A Specifying Method for Real-Time Software Requirement

Jung-sool Kim*

요 약 이 논문은 실시간 소프트웨어의 요구 분석을 위한 구조이다. 이 방법은 실제 사용자들과의 쉬운 통신 수단으로 TNPN(Timed Numerical Petri Net)을 사용하는데, 시스템의 정확성을 위해서 RTTL(Real Time Temporal Logic)에 기초한다. TNPN은 행위 명세언어로서 사용되며, 그 행위의 정당성은 RTTL로 표현되는데, 도달성 그래프를 통해 분석되었다. 그래서 사용자와 시스템의 두가지 요구사항이 모두 만족된다. 공유트랙 시스템의 예를 통하여 실시간의 성질(안전성, 응답성, 생존성, 우선순위)들이 검증되었다. 또한 이 구조는 자연스럽게 객체와 연결된다.

Abstract This paper is on the analysis for the real-time software requirement. This method can be used for TNPN(Timed Numerical Petri Net) as a easy communication means with real-users. It is based on the RTTL(Real Time Temporal Logic) for correctness of the system. TNPN is used to represent a behavior specification language, the validity of specified behaviors in TNPN is expressed in RTTL, and analyzed through the reachability graph. Thus, the requirement between user and system is satisfied. Using the example of shared track, the validity of the property of real-time(safetiness, responsiveness, liveness, priority) is verified. Also, this framework is given to connection with a object, natually.

1. Introduction

In general, the correctness of real-time systems depends not only on the logical result of the system behavior, but also on the time at which the results are produced(stankovic,1988).

Typical applications include process control, flexible manufacturing systems, robotics, avionics and embedded real-time computer systems. Real-time systems composed of interacting processes share

the following characteristics(ostroff,1990).

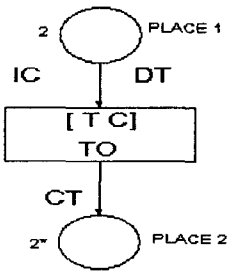
(1) Event occur at discrete times and states have discrete values.

(2) Processes are event-driven rather than clock driven.

(3) Processes are typically nondeterministic.

(4) Interaction and communication between processes may be synchronous or asynchronous.

(5) Hard real-time deadlines must often be met.



(Fig.1) Arch. of basic TNP

The difficulty of modeling real-time systems has long been recognized in software engineering. As a result, a variety of methods have been proposed for the modeling of real-time systems. Although there is a growing theoretic studies on real-time systems, little work has been done on hard real-time systems, i.e.,

* Yeungnam university Dept. of Computer Science and Engineering

systems that are guaranteed to meet timing deadlines (reed(1986), wirth(1977)). So, this paper describes TNPN/RTTL(Timed Numerical Petri Net/Real Time Temporal Logic) framework and provides example for illustration. TNPN is used to represent a behavior of system and RTTL is the assertion language for specifying behavior and verifying that the TNPN satisfies the required properties of real-time systems. Verification is performed by reachability graph of global states. In this paper, we confine the scope of system to finite systems.(ostroff,1989,1990)

2. Related Research

(1) Programming Language : Real-time programming languages suffer from the lack of an abstract mathematical model. As a result, the designer is faced with uncertain semantics, and thus the lack of analysis capabilities(wirth,1977).

(2) Petri Nets : Petri nets nicely model concurrency and nondeterminism, but do not represent numerical information such as data in a message header, time out information, and message sequence numbers found in complex communication protocols. In fact, there is no generally accepted high-level specification language for petri nets, and properties such as net invariance and freedom from deadlock are expressed in diverse and sometimes unrelated formalisms.(billington(1988),murata(1989),pe_terson(1981)

(3) Structured Methods : Structured methods(gomaa, 1984) for real-time systems originated in systems analysis methods used in industry area. These methods have no formal semantics, there is no support for formal verification. So, nondeterministic plant behavior can not suitably modelled.

