams are supported by the profile. We insist on the possibility to express formal temporal requirements using TRDD, a visual language based on Timing Diagrams. The main contribution lies in the possibility to automatically derive observers from temporal requirements defined by Timing Requirement Description Diagram. TTool inserts these observers in the relevant design diagrams (class and activity diagrams) as a premise to guide the verification process.

The observer-based verification approach proposed in the paper reuses the RT-LOTOS code generator included in TTool as well as the RTL verification tool. TTool also generates java code from TURTLE models. We plan to extend the proposed approach to the deployment phase of communicating systems. Observers will be generated in Java in order to become probes for a Java simulator.

VIII Acknowledgements

Acknowledgements are due to Ludovic Apvrille for fruitful discussions on observer synthesis.

IX References

[AML 99] N. Amla, E.A. Emerson, and K.S. Namjoshi. "Efficient Decompositional Model Checking for Regular Timing Diagrams", In Conference on Correct Hardware Design and Verification Methods (CHARME 1999). Springer-Verlag, pp. 67-81, September 1999.

[APV 04] L. Apvrille, J.-P. Courtiat, C. Lohr, P. de Saqui-Sannes, "TURTLE: A Real-Time UML Profile Supported by a Formal Validation Toolkit", IEEE Trans. on Software Engineering, Volume 30, Number 7, page 473-487, July 2004.

[APV 05] L. Apvrille, P. de Saqui-Sannes, F. Khendek, "Real-time UML design synthesis from sequence diagrams" (in French), French conference on protocol engineering (CFIP'05), Bordeaux, France, March 2005.

[BRA 05] V. Braberman, N. Kicillof and A. Alfonso. "A Scenario-Matching Approach to the Description and Model-Checking of Real-Time Properties", Volume 31, Number 12, page 1028-1041, IEEE Trans. on Software Eng. December 2005.

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

[CHO 05] H. Chockel and K. Fisler, "Temporal Modalities for Concisely Capturing Timing Diagrams", Correct hardware design and verification methods, 13th IFIP WG 10.5 advanced research working conference, CHARME 2005, Saarbrücken, Germany, October 2005.

[COU 00] J.P. Courtiat, C.A.S. Santos, C. Lohr., B. Outtaj, "Experience with RT-LOTOS, a Temporal Extension of the LOTOS Formal Description Technique", Computer Communications, Volume 23, Number 12, page 1104-1123, 2000.

[DAM 01] W. Damm and D.Harel. "LSCs:Breathing Life into Message Sequence Charts", Formal Methods in Systems Design, Volume 19, Number 1, Page 45-80, 2001.

[FON 07] B. Fontan, P. de Saqui-Sannes, L. Apvrille. "Génération et synthèse automatique d'observateurs à partir d'exigences temporelles formelles",Technical report ENSICA, February 2007. http://dmi.ensica.fr/spip.php?article776

[FRA 01] M. Fränzle and K. Lüth, "Visual Temporal Logic as Rapid Prototyping Tool", Computer Languages, Volume 27, Page 93--113, 2001.

[HAS 06] J. Hassine, J. Rilling and R. Dssouli. "Timed Use Case Maps", In System Analysis and Modeling: Language Profiles, 5th International Workshop, SAM 2006, Kaiserslautern, Germany, Page 99-114, May-June 2006.

[JAR 88] C. Jar, J.-F. Monin, R. Groz, "Development of Veda, a Prototyping Tool for Distributed Algorithms**,"** IEEE Transactions on Software Engineering, Volume 14, Number 3, Page 339-352, March 1988.

[LAM 06] A. Van Lamsweerde, "Goal-Oriented Requirements Engineering". From System Objectives to UML Models to Software Specifications, Wiley, 2006.

[MIL 89] R. Milner, "Communication and Concurrency," Prentice Hall, 1989.

[OBJ] http://www.objectiver.com/

[OMG 03] Object Management Group, "Unified Modeling Language Specification", Version 1.5, http://www.omg.org/docs/formal/03-03-01.pdf, March 2003.

[RTL] http://www.laas.fr/ RT-LOTOS/

[SYS 06] http://www.SysML.org/docs/specs/SysML-v1-Draft-06-03-01.pdf

[TTOOL] http://labsoc.comelec.enst.fr/turtle/

Coexistence of Time-Triggered and Event-Triggered Traffic in Switched Full-Duplex Ethernet Networks

Joachim Hillebrand, Mehrnoush Rahmani,
Richard Bogenberger
BMW Group, Research and Technology
Hanauer Strasse 46
80788 Munich, Germany
{firstname.lastname}@bmw.de

Eckehard Steinbach
Technische Universität München
Institute of Communication Networks
Media Technology Group
80290 Munich, Germany
eckehard.steinbach@tum.de

Abstract

In the recent years, the Ethernet technology has grown rapidly, mainly due to its applicability in local area networks. High data rates, low cost, collision reduction with the full-duplex approach and the elimination of chaining limits inherent in hubbed Ethernet networks have made the switched Ethernet a dominant network technology. Although the switch technology has improved significantly, the delays appearing in the switches are still not acceptable for time critical applications. This is specially the case when several cascaded switches are applied. Within the scope of developing a new network architecture for the in-vehicle communication, the time constraints of a switched Ethernet network are addressed in this paper. In order not to exceed the delay bounds of time critical applications in the automotive field, a cost-effective approach is proposed and analyzed for several cascaded switches.

1 Introduction

In current automotive communication systems, a significant number of network nodes utilizes a time-triggered communication concept [1]. The nodes obtain network access at specific time periods, also called time slots. Since it is ensured that there is no other network traffic during that time slot, the assigned transmitting network node can exclusively use the network resources at that time. This leads to very short delay times in the transmission. An example for such a system would be the Flexray bus [2], where in practice 4 to 20 network nodes communicate by using total cycle times of 1 to 5 milliseconds.
A different approach is followed in event-triggered networks. Here, the nodes may obtain network access at any time instant. Therefore, it is generally not possible to transmit event-triggered traffic over a time-triggered network. Since event-triggered traffic may happen at any time, it would disrupt time-triggered traffic in dedicated time slots [3]. A very special representative of an event-triggered network is the Full-Duplex Switched Ethernet (FDSE). FDSE network nodes have an exclusive point-to-point link connected to a central Ethernet switch. Even low-cost solutions of the FDSE show high switching performance with low latency and jitter in the range of tens of microseconds [4]. When two end nodes exchange traffic over a simple star topology with one switch, it is ensured that other nodes are not interfered by the traffic due to the switching capabilities in the central switch. Based on realistic automotive network scenarios, we assume the following:

- The amount of time-triggered traffic is small compared to the amount of event-triggered traffic such as bulk and multimedia traffic
- The number of time-triggered nodes is limited in the controlled environment
- Event-triggered traffic is not utilized for high priority control applications unlike the time-triggered traffic
- Allowed delay and jitter for time-triggered traffic is larger than switch latencies (Analyzed in Section 3).
- Event-triggered nodes may not be equipped with the functionality to detect time slots

Several approaches [5] have been introduced for real-time Ethernet switched networks, especially in the automation field. However, those solutions are optimized for industrial control applications where bulk and multimedia traffic are not present. They either employ specific hardware like ProfiNet [6] and EtherCAT [7], or adapt protocols limited for industrial use like the Ethernet Industrial Protocol [8]. The cost of such solutions does not scale to the automotive sector, where a large number of samples is needed for a model range of cars. Another interesting approach is introduced by RTNet [9] that provides a more flexible solution for time critical applications with standardized hardware components. However, RTNet does not allow to connect event-triggered network participants to a switch connected to time-triggered nodes.

In this paper, we introduce a low cost and flexible switching mechanism for FDSE networks that can be utilized by both, time-triggered and event-triggered data.

2 Introduction to the latencies of store and forward switches

Today's Ethernet switches may support priority scheduling by containing two or more output queues per port, where for high and low priority data different queues with different QoS levels are reserved. Depending on the related scheduling schemes, the switch scheduler alternates between the priority queues as shown in Fig.1. Priority identification can be performed based on physical Ethernet ports, MAC addresses, priority tagging according to IEEE 802.1p [10] or higher layer information.

Independent of the applied scheduling algorithm, packets

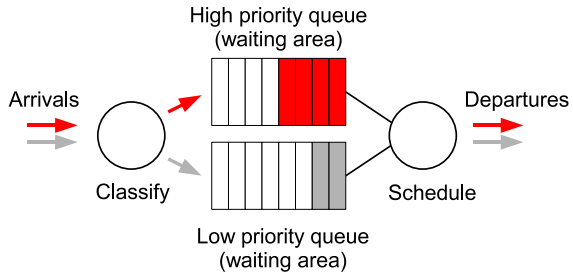


Figure 1. Priority queues in a switch port

in the queue with the highest priority can be delayed due to head-of-line-blocking (HoLB) as it is illustrated in Fig.2 with an example. Head-of-line-blocking is a common

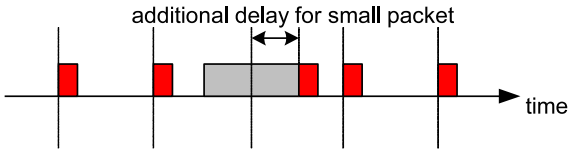


Figure 2. Head-of-line blocking

problem for networks conveying different sized packets [11]. The delay occurs when a high priority packet enters its related queue while a large packet from a lower priority queue is being sent. In general, the delay for a packet passing a switch can be written as:

$$t_{sw} = t_{sf} + t_{sp} + t_{holb} \quad (1)$$

Here t_{sf} represents the store-and forward time, t_{sp} represents the switch processing time and t_{holb} the delay due to head-of-line blocking [12]. In the following section, we analyze the impact of t_{holb} in the packet end-to-end transmission time.

3 Cooperative time slot mechanism

3.1 Performance analysis for cascaded switches with a constant data rate

When using a time slot mechanism in a switched network, it has to be ensured that frames are always transmitted within the respective time slot intervals. For store and forward switches the frame size may vary as much as the transmission still fits into the respective time slot. Depending on the network topology, the delay on each transmission path may consist of one or several switch delays. In the following, we consider several cascaded store and forward switches between the sender and receiver nodes. In order to manage both, the time-triggered and the event-triggered applications in a network, we propose different prioritized queues for switch ports. As a compromise, two queues per each port, one for the time-triggered and one for the event-triggered data seem to be sufficient. We call this approach $Cooperative\ time\ slot\ mechanism$, because it enables the interconnection of time-triggered and event-triggered devices via one switch. Figure 3 shows this idea for one switch, two event-triggered and two time-triggered nodes. The time slots are generated by a clock generator connected to the switch. In the case of TDMA-based synchronization, the clock is treated like a time-triggered node, because it can access the network only within a time slot. As mentioned in Section 5, the incom-

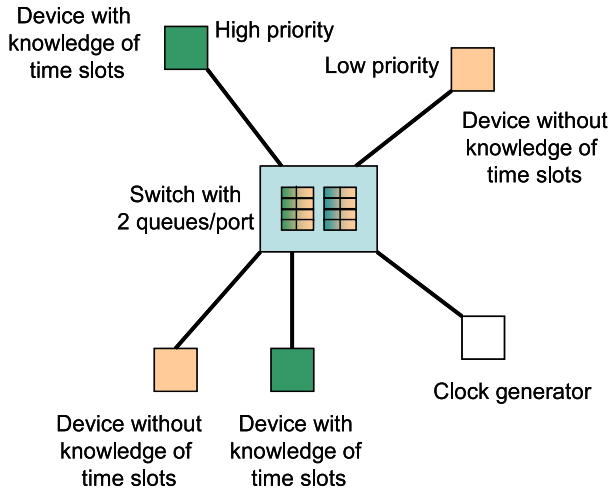


Figure 3. Cooperative time slot mechanism

ing packets are assigned to appropriate queues depending on their priorities. Time-triggered packets are assigned to the high priority queue while event-triggered packets are routed to the low priority queue. If a time triggered node sends a data packet to an event-triggered node, it first uses its respective time slot to access the network at the predefined time. In the switch, the data packet will be forwarded to an appropriate queue due to its destination address and priority. The packet will be sent to the event-triggered node as soon as required resources are available.

If vice versa, the event-triggered node sends the packet to the switch as soon as the required resources are available. In the switch the packet will be forwarded to the appropriate output queue and sent in respective time slots to the time-triggered destination.
In order to determine the efficiency of the cooperative time slot mechanism, we calculate the end-to-end worst case latency for the time-triggered data caused by store and forward switches. We first need to make certain assumptions about the network we are dealing with. Following assumptions are set forth:

- All switches in the network are store and forward switches with the values defined for the switch delay t_{sw} in equation (1)
- All receiving and sending ports are functioning independently (HW router and full-duplex)
- Packet source and sink are separated by n switches
- There is no gap between the time slots for the high priority data transmission

Equation (2) gives the store and forward delay as a function of the high priority frame size (P_{TT}) in bytes and the transmission bit rate b in bits/s. In the same way, Equation (3) considers the fact that head-of-line blocking is caused by frames of the size P_{ET}.

$$t_{sf} = \frac{P_{TT} \cdot 8}{b} \tag{2}$$

$$t_{holb} = \frac{P_{ET} \cdot 8}{b} \tag{3}$$

In a time slot method, the maximum number of time-triggered nodes depends on the cycle time t_{cycle}, the number of cascaded switches n as well as the worst case switch delay t_{sw}. Considering the assumptions mentioned above, equations (2) and (3) and a constant network throughput capacity between the packet source and sink, we achieve a number k for the possible time-triggered nodes in the network:

$$k = \frac{t_{cycle}}{n \cdot t_{sw}} = \frac{t_{cycle}}{n \cdot (t_{sf} + t_{sp} + t_{holb})} = \frac{t_{cycle}}{n \cdot \left(\frac{P_{TT} \cdot 8}{b} + t_{sp} + \frac{P_{ET} \cdot 8}{b}\right)} \tag{4}$$

In the worst case, the low priority packet P_{ET} entailing the head-of-line blocking has the maximum packet size, e.g., 1518 bytes for Ethernet packets while the high priority packet P_{TT} is small, e.g., 64 bytes. By applying equation (4) and the minimum switch processing time t_{sp} = 10 μs from [12], we achieve the result presented in Figure 4 for the number of time-triggered nodes depending on the number of cascaded switches in the network. It can be seen that the possible number of time-triggered nodes decreases by increasing the number of switches. This result confirms our statement that the delay caused by switches influences the entire transmission time. In order to fulfill the time-triggered communication requirements in a switched Ethernet network a tradeoff should be made between the number of time-triggered nodes and switches according to the results achieved in Figure 4. In the same way, the number of possible time-triggered nodes in a switched network can be calculated depending on the size of the event-triggered packets entailing head-of-line blocking. Figure 5 shows the result when assuming three switches between the packet source and sink. According to Figure 5, the larger the event-triggered frame size is, the lower the number of time-triggered nodes should be in order to be able to fulfill the timing requirements.

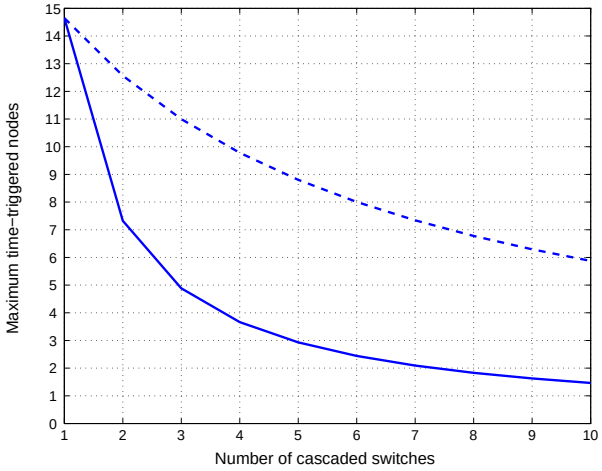


Figure 4. Maximum number of time-triggered nodes with t_{sp}= 10 μs, P_{TT} = 64 bytes, P_{ET} = 1518 bytes, t_{cycle}= 2 ms and b = 100 Mbit/s as a function of the number of switches (solid curve) compared with a network with different bit rates b_1 = 100 Mbit/s and b_2 = 1000 Mbit/s (dashed curve).

3.2 Performance improvement with high data rate inter-switch connections

So far, we analyzed the performance of the cooperative time slot mechanism for time-triggered applications in a network with a constant transmission rate of 100 Mbit/s. However, the performance of a switched network can be improved by optimizing its design. Considering a design with two different throughput capacities, i.e., 1000 Mbit/s segments for the inter-switch connections and 100 Mbit/s segments for the connections to end nodes, we continue our calculations in the following. The number of possible time-triggered nodes k can now be calculated as:

$$k = \frac{t_{cycle}}{n \cdot t_{sp} + \frac{P_{TT} \cdot 8}{b_1} + \frac{P_{ET} \cdot 8}{b_1} + (n-1) \cdot \left(\frac{P_{TT} \cdot 8}{b_2} + \frac{P_{ET} \cdot 8}{b_2}\right)} \tag{5}$$

where b_1 is equal to 100 Mbit/s and b_2 is 1000 Mbit/s. Figures 4 and 5 show the corresponding improvements comparing with the results achieved by only 100 Mbit/s segments. It can be seen that by optimizing the network design, the number of possible time-triggered nodes in-

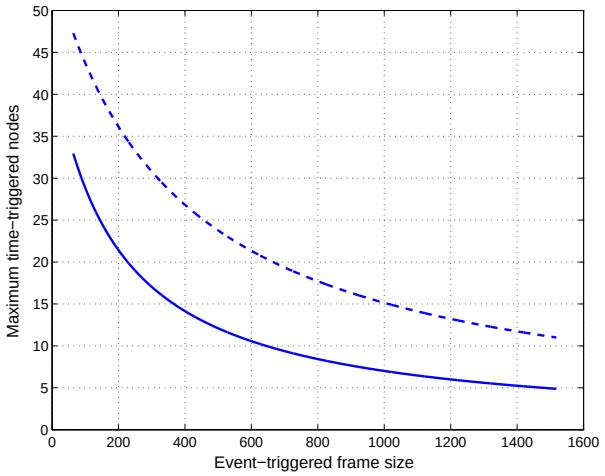


Figure 5. Maximum number of time-triggered nodes with t_{sp}= 10 μs, P_{TT} = 64 bytes, n = 3, t_{cycle}= 2 ms and b = 100 Mbit/s as a function of P_{ET} (solid curve) compared with a network with different bit rates b_1 = 100 Mbit/s and b_2 = 1000 Mbit/s (dashed curve).

creases significantly for the same number of cascaded switches and event-triggered frame size.

