멀티미디어 협동 작업 환경에서의 오류 감지 및 복구 시스템

고 응 남† · 황 대 준††

요 약

멀티미디어는 현재 다양한 실세계의 분야에 적용되고 있다. 특히 멀티미디어 시스템을 위한 컴퓨터 협동 작업 환경에 대한 관심이 집차로 고조되고 있다. 그러나, 이러한 현재의 방향에도 불구하고 컴퓨터 협동 작업 환경에서의 결함 허용에 대한 연전이 충분하게 이루어지지 못하고 있다. 본 논문에서는 EDR_MCSCW를 제안한다. 이 시스템은 소프트웨어 기법을 사용함으로써 두레처럼 멀티미디어 작업 환경에서의 소프트웨어 오류를 감시하고 복구하는데 적합한 시스템이다. 두레는 컴퓨터 협동 작업 환경을 위한 멀티미디어 응용에 관한 개발을 지원하기 위한 프레임웍이다. 오류가 발생했을 때 EDR_MCSCW는 윈도우에 있는 API함수에 있는 훅 방법을 사용함으로써 오류를 감시한다. 오류가 발견되면, 스택을 사용함으로써 멀티미디어 공동 작업 환경 상에서 복구하기 위하여 도미노 효과를 제거하는 기능을 가지고 있는 검사점과 복구 알고리즘을 제안한다.

An Error Detection and Recovery System based on Multimedia Computer Supported Cooperative Work

Eung-Nam Ko† · Dae-Joon Hwang††

ABSTRACT

Multimedia is now applied to various real world areas. In particular, the focus on multimedia system and CSCW (Computer Supported Cooperative Work) has increased. In spite of this current trend, however, the study of fault tolerance for CSCW has not yet fully progressed. We propose EDR_MCSCW. It is a system that is suitable for detecting and recovering software error based on multimedia computer supported cooperative work as DOORAE by using software techniques. DOORAE is a framework for supporting development on multimedia applications for computer-based collaborative works. When an error occurs, EDR_MCSCW detects an error by using hooking methods in MS-Windows API (Application Program Interface) function . If an error is found, we present a checkpointing and recovery algorithm which has the removal function of the domino-effect for recovering multimedia and CSCW by using stack.

1. 서 론

멀티미디어 응용으로는 사무 자동화, 가정용 응용, 과학 기술 분야, 도소매업, 교육/금융/보건 응용, 협동 작업 등을 들 수 있다. 이러한 멀티미디어 응용은 모두 멀티미디어 단말기, 멀티미디어 정보 통신망, 멀티미디어 데이터베이스, 멀티미디어 정보처리 기술을 분산환경에서 복합시킨 것이다[1]. 즉, 지난 10여 년 동안 멀티미디어 기술과 컴퓨터 네트워크 기술이 급속하게 발전하였고 이 기술 들의 결합으로 최근에 있던 컴퓨터 협동 작업 환경 (CSCW : Computer Supported Co-

† 정 회 원 : 신경대학 컴퓨터지열 교수
†† 정 회 원 : 성균관대학교 정보공학과 교수
논문접수 2000년 2월 16일, 심사완료 2000년 4월 27일

operative Work) 분야의 급속한 발전과 더불어 원격 회의, 원격 교육, 원격 자문, 공동 저작 등에 대한 요구가 날로 커지고 있다[2-5].

이러한 현재의 방향에도 불구하고 상호작용(interactive)하는 멀티미디어 환경의 구성 요소에서는 그 시스템에서 계산될 수 있는 결함 허용 응용에서조차도 충분한 신뢰성(reliability)을 항상 보장하는 것은 아니다[6]. 시스템의 신뢰성을 증가시키는 방법에는 임격한 테스팅이나 결함 허용을 통한 방법 등이 있다[7, 8].

따라서, 본 연구에서는 멀티미디어 공동 작업 환경에서 응용 소프트웨어와 미디어 등의 프로세스(process) 요소들에 대한 오류를 미리 감지하여 알려주고 복구할 수 있는 시스템을 제안한다. 오류 감지 시에 훅킹(*hooking*) 방법을 이용한다. 오류 복구는 오류 중에서 소프트웨어 오류인 경우에만 복구할 수 있다. 오류 복구는 먼저 검사점 설정(check point)을 한 후에 오류 감지 발생하면 그 검사점 까지 되돌아가서 재 수행하는 방식을 제안한다. 이 때 프로세스들 간에는 메시지(*message*) 교환이 일어나는 경우도 발생한다 메시지 교환 시스템에서는 도미노 효과(domino effect)가 발생할 수도 있다 프로세서들 간의 정보 교환과 복구점이 조화되지 않으면 프로세스 사이에 계속적인 롤백(rollback) 전달의 사태가 일어나는 것을 도미노 효과라 한다[9]. 본 논문에서는 도미노 효과를 제거할 수 있는 방법을 제안한다.

본 논문의 구성은 2에서 관련 연구를 기술하고, 3에서는 제안하는 오류 감지 및 복구 시스템에 대해서 기술하고, 4에서는 시스템 평가, 5에서는 결론을 기술한다

2. 관련 연구

본 절에서는 기존의 결함 허용 기법 및 한계점에 대해서 기술한다.

