시스템 오류 분석*

성순용

부산외국어대학교

An Analysis of System Fault

Soonyong Seong

Pusan University of Foreign Studies

E-mail : syseong@pufs.ac.kr

요 약

ACSR은 실시간 시스템을 기술하고 분석하기 위한 시간 프로세스 대수학으로서, 동기적으로 발생하는 timed action과 비동기적으로 발생하는 event의 기술을 지원한다. ACSR의 선택 연산에 확률 개념을 도입하여 확장한 대수학이 PACSR이다. 이 논문은 PACSR을 이용하여 일반적인 자원할당시스템에서 시스템 오류의 발생 및 그 오류로부터의 복구 과정을 기술하고자 한다. 시스템 오류 발생 과정이 오류 발생 확률과 복구 확률로부터 분석 가능함을 보였다.

ABSTRACT

ACSR is a timed process algebra for the specification and analysis of real-time systems, which supports synchronous timed actions and asynchronous instantaneous events. PACSR is an extended ACSR with the notion of probabilities in selection operation. Using PACSR, this paper represents a system fault occurrence and recovery from the fault in the general resource allocation system. The result shows that system fault occurrence can be analyzed from the fault occurrence probability and the recovery probability.

키워드

프로세스 대수학, PACSR, 오류 발생률, 오류 복구율

I. 서 론

ACSR(the Algebra of Communicating Shared Resources)[1]은 프로세스 동기화로 인한 시간 지연과 공유 자원 경합으로 인한 시간 지연을 동시에 기술할 수 있는 프로세스 대수학이다. 그 결과 다른 프로세스 대수학 보다 구체적이고 현실적으로 실시간 시스템을 표현할 수 있는 기법으로 평가된다.

PACSR은 ACSR에 확률적 선택 기능을 추가한 것으로서[2,3], 선택연산을 수행할 때 확률에 의한 다음 동작 선택이 가능하도록 처리하고 있다. 이 논문에서는 이와 같은 PACSR을 이용하여 시스템의 오류 발생 현상을 기술하고 분석할 수 있음을 보이고자 한다.

* 이 논문은 2004학년도 부산외국어대학교 학술연구조성비에 의하여 연구되었음

II. PACSR

PACSR은 시간과 자원을 소모하며 발생하는 timed action과 동기화를 위해 필요한, 순간적으로 발생하는 event, 이 두 가지 형태의 동작을 기본으로 하여 프로세스를 기술한다.

timed action은 순차적으로 재사용 가능한 자원과 우선 순위로 이루어진 순서쌍의 집합으로 한 단위 시간을 소모하는 동작을 표기한다. 이때 자원 집합 R의 각 자원은 한 단위씩으로 구성되며, 그 결과 어느 시점에 있어서 최대 한 번만 표기 가능하다. timed action A에 대해 $\rho(A)$는 A가 사용하는 자원의 집합을, $\pi_r(A)$는 A의 자원 r에 대한 우선순위 값을 나타낸다. {}는 자원은 전혀 사용하지 않으면서 한 단위 시간을 소모하는 timed action이다.

event는 그 실행에 시간을 소모하지 않으며 (a, p)와 같은 쌍으로 표기한다. 이때 a는 event의 라벨이고, p

는 그 실행 우선순위이다. event e에 대해 l(e)는 e의 라벨을, π(e)는 우선순위를 나타낸다. event 발생 시 값의 전송도 가능하다[4,5]. 예를 들어 (a, p)?x는 input event로서 전송 받은 값을 x에 주는 것을 표기하고, (a, p)!x는 output event로서 x의 값을 전송함을 표기한다. 특수 라벨 τ를 사용하여 같은 라벨을 갖는 input event와 output event가 동기화되어 동시에 실행될 때 그 결과의 event를 표시한다.

이 두 가지 실행 동작으로부터 프로세스의 기술이 가능하다. PACSR의 프로세스 P는 다음과 같은 문법으로 설명할 수 있다.

$$P ::= NIL \mid A : P \mid e\ .\ P \mid be \rightarrow P \mid P_1 + P_2 \mid P_1 \parallel P_2 \mid [P]_I \mid P\backslash F \mid C(x)$$

NIL은 아무런 실행 동작도 취하지 않는, 교착상태의 프로세스다. A : P는 한 단위 시간을 소모하는 timed action A를 실행하고 다음 프로세스 P로 진행하고, e . P는 event e를 실행하고 P로 진행한다.

be → P 는 부울 식 be가 참일 때만 P를 실행한다.