4 Conclusion and future work

In this paper, we have discussed the possibility of transmitting time-triggered traffic in combination with event-triggered traffic over Full-Duplex Switched Ethernet networks. A new approach called Cooperative Time Slot Mechanism has been introduced. By taking advantage of parallel queuing mechanisms in switches, the method allows time-triggered and event-triggered traffic to pass switches without interferences. The approach is based on the assumption that event-triggered traffic is made up of bulk or multimedia traffic with generally lower priority than the time-triggered traffic. The analysis of delay restrictions shows the possibility to design such a network by limiting the number of switches, or limiting the size of event-triggered frames, or by adding high data rate inter-switch connections. Based on the choice of parameters, a network can be realized to support time-triggered and event-triggered traffic without the need for two separate networks. In the future work, we will analyze the possibilities to add event-triggered traffic with high priority to the Cooperative time slot mechanism. Furthermore, synchronization mechanisms for the time-triggered traffic will be investigated.

References

[1] Nicolas Navet et al. Trends in automotive communication systems. *Proceedings of IEEE*, 93(6), June 2005.

[2] FlexRay Consortium. *FlexRay Communications System, Protocol Specification*, version 2.1 edition, 2005.

[3] Amos Albert. Comparison of event-triggered and time-triggered concepts with regard to distributed control systems. *Embedded World 2004*, pages 235–252, 2004.

[4] John Wernicke. Simulative analysis of QoS in avionics networks for reliably low latency. `http://www.clas.ufl.edu/jur/200601/papers/paper_wernicke.html`, 2006.

[5] Kai Lorenz Arndt Lder, editor. *IAONA Handbook - Industrial Ethernet*. Industrial Automation, Open Networking Alliance e.V., Universittsplatz 2, 39106 Magdeburg, Germany, 3rd edition, 2005.

[6] Siemens. IEC/PAS 62407 Real-time Ethernet PROFINET IO. International Electrotechnical Commission (IEC), `http://webstore.iec.ch/webstore/webstore.nsf/artnum/034395`, 2005.

[7] EtherCAT Technology Group (ETG). Ethercat ethernet control automation technology, *publicly available specification*. International Electrotechnical Commission (IEC), `http://webstore.iec.ch/webstore/webstore.nsf/artnum/034392`, 2004.

[8] ODVA Association. Common industrial protocol (cip). `http://www.odva.org`.

[9] Jan Kiszka et al. RTNet - a flexible hard real-time networking framework. *Emerging Technologies and Factory Automation 2005, EFTA 2005*, 1, 2005.

[10] IEEE Project 802. IEEE 802.1p: Supplement to MAC Bridges: Traffic calss expediting and dynamic multicast filtering. Incorp. in IEEE Standard 802.1D, Part 3: Media Access Control (MAC) Bridges, 1998.

[11] SMSC Max Azarov. Approach to a latency-bound ethernet. IEEE 802.1 AVB group, 2006.

[12] Oe. Holmeide and T. Skeie. VoIP drives the Real-time Ethernet. *Industrial Ethernet Book (IEB)*, 5, 2001.

Abstraction Techniques for Extracted Automata Models

Susanne Kandl
Institute of Computer Engineering
Vienna University of Technology
Vienna, Austria
Email: susanne@vmars.tuwien.ac.at

Abstract — *In this paper we present the application of abstraction techniques for automata models. We give an overview on a state-of-the-art method to reduce the complexity of an automaton model without loosing essential information on the behavior of the modeled system (predicate abstraction). We focus on the applicability of the presented methods, especially on models that are directly extracted from the C source code of a system. We present the process for automated model extraction that yields an automaton model we can use for a verification and testing framework. We show how to apply different abstraction techniques on a case study from the automotive domain and evaluate the resulting state space reduction.*

Keywords: *Modeling, Abstraction, Predicate Abstraction, Model Checking, Test Case Generation*

I. Introduction

Formal methods, like model checking [1], can be used to verify a system or for the automated generation of test cases to test the system [2]. For both, the verification and test case generation, an *automaton model* of the system has to be built. This can be done manually by a human by analyzing the system and modeling the relevant aspects of the system as a transition system. Another way to derive a model is to automatically extract it from a description of the system given as C source code. Especially the second approach lacks of the state space explosion for the model, because all the system behavior (including the semantics of the C program statements) is directly transformed into the automaton model. The resulting model has to be further adapted to simplify the model, and thus to reduce the state space.

The article is organized as follows: Section 2 describes the principles of predicate abstraction. In Section 3 our approach for the model generation is explained. In Section 4 the appliance of abstraction techniques to our case study is shown. Section 5 gives an overview on related work and available tools. Finally we conclude with preliminary results and future work.

II. Predicate Abstraction

In predicate abstraction the system is described as a set of logical formulas [3], [4]. The idea is to find an appropriate abstract description of the system that can be mapped to the concrete transition system. A transition system is given by a set of *states* Q, a set of *symbols* Σ, the *transition function* f, an *initial state* q_0 and a set of *accept states* F. The transition system is defined by a set of transition rules. Each rule defines a transition function f. An execution of the system is a sequence of states $q_0, q_1, ..., q_n, q_{n+1}$, where q_0 is the initial state and $q_{i+1} = f(q_i)$ with $0 \leq i \leq n$. An *abstraction function* α maps a set of concrete states to a set of abstract states, while reversely the *concretization function* γ maps the abstract states to the concrete states. In detail, the abstract system is defined by a concrete system and a set of n predicates: $\phi_1, \phi_2, ..., \phi_n$. Each state q_A of the abstract state space is an assignment to the indices 1 through n to the abstract form of the state q_C of the concrete state space:

$q_A = \alpha(q_C)$ whenever $\forall i : q_A(i) = \phi_i(q_C)$.

A detailed introduction to predicate abstraction can be found in the master thesis from Satyaki Das (Stanford University, 2003) [5].

The main challenge is to find suitable predicates for the abstract system, in this way, that the resulting abstract system shows the same behavior as the underlying concrete system regarding to the system properties. That means, that a property g that holds on the original concrete system should still hold on the abstract system, whereas a property h which is violated in the concrete model should also be not valid within the abstract representation. A common technique to automatically determine the abstrac-

tion predicates is called *counterexample-guided abstraction refinement*: The atomic predicates in the verification condition are used as the initial set of predicates. Then the abstract system is constructed. If a property that holds on the concrete system can still be verified on the abstract model, the abstraction process was successful, otherwise an abstract counterexample is produced. If a concrete counterexample exists that corresponds to the abstract trace, a concrete counterexample has been detected. Otherwise the abstract counterexample can be analyzed and used to discover new predicates for the predicate abstraction. Figure 1 [4] shows the principle of counterexample-guided predicate abstraction.

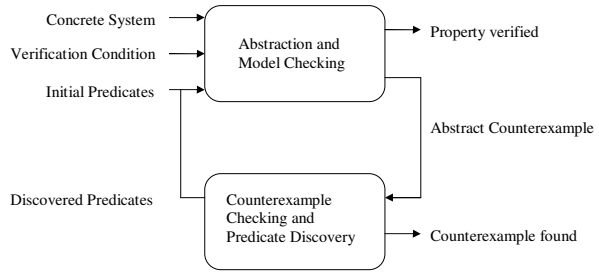


Figure 1: Counterexample-Guided Predicate Abstraction [4]

III. Model Generation

As mentioned before, we need an automaton model for purposes of verification and test case generation. This model is automatically extracted from the C source code. This is realized by static analysis of the C source code, building the syntax tree and interpreting the basic statements of the syntax tree to generate the automaton model. The model is formulated in the automaton language of the model checker NuSMV[1], we are using for our verification and test case generation framework.

Listing 1 shows a small C program, for which the automatically generated NuSMV-model is given in Listing 2 (the additional variable *sequence_nr* represents the program counter).

[1]http://nusmv.irst.itc.it/

```
1 int test(int x, int a) {
2   if (x == 1) {
3     x=10;
4   } else if (a == 2) {
5     x=20;
6   } else {
7     x=x+1;
8   }
9 }
```

Listing 1: C source code

```
1  MODULE main
2  VAR
3    sequence_nr: 0..255;
4    v0_x: 0..255;
5    v1_a: 0..255;
6  ASSIGN
7    init(sequence_nr):= 16;
8    next(sequence_nr):= case
9      sequence_nr= 2: 1;
10     sequence_nr= 5: 1;
11     sequence_nr= 8: 1;
12     sequence_nr= 12 & (v1_a=2) : 5;
13     sequence_nr= 12 & !(v1_a=2) : 8;
14     sequence_nr= 16 & (v0_x=1) : 2;
15     sequence_nr= 16 & !(v0_x=1) : 12;
16   esac;
17   next(v0_x):= case
18     sequence_nr= 2: 10;
19     sequence_nr= 5: 20;
20     sequence_nr= 8: v0_x + 1;
21   esac;
```

Listing 2: NuSMV model

IV. Application of Abstraction Techniques

For our verification and testing framework we used a case study from the automotive domain provided by one of our industry partners. The case study is a control system given as an ANSI-C program. It has approximately 500 lines of code and inhabits 16 variables, thirteen of them of boolean data type, two 16-bit integers and one (unsigned) 8-bit integer. The control flow graph has 76 branches. In the first pass the model is extracted in a straightforward way. This yields a model with a state space of approximately 2^140 states. The applied model

checker NuSMV is not able to deal with models of this size. In a second phase we are simplifying the model by the application of a method, we want to refer as *data type reduction*. Analyzing the automaton model yields the interesting result that within the program execution not all values of a variable are indeed assigned to the variable. That means, from a variable defined as an 8-bit integer, only a discrete list of actual values is used within the program and thus has to be considered in the model. Depending on, whether a list of values or value ranges have to be modeled, the data type definition of a variable can be reduced to the definition of an enumeration of concrete numbers or the specification of a list of equivalence classes. Applying this simplification method yields in a model that has a state space of 2^53 states. The model checker NuSMV has no problems to process this model. Last but not least, we identified predicates for further abstraction of the model. The predicate-based abstracted model is again verified against all the system properties that hold on the original model, to ensure that the predicate abstraction has not affected the system behavior. Integrating this last step of abstraction results finally in a model with a state space of 2^39 states. The model checker NuSMV performs well on models of this size.

Table 1 summarizes the applied abstraction techniques and the model size of the resulting models. In addition also a manual model of the system was built. *ExtrMod* is the extracted model, *DTRed* refers to the data type reduction, *PredAbstr* stands for predicate abstraction and *ManMod* is the manual model. The numbers are given as the state space of the different representations of the automaton model.

ExtrMod	DTRed	PredAbstr	ManMod
2^140	2^53	2^39	2^43

Table 1: Comparison of Model Size

Reducing the state space with the above described methods improved the performance and scalability of our verification and test case generation framework significantly.

V. Related Work

First works concerning abstraction considerations for model checking techniques were published in the mid of the 90's, to mention the master thesis of Long *Model Checking, Abstraction and Compositional Verification* at CMU (Carnegie Mellon University) [6], works from Clarke et al. [7] and Alur et al. [8]. Experiences with a prototype implementation for predicate abstraction are described in Das et al. [3]. The principle of *counterexample-guided refinement* is described in Clarke et al. [9] or Das et al. [4]. Alternative approaches for automated abstraction are using proofs of unsatisfiability, introduced in McMillan and Amla [10], or *thread-modular abstraction refinement* (using thread-modular assume-guarantee reasoning), described in Henzinger et al. [11]. Also works from Shankar (SRI International) have to be mentioned, where deduction is used to construct a finite-state approximation of a program that preserves the property of interest [12]. Recent works deal with the improvement of the efficiency of automated abstraction techniques, for instance Das and Dill [13], Henzinger et al. [14], Clarke et al. [15] or Henzinger et al. [16]. Ball (Microsoft Research) et al. describe boolean and cartesian abstraction for model checking C programs in [17]. The techniques are implemented in tools like C2BP (a tool for automatic predicate abstraction of C programs) [18] or SLAM (a model checker, that integrates predicate abstraction with heuristic approximations) [19], furthermore the model checker BLAST [20] also incorporates automatic abstraction. Another tool for automated abstraction for ANSI-C programs based on predicate abstraction is SATABS, described in Clarke et al. [21]. This tool also supports the model checker NuSMV, we are interested in.

VI. Preliminary Results and Future Work

So far, we are able to generate an automaton model automatically from the C source code of the system we want to formally verify and we need for the automated generation of test cases. With the introduced abstraction techniques we achieved a significant reduction of the model size and thus an improvement of the performance of the applied model checker.

Due to the availability of others tools that incorporate abstraction refinements automatically, we plan to evaluate a predicate abstraction tool (e.g. SATABS) to our case study and compare the results with our preliminary results achieved in the ongoing research project. The above mentioned related works and tools have been also applied to industrial case studies, but it has to be evaluated of which complexity the treated case studies were. Many model checking techniques are, in general, only applied to systems dominated by boolean variables and characterized by a simple control flow graph. The case studies we are concerned with in the recent research project are, amongst others, adaptive control systems and it has to be analyzed, how applicable the methods described in this paper are to this kind of systems.

VII. Acknowledgments

This work has been partially supported by the FIT-IT research project "Systematic test case generation for safety-critical distributed embedded real time systems with different SIL levels (TeDES)"; the project is carried out in cooperation with TU-Graz, Magna Steyr, and TTTech.

References

[1] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, 2000.

[2] Susanne Kandl, Raimund Kirner, and Peter Puschner. Development of a framework for automated systematic testing of safety-critical embedded systems. In Wilfried Elmenreich, Gregor Novak, and Ralf E.D.Seepold, editors, *Proc. of WISES'06*, June 2006.

[3] Satyaki Das, David L. Dill, and Seungjoon Park. Experience with predicate abstraction. In *CAV*, 1999.

[4] S. Das and D. Dill. Counter-example based predicate discovery in predicate abstraction. In *In Formal Methods in Computer-Aided Design*. Springer, 2002.

[5] Satyaki Das. *Predicate Abstraction*. PhD thesis, Stanford University, 2003.

[6] D.E. Long. *Model Checking, Abstraction, and Compositional Verification*. PhD thesis, Carnegie Mellon University, 1993.

[7] E. M. Clarke, O. Grumberg, and D. E. Long. Model checking and abstraction. *ACM Transactions on Programming Languages and Systems*, 16(5), September 1994.

[8] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P. Ho, X. Nicolin, A. Olivero, J. Sifakis, and S. Yovine. Discrete abstractions of hybrid systems. In *The algorithmic analysis of hybrid systems. Theoretical Computer Science, 138:*, 1995.

[9] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In *CAV*, 2000.

[10] Kenneth L. McMillan and Nina Amla. Automatic abstraction without counterexamples. In *TACAS*. Springer, 2003.

[11] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Shaz Qadeer. Thread-modular abstraction refinement. In *CAV*. Springer, 2003.

[12] Natarajan Shankar. Verification by abstraction. In *LNCS*, 2002.

[13] Satyaki Das and David L. Dill. Successive approximation of abstract transition relations. In *Proc. of the 16th Annual IEEE Symposium on Logic in Computer Science*, 2001.

[14] Thomas A. Henzinger, Ranjit Jhala, Rupak Majumdar, and Gregoire Sutre. Lazy abstraction. In *Symposium on Principles of Programming Languages*, 2002.

[15] Edmund Clarke, Orna Grumberg, Muralidhar Talupur, and Dong Wang. Making predicate abstraction efficient: How to eliminate redundant predicates. In *CAV*, 2003.

[16] T. A. Henzinger, R. Jhala, R. Majumdar, and K. L. McMillan. Abstractions from proofs. In *Proc. of the 31st ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, New York, NY, USA, 2004. ACM Press.

[17] Thomas Ball, Andreas Podelski, and Sriram K. Rajamani. Boolean and Cartesian abstraction for model checking C programs. *LNCS*, 2031, 2001.

[18] Thomas Ball, Rupak Majumdar, Todd D. Millstein, and Sriram K. Rajamani. Automatic predicate abstraction of C programs. In *SIGPLAN Conf. on Programming Language Design and Implementation*, 2001.

[19] T. Ball, B. Cook, S. Das, and S. Rajamani. Refining approximations in software predicate abstraction. In *TACAS*. Springer, 2004.

[20] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Software verification with Blast. In *In 10th International Workshop on Model Checking of Software (SPIN)*. Springer, 2003.

[21] Edmund Clarke, Daniel Kroening, Natasha Sharygina, and Karen Yorav. SATABS: SAT-based predicate abstraction for ANSI-C. In *TACAS*, volume 3440 of *LNCS*. Springer, 2005.

A Penalty Upper Bound in an Optimal Schedule of a Set of Soft Real-Time Tasks*

Arezou Mohammadi and Selim G. Akl
School of Computing, Queen's University
Kingston, Ontario, Canada K7L 3N6
{arezou, akl}@cs.queensu.ca

Abstract

A soft real-time task is one whose completion time is recommended by a specific deadline. However, should the deadline be missed, such a task is not considered to have failed; only the later it finishes, the higher the penalty that is paid. For a set of soft real-time tasks that are to be scheduled on a single machine, our objective is to minimize the total penalty paid. This optimization problem is NP-hard. We give a formal definition of this problem. Then, we determine an upper bound for the optimal solution of the problem. Numerical results that compare the upper bound with the optimal solution are also provided.

Keywords: soft real-time tasks, upper bound, penalty minimization, optimal scheduling algorithm, simulation results.

1 Introduction

The purpose of a real-time system is to produce a response within a specified time-frame. In other words, for a real-time system not only the logical correctness of the system should be satisfied, but also it is required to fulfill the temporal constraints of the system. Although missing deadlines is not desirable in a real-time system, *soft real-time tasks* could miss some deadlines and the system will still work correctly while certain penalties will have to be paid for the deadlines missed. In this paper, we focus our attention on scheduling of a set of soft real-time tasks.

Consider a system that consists of a set of soft real-time tasks, $T = \{\tau_1, \tau_2, ..., \tau_n\}$. Task τ_i is a **soft real-time** task, meaning that the later the task τ_i finishes its computation after its deadline, the more penalty it pays. A release time r_i, an execution time e_i and a deadline d_i are given for each task $\tau_i \in T$ (see Section 2 for the definition of these terms). The finishing time of each task $\tau_i \in T$, denoted by F_i, depends on the scheduling algorithm which is used to schedule the execution of the tasks [14]. Suppose that the tasks are scheduled by some scheduling algorithm A. A penalty function $P(\tau_i)$ is defined for the task. If $F_i \leq d_i$, $P(\tau_i) = 0$; otherwise $P(\tau_i) > 0$. The value of $P(\tau_i)$ is a non-decreasing function of $F_i - d_i$. The penalty function of a given scheduling algorithm A for a given set T is denoted by $P(T) = \sum_{i=1}^{n} P(\tau_i)$.