3. TNPN/RTTL

3.1 Syntax Definition of TNPN

A TNPN M of M is a four-tuple given by M = (V,T,S,E). V is a set of variables. T is a finite set of transitions, S is a finite set of states, E is a set of all events. here, $T \ni \tau i$ = (Ssource, E, C, A, Sdestination). See the Fig.1.

3.2 Semantics Definition of TNPN

Given an TNPN M with state-space S a trajectory is any path in the state-space consisting of an infinite sequence of state and transitions. The set of all trajectories of M is denoted Sw. Not all trajectories represent possible behaviors of M(denoted Σ). The set of legal trajectories of M contains just those trajectories that are possible behaviors($\Sigma \subseteq Sw$). All necessary information is collected into a 5-tuple called a generator of trajectories. GM = (GV, R, GS, GT, init), here, GV(all variables) = Vi $\cup$ TSi $\cup$ {ni, ti, IEi, TEi, SAi}, V is called the variables set, TS(Token State) is a current state in system, ni is next transition variable with type(n) = T, t is time integer, IE is a set of internal event, ie $\in$ IE = {event, ON(state),OFF(state)}, TE is set of event not exceed time after event, te $\in$ TE = (te(event name, remaining time)), SA(scheduled action), sa $\in$SA = (action name, remaining time), R is the union of all types and is thus the range of all values that variables in V can have. GS is the set of all states, GT is the set of all transitions, $g\tau = (\alpha\tau, e\tau, h\tau, l\tau, u\tau)$, $l\tau \leq u\tau$. here, α is occuring event, e is enabling condition, h is a transform funtions, l is lower time for transition, u is upper time. init is the set of initial states, i.e. s0(n) = initial.

This is a definition on the global state transition (ko,1994)allowing in this paper.

· simple state transition

Simple transition is composed of $g\tau = (\alpha, e, h, l, u)$ for τ = (Ss,E,C,A,Sd). It sames as the general petri net.

· join state transition

$\tau 1$ ... τn is followed fig.1. It allows for petri net's join property.

· conflict state transition

This transition is sames as petri net's conflict transiton.

· initial state transition

This is satisfied with initial or reboot in system. It expressed as α = init, e = TRUE, g τ = (TRUE,TRUE,[],0,0)

· tick state transition

The clock ticks infinitely often, there are an infinite number of states S in initialized trajectories.

tick = (tick,TRUE,[t:t+1],0,∞).

· timeout state transition

It expressed as α = TRUE, e= timeouevent(to) when any event has exceed time.

· scheduled action state transition

It expressed as α = TRUE, e= scheduled(sa) when time = 0.

· terminate state transition

It is satisfied with when system arrive terminate state. In this paper, we can use initial or terminate transition implictly.

The transform function h of transition τ when applied to state-assignment[1] q yields a set of successor state assignments. In this paper, we introduce state assignment(Q = (V-{n}) and state map(Sm = (V-{n,t}). state map is availables to draws reachability graph. Transform function h is defined belows in each basic behaviors, i.e. event, action. Basical transform representations is as follows.

. h(q) in behavior E = {q,IE:IE∪{E}}
. h(q) in behavior ON(s) = {q,ts,ts∪{s}, IE :∪{ON(s)}}
. h(q) in behavior OFF(s) = {q,ts:ts-{s},IE :∪{OFF(s)}}
. h(q) in behavior TRUE(c) = {q,c:TRUE, IE: IE ∪ {TRUE(c)}}
. h(q) in behavior FALSE(c) = {q, c:FALSE,IE: IE ∪ {FALSE(c)}}
. h(q) in behavior TO(e,t) on E = {q,TE: TE ∪ {[TO(e,t),t]}
. h(q) in behavior TO(e,t) on t = {q,TE :TE-{[TO(e,t),t:t-1]}, IE: IE ∪ {TO(e,t)}}
. h(q) in behavior SA(a,t) on a = {q,SA: SA ∪ {SA(a,t)}}
. h(q) in behavior SA(a,t) on t = {q,SA :SA - {[SA(a,t),t:t-1],q}}
. h(q) in behavior tick = {q,t: t+1, SA:SA - (*,1),TE : TE - (*,1)}