$P_1 + P_2$ 는 P_1과 P_2 중에 하나를 선택하여 실행한다. 이때 $p_1 \bullet P_1 + p_2 \bullet P_2$ ($p_1+p_2=1$)와 같이 확률을 부여할 수 있다. p_1과 p_2는 각각 프로세스 P_1과 프로세스 P_2를 선택할 확률을 나타낸다. 이 확률들에 의해 두 프로세스 중 하나의 동작을 시도하게 된다는 것으로, 이 확률적 선택이 우선순위에 의한 선점 관계보다 우선하는 것으로 정의한다. 확률이 명시되지 않은 선택연산의 경우에는 우선순위로 인한 선점관계가 존재할 경우 그 프로세스를 시도하고, 선점관계가 존재하지 않을 경우에는 작동 가능한 모든 프로세스 중에서 하나를 같은 확률로 선택하게 하는 것으로 정의한다.

$P_1 \parallel P_2$ 는 두 프로세스의 실행이 동시에 병렬로 진행된다. 즉 $P_1 \parallel P_2$ 에서 event는 시간을 소모하지 않으니까 각각 순서적으로 발생하든지, 또는 같은 라벨을 갖는 input, output event라면 동기적으로 동시 발생 가능하며, timed action은 서로 같은 자원을 사용하지 않는 한, 각각 한 단위의 시간을 소모하며 동시에 병렬 실행된다.

$[P]_I$는 P가 집합 I에 있는 자원을 독점적으로 사용한다. P\F는 P의 실행에 있어서 집합 F에 포함된 라벨을 갖는 event의 실행이 제한된다. 즉 a ∈F라면 (a, 1)?, (a, 1)!과 같은 event의 실행은 불가능하고, 두 event가 동기화된 (τ, 2) event만 실행 가능해져 프로세스 동기화를 제어할 때 유용하게 이용될 수 있다.

C(x)는 C(x) ≡ P와 같이 프로세스를 재귀적으로 정의할 때 사용할 수 있다.

우선순위를 고려하여 실행을 제어하고자 할 때 선점관계(preemption relation) " ≺"을 다음과 같이 정의한다. 이때 두 실행 동작 α, β에 대해 α ≺β라 함은 어느 시점에 α 또는 β를 선택해야 하는 경우 항상 β를 우선적으로 실행하도록 제어함을 의미한다.

정의) 선점 관계: 두 실행동작 α, β에 대해 다음 세 조건 중의 하나를 만족하면

α ≺β (β가 α를 선점한다)라고 한다.

1) α, β가 모두 timed action 이고
$(\rho(\beta)\subseteq\rho(\alpha)) \wedge (\forall r\in\rho(\alpha).\pi_r(\alpha)\le\pi_r(\beta)) \wedge (\exists r\in\rho(\beta).\pi_r(\alpha)<\pi_r(\beta))$
2) α, β가 모두 event이고
$\pi(\alpha) < \pi(\beta) \wedge l(\alpha) = l(\beta)$
3) α는 timed action, β는 event이고
$l(\beta) = \tau \wedge \pi(\beta) > 0$

지금까지 PACSR을 이용하여 프로세스를 기술하는 문법에 대해 살펴보았다. 이와 같이 기술된 프로세스는 간단한 대수학 법칙을 사용하여 보다 표준화된 형태로 전이 가능하며 이와 같은 전이를 통하여 시스템 분석이 가능해진다[1].

III. 시스템 오류 기술 및 분석

앞에서 기술한 PACSR을 이용하여 시스템의 오류발생 현상을 기술하고 이를 분석하고자 한다. 앞으로 분석할 대상 시스템을 다음과 같이 정의한다.

시스템에는 n 개의 동종 프로세스(homogeneous process)와 서로 다른 r 종류의 재사용 가능 자원이 한 단위씩 존재한다. 시스템 전체에서 프로세스들의 자원 요구 및 반환 연산이 반복 수행된다. 이때 연산을 수행하는 프로세스는 자신이 현재 가지고 있지 않은 자원들을 한 번에 하나씩 요구하며, 자신이 갖고 있는 자원들을 한 번에 하나씩 반환한다. 요구연산에서 요구하는 자원의 선택은 가지고 있지 않는 자원 모두에게 같은 확률로 주어지며, 반환연산에서도 반환하는 자원의 선택은 갖고 있는 자원 모두에게 같은 확률로 주어진다. 각 프로세스의 자원에 대한 요구 연산은 현재 보유한 자원의 수가 r 보다 적을 때 λ의 비율로, 반환 연산은 보유한 자원의 수가 하나 보다 많을 때 μ의 비율로 발생한다.