In fact, the problem under study in this paper occurs in overload conditions where it can not guaranteed that all tasks can meet their deadlines. In this case, it is compelled to miss some deadlines, while we aim to minimize the penalty that should be paid.

The only fact known about our problem, as is true for most of the problems in this class, is that it is NP-hard [8]. Recently, there has been a lot of progress in the design of approximation algorithms for a variety of scheduling problems in the aforementioned class [9, 2, 16, 3, 1, 5, 6, 10]. Also, in real-time literature, several scheduling algorithms have been proposed to deal with overloads. For instance, one may refer to [4, Chapter 2] and the references therein. A relevant and recent work is [15], in which the problem is studied for the special case of non-preemptive tasks (see Section 2). The authors addressed a more general problem in [13]; namely, the scheduling of a set of preemptive soft real-time tasks (see Section 2) where the objective function is to minimize the total penalties that should be paid for the deadlines missed. In [13], we provided a class of heuristic algorithms and presented simulation results and compared the performances of the proposed algorithms. In this paper, we derive an upper bound for the optimal solution.

The remainder of this paper is organized as follows. We introduce the terminology in Section 2. In Section 3, we formally define the problem to be solved. In Section 4, we find an upper bound for the objective function, and we provide an algorithm that finds the optimal solution. The run time of the algorithm grows exponentially with the number of tasks. Then, in Section 5 we present the simulation results and compare the upper bound with the optimal solution. Section 6 contains the conclusions.

*This work was supported by the Natural Sciences and Engineering Research Council (NSERC) of Canada.

2 Terminology

For a given set of tasks the *general scheduling problem* asks for an order according to which the tasks are to be executed such that various constraints are satisfied. For a given set of real-time tasks, we want to devise a feasible allocation/schedule to satisfy timing constraints. The timing properties of a given task τ_j, where $\tau_j \in T$, refer to the following [11, 12, 17, 7]:

- *Release time* (or *ready time* (r_j)): Time at which the task is ready for processing.
- *Deadline (d_j)*: Time by which execution of the task should be completed.
- *Completion time (C_j)*: Maximum time taken to complete the task, after the task is released.
- *Finishing time (F_j)*: Time at which the task is finished: $F_j = C_j + r_j$.
- *Execution time (e_j)*: Time taken without interruption to complete the task, after the task is started.
- *priority function*: Priority of task τ_j is defined as relative urgency of the task. Priority function is the relative urgency of a task in a given algorithm.
- *Penalty factor (P_j)*: Penalty that should be paid per each time unit after the deadline of task τ_i.
- *Makespan factor (α_j)*: Ratio of C_j to e_j, i.e., $\alpha_j = C_j/e_j$, where e_j and C_j are respectively the execution time of task τ_j and the completion time of the task in the schedule. This factor depends on schedule.

3 Problem Definition

Consider a set $T = \{\tau_1, \tau_2, ..., \tau_n\}$ of n soft real-time tasks. There exists one processor. The tasks are preemptive and aperiodic. For each task τ_i, we assume that r_i, e_i, P_i, and d_i ,which are respectively the release time, execution time, penalty factor and deadline of the task, are known.

We define the penalty function of task τ_i as

$$P(\tau_i) = (F_i - d_i)^+ P_i, \tag{1}$$

where $F_i = r_i + \alpha_i e_i$ is the finishing time of task τ_i, α_i is the makespan factor ($\alpha_i \geq 1$), and

$$(F_i - d_i)^+ = \begin{cases} F_i - d_i & \text{if } F_i - d_i > 0 \\ 0 & \text{otherwise.} \end{cases}$$

A slot is the smallest time unit.

The objective is to minimize $\sum_{i=1}^{n} P(\tau_i)$. Therefore, we can formally express the objective function as follows. Let us define

$$x_{i,t} = \begin{cases} 1 & \text{if the processor is assigned to task } \tau_i \\ & \text{at time slot } t \\ 0 & \text{otherwise} \end{cases}$$

Our goal is to minimize the objective function

$$\sum_{i=1}^{n} (r_i + \alpha_i e_i - d_i)^+ P_i, \tag{2}$$

subject to the following conditions $\Sigma_{i=1}^{n} x_{i,t} = 1$, which means only one processor is working at any given time t, and $\Sigma_{t=1}^{\infty} x_{i,t} = e_i$, meaning that the total time slots assigned to any given task i over time is equal to its execution time.

As mentioned earlier, the problem defined in this section is known to be NP-hard. Thus, the only known algorithm for obtaining an optimal schedule requires time that grows exponentially with the number of tasks. It is desired to find an upper bound for the objective function which, unlike the optimal algorithm, it would be computationally feasible. The upper bound may also be useful for design and comparison purposes.

4 An Upper Bound

In order to determine the upper bound, we refer to the results in [13] and select the priority function of an algorithm, namely algorithm S_8, which has the best solution as compared with the other algorithms discussed in [13]. The priority assigned to each task in algorithm S_8 is in non-decreasing order of P_i/e_i, where for a given task τ_i, P_i and e_i are respectively the penalty factor and the execution time.

We find the upper bound as follows. As mentioned in Section 2, $C_i = \alpha_i e_i$ is maximum time taken to complete the task, after the task is released. Therefore,

$$\alpha_i e_i = e_i + \sum_{\substack{k=1, \\ P_i/e_i < P_k/e_k, r_k < r_i < F_k}}^{n} e_k + \sum_{\substack{l=1, \\ P_i/e_i < P_l/e_l, r_i < r_l < F_i}}^{n} e_l, \tag{3}$$

where τ_k is any task which has arrived before τ_i, has a higher priority than τ_i, and has not been finished when τ_i arrives, and τ_l is any task which arrives after r_i and has a higher priority than τ_i, and finishes before F_i. As a matter of fact, a task cannot be preempted more than once by another task.

It can be verified that

$$\sum_{\substack{k=1,\\ P_i/e_i<P_k/e_k, r_k<r_i<F_k}}^{n} e_k + \sum_{\substack{k=1,\\ P_i/e_i<P_k/e_k, r_i<r_k<F_i}}^{n} e_k \leq \sum_{\substack{j=1,\\ P_i/e_i<P_j/e_j}}^{n} e_j. \quad (4)$$

, where $\sum^{n}_{\substack{k=1,\\ P_i/e_i<P_k/e_k, r_k<r_i<F_k}} e_k$ represents sum of the execution times of the tasks that have arrived before τ_i, but their execution in not finished when task τ_i arrives, and their priority is higher than the priority of task τ_i, and $\sum^{n}_{\substack{k=1,\\ P_i/e_i<P_k/e_k, r_i<r_k<F_i}} e_k$ represents sum of the execution times of the tasks that have arrived after task τ_i and before the finishing time of task τ_i, and their priority is higher than the priority of task τ_i.

Therefore, from (3) and (4), we obtain the following inequality

$$\alpha_i e_i \leq e_i + \sum_{\substack{j=1,\\ P_i/e_i<P_j/e_j}}^{n} e_j.$$

Therefore, we conclude that

$$\sum_{i=1}^{n}(r_i + \alpha_i e_i - d_i)^+ P_i \leq$$

$$\sum_{i=1}^{n}\left(r_i + e_i + \sum_{\substack{j=1,\\ P_i/e_i<P_j/e_j}}^{n} e_j - d_i\right)^+ P_i.$$

We hence obtain the following upper bound for the optimal penalty function

$$\min\sum_{i=1}^{n}(r_i + \alpha_i e_i - d_i)^+ P_i \leq \sum_{i=1}^{n}\left(r_i + e_i + \sum_{\substack{j=1,\\ P_i/e_i<P_j/e_j}}^{n} e_j - d_i\right)^+ P_i. \quad (5)$$

Note on the right hand side of (5) that all of the parameters in this upper bound are known before scheduling and it is not needed to run a scheduling algorithm to find them. Also, the upper bound can be calculated in $O(n^2)$ time, where n is the number of tasks, while finding $\min\sum_{i=1}^{n}(r_i+\alpha_i e_i-d_i)^+ P_i$ is an NP-hard problem.

We need to find the optimal solution to compare it with the results of the upper bound and do not claim that it is the best possible optimal algorithm for the problem. In order to find the optimal solution, we use the following steps: we find all of the $n!$ possible permutations of order of priorities, which are assigned to a set of n soft realtime tasks. Then, we call algorithm A for any individual permutation of priorities, which computes $\Sigma_{i=1}^{n}(F_i-d_i)^+ P_i$ for each of them separately. Finally, we find the minimum of $\Sigma_{i=1}^{n}(F_i - d_i)^+ P_i$ that corresponds to the optimal schedule. The running time of the optimal scheduling algorithm proposed in this section is $O(n!)$.

5 Simulation Results

We have implemented the optimal algorithm and computed the upper bound for $n = 1, 2, \cdots, 8$ for simulation purposes and comparison. Simulation conditions are as follows. Each set of data includes n soft real-time tasks. For each task, we randomly generate r_i, e_i, d_i and P_i. When randomly generating d_i, the condition that $e_i + r_i \leq d_i$ should hold. We generate 20 different data sets with size n and execute the optimal algorithm and compute the upper bound on each data set. We compute the average of the aggregations of the termination times of the 20 simulations for data set with size n. The simulation is done for the algorithms for $n = 1$ to 8.

The optimal algorithm finds $\min\Sigma_{i=1}^{n}(F_i - d_i)^+ P_i$, where F_i is the finishing time of task τ_i. Figure 1 compares the results of the simulations by plotting the penalty to be paid versus the number of tasks. Note in the figure that the upper bound coincides with the optimal solution at $n = 1$. Also, we observe that the ratio of the upper bound to the optimal solution is less than 1.09. We have also computed and plotted, in Figure 2, the upper bound for the penalty to be paid versus the number of tasks for $n = 1, 2, \cdots, 500$. Note that while it is computationally infeasible to find the optimal penalty for $n > 8$, our upper bound can be easily calculated for large numbers of tasks.

6 Conclusions

In this paper, we studied the problem of scheduling a set of soft real-time tasks under overload conditions such that the total penalty to be paid is minimized. The problem is NP-hard. In other words, it is not known whether an optimal schedule can be found in polynomial time.

We have provided an upper bound for the objective function. The running time of computing the upper bound is $O(n^2)$, where n is the number of tasks. Therefore, it is feasible to compute the upper bound for a set of large real-time tasks in a short time. In order to determine the upper bound, we selected the priority function of an algorithm which has the best solution as compared with the other algorithms discussed in [13].

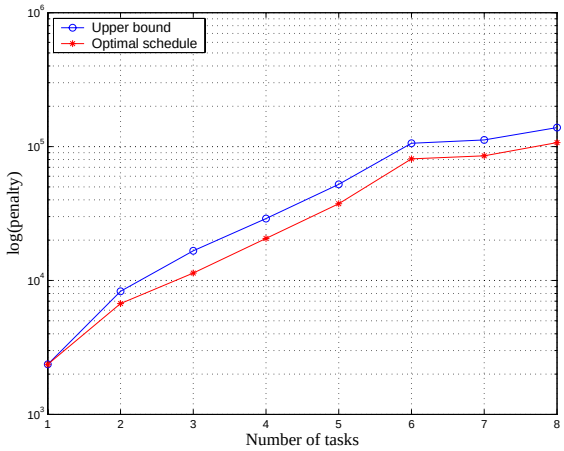


Figure 1: The total penalty of the optimal solution and the upper bound in (5).

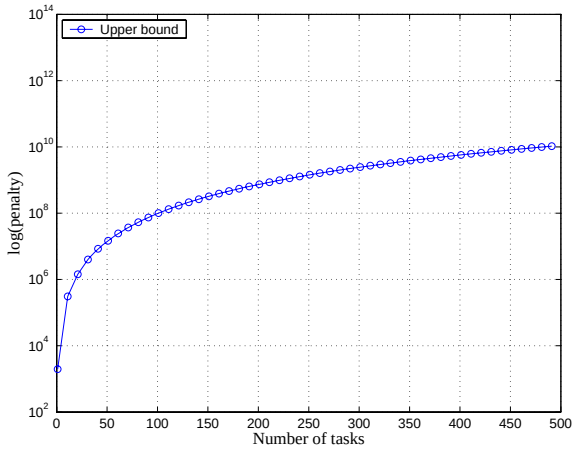


Figure 2: The upper bound of the total penalty for a large number of tasks.

Future work may include computing the ratio of our upper bound to the optimal solution.

References

[1] J. M. Arnaout and Gh. Rabadi, "Minimizing the toal weighted completion time on unrelated parallel machines with stochastic times," *Proceedings of Simulation Conference*. Winter 2005.

[2] I. D. Baev, W. M. Meleis, and A. Eichenberger , "An experimental study of algorithms for weighted completion time scheduling," *Algorithmica*, Vol. 33, 2002.

[3] I. D. Baev, W. M. Meleis, and A. Eichenberger, "Algorithms for total weighted completion time scheduling," *SODA: ACM-SIAM Symposium on Discrete Algorithms (A Conference on Theoretical and Experimental Analysis of Discrete Algorithms)*, 1999.

[4] G. Buttazzo, G. Lipari, L. Abeni, and M. Caccomo "*Soft Real-Time Systems, Predictability vs. Efficiency, "* Springer company, 2005, NY, USA.

[5] C. Chekuri and R. Motwani, "Precedence constrained scheduling to minimize weighted completion time on a single machine," *Discrete Applied Mathematics*, Vol. 98, pp. 29-39, 1999.

[6] C. Chekuri, R. Motwani, B. Natarajan, and C. Stein, "Approximation techniques for average completion time scheduling," *SIAM Journal on Computing*, Vol. 31, No. 1, pp. 146-166, 2001.

[7] W. Fornaciari, P. di Milano, *Real-time operating systems scheduling lecturer*, www.elet elet.polimi polimi.it/ fornacia it/ fornacia.

[8] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of NP-completeness," W. H. Freeman, January 15, 1979.

[9] M. X. Goemans, M. Queyranne, A. S. Schulz, M. Skutella, and Y. Wang, "Single machine scheduling with release dates," 2001.

[10] L. A. Hall, A. S. Schulz, D. B. Shmoys, and J. Wein, "Scheduling to minimize average completion time: off-line and on-line approximation algorithms," *Mathematics of Operations Research*, Vol. 22, pp. 513-544, August 1997.

[11] M. Joseph, *Real-time systems: specification, verification and analysis*, Prentice Hall, 1996.

[12] P. A. Laplante, *Real-time systems design and analysis, an engineer handbook*, IEEE Computer Society, IEEE Press, 1993.

[13] A. Mohammadi and S. Akl, "Heuristic Scheduling Algorithms Designed Based on Properties of Optimal Algorithm for Soft Real-Time Tasks," submitted to 2007 Summer Computer Simulation Conference (SCSC'07).

[14] A. Mohammadi and S. Akl, *Scheduling algorithms for real-time systems*, Technical Report 2005-499, School of Computing, Queen's University, 2005.

[15] M. W. P. Savelsbergh, R. N. Uma, and J. Wein, "An experimental study of LP-based approximation algorithms for scheduling problems," *INFORMS Journal of Computing*, Vol. 17, No. 1, Winter 2005, pp. 123-136.

[16] M. Skutella, "List scheduling in order of alpha-points on a single machine," in *Efficient Approximation and on-line algorithms*, editted by E. Bampis, K. Jansen and C. Kenyon, 2002.

[17] J. A. Stankovic and K. Ramamritham, *Tutorial on hard real-time systems*, IEEE Computer Society Press, 1988.

DARTSVIEW, A Toolkit for DARTS in LabVIEW

NGO Khanh Hieu, GROLLEAU Emmanuel
Laboratory of Applied Computer Science
LISI-ENSMA, 1 avenue Clément Ader, Téléport 2
BP. 40109 – 86961 Futuroscope Cedex – France
ngo@ensma.fr, grolleau@ensma.fr

Abstract

DARTS (Design Approach for Real Time Systems) [Gom93] is a software design method for real time systems. LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical application development environment developed by National Instruments Corporation based on the dataflow representation of the "G" language [Lab98][Cot 01]. LabVIEW is implicitly multithreaded and has high level functions for communication/synchronization, allowing it to be used as a programming language for control/command and soft real-time applications. In order to help a designer to develop a real-time application, we propose the library DARTSVIEW, which simplifies the passage from the conception of a "multitasking" application to the implementation [NG03]. One can use DARTSVIEW in different phases of the life cycle of a real-time system software. The last version of DARTSVIEW, allows to define in XML several real-time programming normalized languages, and to generate a part of the code for different specific programming languages (Ada, POSIX 1003.1, VxWorks, OSEK/VDX, etc.). The flexibility introduced by the use of XML allows a designer also to generate some code targeting real-time scheduling analysis tools in order to achieve the temporal validation. The objective of this article is to present an overview of DARTSVIEW, a Toolkit for DARTS in LabVIEW, the role of DARTSVIEW in the software life-cycle, and some perspectives for the extensibility of this Toolkit in the future.

1. Introduction

The "concurrency" is one of the problems that we have to face frequently in real time systems. A concurrent system has many activities (or tasks) occurring in parallel. Usually, the order of incoming event is not predictable and these events may overlap [Gom93]. So several tasks may handle the data-acquisition at different rates, some other tasks may be dedicated to the calculation of commands, and some others to the commands of several devices. When these activities (or tasks) synchronize and communicate, the conformance with rules of the mutual exclusion, of the synchronization, and of the communication is actually a key issue to be addressed:

- mutual exclusion is the mechanism for ensuring that only one process at a time performs a specified action . Hence, it guarantees shared access to data (or resources) to the tasks,
- synchronization is the control of the execution of two or more interacting processes so that they perform the same operation simultaneously. It allows to block a task until another one awakes it,
- communication is a mechanism permitting the tasks to exchange the data.

DARTS (Design Approach for Real Time Systems) is a software design method, which emphasizes the decomposition of a real-time system into concurrent tasks and defines the interfaces between these tasks. In a DARTS diagram, each task is presented by a parallelogram (Fig. 1). It can be either a hardware task (released by an external event, such as an interrupt or a real-time clock), or a software task (released by another task) [NG03]. DARTS can be used as a conception method for multitask systems (including real-time and control/command systems), since it focuses on the task decomposition, and thus is really close to the implementation process [CG05].