3.3 Legal trajectories

Let S be the state space of an TNPN. A trajectory σ = S0S1S2 ... = init→q0 g τ 0→q1 g τ 1→q2 ... The notation init→q0 g τ 0→q1 g τ 1→q2 ... for a trajectory is often preferred as it allows us to picture a trajectory as a sequence of state assignments with transition taking us from one state assignment to the next. The legal trajectory set ΣM is the set of all initialized trajectories of M together with all suffixes of initialized trajectories. An initialized trajectory σ satisfying the following requirements. (a) initialization : The state S0 ∈ init, and initial never occurs again in σ. (b) succession : For each i, if si(n) = τ, then si(e) = true, qi+1 ∈ h(qi). (c) ticking. (d) upper time, lower time bound(ostroff(1989,1990)).

3.4 Reachability Graph for TNPN

Earlier we have defined a finite state in this system. So, these finite global state TNPN has a finite reachability graph. To reachability graph, we have introduced a state mapping Q. Reachability graph RG = (Q,E), Q is a all global state, E is a set of global state transition component. So initial state map, init is defined follows.

init = { state s : s(init) = true }, and if a state mapping is a node of the RG, RG(q) is defined as follows.

node(RG(q)) : {q} ∪ {q′ ∈ node(RG) | q′ is reachable state from q},

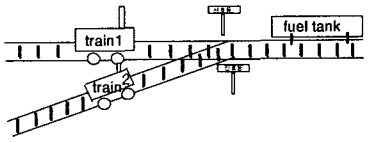
edge(RG(q)) : {(qs,g τ _set,qd) ∈ edge(RG) | qs,qd ∈ node(RG(q))}, and the algorithm for RG is presented as follows.

WHILE there is a reachable node in node(RG) DO
check factor SAi ∈ qs in initial state qs(s=0).
    if (SA = Ø) then {at q0, find e τ
        same as TS component in GT }

{find IE component in Q equals to
fined g τ component}
{fined component qi assign to next
state input}
else {compute t of sa(a,t) in qi}
{occurs tick transition as
much as t}
if (t = 0) then {find IE equals
to "a" of SA(a,t) in GT}
{fine IE equals to fined IE
in Q, assign to next
state input}
endif
endif
endwhile

4. Applying example

Consider a object which consists of two diesel trains which share a common section of railway track. On the shared track, there is a diesel pump for refuelling the trains. To prevent the potentially disastrous situation of two trains simultaneously entering the shared track, two traffic lights have been installed. See the Fig.2.



(Fig.2) shared track system

(1) syntax definition

Mst = (V,T,S,E)
V = all variables
E = {wait, enter, connect, pump, pumped, disconnect, exit}
S = {ST, train1, train2, pump, travel, wait, shared track, pump connect, pump disconnect, off, connecting, pumping,}
T = { τ0, τ1, τ2, τ3, τ4, τ5, τ6, τ7, τ8}

τ0 = (initial, true, true, $\varnothing$)
τ1 = (travel, wait, wait, true, move())
τ2 = (wait, shared track, enter, true, move())
τ3 = (shared track, pump connect, connect, true, connect())
τ4 = (pump connect, pump disconnect, disconnect, true, disconnect())
τ5 = (pumpdisconnect, travel, exit, true, move())
τ6 = (off, connecting, true, pump)
τ7 = connecting, pumping, pump, true, SA(pumped,d(r,2r))
τ8 = (pumping, exit, pumped, true, disconnect)