각 자원 r_i는 현재 임의의 프로세스에게 할당된 상태 R_i'이거나 할당이 가능한 상태 R_i로 존재한다. 또한 각 상태 R_i와 R_i'는 결함이 발생하면 각각 상태 $\underline{R_i}$, $\underline{R_i'}$로 전이한다.

자원 r_i의 결함 발생 비율은 현재 결함이 없는 경우에 매 단위시간마다 p_i의 확률로 발생한다고 가정한다. 그리고 일단 발생한 결함은 매 단위시간마다 q_i의 확률로 치료가능하다고 가정한다. 이와 같은 시스템에서 자원 r_i의 상태는 다음과 같이 진행한다.

$R_i = (r_i, 1)?.R_i' + (p_i \bullet \{\} : \underline{R_i} + (1-p_i) \bullet \{\} : R_i)$
$R_i' = (r_i', 1)?.R_i + (p_i \bullet (\underline{r_i}, 1)!.\{\} : \underline{R_i'} + (1-p_i) \bullet \{\} : R_i')$
$\underline{R_i} = q_i \bullet \{\} : R_i + (1-q_i) \bullet \{\} : \underline{R_i}$
$\underline{R_i'} = q_i \bullet (\underline{r_i'}, 1)!.\{\} : R_i' + (1-q_i) \bullet \{\} : \underline{R_i'}$

상태 R_i에서 요구연산을 나타내는 $(r_i, 1)!$ event와 동기화되면 R_i'로 전이한다. 요구연산 event가 없을 경우엔 결함이 발생하느냐의 여부에 따라 R_i, 또는 $\underline{R_i}$로 이

동한다. R_i'에서도 반환연산 event $(r_i', 1)!$과 동기화되어 R_i로 전이되어진다. 역시 결함 여부에 따라 R_i', 또는 $\underline{R_i'}$로 전이하게 된다. R_i'에서 $\underline{R_i'}$로 전이하게 되는 경우 이 자원을 소유한 프로세스에게 결함이 발생했다는 사실을 $(r_i, 1)!$ event로 알려준다.

상태 $\underline{R_i}$는 결함이 존재하는 경우이므로 결함이 복구될 때까지 같은 상태를 유지하며, 이 상태에서는 할당이 이루어지지 못한다. 그러나 $\underline{R_i'}$는 이미 할당된 상태에서 결함이 발생한 것이므로 계속 같은 프로세스에게 할당된 상태를 유지하며, 결함으로부터 회복될 때 해당 프로세스에게 $(\underline{r_i'}, 1)!$ event로서 회복사실을 알려준다.

자원의 요구 연산 시 해당 자원이 다른 프로세스에 이미 할당된 상태이면 그 자원이 반환될 때까지 기다려야 한다. 또한 그 자원에 결함이 발생했을 때도 그 결함이 고쳐질 때까지 기다려야 한다. 이와 같이 대기하는 상태에 있는 프로세스를 보류 프로세스라 하고 보류 프로세스가 아닌 프로세스를 실행 프로세스라 한다. 실행 프로세스만이 새로운 자원을 하나 더 요구하거나 자신이 보유한 자원 중의 하나를 반환할 수 있다. 보류 프로세스는 기다리는 자원이 자신에게 할당되어야만 실행 프로세스로 전환된다.

자원이 할당된 상태에서 결함이 발생했을 때는 그 결함으로부터 회복될 때까지 그 자원에 대한 소유권을 유지한다. 즉 결함을 먼저 고친 후 자원을 이용한 나머지 작업을 모두 마친 뒤 반납한다고 가정한다.

시스템의 자원 할당 상태는 즉시 할당 상태(expedient state)로 가정한다. 즉 요구한 자원이 현재 할당된 상태가 아니고 결함상태가 아니라면 바로 할당받는다.