2.1 기존 결함 허용 기법

결함 허용 시스템이란 하드웨어 오동작, 소프트웨어 오류 또는 정보 오염이 일어날지라도 주어진 임무를 올바르게 수행할 수 있는 시스템을 말한다[10]. 결함 허용성을 부여하는 방법에 따라 3가지로 나눌 수 있다. 첫째, 소프트웨어 기법은 운영체제에 의해 이루어지는 기법으로 소프트웨어에 의한 오버헤드로 시스템 성능 하락에 대한 희생이 따른다. 둘째, 하드웨어 기법은 하드웨어 다중화를 통해 결함 탐지 및 복구가 수행되는 기법이다. 셋째, 혼합 기법은 하드웨어로 결함을 탐지하고 소프트웨어로 결함 복구를 하게함으로써 소프트웨어 오버헤드와 하드웨어 비용을 줄일 수 있는 장점이 있다[11]. 소프트웨어 결함허용 기법은 소프트웨어 모듈의 중복이나 재수행(rollback and retry),또는 이 두가지 방식의 혼용에 기초를 두고 있다. 검사점(*Check pointing*)은 소프트웨어 실행 중에 검사지점을 설정하여 오류가 발생했는지를 검사하여 이상이 없으면 계속 수행하고 이상이 감지되면 그 이전의 검사지점으로 되돌아가 재수행 하는 방식이다. 복구 블록(Recovery block)은 재수행 (rollback and retry)에 근거한 기법으로 검사지점에서 오류가 감지되면 지정된 이전 시점으로 되돌아가 같은 기능을 가진 다른 소프트웨어 모듈을 실행하는 방식으로 단일 프로세스내에 적용될 수 있다. 분산 복구 블록(Distributed recovery block)은 복구 블록을 분산 환경으로 확장 적용하여 하드웨어 결함과 소프트웨어 결함을 동일한 방법으로 극복할 수 있도록 한 기법이다 N-버전 프로그래밍(version programming)은 하드웨어 결함허용 기법 중 TMR(Triple Modular Redundancy)과 유사한 기법으로 N개의 독립적인 소프트웨어 모듈이 수행한 결과를 비교하여 다수의 동일한 결과를 채택하는 기법이다 TMR은 세 개의 동일한 모듈에서 실행 결과를 받아 이 들의 값을 서로 비교하여 세 개 중에 두 개 이상의 결과가 같으면 이를 출력으로 발생시키는 보팅(voting) 작동에 기초한다[12]. 이외에도 목적한 응용 분야에 적합하도록 설계된 여러 가지 형태의 소프트웨어 결함허용 기법이 있다[13, 14].

2.2 기존 결함 허용 기법의 한계점

소프트웨어 고장은 검사점(checkpoint) 설정, 재구성(reconfiguration) 및 복구 과정으로 처리되며, 재 수행량을 줄이며 log 정보 분실에서도 일치된 상태를 유지시켜야 한다[15, 16] 검사점은 다른 노드 설정과 무관하게 설정하므로 도미노 효과를 유발한다. 특히, 롤백(rollback) 정보 전송시 무한정 그 실행을 반복하는 순환적 재시작 현상(livelock effect)이 발생할 수 있다[17].

기존의 분산 시스템에서 사용되는 검사점 설정 방법은 동기화된 방법과 비동기화된 검사점 설정 방법으로

구분된다. 기존의 동기적 또는 비동기적 검사점 설정 기법에는 정기적(periodic) 검사점 알고리즘이 기본적으로 사용된다. 이 방법은 각 프로세스가 일정한 시간 간격마다 체크포인트하는 방식이다. 기존의 동기적(synchronous) 검사점 설정 기법들은 각 사이트에서 검사점을 설정하는 시점에 다른 사이트의 검사점들과 완전 동기화를 이루도록 검사점을 설정한다. 따라서 고장이 발생한 경우에 빠른 시간에 일관성이 보장된 검사점을 찾아서 복구를 쉽게 할 수 있다. 그러나 이러한 방법은 검사점을 설정하는 시간이 많이 걸리고, 분산 트랜잭션들의 수행에 간섭 현상을 초래하게 되는 단점이 있다. 비동기적인(asynchronous) 검사점 기법들은 검사점을 각 사이트에서 독립적으로 설정함으로써 쉽게 검사점을 설정할 수 있지만 고장이 발생하는 경우 일관성이 보장된 검사점을 찾아서 복구하는 작업을 어렵게 한다. 이러한 기법들은 최악의 경우 일관성이 보장된 검사점을 찾기 위해 시스템 초기 시점까지 복구 작업을 하는 도미노 효과를 발생하게 된다[18-20].

3. 오류 감지 및 복구 시스템

본 절에서는 멀티미디어 협동 작업 환경에서의 응용 소프트웨어에 대한 결함을 발견 및 복구할 수 있는 시스템인 EDR_MCSCW(An Error Detection and Recovery System based on Multimedia Computer Supported Cooperative Work)를 기술한다.

결함 감지시 훅(hook) 기술을 이용하고 오류 복구시 검사점까지 복구하는 롤백 알고리즘의 변형된 알고리즘을 제안한다. 이 때 도미노 효과도 제거한다.