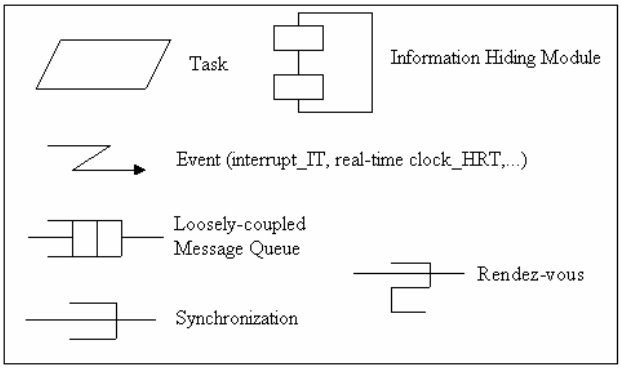


Figure 1: Elements of a DARTS diagram

LabVIEW (Laboratory Virtual Instrument Engineering Workbench) is a graphical application development environment in the G language. LabVIEW is very well suited for data-acquisition, signal processing, and (soft real-time) control/command of process. The LabVIEW programming language is naturally parallel: when parts of the data flow are independent, the runtime can map them in several system threads. However, the difference between the notion of parallelism in LabVIEW and the semantics of dataflow associated in the G language does not allow to make a direct communication between them

by a dataflow (in this case, the second function has to wait for the completion of the first one in order to start its execution). Therefore, following the work of [Gev98], LabVIEW integrates intertask communication tools.

During several case studies, we realized that it was quite interesting to help the designer to create a multitask application in LabVIEW with DARTS based bricks provided as a Toolkit named DARTSVIEW, in order to get past of the classic multitask implementation process, and to focus on the behavior of the tasks.

One of the important roles of DARTSVIEW is to help the designer to represent a DARTS diagram directly in LabVIEW. And in the software life-cycle like the classic V model given on Figure 2, the functional aspect of the system may be tested.

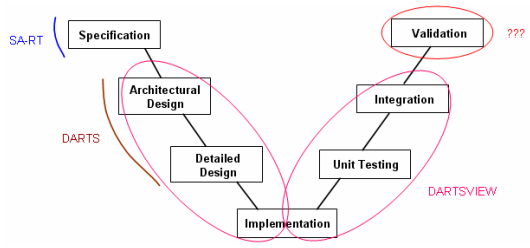


Figure 2: Software life-cycle in V

The temporal validation usually consists in a schedulability analysis based on a temporal model of the tasks [CDKM02]. The target programming language will likely be an imperative language (like C-based, Ada), and several tools can be used, Response-Time Analysis based, like MAST [MAST01], or building a feasible schedule, like PeNSMARTS [Gro99]. The choice that has been made for DARTSVIEW was to use the flexibility of XML for the temporal validation in the same way as it has been used to generate program in several programming standards: an XML model can be used to output a task model in the required format of several validation tools.

Therefore, in this case the software life-cycle based on a classic V model would be extended with the second V porting the workstation code whose parallel behavior has been tested on the workstation, and on the embedded target. The Figure 3 represents a software life-cycle in W, and the role of DARTSVIEW in this life-cycle.

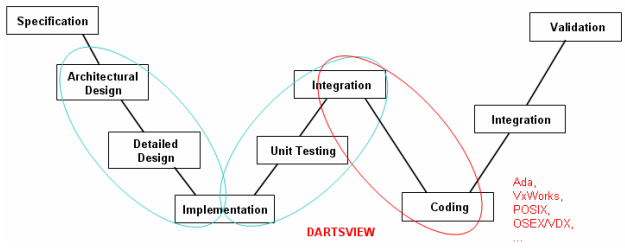


Figure 3: Software life-cycle in W

In the sequel, the following aspects of DARTSVIEW approach will be presented: section 2 presents the main multitasking LabVIEW concepts, and how these concepts are used in DARTSVIEW v7.1. The DARTSVIEW Toolkit and a case study are presented in section 3. Section 4 presents some perspectives.

2. DARTS and multitasking LabVIEW concepts

The implicit notion of parallelism inherent in LabVIEW allows multitask programming transparently: in fact, two loops running in parallel are mapped on different threads, and hence are executed in parallel. This characteristic permits to implement directly an abstraction of the tasks based on a DARTS representation. So, in LabVIEW a task DARTS can be simply modeled by an infinite While loop. However, the difference between the notion of parallelism in LabVIEW and the notion of data flows in G language does not allow exchanging the data directly between the tasks. Then this section presents how LabVIEW implements the interfaces between these tasks.

2.1. Task synchronization

Synchronizing two tasks consists in introducing a precedence constraint at a certain place of code: the destination task has to wait for an event sent by the source task in order to execute an action. In DARTS, synchronization between two tasks is presented on Fig. 4. For this type of synchronization, programming languages usually use a counting semaphore [CG05]. However the Semaphore tools proposed after LabVIEW v7.1 are bounded semaphore (LabVIEW requires that a semaphore can not have a count greater than its initial count). In order to solve this problem, we had to modify the implementation of the task synchronization in the way that firstly we decrease the count of a semaphore to 0, and then the release of semaphore must be verified to insure that the count is always smaller than or equal to the initial count.

2.2. Loosely-coupled communication

The communication is the transfer of data from one task to another. It is either based on a send and forget paradigm (see Fig. 4) when the size of the message queue is unbounded, or when a recent message replace the oldest one, or on a producer/consumer paradigm when the size of the queue is bounded and no message can be lost (default behaviour of the message queue tool in LabVIEW).

We notice that after LabVIEW 7.1, the data is casted to a variant data type, and allows sending a message of any data type to the message queue. Retrieving the message consists in casting back the variant to the original data type. LabVIEW is checking the coherence when casting from a variant type to another data type, so the user can not make any typing mistake without being warned at runtime.

In the send and forget paradigm (see Fig.4), the writing consists in emptying first, and then writing into the message queue in order to replace the oldest message in the case it would not have been read by the time the new message is sent. So the producer could send a message and then continue its execution without care of the reception of it in the consumer. This communication

is very useful in the case of dense task producer; it allows to control the reception-rate in the task consumer.

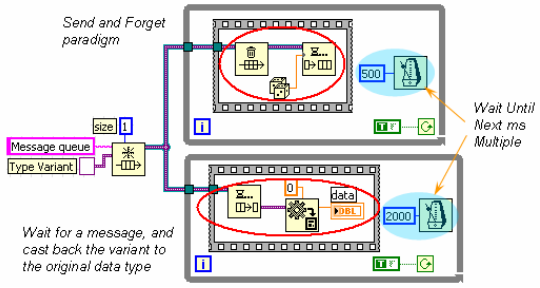


Figure 4: Message queue with replacement

2.3. Tightly-coupled message communication

The synchronous message communication, represented by the tightly-coupled message communication (Ada 83 rendez-vous), is a mechanism with which the producer sends a message to the consumer, and then immediately waits for a response (a message, or an event). In LabVIEW, this kind of communication can be implemented by two message queues (hence one for the producer, another for the consumer). A model of producer is presented on Figure 5: the producer firstly sends a message to the queue of the consumer, and then waits for the response sent by the consumer in order to continue its execution. The model of consumer is symmetric.

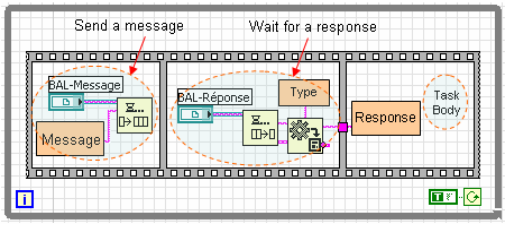


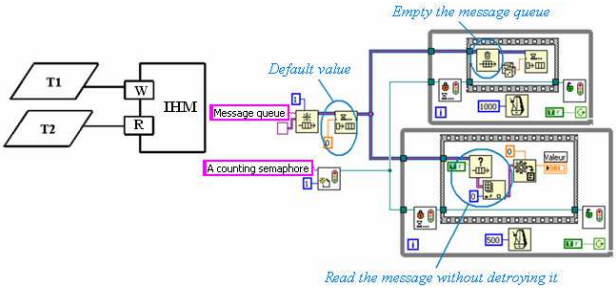
Figure 5: Producer, Tightly-coupled message communication with response

In the case of the tightly-couple message communication without response, the message queue for the consumer might be replaced by a tool of synchronization (i.e., a semaphore in LabVIEW) for the signal of the consumer to the producer when it receives successfully the message sent by the producer.

2.4. Information hiding module (IHM)

Information hiding is used as a criterion for encapsulating data stores. In DARTS, IHMs are used for hiding the contents and representation of data stores and state transition tables. When an IHM is accessed by more than one task, the access procedures must synchronize the access to the data [Gom93]. The Figure 6 represents a simple implementation of IHM in LabVIEW. An IHM for encapsulating data stores of type "Reader/Writer" is compounded of two atomic operations, Read and Write, acting on an internal data. Note that we use a message queue of size 1 to store this data, and a counting semaphore of size 1 too in order to insure atomicity.

Hence, writing consists in emptying the message queue firstly, and then inserting the new value, while reading consists in getting the data value without destroying it. For the other IHMs (i.e., IHM de type Multiple-Readers/Writer, State Transition Modules, Device Interface Modules, etc.), thanks to the VIs in the palettes "Queue" and "Semaphore" of LabVIEW, we

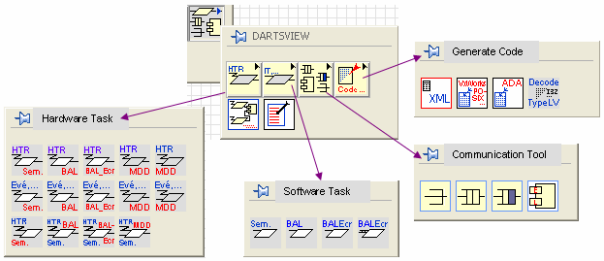


could implement them easily and intuitively.

Figure 6: Communication by IHM

3. DARTSVIEW Toolkit

The DARTSVIEW Toolkit is presented on the Figure 7. It is a LabVIEW library abstracting the DARTS concepts into LabVIEW programming elements. The library has four mains palettes, named "Hardware Task" (task is activated by either a real-time clock or an external event), "Software Task" (task is activated by another by means of the synchronization/communication tool),



"Communication Tool", and "Generate Code".

Figure 7: DARTSVIEW Toolkit

The designer of a real-time application can program his system directly from the DARTS conception, and will obtain a program that can be tested, and used in order to generate some code targeting different real-time programming standards or temporal validation tools. To illustrate the role of DARTSVIEW in a W life-cycle, a simple example is presented on Figure 8. This is a building's heating system; its brief behavior is the following: the ignition system is run if the air (controlled by a fan) and the gas (controlled by a valve) are supplied. If during the operation of the system one of these two sources is closed, or the combustion is turned off, an alarm will be raised.

Thus, depending on the states of the sensors (the toggle switch state, the thermostat, the thermocouples, the flow meters, etc.), a central control task decides to send the commands to the tasks commanding the actuators, while another task is in charge of calculating the

difference between the actual temperature and the required one.

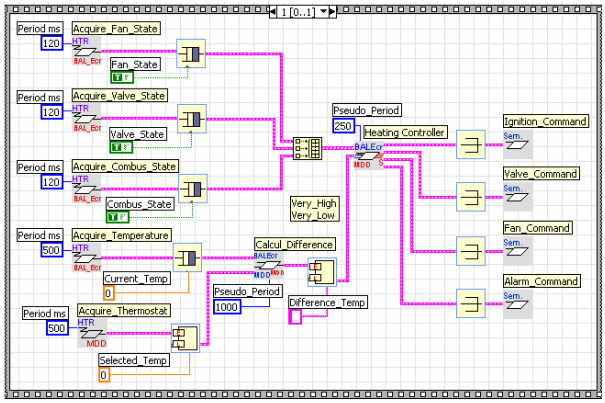


Figure 8: DARTSVIEW diagram of a heating system

The DARTSVIEW diagram on Fig. 8 is really similar to the DARTS diagram. Thanks to the simple and intuitive implementation, the designer could create in LabVIEW a software simulator in order to test the global behavior (the functional aspect) of the tasks system in the first V of the W life-cycle by means of the DARTSVIEW diagram. Moreover, all of the information about the tasks system will be recorded, and will be generated to the designer in form of a XML document (see Fig. 9) for the use in the second V of the W life-cycle: code generation.

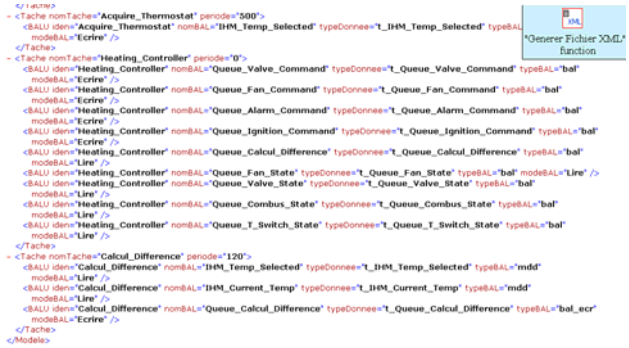


Figure 9: XML representation

LabVIEW allows a rapid development of a control/ command or soft real-time application, but it is less used for embedded hard real-time systems. Several standards and proprietary extensions are used, depending on the application area (aerospace, aeronautics, car, manufacturing, UAV, electronic devices…): Ada, ARINC 653, OSEK/VDX, POSIX 1003.1, VxWorks, TRON, etc. So it is convenient to find a flexible way to be able to generate the specific multitask code parts targeting these languages/standards from DARTSVIEW. The same problem arises when we want to generate the code in order to validate the application by a third-party tool. A flexible choice seems to be the use of XML in a schema of the code generation from DARTSVIEW like the one shown on Figure 10. A new standard or third-party tool is then targeted by LabVIEW using an XML file to describe the code generation for the tests, the calculation of the temporal parameters of each task, and the feed-back of these results to the DARTSVIEW model. Consequently, thanks to DARTSVIEW the time-to-market of the development of system will be better than the one using the traditional approach.

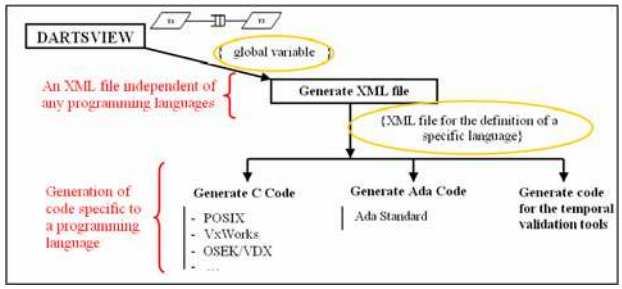


Figure 10: Schema of code generation from DARTSVIEW

4. Conclusion

DARTSVIEW Toolkit is a helpful tool for the DARTS development of control-command applications in LabVIEW, as well as a helpful tool for the development of embedded applications using a target language based on a specific standard or proprietary library. The use of XML-DTD facilitates the generation of code from the DARTSVIEW model, and allows the designer to choose a third-party tool for the validation of the timing requirements.

DARTSVIEW Toolkit is already used as a first multitasking environment for students in two French schools.

Reference

[CDKM02] F. Cottet, J. Delacroix, C. Kaiser, Z. Mammeri, «Scheduling in Real-Time Systems», J. W. & Son, 2002.

[CG05] F. Cottet, E. Grolleau, «Systèmes temps réel de contrôle-commande, Conception et Implémentation», Dunod, 2005.

[Gev98] Emmanuel Gerveaux, «Conception d'un environnement de développement des applications de contrôle de procédé basé sur le modèle formel GRAFCET et fondé sur un langage graphique flot de données», rapport de Thèse, LISI-ENSMA, 29 Septembre 1998.

[Gom93] Hassan Gomaa, «Software Design Methods for Concurrent and Real-Time Systems», Addison Wesley, SEI Series in Software Engineering, 1993.

[Gro99] Emmanuel Grolleau, «Ordonnancement temps réel hors-ligne optimal à l'aide de réseaux de Petri en environnement monoprocesseur et multiprocesseur», rapport de Thèse, LISI-ENSMA, 29 Novembre 1999.

[Lab98] National Instruments, LabVIEW™ 5 «Software Reference and User Manual», February 1998.

[MAST01] M. G. Harbour, J.J. G. Garcia, J.C. P. Gutierrez, J.M. D. Moyano, «MAST: Modeling and Analysis Suite for Real-Time Applications», Proc. Of the 13th IEEE Euromicro Conference in Real-Time Systems, 2001.

[NG03] K.H. Ngo, E. Grolleau, «La Méthode DARTS et La Programmation Multitâche en LabVIEW», FuturVIEW'2003, ENSMA, 12-13 Juin 2003.

Timing Properties of Removable Flash Media

Daniel Parthey and Robert Baumgartl
Real-Time Research Group
Department of Computer Science
Chemnitz University of Technology
{daniel.parthey, robert.baumgartl}@cs.tu-chemnitz.de

Abstract

Flash memories could be a basis for mass storage with real-time guarantees if a suitable model for access timing could be established. To characterize the operation timing of removable flash media such as Compact Flash (CF) and MultiMedia Card (MMC), we propose a set of benchmarks. The results of our study indicate that a simple timing model for current media cannot be established. The timing of individual read and write operations depends on the address and block size of accessed data as well as the written bit pattern. Many timing anomalies were observed.

Keywords

Flash Storage Media, Flash Memory, Worst-Case Access Time, Embedded Systems

1. Introduction

In the past, quite some research efforts have been made to analyze hard disk access timing and to establish precise timing models [9]. Unfortunately, today's hard disks are very complex systems. Most of their inner workings are hidden from the user and must be deduced by complex experiments [12]. The continuous capacity and (to a lower degree) bandwidth improvements require sophisticated techniques. Hence, as Ruemmler concludes, access operations cannot be modeled with any accuracy [9]. As a component for hard real-time systems, hard disks usually are not used with explicit timing guarantees.

The advent of flash memories has changed this situation. Because they provide a very simplistic interface and operation timing is usually well-documented (cf. [10] for example), it seems that they could be the basis for mass storage with very precise and easy-to-determine access timing. In comparison to hard disks, a new quality of timing predictability for soft and hard real-time systems seems possible. Additionally, flash media are very robust and therefore ideal under rough conditions.

Two different strategies to manage flash memory can be distinguished. The first one is the design of specialized file systems such as JFFS [11], YAFFS [1] and LogFS [4]. The second strategy is to emulate a block device on top of flash memory. The Flash Translation Layer (FTL) [5] is an example as well as all removable flash media such as CompactFlash [3] or MultiMedia Card [6]. With the exception of the xD Picture Card (which we did not test) all removable flash media consist of one or several flash memory circuits and a microcontroller which emulates the block device by providing the communication interface to the outside, translating protocol commands into flash memory accesses and keeping track of wear-levelling.

The file system approach has attracted considerable research attention. For example, efficient techniques for initialization and crash recovery [13] as well as efficiently managing metadata information [2] have been published. Removable flash media, on the other hand, did not attract the same amount of research. Media comparisons in technical journals or the web usually focus on achievable throughput only [8]. No thorough analysis concerning real-time access timing of removable flash media is known to the authors. The goal of this study is to obtain a first impression on the subject.