자원 r_i에 대한 요구연산은 아래와 같이 기술될 수 있다. 현재 시스템에는 자원이 r_i 하나만 존재한다는 가정 하에 프로세스 P가 자원 r_i를 요구하고 반납하는 과정을 기술한다.

$$P = (1-\lambda) \bullet \{\} : P + \lambda \bullet P^i$$
$$P^i = (r_i, 1)!.\{(r_i,1)\} : P' + \{\} : P^i$$
$$P' = (\underline{r_i}, 1)?.\underline{P'} + ((1-\mu) \bullet \{(r_i,1)\} : P' + \mu \bullet (r_i', 1)!.\{\} : P)$$
$$\underline{P'} = (\underline{r_i'}, 1)?.P' + \{(\underline{r_i}, 1)\} : \underline{P'}$$

상태 P는 요구연산이 발생하면 P^i로 전이한다. P^i에서는 현재 자원이 상태 R_i에 있을 때에만 할당 가능하며, 이때 $(r_i, 1)?$ event와 동기화되어 자원을 할당받고 P'으로 전이된다. P'에서는 자원에 결함이 생겼다는 event $(r_i, 1)!$와 동기화되면 $\underline{P'}$로 전이하고 이후 결함을 복구한 $(\underline{r_i'}, 1)!$ event를 받으면 다시 P'로 회복된다. timed action $\{(\underline{r_i}, 1)\}$은 자원 r_i가 결함있는 상태에서 한 단위시간을 소모함을 나타낸다. 상태 P'에서 자원을 반납할 때는 $(r_i', 1)!$ event를 이용해 R_i' 상태의 자원에게 알려준다.

지금까지 기술한 시스템에서는 자원을 할당받은 상태에서 해당 자원에 결함이 발생하면 그 자원을 소유한 채 결함으로부터 회복될 때까지 기다리도록 하였다.

그러나 시스템에서 오류가 발생하면 그 자원을 소유한 프로세스를 중단시키고, 중단된 프로세스가 보유한 모든 자원을 반납하게 하는 처리과정도 가능하다. 이러한 기법을 적용한 경우의 자원과 프로세스의 상태전이를 도식화한 것이 각각 <그림 1>과 <그림 2>다.

그림에서 상태의 진행방향을 나타내는 화살표에는 각각 라벨이 표기되어 있다. 이 라벨에서 수치 p_p 등은 선택 확률을 나타낸다. 반면에 $(r_i, 1)?$와 같은 event와 {}와 같은 timed action은 각각 그 동작을 수행하고 상태가 전이됨을 나타낸다. 다만 event의 경우에는 그 상대 event와 동기화되어야만 전이가 일어난다.

<그림 1>에서는 $\underline{R_i'}$ 상태가 존재하지 않는다. <그림 2>에서 상태 $\underline{P'}$는 상태 P로 즉시 전이되는, 일시적으로 거쳐 가는 상태다. 여러 개의 자원이 존재하는 일반 시스템에서, 상태 $\underline{P'}$에서는 오류가 발생한 자원을 포함하여 프로세스가 보유한 모든 자원을 반납하게 된다.

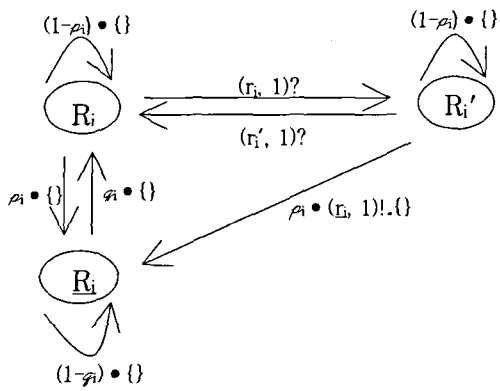

그림 1. 자원의 결함 발생 전이

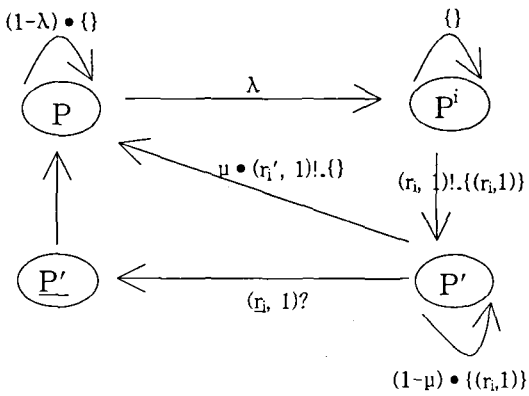

그림 2. 프로세스의 자원할당 상태 전이

이러한 오류 발생 시스템을 PACSR로 표기하면 아래와 같다. 문제를 단순화하기 위해 두 종류의 자원 r_1과 r_2만 존재하는 경우를 다루었다. r 개의 자원을 갖는 시스템으로의 확장은 같은 방식으로 기술 가능하다.