(2) semantic definition

Gst = {GV, R, GS, GT, init}
GV = {TS, n, t, IE, TE, SA}
R = {power(TS) $\cup$ power(IE) $\cup$ power(n) $\cup$ SA $\cup$ t}
TS = {travel,wait,shared track, pump, connect, pump disconnect, off, connecting, pumping}
IE = {on(travel), on(wait), on(shared track), on(pump connect), on(pump disconnect), on(off), on(connecting), on(pumping), off(travel), off(wait),off(shared track), off(pump connect), off(pump disconnect), off(off), off(connecting), off(pumping), connect, pump, pumped, disconnect, sa(pumped, d(r,2r))}
SA = {(pumped,d(3,6)), (pumped,d(4,6) ,(pumped,d(5,6) , (pumped,d(6,6)}
n = { τ0, τ1, τ2, τ3, τ4, τ5, τ6, τ7, τ8}
t = {0,1,2, ... ∞}
GS = {TS:R(2TS), IE:R(2IE), TE:$\varnothing$, SA:R(sa), n:R(2n), t:t}
init = {TS:{travel, off}, IE:$\varnothing$, TE:$\varnothing$, SA:$\varnothing$, n: τ0, t:0}
GT = {g τ0, g τ1, g τ2, g τ3, g τ4, g τ5, g τ6, g τ7, g τ8}
g τ0 = (true, true, [TS:{travel, off}, IE : {on(travel), on(off)}, 0, 0)]
g τ1 = (wait, on(travel), [TS:TS$\cup${wait} - {travel}, IE : IE$\cup$ {on(wait), off(travel)}, 0, 1]
g τ2 = (enter,on(wait), [TS:TS$\cup${shared track} - {wait}, IE : IE $\cup$ {on(shared track), off(wait)},0,1]
g τ3 = (connect, on(shared track), [TS:TS$\cup${pump connect} - {shared track}, IE :IE{on(pump

connect), off(shared track)},0,1]

g τ4 = (disconnect, on(pump connect), [TS:TS∪{pump disconnect} - {pump connect}, IE : IE ∪ {on(pump disconnect), off(pump connect)},0,1]

g τ5 = (exit,on(pump disconnect), [TS:TS∪{travel}-{pump disconnect}, IE: IE∪{on(travel), off(pump disconnect)},0,1]

g τ6 = (connect, on(off), [TS:TS∪{connecting}-{off},IE : IE∪{on(connecting), off(off)},0,1]

g τ7 = (pump,on(connecting), [TS:TS∪{pumping} - {connecting}, IE:IE∪{on(pumping), off(connecting)},SA:SA∪{pumped, d(r,2r),0,3]

g τ8 = (pumped,on(pumping),[TS:TS∪{off} - {pumping}, IE:IE∪{disconnect,on(off),off(pumping)},0,1]

g τ9 = (true,scheduled_action(sa),[IE : IE∪{pumped},0,1]

(3) reachability graph

RGst = (Q,E)

Q = {q0,q1,q2,q3,q4,q5,q6,q7,q8,q9,q10,q11}

q0 = (TS:{travel,off}, IE:{on(travel),on(off)}, TE:∅, SA:∅)

q1 = (TS:{wait,off}, IE:{on(wait),off(travel)}, TE:∅,SA:∅)

q2 = (TS:{shared track,off},IE:{on(shared track),off(wait),TE:∅,SA:∅)

q3 = (TS:{pump connect,connecting}, IE:{on(pump connect),on(connecting),off(shared track),off(off),TE:∅,SA:∅)

q4 = (TS : {pump connect, pumping}, IE:{on(pumping),off(connecting)},TE:∅,SA:{(pumped),d(3,6)})

q5 = (TS : {pumpconnect,pumping}, IE:∅,TE:∅,SA:{(pumped),d(4,6)})

q6 = (TS:{pump connect,pumping} ,IE:∅,TE:∅,SA:{(pumped),d(5,6)})

q7 = (TS:{pump connect,pumping} ,IE:∅,TE:∅,SA:{(pumped),d(6,6)})

q8 = (TS:{pump connect,pumping},IE:{pumped},TE:∅,SA:∅)

q9 = (TS:{pump connect,off},IE:{on(off),off(pumping)},TE:∅,SA:∅)

q10 = (TS : {pump disconnect, off}, IE:{on(pumpdisconnect),off(pump connect)},TE:∅,SA:∅)

q11 = (TS : {travel, off}, IE : {on(travel),off(pump disconnect)},TE:∅,SA:∅)