2. Experimental Setup

Because removable flash media cannot be opened without destruction we relied on black-box testing only. We designed a suite of seven simple tests to reveal the following properties of the media:

- dependence of timing on accessed address,
- dependence of timing on accessed block size,
- dependence of timing on written and overwritten bit pattern,
- identify and quantify potential caching or buffering (read, read-ahead, write)
- try to deduce the erase block size by measuring the time to write blocks of different sizes.

As an example, to assess a potential timing influence of the accessed address, we measured the time to read and write four consecutive blocks of 64 KiB at ten different positions across the medium. Describing all tests in detail is beyond the scope of this paper. We refer the interested reader to [7].

All tests were performed on an AMD64 Athlon 3000+ with 1 GiB RAM running Linux 2.6.18. The media were accessed via a USB 2.0 Hama card reader with a GL819 chip. No other devices were connected to the USB. Due to the comparatively long operation times we deemed standard Linux suitable as measurement platform.

To estimate the influence of the card reader onto the measured performance some of the measurements were reiterated with a different card reader under otherwise identical conditions. A difference could be noticed but is negligible.

All read and write operations were performed on raw devices without any file system. A total of 19 different media were analyzed, among them Compact Flash (CF), Secure Digital (SD) Card, Memory Stick (MS) and Multimedia Card (MMC).

3. Results

3.1. Data Throughput

We begin our analysis with a conventional performance test. The achievable throughput for read and write accesses is measured depending on block size.

Figure 1 shows an exemplary result. For small block sizes below the physical erase block size throughput is limited by overhead. As soon as the transferred block size reaches the erase block size, maximum throughput can be observed. As in the depicted example, these limits usually differ for read and write operations.

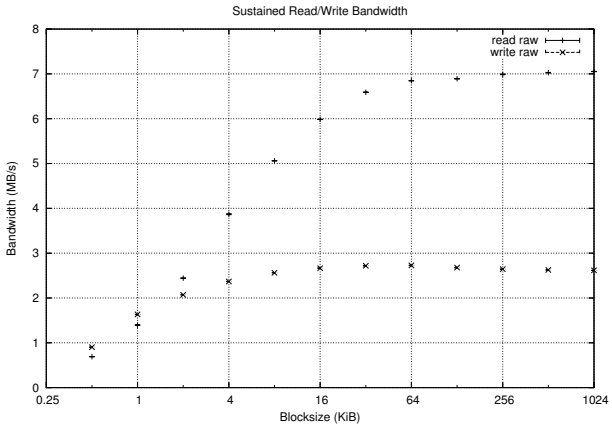


Figure 1. Data Throughput for different Block Sizes (MS 32 MiB SanDisk)

Usually, write operations are slower than reads, but we also observed identical maximum read and write bandwidths for modern CF media. By looking at throughput

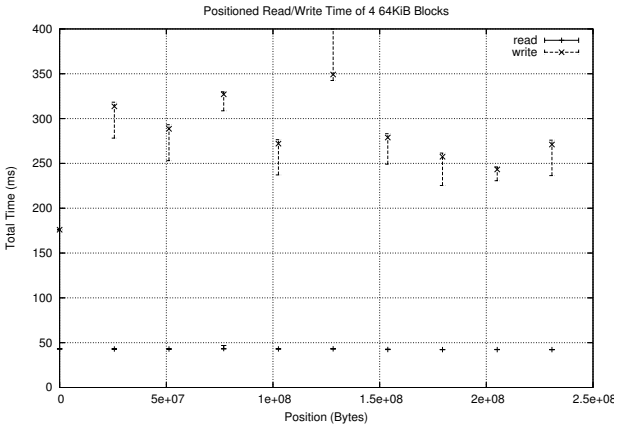


Figure 2. Time to access a 64 KiB Block at different Locations (CF 256 MiB Toshiba)

only, removable flash media seem to exhibit very deterministic access timing.

3.2. Sensitivity of Position

Some but not all media have a special short access time for write operations at address 0. Figure 2 shows an example.

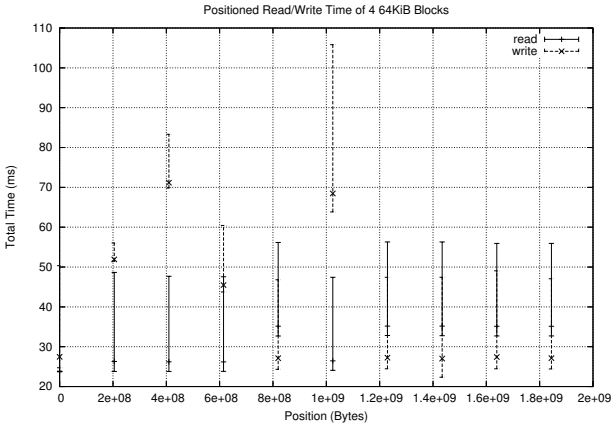


Figure 3. Time to access a 64 KiB Block at different Locations (CF 2 GiB SanDisk)

Writing four 64 KiB blocks from address zero needs between 24 and 48 percent less time than writing the same amount of data to other locations. Additionally, the timing of writing an individual block varies considerably except for address 0. The read operation is not affected.

A possible partial explanation could be that the VFAT file system stores its main metadata structure, the File Allocation Table (FAT), at that address. Frequent write accesses to that location are therefore very likely and are consequently optimized by the controller.

The same phenomenon was observed for a more recent medium as depicted in figure 3. Again, writing to address

zero is considerably faster than to any other address. Further, writing to address 1 GiB is especially slow. Note also the high read and write timing variance.

Another anomaly is depicted in figure 4. Here, all read operations above 256KiB need more than triple the time than reads below that boundary (150 vs 44 milliseconds). Write timing is uniform across the medium. It seems that

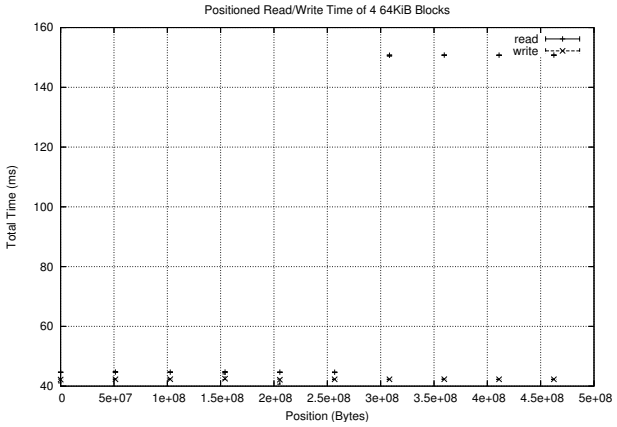


Figure 4. Time to access a 64 KiB Block at different Locations (MMC 512 MiB Extrememory)

two different flash memory circuits were used for that specific medium.

3.3. Sensitivity of Block Size

As figure 5 indicates, some media have an extremely poor write performance when accessing very small blocks. In our benchmark, exactly one megabyte of data was written using blocks of a constant size. The block size was varied and the time necessary for completing all write operations was measured. As can be seen, write operations with block sizes $\leq$ 2 KiB need almost a magnitude of order more time to complete. This phenomenon could only be observed for exactly one medium. Many other media exhibit poor access performance for very small blocks but the performance follows more or less the "classic" smooth curve and can be attributed to normal overhead increase.

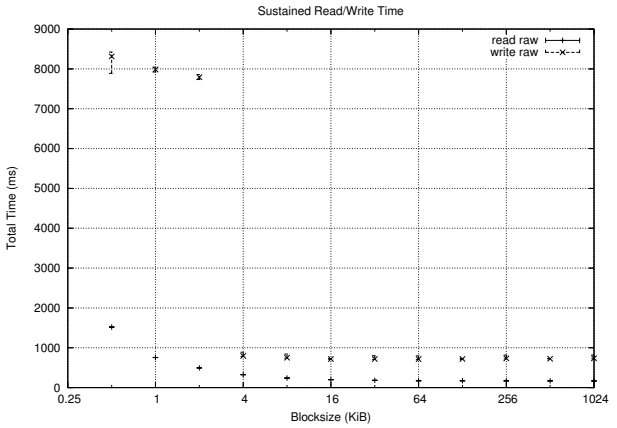


Figure 5. Time to access 1 MiB with Blocks of constant Size (CF 256 MiB Toshiba)

Figure 6 depicts a similar phenomenon. This time, reading small blocks is anomalously slow. Whereas for block sizes between 1 and 1024 KiB read and write operations behave inconspicuously, reading with a block size of 512 Bytes slows down pathologically. Reading 1 MiB of data requires over 10 seconds on average; in the worst case more than half a minute is needed! We have no explanation for that phenomenon so far. Neither a different MMC medium of different size nor media of different type did exhibit a similar anomaly.

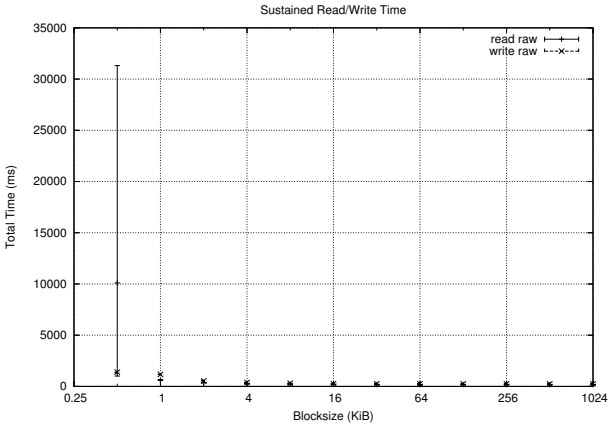


Figure 6. Time to access 1 MiB with Blocks of constant Size (MMC 512 MiB Extrememory)

A similar poor access performance for large block sizes has not been observed.

3.4. Sensitivity of Written Value

A third influence factor on access speed is the actual value to be written. For one medium, writing the value 0xff needs approximately 284 milliseconds, whereas writing any other bit pattern needs only 252 milliseconds.

More common is the reverse case: writing 0xff needs *less* time than writing other values. A tentative explanation for that phenomenon could be that setting all bits corresponds to an erase operation whereas writing a pattern containing at least one zero bit requires an additional operation after the erase. Usually the difference is below 10 percent of the average write time, but in one case the 0xff-write needed 158 milliseconds whereas the generic write needed up to 260 milliseconds.

Sometimes, also the overwritten value influences access timing, but the impact is negligible.

Some media exhibit almost no sensitivity on write bit patterns. The MMC from figure 6 performs all four 64 KiB writes in 44.5 ± 0.3 milliseconds.

3.5. Caching

We could not detect the existence of any read or read-ahead cache in the media we analyzed. Because the predominant access order for mass storage is strictly sequential, a read cache would hardly improve performance. On the other hand, a read-ahead cache seems only reasonable when there is some kind of penalty for late read accesses (such as the rotational latency in hard disks).

In a single case (an SD card of 1 GiB size), our test indicated the existence of a write cache. The idea of the test is to write several consecutive blocks of random data to pollute a potentially-existing cache, read a single block, write random data to that block and read it again. In the presence of a write cache, the second read operation will perform much faster than the first.

In the described case, we measured an average of 18 milliseconds for the first and almost no time for the second read operation when accessing blocks of 512 bytes size. For all other tested block sizes (1 KiB, 2 KiB, ..., 1 MiB), both read operations needed identical times. Because the test only succeeds for one block size, we are not sure whether we really identified a cache or simply another timing anomaly. Certainly, that aspect needs further research.

4. Conclusions & Outlook

Our tests indicate that there is no such thing as a uniform access time for removable flash media. Whereas most read and write accesses behave as expected, surprisingly many of them do not. We identified three main sources of access timing indeterminism: (I) The address of the accessed data block: some media access address zero especially fast, other locations are sometimes very slow. (II) Accessing very small blocks can be prohibitively slow. (III) The written bit pattern also influences access timing, but to a lesser degree. The value of irregular behavior is usually 0xff. Our tests showed a high degree of reproducibility of all timing parameters. Media of identical type seem to behave identically.

It is impossible to explain most of the observed timing anomalies without knowing the internals of the flash media. Information on the number and exact type of flash memory circuit as well as on the microcontroller type and program are necessary. Existing documentation by vendors is usually poor.

Hence, it is impossible to describe access timing of removable flash media with a few parameters such as minimum and maximum read and write time or bandwidth. Instead, media must be precisely analyzed to identify possible timing variances and anomalies. A basis for such an analysis could be our set of benchmarks.

We feel that our set of analyzed media is yet too small to be representative. Therefore, we will make the measurement scripts available to the public at our website. We will encourage anybody interested to apply the measurements to whatever removable flash media and hope to collect a reasonable amount of interesting timing data to assess our results.

An interesting question is whether our set of benchmarks is adequate for fully characterizing access timing of a removable flash medium. Many of the measurement parameters such as the number and size of the blocks to access were chosen arbitrarily. Therefore, we do not have a guarantee that we actually really captured worst case timing. As an example, analyzing the influence of the accessed position on timing needs refinement. Additionally, we did not investigate whether access timing changes during media lifetime. These and other questions are subject of further research.

As a next step of our project we plan to analyze raw flash circuits in a similar manner. The Memory Technology Device (MTD) abstraction of Linux seems a suitable basis for that. We hope that direct access to flash memory will improve timing predictability. Having established a firm timing model for elementary flash operations, we want to simulate and analyze the real-time properties of typical flash file systems such as JFFS2 and LogFS.

References

[1] Aleph One Limited. YAFFS Overview. available from http://www.aleph1.co.uk/yaffsoverview, 2006.

[2] L.-P. Chang and T.-W. Kuo. An Efficient Management Scheme for Large-Scale Flash-Memory Storage Systems. In *Proceedings of the 2004 ACM Symposium on Applied Computing (SAC'04)*, pages 862–868, Nicosia, Cyprus, Mar. 2004.

[3] CompactFlash Association. *CF+ and CompactFlash Specification Revision 4.1*, Feb. 2007. available from http://compactflash.org.

[4] J. Engel and R. Mertens. LogFS – Finally a scalable flash file system. http://wh.fh-wedel.de/~joern/logfs.pdf, 2005.

[5] Intel Corporation. *Understanding the Flash Translation Layer (FTL) Specification*, Dec. 1998. Application Note AP-684.

[6] MMCA Technical Committee. *The MultiMedia Card – System Summary*, Apr. 2005. http://www.mmca.org.

[7] D. Parthey. Analyzing Real-Time Behaviour of Flash Memories. Master's thesis, Chemnitz University of Technology, Apr. 2007. (to appear).

[8] H.-J. Reggel. Cardspeed – Card Readers and Memory Cards. http://www.hjreggel.net/cardspeed/, 2007.

[9] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–29, 1994.

[10] ST Microelectronics. M29W640DT Datasheet. available from http://www.datasheetcatalog.com, Dec. 2004.

[11] D. Woodhouse. JFFS2: The Journalling Flash File System, version 2. available from http://sourceware.org/jffs2/, 2003.

[12] B. L. Worthington, G. R. Ganger, Y. N. Patt, and J. Wilkes. On-Line Extraction of SCSI Disk Drive Parameters. In *Proceedings of the ACM Sigmetrics Conference (SIGMETRICS 95)*, pages 146–156, May 1995.

[13] C.-H. Wu, T.-W. Kuo, and L.-P. Chang. The Design of Efficient Initialization and Crash Recovery for Log-based File Systems Over Flash Memory. *ACM Transactions on Storage*, 2(4):449–467, Nov. 2006.

Comparison of two worst-case response time analysis methods for real-time transactions

A. Rahni, K. Traore, E. Grolleau and M. Richard
LISI/ENSMA
Téléport 2, 1 Av. Clément Ader
BP 40109, 86961 Futuroscope Chasseneuil Cedex
{rahnia,traore,grolleau,richardm}@ensma.fr

Abstract

This paper presents a comparison of two worst case response time analysis methods in the context of transactions. In the general context of tasks with offsets (general transactions), only exponential methods are known to calculate the exact worst-case response time of a task. We focus more precisely on monotonic transactions. In this context, we present the fast and tight analysis, proposed in [7, 6], and the analysis technique of monotonic transaction that we have proposed in [14]. We compare them on a case study and on several configurations generated randomly.

Keywords: Response Time, Transactions

1. Introduction

The Response-Time Analysis [1] is an essential analysis technique that can be used to perform schedulability tests. Tindell proposed in [11] a new model of tasks with offset (transactions) extending the model of Liu and Layland [5]. Since the transactions are non-concrete(the transaction release times are not fixed a priori), the main problems is to determine the worst case configuration for a task under analysis (its critical instant). Tindell showed that the critical instant for a task under analysis (τ_{ua}) occurs when one task of higher priority in each transaction is released at the same time as τ_{ua}.

An exact calculation method has been proposed in [10], but has an exponential complexity and is intractable for realistic task systems; Tindell [11] has proposed a pseudo-polynomial approximation method providing an upper bound of the worst-case response-time. Later, this approach has been improved in [4, 6, 7, 8]

In the sequel, we present the model of tasks with offsets (a.k.a. transaction), then we present the best known approximation method proposed by Turja and Nolin [8]. Section 4 presents the monotonic transactions exact analysis [13] and these two methods are used on the same tasks system. In the last section their performance are compared on randomly generated transaction systems.

2. Model of transactions

A tasks system Γ is composed of a set of $|\Gamma|$ transactions Γ_i, with $1 \leq i \leq |\Gamma_i|$ (where $|\Gamma_i|$ is the number of elements in the set Γ_i).

$$\begin{aligned} \Gamma &: \left\{\Gamma_1, \Gamma_2, .., \Gamma_{|\Gamma|}\right\} \\ \Gamma_i &: \left\{\tau_{i1}, \tau_{i2}, ..., \tau_{i|\Gamma_i|}, T_i\right\} \\ \tau_{ij} &: < C_{ij}, O_{ij}, D_{ij}, J_{ij}, B_{ij}, P_{ij} > \end{aligned}$$

Each transaction Γ_i (see figure 1) consists of a set of $|\Gamma_i|$ tasks τ_{ij} released at the same period T_i , with $0 < j \leq |\Gamma|$. Without loss of generality, we suppose that the tasks are ordered in the set by increasing offset. A task τ_{ij} is defined by : a worst-case execution time (WCET) C_{ij}, an offset O_{ij} related to the release date of the transaction Γ_i, a relative deadline D_{ij}, a maximum jitter J_{ij} (the activation time of task τ_{ij} may occur at any time between $t_0 + O_{ij}$ and $t_0 + O_{ij} + J_{ij}$, where t_0 is the release date of the transaction Γ_i, a maximum blocking factor B_{ij} due to lower priority tasks (e.g. priority ceiling protocol [9]), and P_{ij} is its priority (we assume a fixed-priority scheduling policy). The figure 1 presents an example of transaction Γ_i composed of three tasks with period $T_i = 16$.
Let us note $hp_i(\tau_{ua})$ the set of indices of the tasks of Γ_i with a priority higher than the priority of a task under analysis τ_{ua}, assuming that the priorities of the tasks are unique.