3.1 가정 및 정의

EDR_MCSCW에 대한 설명과 분석을 위해서 필요한 정의 및 표기는 다음과 같다

[정의 1]

멀티미디어 공동 작업 환경을 EDR_MCSCW라고 표시하면 EDR_MCSCW = < P, L, M > 이다.

여기서 $P = \{ p_1, p_2, \ldots, p_n \}$이며 프로세스(process)들의 유한 집합(finite set)이다. $L \subseteq P^2$이며 채널(channel)들의 부분 집합이다.

$L = \{ < p_i, p_j > \mid p_i$: 메시지 보내는 프로세스,
p_j : 메시지 받는 프로세스$\}$

M은 메시지들의 유한 집합이다.

$M = \{ m < p_i, p > \mid p_i$: 메시지 보내는 프로세스,
p_j : 메시지 받는 프로세스$\}$

[정의 2]

세션이 진행 중일 때 실행되는 프로세스를 P라고 정의하기로 한다 즉, 오류 감지 및 복구 대상이 되는 미디어, 미디어 인스턴스 및 응용 프로그램들의 집합은 다음과 같다.

모든 프로세스들의 집합인 P를 공집합이 아닌 임의의 집합이라고 하면, 집합 P의 분할(partitions) πP는 다음과 같은 집합이다

πP = {GSM, LSM, PSM, DM, EDR,
VD, VDI, AD, ADI, WB, AP, A}이고
P = GSM ∪ LSM ∪ PSM ∪ DM ∪ EDR ∪
VD ∪ VDI ∪ AD ∪ ADI ∪ WB ∪ AP ∪ A이다.

GSM = $\{P_{gsm} \mid P_{gsm} \in P, gsm \in N, N$: 자연수$\}$
LSM = $\{P_{lsm} \mid P_{lsm} \in P, lsm \in N, N$: 자연수$\}$
PSM = $\{P_{psm} \mid P_{psm} \in P, psm \in N, N$ 자연수$\}$
DM = $\{P_{dm} \mid P_{dm} \in P, dm \in N, N$: 자연수$\}$
EDR = $\{P_{edr} \mid P_{edr} \in P, edr \in N, N$: 자연수$\}$
VD = $\{P_{vd} \mid P_{vd} \in P, vd \in N, N$: 자연수$\}$
VDI = $\{P_{vdi} \mid P_{vdi} \in P, vdi \in N, N$: 자연수$\}$
AD = $\{P_{ad} \mid P_{ad} \in P, ad \in N, N$: 자연수$\}$
ADI = $\{P_{adi} \mid P_{adi} \in P, adi \in N, N$ 자연수$\}$
WB = $\{P_{wb} \mid P_{wb} \in P, wb \in N, N$: 자연수$\}$
AP = $\{P_{ap} \mid P_{ap} \in P, ap \in N, N$ 자연수$\}$
A = $\{P_a \mid P_a \in P, a \in N, N$: 자연수$\}$이고

$GSM \cap LSM \cap PSM \cap DM \cap EDR \cap VD \cap VDI \cap AD \cap ADI \cap WB \cap AP \cap A = \phi$ 이다

여기서

GSM : 전체 세션 관리자들의 집합
LSM : 지역 세션 관리자들의 집합
PSM : 참여자 관리자들의 집합
DM : 데몬들의 집합
EDR : 오류 감지 및 복구 시스템들의 집합
VD : 비디오 서버 들의 집합,
VDI : 비디오 서버 인스턴스 들의 집합.

AD : 오디오 서버 들의 집합,
ADI : 오디오서버 인스턴스 들의 집합,
WB : 화이트보드 들의 집합,
AP : 응용 공유 들의 집합,
A : 응용 프로그램 들의 집합이라고 정의 한다.

[정의 3]

본 논문에서 오류 감지 및 복구 시스템에 관련되어 있는 에이전트들의 집합은 다음과 같다.

세션이 개설되어 있을 때 여러 플랫폼(platform) 중에서 i번째 플랫폼에 실행하는 오류 감지 및 복구 프로세스들을 EDR_i라고 정의한다. 정의된 오류 감지 및 오류 복구에이전트들 EDR_i, ED_i 및 ER_i 사이의 관계는 다음과 같다. 분할 $\pi EDR_i = \{ ED_i, ER_i \}$이고 $EDR_i = ED_i \cup ER_i (i \in N)$이다.

[정의 4]

오류 감지 및 복구 에이전트들의 집합인 EDR을 공집합이 아닌 임의의 집합이라고 하면, 집합 EDR의 분할(partitions) πEDR은 다음과 같은 집합이다.

$$\pi EDR = \{ EDR_1, EDR_2, .. EDR_i .. EDR_k \} \quad (i, k \in N)$$

여기서 EDR은 다음을 만족한다.

(1) i = 1, ..., k 에 대하여 EDR_i는 공집합이 아닌 집합 EDR의 부분 집합이다.
(2) $EDR = EDR_1 \cup EDR_2 \cup ... \cup EDR_k$ $(k \in N)$
(3) EDR_i들 사이에서는 서로소이다. 즉, $i \neq k$이면 $EDR_i \cap EDR_k = \emptyset$ 이다.

같은 방법으로 ED를 오류 감지 에이전트 들의 집합이라고 정의하면 집합 ED와 분할 πED는 다음과 같다.