오류가 발생한 자원이 회복될 때까지 그 자원을 반납하지 않는 시스템에 대한 PACSR은 [6]에서 기술한

바 있다.

전체 시스템은 다음과 같이 자원 r_1과 r_2를 경합하는 n 개의 프로세스 P_i로 구성된다.

$$SYSTEM = [(P_1 \,||\, \ldots \,||\, P_i \,||\, \ldots \,||\, P_n \,||\, R_1 \,||\, R_2) \setminus \{r_1, r_2, r_1', r_2', \underline{r_1 r_2}, \underline{r_1 r_2'}\}]_{\{r1,r2\}}$$

$$P_i = P_{i\{\}}$$

$$P_{i\{\}} = (1-\lambda) \bullet \{\} : P_{i\{\}} + \lambda/2 \bullet P_{i\{\}}^{(1)} + \lambda/2 \bullet P_{i\{\}}^{(2)}$$

$$P_{i\{\}}^{(m)} = (r_m, 1)!.\{(r_n,1)\} : P_{i\{m\}} + \{\} : P_{i\{\}}^{(m)}$$

$$P_{i\{m\}} = (\underline{r_m}, 1)?.P_{i\{\}} + ((1-\lambda-\mu) \bullet \{(r_n,1)\} : P_{i\{m\}} + \lambda \bullet P_{i\{m\}}^{(n)} + \mu \bullet (r_m', 1)!.\{\} : P_{i\{\}})$$

$$P_{i\{m\}}^{(n)} = (r_n, 1)!.\{(r_m,1), (r_n,1)\} : P_{i\{m,n\}} + \{(r_m,1)\} : P_{i\{m\}}^{(n)}$$

$$P_{i\{m,n\}} = (\underline{r_m}, 1)?.P_{i\{\}} + (\underline{r_n}, 1)?.P_{i\{\}} + ((1-\mu) \bullet \{(r_m,1), (r_n,1)\} : P_{i\{m,n\}} + \mu/2 \bullet (r_m', 1)!.\{(r_n,1)\} : P_{i\{n\}} + \mu/2 \bullet (r_n', 1)!.\{(r_m,1)\} : P_{i\{m\}})$$

$$R_m = (r_m, 1)?.R_m' + (p_m \bullet \{\} : \underline{R_m} + (1-p_m) \bullet \{\} : R_m)$$

$$R_m' = (r_m', 1)?.R_m + (p_m \bullet (\underline{r_m}, 1)!.\{\} : \underline{R_m} + (1-p_m) \bullet \{\} : R_m')$$

$$\underline{R_m} = q_m \bullet \{\} : R_m + (1-q_m) \bullet \{\} : \underline{R_m}$$

위의 SYSTEM 기술에서 집합 A에 대해 P_{iA}는 $m \in A$인 경우 프로세스 P_i가 자원 R_m을 보유하고 있는 상태를 나타낸다. $P_{iA}^{(m)}$은 프로세스 P_i가 $m \notin A$인 자원 R_m을 요구하고 있는 상태이다.

우선 각 프로세스의 초기상태는 아무 자원도 보유하지 않은 상태 $P_{i\{\}}$에서 시작한다. $P_{i\{\}}$는 현재 자원을 전혀 보유하고 있지 않은 상태이므로 반환연산은 발생하지 않으며, λ의 확률로 자원을 요구하거나, 1-λ의 확률로 자원을 보유하지 않은 상태에서 그대로 한 단위시간만큼 작업을 수행한다. 요구 자원은 R_1, R_2에 대해 같은 확률을 가지므로 각각 1/2의 확률로 선택하게 된다. 위의 기술에서 m=0이면 n=1이고, m=1이면 n=0이다.

$P_{i\{\}}^{(m)}$은 자원 R_m을 사용 가능하여 얻게 되면 timed action $\{(r_m,1)\}$을 한 단위시간만큼 수행한 후 $P_{i\{m\}}$으로 진행되고, 자원 r_m을 얻지 못하면 얻을 때까지 그 상태에서 계속 기다린다.