3 Fast and Tight Analysis

This method provides an efficient implementation to calculate an upper-bound of the worst-case response times [7]. The main idea is to represent the periodic interference function statically, and during

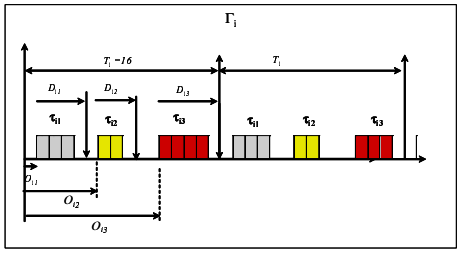


Figure 1. Example of transaction.

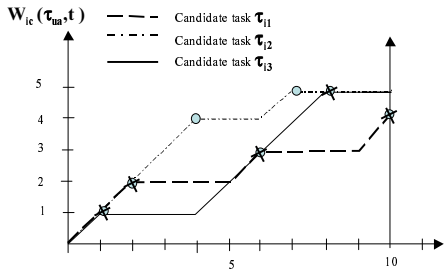


Figure 2. Interference of transaction.

the response-time calculation, to use a simple lookup function in order to compute its value. The interference imposed by the transaction Γ_i on a task under analysis τ_{ua} during a busy period of length t starting at the release of τ_{ua} and corresponding to the release of τ_{ic} is noted $W_{ic}(\tau_{ua}, t)$ (τ_{ic} is then called the critical instant candidate in Γ_i). In order to simplify, we suppose no Jitter in the transaction (i.e $J_{ij} = 0$).

$$
\begin{aligned}
W_{ic}(\tau_{ua}, t) &= \sum_{j \in hp_i(\tau_{ua})} \left(\left(\left\lfloor \frac{t^*}{T_i} \right\rfloor + 1 \right) * C_{ij} - x_{ijc}(t) \right) \\
t^* &= t - \Phi_{ijc} \\
\Phi_{ijc} &= (T_i + (O_{ij} - O_{ic})) \ \% \ T_i \\
x_{ijc}(t) &= \begin{cases} 0 \text{ for } t^* < 0 \\ \max(0, C_{ij} - (t^* \ \% \ T_i)) \text{ otherwise} \end{cases}
\end{aligned}
$$

Φ_{ijc} is the phase between τ_{ic} and τ_{ij}. $x_{ijc}(t)$ corresponds to the part of the task τ_{ij} that cannot be executed in the time interval of length t (note that this part is the main difference between the methods presented in [4] and [7]).
In order to find the critical instant, one would have to compare every combination of critical instant candidates, making the exact test intractable. The fast and tight analysis method consists of creating a global interference function $W_i(\tau_{ua}, t)$ for each transaction Γ_i, in choosing the maximum value of each interference function.

$$W_i(\tau_{ua}, t) = \max_{\forall c \in hp_i(\tau_{ua})} W_{ic}(\tau_{ua}, t)$$

The figure 2 shows the graphical representation of the interference of a transaction : each curve represents the interference function for each critical instant candidate. Since the transaction is in normal form, the derivative of each curve is either 0 or 1. $W_i(\tau_{ua}, t)$ that has to be computed is the maximum of all the curves. The efficient implementation proposed in [7] stores the set of points P_{ic}, where each point $P_{ic}[k]$ has an x (representing time) and a y (representing interference) coordinate. These points correspond to the convex corners of the curve $W_{ic}(\tau_{ua}, t)$.
The calculation of the upper bound on the worst-case response time R_{ua} of τ_{ua} is obtained by an iterative fix-point lookup.

$$
\begin{aligned}
R^0_{ua} &= C_{ua} \\
R^{(n+1)}_{ua} &= C_{ua} + \sum_{\Gamma_i \in \Gamma} (W_i(\tau_{ua}, R^n_{ua}))
\end{aligned} \tag{1}
$$

where $W_i(\tau_{ua}, t)$ is deduced from the static representation of the transactions interferences.

3.1 Normal form of a transaction

Without loss of generality, we consider that all the tasks of Γ_i have a higher priority than the task under analysis τ_{ua}. Since some tasks of a transaction may have to overlap, issuing in an interference function which derivative would be greater than 1, a normal form of the transaction is first obtained using three operations: order, merge, and split [7]. For each critical instant candidate τ_{ic}, the transaction Γ_i is put in normal-form. We start with all the tasks numbered in increasing Φ_{ijc} (phase between τ_{ij} and τ_{ic}). Thus the task τ_{ic} in the original transaction is named τ_{i1} at the beginning of the normal-form processing.
The underlying idea behind the merge operation is that if two tasks τ_{ij} and τ_{ij+1} overlap, then the longest busy period initiated by τ_{ij} is always including the longest busy period initiated by τ_{ij+1}. The split operation is used when a part of a task has to be executed during the next period of the transaction : in this case the spilling task is splitted into two tasks. The spilling part is taken into account as a task with an offset equal to 0 in the second period of the transaction. Thus, since the tasks are numbered according to the increasing value of Φ_{ijc}, the tasks can be renumbered (ordering operation) in the second period of the transaction. These operations are used until no task in the transaction is forced to overlap on the next one, and until no task is forced to spill in the second period of the transaction.
Note that the first period of a transaction may differ in the number of tasks from the second period due to the spilling tasks.
Note that if a jitter has to be taken into account, this operation has to be done for every critical instant candidate.

3.2 Example

We apply this method on the transaction Γ_i that contains twelve tasks with no jitter ($J_{ij} = 0$). and the task τ_{ua} with a WCET $C_{ua} = 9$ and a lower priority

than all the tasks of Γ_i.

$$\Gamma_i := \{< \tau_{i1}\tau_{i2}, ..., \tau_{i12} >, 60\}$$

$$\begin{array}{ll} \tau_{i1} :< 3, 1, D_{i1}, 0, 0, 1 > & \tau_{i7} :< 2, 36, D_{i7}, 0, 0, 7 > \\ \tau_{i2} :< 4, 9, D_{i2}, 0, 0, 2 > & \tau_{i8} :< 5, 43, D_{i8}, 0, 0, 8 > \\ \tau_{i3} :< 2, 11, D_{i3}, 0, 0, 3 > & \tau_{i9} :< 3, 46, D_{i9}, 0, 0, 9 > \\ \tau_{i4} :< 3, 20, D_{i4}, 0, 0, 4 > & \tau_{i10} :< 1, 49, D_{i10}, 0, 0, 10 > \\ \tau_{i5} :< 4, 29, D_{i5}, 0, 0, 5 > & \tau_{i11} :< 4, 56, D_{i11}, 0, 0, 11 > \\ \tau_{i6} :< 5, 31, D_{i6}, 0, 0, 6 > & \tau_{i12} :< 2, 57, D_{i12}, 0, 0, 12 > \end{array}$$

Obtaining the normal-form for Γ_i for the critical instant candidate τ_{i1}: the three operations (order, merge and spill), merge τ_{i3} in τ_{i2}, τ_{i6} and τ_{i7} in τ_{i5}, τ_{i9} and τ_{i10} are merged in τ_{i8}, and τ_{i11} is merged in τ_{i12}. The last task spills in the next period, thus the second period of the transaction (and the following) will have a additional task The resulting transaction in normal-form, for the first period is:

$$\begin{array}{ll} \tau_{i1} :< 3, 0, D_{i1}, 0, 0, 1 > & \tau_{i2} :< 6, 8, D_{i2}, 0, 0, 2 > \\ \tau_{i3} :< 3, 19, D_{i3}, 0, 0, 3 > & \tau_{i4} :< 11, 28, D_{i4}, 0, 0, 4 > \\ \tau_{i5} :< 9, 42, D_{i5}, 0, 0, 5 > & \tau_{i6} :< 5, 55, D_{i6}, 0, 0, 6 > \end{array}$$

For the second period, the spilling time of the original τ_{i12} is taken into account in the first task τ_{i1} of the second period of the transaction, obtaining a WCET of 4.

$$\begin{array}{ll} \tau_{i1} :=< 4, 0, D_{i1}, 0, 0, 1 > & \tau_{i2} :=< 6, 8, D_{i2}, 0, 0, 2 > \\ \tau_{i3} :=< 3, 19, D_{i3}, 0, 0, 3 > & \tau_{i4} :=< 11, 28, D_{i4}, 0, 0, 4 > \\ \tau_{i5} :=< 9, 42, D_{i5}, 0, 0, 5 > & \tau_{i6} :=< 5, 55, D_{i6}, 0, 0, 6 > \end{array}$$

The same operation is done for all the task candidates τ_{ic} for $c = 2..12$. The upper bound of the worst-case response time is then obtained using formula 1:

$$\begin{array}{lcl} \text{Iteration } 0 & : & R^0_{ua} = 9 \\ \text{Iteration } 1 & : & t = 9 : R^1_{ua} = 9 + 11 = 20 \\ \text{Iteration } 2 & : & t = 20 : R^2_{ua} = 9 + 20 = 29 \\ \text{Iteration } 3 & : & t = 29 : R^3_{ua} = 9 + 29 = 38 \\ \text{Iteration } 4 & : & t = 38 : R^4_{ua} = 9 + 29 = 38 \end{array}$$

4 Monotonic Transactions

In this section we present the different steps of monotonic transaction analysis [13, 14, 12]. Monotonic transaction analysis relies on transactions for which one interference curve (for one candidate) is always greater or equal than the other curves. In this way, it's close to the concept of accumulatively monotonic generalized multiframe task sets [2]. In this case, if a transaction Γ_i is monotonic then the critical instant occurs when the task under analysis is released at the same time as the first task of the pattern of the normal form of Γ_i. Therefore, for the case where all the transactions of the task system are monotonic for a task under analysis, the computed worst-case response time is exact.

Since there is only one possible candidate in a monotonic transaction, there is only one normal-form to compute.

4.1 Finding the monotonic pattern

Let Γ^*_i be the normal form of the transaction Γ_i where $\Gamma^*_i :< \left\{\tau^*_{i1}, \tau^*_{i2}, ..., \tau^*_{i|\Gamma^*_i|}, T_i\right\}, T_i >$ and $\Gamma_i :< \left\{\tau_{i1}, \tau_{i2}, ..., \tau_{i|\Gamma_i|}, T_i\right\}, T_i >$. Let us note:

$$\alpha_{ij} = O^*_{i(j+1)} - (O^*_{ij} + C^*_{ij}) \; for \; 1 \leq j < |\Gamma^*_i|$$

$$\alpha_{i|\Gamma^*_i|} = (T_i + O^*_{i1}) - (O^*_{i|\Gamma^*_i|} + C^*_{i|\Gamma^*_i|})$$

where $\alpha_{ij} > 1$ since Γ^*_i is in normal form.
Note that since it's not necessary to statically store the interference function, there is no need to make a difference between the first and the second period of the transaction.

Γ_i is a monotonic transaction for the task τ_{ua} (we consider that all the tasks of Γ_i have a higher priority than the one of τ_{ua}) if the WCET of the tasks of Γ^*_i have decreasing values while the idle slots α_{ij} have increasing values i.e: $C^*_{i(p+1)} \leq C^*_{ip}$ for all $1 \leq p < |\Gamma^*_i|$ and $\alpha_{ip} \leq \alpha_{i(p+1)}$ for all $1 \leq p < |\Gamma^*_i|$.
Γ_i is monotonic if we can find a monotonic pattern in Γ^*_i by rotating the tasks of Γ^*_i. We know that for a monotonic pattern the first task has the highest WCET. In order to look for a monotonic pattern, we start by inventorying all the tasks with the maximum WCET. Then, we consider alternatively each of these tasks τ^*_{ik} as the first task of the transaction Γ^*_i by rotating the tasks of Γ^*_i; and we verify if the conditions of monotony (on C^*_{ij} and α_{ij}) are respected; if so, Γ_i is monotonic and τ^*_{ik} become the first task of Γ^*_i.

4.2 Example

We apply this method on the same example as the one we used for the fast and tight analysis.
We find Γ^*_i the normal form of the transaction Γ_i by applying the operations of normalization process:

$$\Gamma^*_i : \{< \tau^*_{i1}, \tau^*_{i2}, ..., \tau^*_{i5} >, 60\}$$

$$\begin{array}{ll} \tau^*_{i1} :< 6, 9, D_{i1}, 0, 0, 1 > & \tau^*_{i2} :< 3, 20, D_{i2}, 0, 0, 2 > \\ \tau^*_{i3} :< 11, 29, D_{i3}, 0, 0, 3 > & \tau^*_{i4} :< 9, 43, D_{i4}, 0, 0, 4 > \\ \tau^*_{i5} :< 9, 56, D_{i5}, 0, 0, 5 > & \end{array}$$

We have: $\alpha^*_{i1} = 5, \alpha^*_{i2} = 6, \alpha^*_{i3} = 3, \alpha^*_{i4} = 4, \alpha^*_{i5} = 4$
There is a monotonic pattern starting from task τ^*_{i3} where:

$$C^*_{i3} \geq C^*_{i4} \geq C^*_{i5} \geq C^*_{i1} \geq C^*_{i2}$$
$$\alpha^*_{i3} \leq \alpha^*_{i4} \leq \alpha^*_{i5} \leq \alpha^*_{i1} \leq \alpha^*_{i2}$$

Hence the transaction is monotonic, and the critical instant of τ_{ua} corresponds to the release of the task τ^*_{i3}, we apply the iterative fix-point lookup in order to calculate the worst-case response time of τ_{ua}.
In the case of monotonic transactions, the two methods presented provide the same exact worst-case response time with the same number of iterations in the process of calculation, because in every iteration, the task that initiates the critical instant is the same. The only difference between these two methods comes from the stage of static representation in the fast and tight analysis (stage A) and the research of the monotonic pattern for the method of monotonic transaction (stage B). Stage A for n tasks requires at least n normal-form processing, plus computing the static tables. Stage B requires only one normal-form processing operation and a linear complexity test in order to check the conditions related to C^*_i and α^*_i.

5. Performance comparison and future works

We have implemented the two methods in order to compare their respective performance. The figure 3 shows that the time used by the method proposed in [8]increases linearly with the number of transactions, while the method proposed in [14] is less sensitive to the size of the system when only monotonic transactions are involved. The tests have been led on a Pentium IV processor, for sets of 20 configurations per number of transactions. The transactions are monotonic, and contain 15 tasks, while the workload is fixed around 0.8. The random generation system is based on the UUniFast algorithm presented in [3].
The bound on the worst-case response time is the same for both methods, since at this stage, montonicity is more a characterization allowing an optimization than a method by itself (it has still to be coupled with the method of [8] because in a system of transactions, the transactions don't have to be all monotonic).
In future works, we will use the monotonicity property as a basement to introduce a new evaluation method in order to decrease the pessimism of [8] for the upper bound on the worst-case response times.

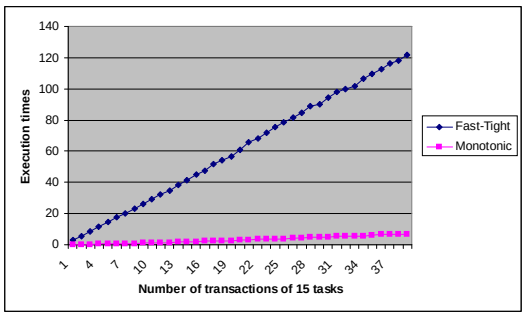


Figure 3. Execution time

References

[1] N. Audsley, A. Burns, R. Davis, K. Tindell, and A. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems 8*, pages 129–154, 1995.

[2] S. Baruah, D. Chen, S. Gorinsky, and A. Mok. Generalized multiframe tasks. *The International Journal of Time-Critical Computing Systems*, (17):5–22, 1999.

[3] E. Bini and G. Buttazzo. Biasing effects in schedulablity measures. *IEEE Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 04), Catania, Italy*, (16), July 2004.

[4] J. P. Gutierrez and M. G. Harbour. Schedulability analysis for tasks with static and dynamic offsets. *Proc IEEE Real-time System Symposium (RTSS)*, (19), December 1998.

[5] C. Liu and J. Layland. Scheduling algorithms for multiprogramming in real-time environnement. *Journal of ACM*, 1(20):46–61, October 1973.

[6] J. Maki-Turja and M. Nolin. Faster response time analysis of tasks with offsets. *Proc 10th IEEE Real-Time Technology and Applications Symposium (RTAS)*, May 2004.

[7] J. Maki-Turja and M. Nolin. Tighter response time analysis of tasks with offsets. *Proc 10th International Conference on Real-Time Computing and Applications (RTCSA'04*, August 2004.

[8] J. Maki-Turja and M. Nolin. Fast and tight response-times for tasks with offsets. *17th EUROMICRO Conference on Real-Time Systems IEEE Palma de Mallorca Spain*, July 2005.

[9] L. Sha, R. Rajkumar, and J. Lehoczky. Priority inheritance protocols : an approach to real-time synchronization. *IEEE Transactions on Computers*, 39(9):1175–1185, 1990.

[10] K. Tindell. Using offset information to analyse static priority pre-emptively scheduled task sets. *Technical Report YCD-182,Dept of Computer Science, Unoversity of York, England*, 1992.

[11] K. Tindell. Adding time-offsets to schedulability analysis. *Technical Report YCS 221, Dept of Computer Science, University of York, England*, January 1994.

[12] K. Traore. *Analyse et Validation des Applications Temps Réel en Présence de Transactions : Application au Pilotage d'un Drone Miniature.* Thèse, ENSMA-Université Poitiers, 2007.

[13] K. Traore, E. Grolleau, and F. Cottet. Characterization and analysis of tasks with offsets: Monotonic transactions. *Proc 12th International Conference on Embedded and Real-Time Computing Systems and Applications. RTCSA'06*, (12), August 16-18th Sydney, Australia 2006.

[14] K. Traore, E. Grolleau, A. Rahni, and M. Richard. Response-time analysis of tasks with offsets. *12th IEEE International Conference on Emerging Technologies and Factory Automation ETFA'06*, September 2006.