즉, $\pi ED = \{ED_1, ED_2, ... ED_i ... ED_k\}$ (i, k ∈ N)이고
$ED = ED_1 \cup ED_2 \cup ... \cup ED_k (k \in N)$이다.

같은 방법으로 ER를 오류 복구 에이전트 들의 집합이라고 정의하면 집합 ER과 분할 πER은 다음과 같다

즉, $\pi ER = \{ ER_1, ER_2, .. ER_i .. ER_k \}$ (i, k ∈ N) 이고
$ER = ER_1 \cup ER_2 \cup .. \cup ER_k (k \in N)$이다.

[정의 5]

$E_i(j)$는 프로세스 p_i가 실행하고 있을 때 그 프로세스 p_i에서 j번째 발견되는 오류(error)들의 집합으로 정의한다. 즉, $E_i(j) = \{ e_i(j) \mid i \in N, j \in N \}$이다.

$S_i(j)$는 프로세서 p_i가 실행하고 있을 때 그 프로세스 p_i에서 j번째 메시지를 받았을 때 스택(stack)에 저장되는 항목(item) 들의 집합으로 정의한다. 즉, $S(j) = \{ s_i(j) \mid i \in N, j \in N \}$이다

$K_i(j)$는 프로세스 p_i가 실행하고 있을 때 그 프로세스 p_i에서 j번째 발견되는 오류(error)를 복구하는 모듈들의 집합으로 정의한다. 즉, $K_i(j) = \{ k_i(j) \mid i \in N, j \in N \}$이다

$B_i(j)$는 프로세스 p_i가 실행하고 있을 때 그 프로세스 p_i에서 j번째 발견되는 오류(error)의 정보를 갖고 있는 프로세서 데이터베이스(PDB : Process Data Base)에서의 리턴 코드 값을 집합으로 정의한다. 즉, $B_i(j) = \{ b_i(j) \mid i \in N, j \in N \}$이다.

[정의 6]

$C_i(j)$는 프로세스 p_i가 실행하고 있을 때 그 프로세스 p_i에서 j번째의 검사점(checkpoint) 들의 집합으로 정의한다 즉, $C_i(j) = \{ c_i(j) \mid i \in N, j \in N \}$이다.

$R_i(j)$는 프로세스 p_i가 실행하고 있을 때 그 프로세스 p_i에서 j번째의 검사점(checkpoint)과 j+1번째의 검사점사이에 있는 r에서 오류가 발생하여 복구 시킬 때 $C_i(j)$에서부터 롤백(rollback)하여 재시작해야 한다

즉, $R_i(j) = \{ r_i(j) \mid i \in N, j \in N, j \leq r < j+1 \}$이다. 단, $r_i(j)$는 프로세서 p_i에서 $c_i(j)$로 롤백할 때만 존재한다. $c_i(j)$가 활성화(active)란 것은 $r_i(j)$가 발생하기 직전까지의 상태를 말하며 $c_i(j+1)$이 생성되면 $c_i(j)$는 소멸된다.

[정의 7]

검사점(checkpoint)의 우선 순위는 다음과 같이 정의할 수 있다.

프로세서 p_i, 프로세서 p_j가 각각 활성화 검사점 $c_i(s)$, $c_j(t)$라고 하면 메시지 m을 보내는 임의의 채널 $< p_i, p_j >$에서 활성화 검사점 $c_i(s)$, $c_j(t)$ 각각이 메시지 주고받는 사건(event)보다 앞서서 발생한다고 하면 검사점 사이의 관계는 다음과 같이 표시한다.

$< c_i(s), c_j(t) >$이다

[정의 8]

롤백 영역(rollback domain) $W(p_i)$는 프로세서 p_i에서 다음과 같이 정의할 수 있다

첫째, 프로세스 p_i에서 활성화 검사점 $c_i(s)$가 포함되어 있다면 $p_i \in W(p_i)$라고 표시한다

둘째, 프로세스 p_i가 활성화 검사점 $c_j(t)$가 포함되어 있고 $< c_i(s), c_j(t) >$이거나, 또는 $< c_j(t), c_i(s) >$이면 $p_j \in W(p_i)$라고 표시한다

즉, $W(p_i) = \{ p_i, p_j \}$

같은 방법으로 롤백 영역(rollback domain) $W(p_j)$는 프로세스 p_j에서 다음과 같이 정의할 수 있다.

$W(p_j) = \{ p_i, p_j \}$

그러므로 $W(p_i) = W(p_j) = \{ p_i, p_j \}$이다.

이 때 하나의 집합 $S' = W(p_i) = W(p_j)$를 롤백 클래스(rollback class)라고 한다.