$P_{i\{m\}}$은 자원 r_m을 보유하고 있는 상태이므로 다른 자원 r_n을 요구할 수도 있고, 보유한 자원 r_m을 반환할 수도 있다. 그 결과 λ의 확률로 $P_{i\{m\}}^{(n)}$으로 진행하고, μ의 확률로 $P_{i\{\}}$을 수행하며, 나머지 1-λ-μ의 확률로 보유한 자원을 갖고 timed action $\{(r_m,1)\}$을 한 단위만큼 수행한 후 같은 과정을 반복한다. 이때 r_m에 오류가 발생하면 그 자원을 반납하고 초기상태 $P_{i\{\}}$로 돌아간다.

$P_{i\{m\}}^{(n)}$은 자원 r_n도 얻어 timed action $\{(r_m,1), (r_n,1)\}$을 수행할 수도 있고, r_n을 얻을 때까지 r_m을 보유한 상태로 대기 상태에 머문다.

$P_{i\{m,n\}}$은 모든 자원을 보유하고 있는 프로세스 P_i가 더 이상의 요구연산은 수행하지 못하고, μ의 확률로 r_m 또는 r_n 중의 하나를 반환한다. 그리고 1-μ의 확률로 자원 r_m, r_n을 보유한 timed action을 한 단위시간만큼 수행한다. 여기서 하나 이상의 자원에 결함이 생기는 경우에는 그 자원을 포함하여 모든 자원을 반납하고 중단됨으로써 초기상태 $P_{i\{\}}$로 돌아가게 된다.

지금까지 전형적인 자원 할당 시스템에서 오류가 발생하는 상황을 PACSR로 기술해 보았다.

이를 분석하여 각 프로세스 P_i가 $P_{iA}^{(m)}$ 상태에 머무는 비율을 계산하면 자원할당을 위해 대기하는 시간 비율을 계산할 수 있다. 또한 P_{iA}의 집합 A의 크기를 이 상태의 시간만큼 누적하여 평균하면 각 프로세스가 보유한 자원의 평균이 계산된다. 우리의 관심 분야인 교착상태에 대해서도 쉽게 분석 가능하다. SYSTEM이 진행하다가 임의의 서로 다른 두 프로세스 P_i와 P_j에 대해 $P_{i\{0\}}^{(1)} \,||\, P_{j\{1\}}^{(0)}$ 인 상태가 발생하면 교착상태가 된다. 이때 p_m과 q_m의 값을 변화시켜 주면 오류가 발생하는 비율이 미치는 영향에 대해서도 분석가능하다.

IV. 결 론

실시간 시스템을 기술하고 분석하는 도구로서 시간 개념을 도입한 프로세스 대수학이 많이 이용되어지고 있는 바, 그 중에서도 ACSR은 프로세스 동기화로 인한 시간 지연 뿐만 아니라 공유 자원 결합에 따른 시간 지연도 함께 표현할 수 있는 보다 구체적이고 현실성 있는 계산 모델이다. 그리고 ACSR에 확률 개념을 도입한 것이 PACSR이다.

본 논문에서는 PACSR을 이용하여 일반적인 자원 할당 시스템에서 오류가 발생하고 이로부터 복구하는 상황을 표기하여 이때 발생하는 교착상태를 비롯한 제반 기술 및 분석이 효율적으로 이루어질 수 있음을 보였다.

참고문헌

[1] I. Lee, P. Bremond-Gregoire, and R. Gerber, " A Process Algebraic Approach to the Specification and Analysis of Resource-Bound Real-Time Systems" , Proceedings of the IEEE, pp 158-171, Jan. 1994

[2] 성순용, "확률적 ACSR", 외대논총, 2003년 2월

[3] 성순용, 'PACSR : 확률적 ACSR", 한국해양정보통신학회 2005 춘계종합학술대회 9권, pp 720-723

[4] Jin-Young Choi, Insup Lee and Inhye Kang, "Timing Analysis of Superscalar Processor Programs Using ACSR, " 11th IEEE Workshop on real-time operating systems and software, May 1994

[5] Jin-Young Choi, Insup Lee and Hong-Liang Xie, "The Specification and Schedulability Analysis of Real-Time Systems using ACSR," Proc. 16th IEEE Real-Time Systems Symposium, 1995

[6] 성순용, "PACSR을 이용한 자원결함 발생 분석", 외대논총, 27권, 2003년 8월