Stochastic network calculus for buffer overflow evaluation in an avionics switched Ethernet

Frédéric Ridouard, Jean-Luc Scharbarg, Christian Fraboul
Toulouse University - IRIT - ENSEEIHT
2, rue Camichel
31000 Toulouse - France
{Frederic.Ridouard,Jean-Luc.Scharbarg,Christian.Fraboul}@enseeiht.fr

Abstract

AFDX (Avionics Full Duplex Switched Ethernet, ARINC 664) used for modern aircraft such as Airbus A380 represents a major upgrade in both bandwidth and capability for aircraft data networks. Its reliance on Ethernet technology helps to lower some of the implementation costs, though the requirement for guaranteed service does present challenges to system designers. An objective is to prove that no frame will be lost by the network (no switch queue will overflow). Several approaches have been proposed for this evaluation. Deterministic network calculus gives a guaranteed backlog upper bound, while simulation produces more accurate results on a given set of scenarios. In this paper, we propose a stochastic network calculus approach in order to evaluate the probability of buffer overflow in a network node.

Keywords: Network Calculus, stochastic, backlog.

1. Introduction

The evolution of avionics embedded systems and the amplification of the integrated functions number in the current aircraft imply a huge increase in the exchanged data quantity and thus in the number of connections between functions.

The solution adopted by Airbus for the new A 380 generation consists of the Switched Ethernet technology which benefits from a long industrial use [1]. It allows to have confidence in the reliability of the material and on the facility of its maintenance. Hence aeronautical systems can profit of a much more powerful technology than the traditional avionics bus (Switched Ethernet / 100 Mbps).

AFDX (Avionics Full Duplex Switched Ethernet) [2, 3, 4] is a static switched Ethernet network. The full duplex switched Ethernet technology guarantees that there are no collisions on the physical links, compared with a vintage Ethernet solution [8]. So, it eliminates the inherent indeterminism of vintage Ethernet and the collision frame loss. But, it shifts in fact the problem to the switch level where various flows will enter in competition for the use of the resources of the switches. This can lead to temporary congestion on an output port of a switch that can increase significantly end-to-end delays of frames and even lead to frame losses by overflow of queues.

Flows on an AFDX network are statically identified in order to obtain a predictable deterministic behavior of the application on the network architecture. The analysis of the performance bounds for queues is necessary in order to dimension correctly the application. This analysis has to evaluate, on the one hand an upper bound on the backlog of a switch, on the other hand the distribution of this backlog. The first one is mandatory for certification reasons, while the second one can help greatly to evaluate the pessimism of the upper bound. In this paper, we study backlog distribution of nodes with a stochastic network calculus approach.

Section 2 specifies the performance of buffer analysis problem in the context of this paper. Section 3 presents the stochastic network calculus approach. Section 4 gives some results and evaluate their pessimism. Section 5 summarizes the paper and gives some guidelines for future works.

2. Scope of the study

We first give a brief overview of the AFDX network. Then, we formulate the problem of backlog analysis and the way it is addressed in the remaining of the paper.

2.1. The AFDX network

An example of an AFDX network architecture is depicted on Figure 1. It corresponds to a test configuration provided by Airbus for an industrial research study [6]. It is composed of several interconnected switches. There are no buffers on input ports and one FIFO buffer for each output port. The inputs and outputs of the networks are called *End Systems* (the little circles on Figure 1). Each End System is connected to exactly one switch port and each switch port is connected to at most one End System. Links between switches are all full duplex. On Figure 1,

the values on End Systems indicates number of flows that are dispatched between End Systems. Number of input and output End Systems per switch are not specified on Figure 1.

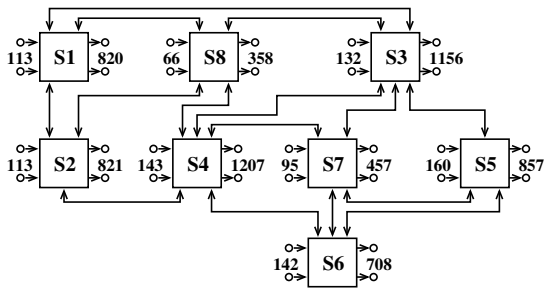


Figure 1. AFDX network architecture

The end-to-end traffic characterization is done by the definition of Virtual Links. As defined by ARINC-664, Virtual Link (VL) is a concept of virtual communication channels; it has the advantage of statically defining the flows which enter the network [4].

End Systems exchange Ethernet frames through VL. Switching a frame from a transmitting to a receiving End System is based on a VL (deterministic routing). The Virtual Link defines a logical unidirectional connection from one source End System to one or more destination End Systems. It is a path with multicast characteristic. The routing of each VL is statically defined. Only one End System within the Avionics network can be the source of one Virtual Link, (i.e., Mono Transmitter assumption).

Traffic on each Virtual Link is sporadic. Most of the time, physical links of an AFDX network are lightly loaded. As an example, on the configuration of Figure 1, most of the links are loaded at less than 15 % and no link is loaded at more than 21 % (see [6] for details). However, a congestion can occur at any time at any output port in case of a transient burst of traffic. Bursts of traffic occur when frames of different VLs reach the same output port at the same time. This event is closely related to the emission of the frames of the different VLs, i.e. the phasing between VLs.

2.2. Scope of the backlog analysis

In our context, no frame will be lost by the network and consequently, no switch queue will overflow. Then, the buffer must be dimension correctly, sufficiently but not too much to minimize the costs. Moreover, the end-to-end delay of a given path of a VL is the sum of the delays in each switch crossed by the path. The delay in a switch is composed of the switching delay (filtering and forwarding operations), the waiting time in the output buffer and the transmission time on the output link. The switching delay is a constant that depends on the switch technology ($16\ \mu s$ for switches used by Airbus). The transmission time is a function of the link rate (typically $100\ Mbps$). The waiting time of a frame depends on the load of the output port (backlog) at the arrival time of the frame. Therefore, the end-to-end delay is not constant due to the waiting times in the switch output ports it crosses.

In this paper, we propose a stochastic network calculus approach in order to obtain a distribution of backlog in a network node. Such an approach could deal with arbitrarily large network configurations. The next section presents the stochastic network calculus approach.

3. Stochastic network calculus analysis

First, we explain why stochastic network calculus theory can be applied in the AFDX context. Then we show how we apply stochastic network calculus results to our context.

3.1. Applicability of the analysis

As mentioned earlier, the aim is to obtain the distribution of backlog for each switch output port. The AFDX networks considered in the present study have a single FIFO buffer for each switch output port. That means that flows (VLs) all have the same priority. Consequently, each switch output port can be considered as servicing an aggregate traffic (all the VLs crossing this port) with a constant rate c which is the capacity of the output link (e.g. $100\ Mbps$). Moreover, the individual flows are shaped separately at network access, by the assumption of the minimum delay between the emission of two consecutive frames, i.e. BAG (Bandwidth Allocation Gap). It corresponds to a network considering *EF PHB* (Expedited Forwarding Per-Hop Behavior) service of *DiffServ* (Differentiated Services) architecture [7]. The nodes (i.e. the switch output ports) are said *PSRG* (Packet Scale Rate Guarantee) nodes [5] and the *EF* traffic at a node is served with a rate independently of any other traffic transiting the same node. The stochastic network calculus approach presented in [9] applies to such network configurations.

More formally, a node is *PSRG* (c,e) for a flow means this flow is guaranteed a rate c, with a latency (error term) e. Therefore if we denote d_n, the departure of the n^{th} packet of the *EF* aggregate flow, in order of arrivals, d_n satisfies

$$d_n \leq f_n + e$$

where f_n is calculated recursively as $f_0 = 0$ and

$$f_n = \max\{a_n, \min\{d_{n-1}, f_{n-1}\}\} + \frac{l_n}{c},\ n \geq 1$$

where the n^{th} packet arrives at time a_n with l_n bits.

The error term e is the extra waiting time due to non *EF* traffic. In our context, there is only *EF* traffic crossing each switch output port. Consequently, we have $e = 0$.

The works presented by *Vojnović* and *Le Boudec* in [9, 10] about networks with *EF PHB* service can be used to calculate the distribution of the backlog for each switch output port. It is based on the probability of bound buffer overflow in the switch output port. Such a problem was

previously addressed in the litterature but the results presented in [9, 10] have proposed the tightest upper bounds.

Vojnović and *Le Boudec* make four assumptions presented in [9]. These assumptions concern the modeling of the network and its elements. The assumption (A1) imposes to define a service curve β for nodes. But a property of *PSRG* is that a *PSRG* $(c, 0)$ implies the service curve $\beta(t) = ct$. Consequently, the property (A1) is respected. As *VL*s are independent at network access, assumption (A2) is respected. Concerning assumption (A3), in the AFDX context, each *VL* is regulated by a leaky-bucket ($\alpha_i(t) = \rho_i t + \sigma_i$) defined in the following way. σ_i is the maximum length of a frame of the VL, denoted S_{max}. ρ_i is the VL maximum flow, $\frac{S_{max}}{BAG}$, where BAG is the minimum delay between the emission of two consecutive frames of the VL by its source end system. Therefore assumption (A4) is valid with $\xi_i = \rho_i$.

Vojnović and *Le Boudec* define the concept of *EF* traffic inputs homogeneously regulated : the *EF* traffic inputs are homogeneously regulated, if they are regulated by the same function : $\alpha_i(t) = \alpha_1(t)$, for all $i \in \{1, \ldots, \mathcal{I}\}$. Otherwise, the *EF* traffic inputs are heterogeneously regulated. In our context, traffic inputs are homogeneously regulated when all VLs have the same S_{max} and BAG and they are heterogeneously regulated otherwise. Results in [9] have been proved for homogeneous and heterogeneous cases, while better results are presented in [10] for homogeneous cases.

As all the assumptions made by *Vojnović* and *Le Boudec* are respected, their results can be applied in our context.

3.2. The analysis

We denote $Q(t)$ the backlog at time t of a studied node. *Vojnović* and *Le Boudec* establish upper bounds of probability that the backlog is above a given level b ($\mathbb{P}(Q(t) > b)$). In [9], Theorem 2 defines the tightest backlog bound (that we denote in the following V1) that holds for homogeneous and heterogeneous regulation of traffic inputs. Whereas, in [10], a tighter bound (denoted V2) is given by Theorem 4, but that holds only for the homogeneous case.

3.3. Application of the analysis

In order to determine the distribution of the backlog of a given switch output port, we first compute

$$\mathbb{P}(Q(t) > b) \; with \; b = 25, 75, 125, \ldots$$

until $\mathbb{P}(Q(t) > b) = 0$

We have
$$\begin{aligned} \mathbb{P}(Q(t) > 0) &= 1 \; and \\ \mathbb{P}(Q(t) > b) &\geq \mathbb{P}(Q(t) > b + 1) \end{aligned}$$

Finally, we consider

$\mathbb{P}(Q(t) = b) = \mathbb{P}(Q(t) > b - 25) - \mathbb{P}(Q(t) > b + 25)$
with b=50, 100, 150, ...

4. First results

The stochastic analysis presented in the previous section has been applied to AFDX network configurations. First results are presented in this section. They all concern switch output ports of one switch, for example, the output port of sy with the destination e_q of Figure 2.

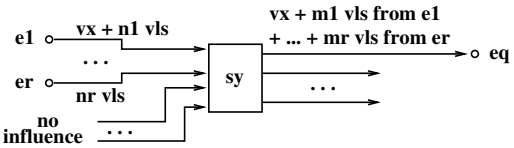


Figure 2. Stochastic analysis context

We are only interested by inputs crossing the switch sy and as destination e_q. We denote e_1 to e_r these End Systems. Each of those End Systems emits n_i VLs and among which m_i have e_q as destination End System.

	S_1	S_2	S_3	S_4	S_5	S_6
m_1	4	26	48	70	92	114
m_2	5	31	57	83	109	135
m_3	2	13	24	35	46	57
load	2 %	12 %	22 %	32 %	41 %	51 %

Table 1. studied configurations

Results presented in this paper concern the configurations of table 1. We consider three end systems emiting VLs ($r = 3$). $S1$ corresponds to a typical VL path of the industrial AFDX configuration of Figure 1. The load of the single switch output port crossed by the VL under study is about 2%. Configurations $S2$ to $S6$ concern VLs paths similar to $S1$ with a higher load on the switch output port (between 12% and 51%). We consider homogeneous flows ($S_max = 672bits$ and $BAG = 4000\mu s$). Therefore, the approaches $V1$ and $V2$ can be applied. In order to evaluate the relevance of these stochastic methods, we have to state, on the one hand the pessimism of the obtained distribution, on the other hand the pessimism of their upper bounds.

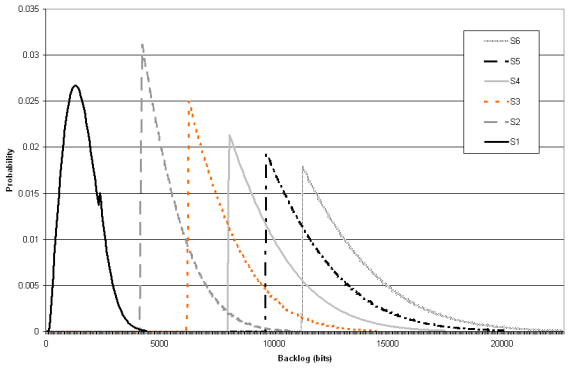


Figure 3. Stochastic network calculus V1

Figures 3 and 4 present the distributions of the backlog in the output port of switch sy, obtained with the V1 and V2 stochastic network calculus approaches presented in section 3.

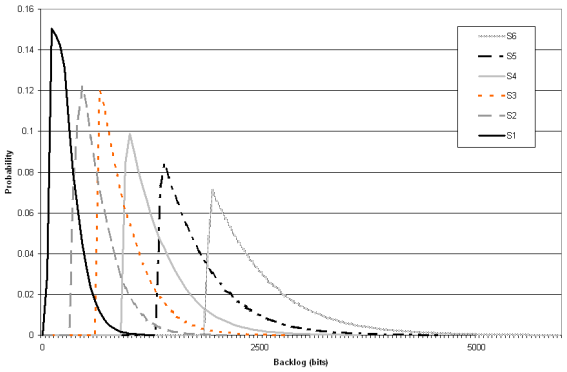


Figure 4. Stochastic network calculus V2

With the $V1$ approach, the interval where backlog are distributed shifts to higher values with the increase of the load on the switch output port (backlog for configuration $S1$ are mostly distributed between 0 and 4200 bits whereas for $S6$, the backlog is distributed between 11200 and 22400 bits). With the $V2$ approach, we obtain similar distributions with smaller intervals (between 0 and 1000 bits for $S1$ and between 1800 and 4600 bits for $S6$). Obliviously, the $V1$ approach is more pessimistic than the $V2$ one because the configurations contain only homogeneous flows. However, the $V2$ approach can't apply when flows are heterogeneous.

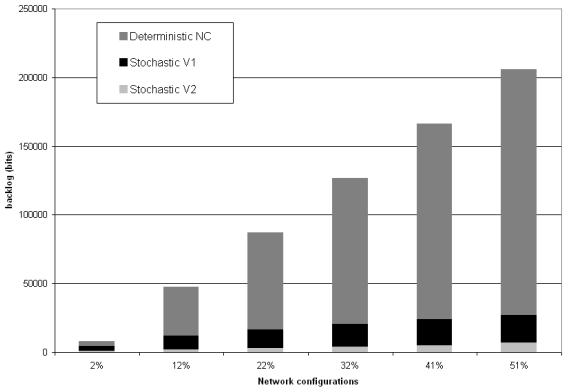


Figure 5. Comparison by upper bounds

Figure 5 presents for each network configuration, the upper bounds that the backlog, presents in the output port of sy, will exceed with a probability of 0.00001. The upper bounds are calculated by the stochastic methods $V1$ and $V2$ and by the deterministic Network Calculus. Considering the deterministic network Calculus approach, the difference with others methods increase with the load. For example, between the deterministic Network Calculus and the $V2$ approach, 7392 vs 1350 for a load of 2% and 205632 vs 7300 for a load of 51%. Consequently, the pessimism of this approach increases with the load of the switch output port. Conversely, the variation between the two stochastic methods ($V1$ and $V2$) remain relatively stable.

5. Conclusion

In this paper, we detail the stochastic network Calculus used to analyse the backlog contained in the output buffer of switches on an industrial switched Ethernet network. Two important characteristics are the upper bound of the backlog and their distribution. The first one is mandatory for certification reasons. The second one can help greatly to evaluate the pessimism of the upper bound. We present a stochastic network calculus approach that gives an evaluation of the backlog present in a output port of a given switch. The upper bound obtained with this method is less pessimistic than the upper bound calculated with the deterministic Network Calculus.

In further works, we focus on the utilization of the stochastic network calculus to determine the end-to-end delay of a given flow that crosses a switch.

References

[1] *IEEE 802.1D, Local and Metropolitan Area Network: Media Access Control Level Bridging.*, 1998.

[2] *ARINC 664, Aircraft Data Network, Part 1: Systems Concepts and Overview.*, 2002.

[3] *ARINC 664, Aircraft Data Network, Part 2: Ethernet Physical and Data Link Layer Specification.*, 2002.

[4] *ARINC 664, Aircraft Data Network, Part 7: Deterministic Networks.*, 2003.

[5] J. Bennett, K. Benson, A. Charny, W. Courtney, and J. Le Boudec. Delay jitter bounds and packet scale rate guarantee for expedited forwarding. *IEEE/ACM Transactions on Networking*, 10(4), August 2002.

[6] H. Charara, J.-L. Scharbarg, J. Ermont, and C. Fraboul. Methods for bounding end-to-end delays on an AFDX network. In *Proceedings of the 18th ECRTS*, Dresde, Germany, July 2006.

[7] B. Davie, A. Charny, J. Bennett, K. Benson, J. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. An expedited forwarding PHB (per-hop behavior). *Network Working Group*, March 2002.

[8] J. Jasperneite, P. Neumann, M. Theis, and K. Watson. Deterministic Real-Time Communication with Switched Ethernet. In *Proceedings of the 4th IEEE International Workshop on Factory Communication Systems*, pages 11–18, Västeras, Sweden, August 2002. IEEE Press.

[9] M. Vojnović and J. Le Boudec. Stochastic analysis of some expedited forwarding networks. *in Proceedings of Infocom, New-York*, June 2002.

[10] M. Vojnović and J. Le Boudec. Bounds for independent regulated inputs multiplexed in a service curve network element. *IEEE Trans. on Communications*, 51(5), May 2003.