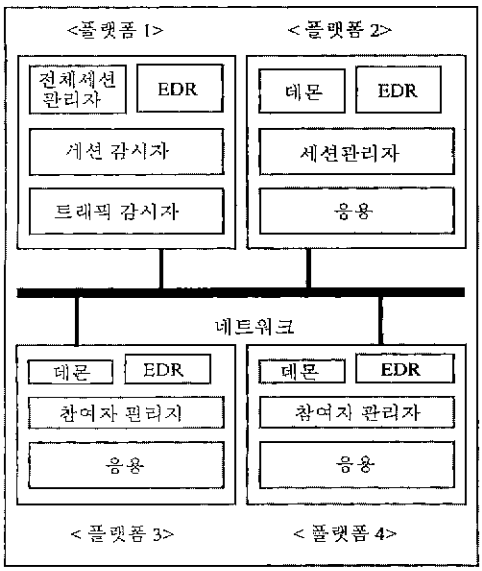
$S' = W(p_i) = W(p_j) = \{ p_i, p_j \}$,

$S'' = W(p_k) = W(p_y) = \{p_k, p_y\}$에서 $1 \le j \le k \le y$일때, 메시지 m을 p_k에서 p_j로 보내면 롤백 클래스는 다음과 같이 변한다. $S = S' \cup S'' = \{ p_i, p_j, p_k, p_y \}$

3.2 EDR_MCSCW의 구조

멀티미디어 협동 작업 환경의 좋은 예는 성균관 대학교에서 1996년부터 개발 운영해온 두레(DOORAE)이다. 두레는 상호 참여형 멀티미디어 응용 개발 환경으로 4계층으로 구성되어 있다. 기존의 단일 멀티미디어 응용 개발에 발생하는 미디어 제어와 세션 관리에 대한 개발 과정의 비용을 줄임으로 상호 참여형 멀티미디어 응용의 개발을 가능하게 지원하는 미들웨어이다. 통신 계층은 분산 처리 환경의 메시지 전송을 담당하는 계층으로 MS-WINDOWS 95/98/NT의 소켓 시스템(socket system)을 활용한다. 즉, 소켓 함수의 패밀리(family)로는 AF_INET를, 유형은 비연결형 또는 연결형을 사용한다. 시스템 계층은 기본적으로 기존의 운영체제(예 : Windows 95/98/NT) 기능을 사용한다 응용 계층은 다양한 응용들이 존재하는 계층으로서 사용자들이 필요로하는 서비스에 대한 선택이 이루어지는 계층이다.

두레 계층은 상호 참여형 멀티미디어 일반적인 응용을 개발하기 위해서 설계된 프레임워크이다. (그림 1)과 같다 두레 계층은 EDR, 전체 세션관리자(GSM), 세션관리자(SM), 서비스 에이전트들로 구성된다. 두레에서 제공되는 서비스 에이전트에는 여러 개의 기능들을 가진다. 플랫폼 1은 전체세션관리자(Global Session Manager)가 활성화되어 있으며 그 안에는 세션 감시자(Session Monitor)와 트래픽 감시자(Traffic Monitor)가 활성화되어 실행되고 있는 상태이다 플랫폼 2는 세션의 초기 생성자인 세션관리자가 위치하는 곳이다. 플랫폼 2, 3, 4에는 데몬(Daemon)이 각각 하나씩 있는데 이 것은 응용 사용자에게 세션관리 서비스를 제공하기 위한 가장 기본적인 세션관리 서비스 객체이다. 이 데몬 생성시 동시에 오류를 감지하여 자동으로 복구시켜 주는 EDR도 생성된다. 하나의 세션이 생성되면 이 세션에는 하나의 전체 세션 관리자, 하나의 세션 및 트래픽 감시자, 하나의 세션 감시자, 다수의 참여자 관리자가 생성된다



(그림 1) 근거리통신망에서의 단일 세션

3.3 EDR의 알고리즘

본 논문에서 제안하는 EDR은 여러 기능의 에이전트가 존재하며 원활한 오류 감지 및 복구 기법을 수행하는 멀티 에이전트 시스템이다. EDR를 구성하는 구성 모듈로는 ED(Error Detection)와 ER(Error Recovery)이다.

ED는 오류를 감지하는 핵심 에이전트로 고장 감지

정보 흐름은 윈도우의 훅킹(hooking) 방법을 이용하여 그 상태를 분석하여 오류의 발생 여부를 감지한다. 세션의 복원을 진행하기 위해서는 먼저 오류 감지를 하기위한 방법이 필요하다. ED는 윈도우의 훅킹 방법을 이용한다.

오류 감지 방법에 대한 개요(scheme)는 다음과 같다.

Set of Detection = {Set of error, Set of fault, Set of error detector}

여기에서

Set of error = {E, D}
- E : 발생하는 오류
- D : 발생하는 오류의 도메인 위치

Set of fault= {H, C, F}
- H : 감지된 오류
- C : 프로세스 데이터베이스(PDB : Process Data Base)에서 찾은 오류 코드(error code), 즉 오류의 유형
- F : 오류의 원인이 되는 고장(fault)

Set of error detector = {Addr_ED, Method_ED, Func_ED}
- Addr_ED : ED의 주소 정보, 즉 $E_i(j)$ 및 $B_i(j)$에 대한 정보
- Method_ED : 윈도우에서 포인팅하는 함수를 가로채서 전달하는 방식인 훅 킹 방식 사용
- Func_ED : ED의 기능(function)은 세 집합 P, $E_i(j)$, $B_i(j)$에서 R_1을 집합 P에서 $E_i(j)$로의 관계(relation)라 하고, R_2를 집합 $E_i(j)$에서 $B_i(j)$로의 관계(relation)라 하면, 집합 P에서 $B_i(j)$로의 합성관계 R_1R_2는 다음과 같이 정의된다

$$R_1R_2 = \{(p_i, b_i(j)) \mid p_i \in P, b_i(j) \in B_i(j), (p_i, e_i(j)) \in R_1, (e_i(j), b_i(j)) \in R_2\}$$