E = {e0,e1,e2,e3,e4,e5,e6,e7,e8,e9,e10,e11}

e0 = (null, g τ0, q0)

e1 = (q0, g τ1, q1)

e2 = (q1, g τ2, q2)

e3 = (q2, g τ3 || g τ6, q3)

e4 = (q3, g τ7, q4)

e5 = (q4, g τ tick, q5)

e6 = (q5, g τ tick, q6)

e7 = (q6, g τ tick, q7)

e8 = (q7, g τ9, q8)

e9 = (q8, g τ8, q9)

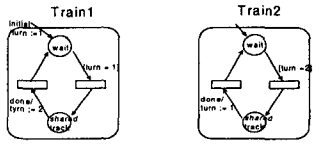
e10 = (q9, g τ4, q10)

e11= (q10, g τ5, q11)

(4) Property of real-time software

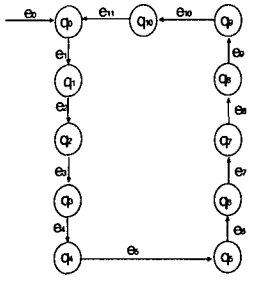
. Safety : Both trains may not simultaneously use the shared track. It is express in RTTL as follows. It sames as semaphore. See the figure 4.

true → □(TStrain1(travel ∨ wait) ∨ (TStrain2(wait ∨ travel))



(Fig.4) Concurrent Train with mutual exclusion

. Priority : If one of the trains has been allowed to use the shared track, then the currently waiting train must have first priority to use the shared track once the track is vacated. It is expressed as follows.



(Fig.3) Reachability graph of ST

$(n = \alpha 1 \wedge TStrain1(wait)) \rightarrow \bigcirc(n = \alpha 2 \; P\, n = \alpha 1)$

$(n = \alpha 2 \wedge TStrain2(wait)) \rightarrow \bigcirc(n = \alpha 1 \; P\, n = \alpha 2)$

. Response : No train should be allowed to remain in shared track longer than 2r ticks of the clock. It sames as dead line in transition.

$$(TStrain(pump\ connect) \wedge t = T) \rightarrow \Diamond\, (TSpump(pumping) \wedge t \leq T + 2r).$$

. Liveness : the clock is often infinitely. It same as the response property in that the clock flows.

5. Conclusion

In this paper, we present a specifying method for real-time software requirement. This framework can be used for TNPN(Timed Numerical Petri Net) as a easy communication means with a real-users. It is based on the RTTL(Real Time Temporal Logic) for correctness of the system. So, it will be lead to compensate for trade off relation(easy and correctness)in complex systems. And we present reachability graph algorithm for behavior analysis. So, we verified the properties of the real-time satisfied with the behaviors of TNPN.

References

[1] Billington,J., et al, "PROTEAN : A High-level Petri Net Tool for the Specification and Verification of Communication Protocols", IEEE Trans. SE, vol 14, No3, pp.301-16, March. (1988)

[2] G.M.Reed and A.W.Roscoe, "A timed model for communicating sequential processes", in Proceedings ICALP 86. LNCS 226. NewYork : Springer-Verlag, (1986)

[3] Gomaa, H., "A Software Design Method for Real-time Systems", CACM, vol 27, sep, pp.938-949, (1984)

[4] J.A.Stankovic, "Misconceptions about real-time computing", IEEE computer, vol .21, pp.10-19, Oct. (1988)

[5] J.S.Ostroff, "A logic for Real-Time Discrete Event Process", IEEE control systems magazine, pp95-102, June. (1990)

[6] J.S.Ostroff, "Temporal Logic for Real-time Systems", research studies press. (1989)

[7] Ko,G.I. and Kang,K.C., "A Formal Method of Specificaton and Verification of Real-Time Properties in Statechart", POSTECH/CS/SE-94-TR-8, Dept. of CS, Postech, (1994)

[8] Murata.T, "petri nets:properties, analysis and applications", Proceeding of the IEEE, vol.77, No.4, pp.541-580, April. (1989)

[9] N.Wirth, "Towards a discipline of real-time programming", CACM, vol.20, Aug. (1977)

[10] Peterson,J., "Petri Net Theory and the Modeling of Systems", Prentice-Hall, (1981)