Multilevel Tracing for Real-Time Embedded Systems *

Aitor Viana
Aitor.Viana.Sanchez@esa.int
European Space Agency, Noordwijk NL, 2200AG

O.R. Polo M. Knoblauch P. Parra S.S. Prieto
D. Meziat
{opolo, martin, pablo.parra, ssp, meziat}@aut.uah.es
University of Alcala
Computer Engineering Department

Abstract

Real-time systems development is a complex process. The ability to trace the system execution is very important in order to verify the correct system behavior. System tracing requires instrumented code that alters the behavior of the system, for instance, increasing its response time. In embedded systems, tracing can be used to monitoring the system evolution during its execution. In that way, the instrumented code is neither intrusive nor dead code that should be removed from the final version. Indeed, it is part of the final system and also helps after deployment. We provide a solution capable of tracing all the software levels of the system, including the kernel, the modeling language level and the different application levels. POSIX 1003.1q event tracing standard provides an interface to handle event data, but it lacks of a system level concept and multilevel approach. This paper presents an implementation of the POSIX 1003.1q over the ERCOS-RT operating system, which has been developed to work with the component based graphical modeling and automatic code generation tool named EDROOM. ERCOS-RT and EDROOM have both event tracing capabilities that can be integrated in a multilevel approach including also the top user defined application events.

Keywords*: Embedded Systems, Real-Time Systems, Tracing, POSIX.*

1 Introduction

The development of real-time embedded applications needs of some tracing mechanism in order to ensure a correct system behavior. System monitoring provides a lot of information very handy to certify the system constraints and performance. In embedded systems, the tracing information is a key source of knowledge, not only in validation and verification processes, but also during the whole system execution.

Many works have been done in the tracing area, and a wide variety of profiling, tracing tools and methodologies are available [5, 11], but none of them tackle the problem of tracing an application at different levels. In the last five years, IEEE introduced tracing facilities having resulted in the definition of the POSIX Trace standard 1003.1q [4]. This standard provides an interface allowing to trace the different events from real-time systems.

We use the POSIX Trace standard to implement a multilevel tracing system, capable of tracing all system events ranging from the low kernel level to the high application level, but the standard lacks of both a system level concept and a multilevel approach. Thus, some modifications in the standard are proposed in this paper reaching a more suitable solution.

The tracer is implemented over the ERCOS-RT real-time kernel, which has been developed to work with the EDROOM tool [8].

The rest of the paper is organized as follows: Section 2 describes the POSIX Trace standard aspects. Section 3 introduces the relevant aspects of the EDROOM tool and the ERCOS-RT design. Section 4 deals with the implementation details of the POSIX Trace standard and our proposed extension to handle multiple layers. On section 5 some timing and overhead measurements are reported. Section 6 presents a tracing example with a real system. Finally, the last section shows the conclusions.

2 POSIX Trace standard Overview

The POSIX Trace standard has been developed to provide tracing facilities. It defines two main data types, called *events* and *trace streams*.

2.1 Trace Events

The points where the information must be generated are called *trace points*, and the information itself is called *trace events*. When an instrumented application wants to register a new event, it must invoke the `posix_trace_eventid_open()` routine which returns the event identifier. The event trace mechanism is performed by calling the `posix_trace_event()` routine. The standard specifies the tracing information that must be saved,

*This work has been supported by *Comisión Interministerial de Ciencia y Tecnología (CICYT)* of Spain, grant ESP2005-07290-C02-02

which is: (1) the trace event type identifier; (2) a time-stamp; (3) the process identifier of the traced process; (4) the thread identifier of the traced process, if threads are supported by the operating system; (5) the program address where the trace is being performed; (6) any extra data associated with the event and previously defined by the user; (7) the extra data size.

2.2 Stream Buffers

When any system application traces an event, its information is stored in the stream buffer. The standard specifies that streams must be created by processes and the relationship between streams and processes is many-to-many. By default, all events associated with a process are traced in all stream buffers belonging to that process. Thus, it is possible to trace events from a single process into many streams. The POSIX standard also supports event filtering, allowing events from one process to be associated with a single stream and also tracing the events from various processes into one single stream. The standard defines active streams. An active stream is created to trace events during system execution. It can also be associated with a log file in order to store the information on a persistent object when a flush operation is performed.

2.3 Tracing Roles

The POSIX Trace standard defines three types of roles called *trace processes*: (1) the *trace controller process*, which is in charge of the stream buffer creation; (2) the *traced process*, which is the one being traced; and (3) the *analyzer process*, which is in charge of retrieving the traced events from the stream buffer in order to analyze the system behavior.

3 EDROOM and ERCOS-RT Overview

3.1 EDROOM Overview

EDROOM[1] is a tool inspired on ROOM [9] and UML2 [3] methodologies. This tool provides facilities for modeling real-time systems using the object oriented paradigm, and also integrates an automatic Embedded C++ code generator. EDROOM lets the designer describe the structure, communication and behavior of the real-time systems using diagrams. Both structure and behavior can have several levels of definition in order to ease an iterative design of the system. The components of the multilevel structure are communicated with each other by message passing through their ports.

In figure 1 an EDROOM graphic representation of a system structure and communication with three levels of actors is shown. The behavior of each component is defined using a kind of hierarchical state chart, called ROOMCharts, based on the Statecharts introduced by Harel [6]. The received messages lead the transitions triggering between the states.

3.2 ERCOS-RT Overview

The ERCOS-RT[1] real-time operating system has been designed to support only the services required by the EDROOM tool, which are timing, thread management, synchronization and interrupt handling services. It has been developed over the standard platform of the European Space Agency (ESA) in space missions: the ERC32 [10] architecture.

[1] Available at http://srg.aut.uah.es

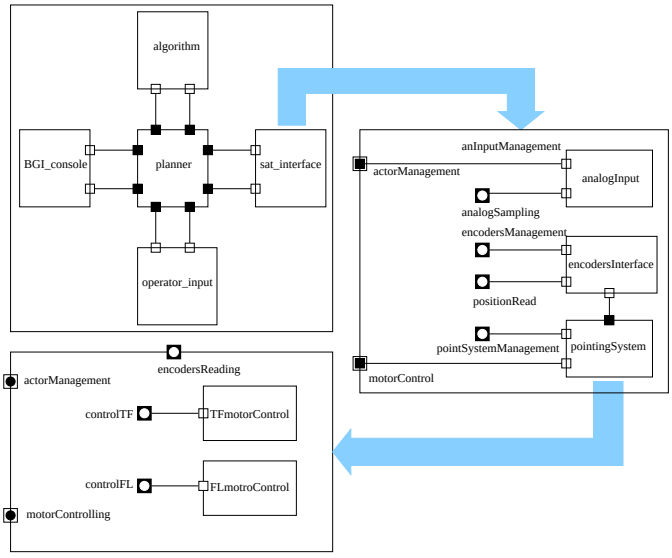


Figure 1. Multilevel structure of a EDROOM model.

ERCOS-RT has four layers: hardware dependent layer, kernel layer, system call and a POSIX interfaces.

The main requirements formulated for this kernel are: (1) to be compliant with the POSIX.13 Minimal Real-Time System Profile; (2) to have a hard real-time performance; and (3) to be easily adaptable to any other platform. It is specially targeted for embedded real-time applications that have defined all the needed resources at compilation time, making the kernel fully configurable. It is already running over the LEON2 and LEON3 architectures [1] and it is also being ported to the M68K 68332 platform.

4 Multilevel Tracing Implementation

4.1 POSIX Trace Standard to perform Multilevel Tracing

The proposed solution is compliant with the POSIX Trace standard but extends its functionality in order to manage different trace-levels. ERCOS-RT does not have the process abstraction, and it implies that the three roles defined in the POSIX Trace standard (controller, traced and analyzer processes) are implemented by only one process at the standard point of view, but they are really implemented over two independent execution units (threads). There is also no persistent mechanism to store any data (no file system is available), so the implementation of the log file is difficult, but it can be achieved by carrying out the logging process in a remote machine. The stream flusher process is not implemented because it would overload the system.

In respect to the multilevel tracing, it could be implemented by using one stream for each level, which should perform filtering operations to avoid tracing information belonged to other levels. The multilevel tracing is not implemented using this approach because it would be not possible to keep the inter-level encapsulation. This encapsulation assumes that one level (and one traced thread) does not know about the others, but the filtering operations need the processes/threads to know all the

events being traced, broken this encapsulation.

For this reason, ERCOS-RT uses only one stream to trace all levels. In this stream all events are recorded automatically and it is unnecessary to implement filtering options. On the other hand, it is necessary to identify the event and the level it corresponds to, but the POSIX standard only considers the event identifier. Therefore, some modifications are done to solve this problem. Figure 2 depicts the tracing mechanism over ERCOS-RT. System threads provoke the kernel to trace the different events when they invoke certain system calls, so they are both the controller and traced threads. This trace mechanism is depicted as "T" in the figure. The information is buffered and the analyzer thread can retrieve it either by sending it to a remote PC (via debugging line) where the information is analyzed on/off-line or by storing it in memory in order to be retrieved if an error or a fail condition are detected.

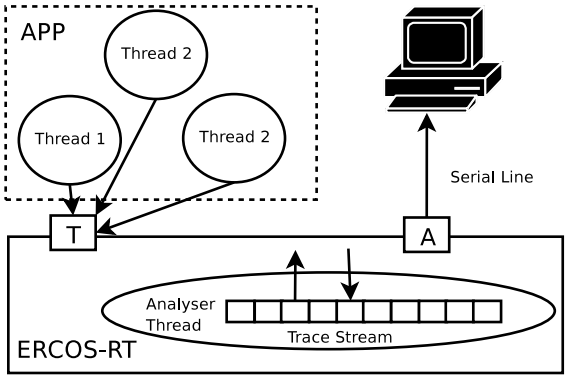


Figure 2. Tracing System.

4.2 POSIX Trace Add-ons

To identify an event belonging to a certain level, two identifiers must be known, the level and the event identifiers. The problem is that the POSIX routines used to register a new event, `posix_trace_eventid_open()`, and to trace an event, `posix_trace_event()` do not consider the level field.

Let's assume that the event identifier field has 32 bit length (any other length could be possible). We propose to divide this field into two parts, the higher 8 bits to identify the level and the lower 24 bits to identify the event. Due to this modification, in the routine `posix_trace_eventid_open()`, the event identifier is passed by reference with its upper 8 bits set to the value of the level identifier. The routine fills only the lower 24 bits in order to identify the event unanimously. The `posix_trace_event()` routine does not suffer any modification, because it is only in charge of storing the information in the stream buffer.

4.3 Implementation Issues

The implementation supports the standard Trace and the Trace Event Log options. The analyzer process has been carried out by the idle thread and the threads being traced acts as the controller and traced processes. Because the analyzer process is the idle thread, it does not interfere with the overall system execution and it does not overload it because it is only executed when no other thread is ready or in execution. In figure 3 a simple example of a tracing execution is shown.

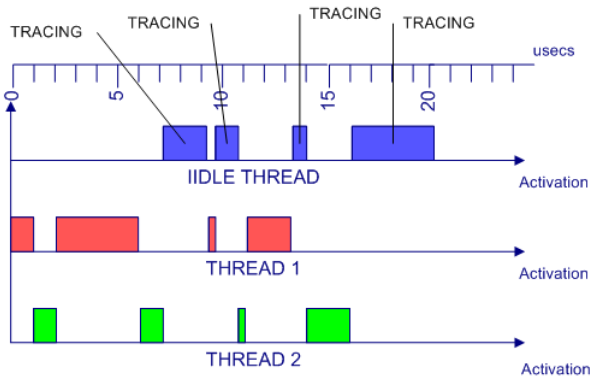


Figure 3. Simple Tracing Diagram

By default, the analyzer process retrieves the event information through a debugging serial line, but it is also possible for the controller process to register a new method in order to retrieve it; for example, storing it in a certain memory location or sending it through any other device.

5 Performance of the Implementation

ERCOS-RT has about 7500 lines of code and only a 3% of these lines are associated with the kernel tracing system. The kernel traces the next events: (1) schedule entry/exit; (2) semaphore wait/signal; (3) thread creation/termination; and (4) thread block. The traced events have a fixed size that can be set at compilation time (plus 16 bytes for the header) and also the amount of memory associated to the stream buffer. The accuracy of the traced events timestamping is 18 microseconds.

The table 1 shows the overload results over the ERC32 real architecture and TSIM simulator, both at 16MHz.

Event data size	TSIM (us)	Real HW (us)
4 bytes	332 us	334 us
8 bytes	346 us	348 us
16 bytes	373 us	375 us

Table 1. Overload Results

In the development phase, the events retrieval through a debug line does not interfere the real-time application because this labor is carried out by the idle thread. In the final system, there is also no overhead in the data delivery because they are sent whenever a system error occurs and that issue is part of the system behavior.

6 Case of Study

The solution presented was used to verify the behavior correctness of the on-board software for the Spanish nano-satellite called NANOSAT [7]. In the development phase, three levels were traced and the event size was fixed to the maximum event size, which was 64 bytes:

The first level is associated with the ERCOS-RT kernel. Despite the kernel uses different interface to that of POSIX, it is low-level compatible with the tracing system. Traced events from this level were wrapped to be analyzed by an application called Kiwi [2], developed by the University of Valencia,

that generates sequence diagrams, signaling task switches and resource accesses. Figure 4 shows an example of this trace.

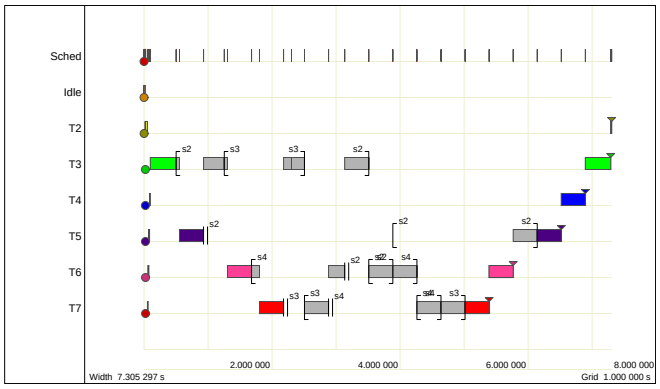


Figure 4. Kernel Tracing Using Kiwi.

The second level is the EDROOM level. The information regarding with the transitions and state changing is automatically generated by the EDROOM tool using the POSIX tracing interface. This information consists in the triggered transitions, time-tagged, between the states associated to each component, allowing an off-line analysis to verify the system design and the fulfilment of the time requirements during system execution. We used the trace information to feed a tool allowing the off-line representation of the system behavior by showing all the system execution at state-machine level. The trace of this level is shown in figure 5.

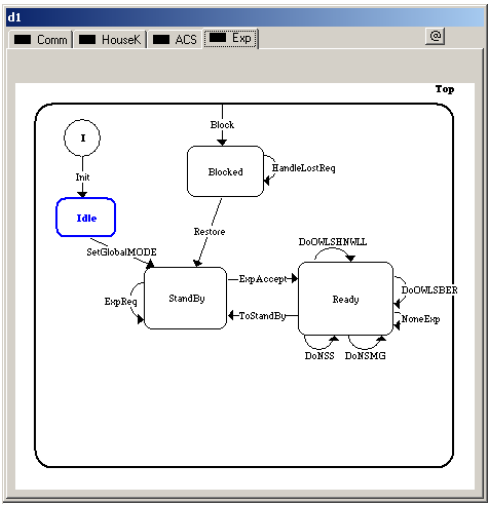


Figure 5. EDROOM Level Behavior Tracing.

The last level is associated with the on-board software. The on-board SW informs about general changes in the system, such as activation of the experiments, communications, housekeeping activities, etc. All the information is sent to the Electrical Ground Support Equipment (EGSE) to monitor all the events maintaining a precise reconstruction of the state of the satellite. This last level allows to monitor the execution of the on-board SW being very useful in the test phases to check the correctness in the timing of the SW events, for instance, the experiments execution, the housekeeping acquisition, etc.

Because of the complexity of the on-board SW, there were a great amount of state machines and a lot of high system states (third level) to be notified, so that the amount of events to be traced was large enough to stress the tracer implementation. Moreover, the kernel level also introduced a huge “event to be traced” overload because each context switch, each resource access, etc. were also traced.

7 Conclusions

Nowadays, it is necessary to be capable of tracing the embedded real-time systems behavior due to its complexity. To perform the trace it is necessary to introduce instrumented code, but it can be also integrated in the final system in order to monitor the system evolution.

The POSIX Trace standard defines an application tracing interface, however, it lacks of a system level concept neither multilevel tracing capabilities. The work explained in this paper extends the POSIX trace functionality, by introducing the tracing level concept, being also compliant with it without affecting the standard interface neither the standard operation.

The final implementation facilitates a multilevel trace mechanism, in which the kernel and all the defined user levels can be traced. All levels are managed by means of a single stream buffer but making possible the selective and independent recovery of each level trace information by using different tools.

Finally, the possibility to extend the POSIX Trace interface, in order to introduce some primitives allowing enable/disable each tracing level independently can be considered in the future. Moreover, it could be possible to carry out this labor even at execution time.

References

[1] Gaisler Research. http://www.gaisler.com.
[2] KIWI. http://rtportal.upv.es/apps/kiwi/.
[3] UML 2.0. www.u2-partners.org/uml2wg.htm.
[4] Standard for Information technology-Portable Operating Systems Interface (POSIX) - Part 1: System Application Program Interface (API) - Admendment 7: Tracing [C Language], 2000.
[5] V. Danjean, R. Namyst, and P.-A. Wacrenier. An efficient multi-level trace toolkit for multi-threaded applications. In *EuroPar*, volume 3648/2005, pages 166–175, Lisbonne, August 2005.
[6] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
[7] O. R. Polo, L. de Salvador, M. Angulo, and J. M. de la Cruz. Development plan of the on board satellite software based on ROOM modelling and evolution of component based prototypes. In *27th IFAC/IFIP Workshop on Real-Time Programming.*, 2003.
[8] O. R. Polo, D. la Cruz J. M., G.-S. J.M., and E. S. Edroom. automatic C++ code generator for real-time systems modelled with ROOM. In *NTCC2001 IFAC Conference*, November 2001.
[9] Selic, B., Gulleckson, G., and W. P.T. *Real-Time Object Oriented Modelling.* John Wiley and Sons, 1994.
[10] V. Stachetti, J. Gaisler, G. Goller, and C. L. Gargasson. 32-bit processing unit for embedded space flight applications. *IEEE Transactions*, 43:873–878, June 1996.
[11] K. Yaghmour and M. R. Dagenais. Measuring and characterizing system behavior using kernel-level event logging. In *Proceedings of the 2000 USENIX Anual Technical Conference*, pages 13–26, 2000.

Institut National Polytechnique de Lorraine
Impressions et Reliures :
INPL – Atelier de reprographie
2, avenue de la forêt de la Haye
B.P. 3. – F-54501 Vandoeuvre Cedex
Tel : 03.83.59.59.26 ou 03.83.59.59.27
Scientific editor : Liliana CUCU (LORIA)