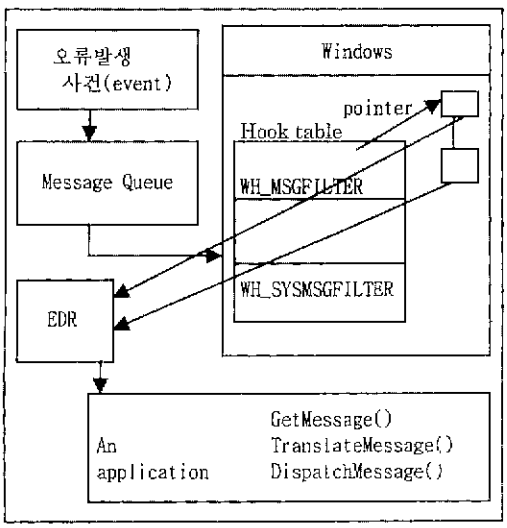
관계 R_1에서는 키보드, 마우스, 시스템 등의 사건을 메시지 형태로 전송한다. 이 메시지를 훅 킹 함수라고 한다 사용자의 키보드나 마우스 같은 입력은 메시지 큐에 저장되고 윈도우즈는 명령어에 대한 훅 테이블을 유지하고, 실행할 함수를 포인팅 한다. 오류를 감지하는 방법도 이와 비슷한 과정을 거친다. 이 과정에서 오류를 감지한 내용, 즉, 포인팅 하는 함수를 가로채서 전달하는 방식이다. 훅 킹 메시지의 종류는 <표 1>과 같다. ED는 저널 레코드 또는 훅 훅 등의 훅 킹 메시지를 사용하여 오류를 감지한다.

뒤에 R_2에서 오류 감지된 프로세스의 오류 유형을 알기 위하여 프로세스데이타베이스(PDB : Process Data Base)를 검색하면 알 수 있다 윈도우 함수 중 GetExitCodeProcess를 사용할 때 리턴코드 값에 의해서 알 수 있다.

〈표 1〉 훅 킹 메시지의 종류

사건유형(Event Types)	기능설명(Description)
WH_MSGFILTER	Dialog box, scroll bar, menu
WH_SYSTEMFILTER	Dialog box, scroll bar, menu
WH_GETMESSAGE	GetMessage, PeekMessage
WH_KEYBOARD	Process, modify keyboard events
WH_MOUSE	Process, modify mouse events
WH_DEBUG	Prevent another filter being called
WH_CALLWINPROC	Send Message intercept
WH_JOURNALRECORD	Record keyboard and mouse event
WH_JOURNALPLAYBACK	Playback keyboard and mouse event

오류 감지 시 훅 킹 과정은 (그림 2)와 같다.



(그림 2) 오류 감지 시 훅 킹 과정

ER은 ED로부터 받은 오류 분류 정보를 바탕으로 오류를 복구하는 모듈이 실행된다. 이 때 검사점 설정까지 설정된 지점까지 롤백(rollback)하여 복구된다. 복구 방법에 대한 개요(scheme)는 다음과 같다.

Set of Recovery = { Set of fault,
Set of checkpoint type,
Set of message information,
Set of recovery agent }

여기에서

Set of fault={C} 오류코드, 즉 PDB에서 찾은 오류의 유형 내용

Set of checkpoint type = {SF, SD, SI}
하나의 클래스 S에서 S={ p_i, p_j, p_k, p_x, p_y } 이며 i ≠ j ≠ k ≠ x ≠ y이다. 하나의 프로세스 p_i가 실제 오류를 발생시킨 프로세스라고 하면 SF = { p_i }이고, SD = {p_j, p_k }는 같은 플랫폼(platform)에 있는 프로세스들이고, SI = {p_x, p_y }는 다른 플랫폼에 있는 프로세스들이다. 즉, S = SF ∪ SD ∪ SI이고 SF ∩ SD ∩ SI = ∅이다. 이때 SF 클래스, SD 클래스, SI 클래스 사이에는 메시지 교환이 발생하는 것을 전제로 한다. 만일 메시지 교환이 발생하지 않으면 SF, SD, SI 간의 롤백 클래스가 다를 수 있다. SF와 SD 사이에 메시지 교환이 일어나면 이때 롤백 클래스 SFD = {p_i, p_j, p_k }. SF, SD, SI사이에 메시지 교환이 일어나면 이 때 롤백 클래스 SFDI = { p_i, p_j, p_k, p_x, p_y }이다.

Set of message information = {m.d, m.h}
메시지 m은 데이터 m.d와 헤더정보 m.h로 표시된다. m.h는 세분하면 다음과 같다.
m.h = {w, x, y, z, a, b}

여기에서 w는 검사점을 표시
x는 통신의 종류(송신, 수신) 표시
y는 프로세스들간의 관계 즉, SD 또는 SI에 속해 있는 프로세스들간의 관계를 표시하는 플래그
z는 주고 받은 프로세스들의 식별 번호 등을 표시하는 플래그
a : SD, SI인 경우에 프로세스 간 주고 받은 누적 번호 기록
b : 메시지의 종류
값 1-정상 데이터, 0-신호 데이터

Set of recovery agent = {Log_ER, Addr_ER, Method_ER, Func_ER}

여기에서

Log_ER : 2개의 스택 구조, 즉 검사점 스택과 복구 스택으로 나누이지고 각 프로세스는 각각 2개의 스택을 가지고 있음. 프로세스간 교환된 메시지의 정보,즉, 프로세스가 m.h 등을 받는 시점에 스택 형식으로 저장하고 읽는 데이터 구조, 즉 LIFO 구조인

Addr_ER : ER의 주소 정보, 즉 $S_i(j)$, $C_i(j)$, $R_i(j)$ 및 $K_i(j)$에 대한 정보

Method_ER : 검사점 설정과 롤백 복구 알고리즘 사용함

Func_ER : ER의 기능(function)은 5개의 집합 P, $S_i(j)$, $C_i(j)$, $R_i(j)$, $K_i(j)$에서 R_3을 집합 P에서 $S_i(j)$로의 관계(relation)라 하고, R_4를 집합 $S_i(j)$에서 $C_i(j)$로, R_5를 집합 $C_i(j)$에서 $R_i(j)$로, R_6를 집합 $R_i(j)$에서 $K_i(j)$로의 관계(relation)라 하면, 집합 P에서 $K_i(j)$로의 합성관계 R_3R_4 R_5R_6은 다음과 같이 정의된다.

R_3R_4 R_5R_6 = {(p_i, $k_i(j)$) | $p_i \in P$, $k_i(j) \in K_i(j)$, (p_i, $s_i(j)$) $\in R_3$, ($s_i(j)$, $c_i(j)$) $\in R_4$, ($c_i(j)$, $r_i(j)$) $\in R_5$, ($r_i(j)$, $k_i(j)$) $\in R_6$ }

관계 R_3에서는 프로세스간에 메시지를 받을 때 받는 즉시 검사점을 설정한 후에 ER에서 관리하고 있는 해당 프로세스의 검사점 스택에 저장한다. 즉, SF, SD, SI에 속한 모든 프로세스들이 메시지를 받으면서 그 프로세스에서 받은 m.h의 내용을 검사점 스택에 저장한다.

관계 R_i에서는 EDR이 고장 프로세스 p_i를 발견하면 프로세서 p_i가 실행하고 있을 때 그 프로세스 p_i에서 j번째의 검사점(checkpoint)과 j+1번째의 검사점 사이에 있는 r에서 오류가 발생하여 복구 시킬 때 $c_i(j)$ 에서부터 롤백(rollback)하여 재시작해야 한다. $c_i(j)$를 알아내기 위하여 스택에 있는 항목을 하나씩 읽어서 $c_i(j)$가 나타날 때까지 읽는다. 이때 클래스 SF에 속한 프로세스 p_i는 관계 R_i에서 $c_i(j)$를 발견할 때까지 검사점 스

택에 있는 내용을 읽으면서 복구 스택에 저장한다. 즉, 검사점 스택의 일부 항목과 복구 스택의 항목은 역순이 된다

관계 R_7에서는 복구 스택의 항목을 하나씩 읽어내면서 롤백 클래스가 SF, SFD, SFDI 중 어느 것인지를 결정한다 검사점의 유형에 따른 복구 방법에 대한 개요(scheme)는 다음과 같다.

관계 R_8에서는 하니의 클래스 S에서도 SF, SFD, SFDI에 따라서 복구 알고리즘이 다르게 실행된다. 이러한 프로세스의 형태를 구별하는 방법은 저장되어 있는 고장난 프로세스의 m.h의 y정보를 보고 구별한다.

(SF인 경우)

고장이 발생하면 EDR은 프로세스에게 검사점을 파악한 후에 롤백한다.

(SFD인 경우)

고장난 프로세스는 클래스 SFD에 복구 신호를 준다. 이 복구신호를 받았을 때 같은 플랫폼에 있는 프로세스 간에는 단순 재 실행 방법으로 복구 한다. 데몬은 할당 받은 네트워크 자원을 가지고 세션관리자를 생성하게 된다. 이 때 생성된 세션관리자는 요구 받은 미디어 자원에 대한 생성을 요구한다. 미디어 서버는 미디어 서버 인스턴스를 생성한다

(SFDI인 경우)

고장난 프로세스는 클래스 SFDI에 복구 신호를 준다. 다른 플랫폼에 있는 프로세스 간에는 메시지 교환이 발생하므로 도미노효과(Domino Effect)를 고려한다. ER에 저장되어 있는 log 파일의 내용을 읽어서 last in-first out 순서로 읽으면서 m.a, m.z 정보를 통하여 검사점을 알아낸다. 검사점을 알아내면 다음과 같은 과정 중에서 그 검사점에서부터 복구한다.

EDR은 전체세션관리자(GSM)에게 모든 진행 중인 세션에 대한 정보를 요청한다.

(1) 전체세션관리자(GSM)는 EDR에게 세션에 대한 정보를 알려준다.
(2) EDR은 그 위치에 있는 데몬(Daemon)에게 복구할 것을 알린다.
(4), (5) 세션 진행 중에 복구 요청을 받은 데몬(Daemon)은 원격지의 데몬(Daemon)들에게 세션 진행중에 복구 요청이 있다는 것을 알린다.
(6) 원격지 데몬(Daemon)은 자신의 세션관리자(SM)에게 세션 진행 중 에 복구 요청이 있다는 것을 알린다.
(7) 원격지 세션관리자(SM)는 자신의 데몬(Daemon)을 통해 오류가 발 생한 곳의 데몬(Daemon)에게 응답을 보낸다.
(8)-(11) 원격지에 모두 알린 데몬(Daemon)은 자신의 참여자 관리자(PM : Participant Manager)를 생성하고 이어서 세션에 필요한 미디어를 실행시킨다.
(12) 사용할 자원의 생성이 모두 끝나면 응용을 실행시킨다.
(13) 응용의 실행이 모두 끝나면 오류 복구 처리는 완료한다

복구된 프로그램들은 세션의 정상적인 참여자가 된다.

4. 시스템 평가

제안된 시스템은 Visual C++로 설계 및 구축 가능하다. 오류 감지에서 제안된 방법의 나은 점을 DEVS 형식론을 이용하여 모델링 및 시뮬레이션을 통하여 비교한다. DEVS(Discrete Event System Specification)는 Bernard P Zeigler에 의해 개발된 이산 사건 모델들의 계층 구조적 모듈화 방법을 제공해주는 형식론이다 시스템을 작은 모듈들로 나누고 그것들로 전체 시스템을 계층적으로 구성해 나간다. 각 모듈들은 원자(atomic) 모델로 표현되며 그것들의 계층적 구성은 커플(coupled) 모델로 표현된다 DEVS 형식론에서 가장 기본이 되는 모델인 원자 모델은 다음과 같은 길합으로 표현된다[21].

$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$

- X : 외부 입력 사건들의 집합
- S : 상태 변수들의 집합
- Y : 외부 출력 사건들의 집합
- δ_{int} : 내부적 상태 변환 함수
- δ_{ext} : 외부적 상태 변환 함수
- λ : 출력 함수
- t_a : 시간 진행 함수

원자모델을 결합하여 새로운 커플모델을 형성한다. 본 논문에서는 전체 응용에 대해서 기존 방식 중의

하나인 폴링 방식을 사용하는 방식과 본 논문에서 제안한 훅킹 방법을 사용하여 오류 감지 시간을 줄이는 방식 2가지를 비교하여 효율성 검토를 한다. 시뮬레이션 모델을 통한 관측 목표와 관측 값 계산에 관련된 변수를 상태 변수로 가지는 모델이다 오류 감지에 대한 효율성 비교는 <표 2>와 같다.

〈표 2〉 오류 감지에 대한 효율성 비교

대상 (object)	관측 지수		관측 모델
	기존 방식	제안된방식	
각 직업 (Each job)	ED1은 EF에 RA1 반응시간 총합을 준다 (종료 시점에서)	ED2는 EF에 HA2는 반응시간 총합을 준나 (종료 시점에서)	transducer

(기존 방식)

DEVS 형식론에서 기존 방식의 변수를 정의하면 다음과 같다. 원자 모델은 EF, RA1, ED1이고 이 원자 모델들을 결합하여 새로운 커플모델을 형성한다. 처음에 외부 입력 사건들 중에서 폴링 시간을 받아들인다. 그 값이 RA1에서 입력 값이 되어 실행 한 후에 그 결과 값이 ED1의 입력 값이 되어서 그 결과 값이 EF의 transducer를 통해서 관측될 수 있다. EF의 상태 변수는 poll-time1, RA1의 상태 변수는 pr-cnt1, ED1의 상태 변수는 tat-t1이다.

(제안된 방식)

DEVS 형식론에서 제안된 방식의 변수를 정의하면 다음과 같다. 원자 모델은 EF, HA2, ED2이고 이 원자 모델들을 결합하여 새로운 커플모델을 형성한다. 처음에 외부 입력 사건들 중에서 훅 킹 방법에 의한 감지 시간을 받아들인다 그 값이 HA2에서 입력 값이 되어 실행 한 후에 그 결과 값이 ED2의 입력 값이 되어서 그 결과 값이 EF의 transducer를 통해서 관측될 수 있다. EF의 상태 변수는 poll-time2, HA2의 상태 변수는 pr-cnt2, ED2의 상태 변수는 tat-t2이다

기존 방식과 제안된 방식의 비교는 다음과 같이 할 수 있다 오류 감지 부분에서 기존 방법의 감지 시간(tat-t1)은 다음과 같다. 만일 프로세스 간의 메시지가 전달될 때 걸리는 시간을 t라고 하면 한번 폴링 시간은 2t가 된다. tat-t1 = 2t*pr-cnt1

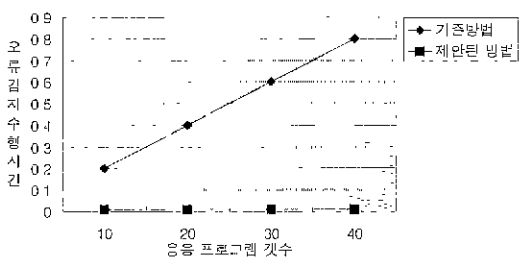
제안된 방법의 감지 시간(tat-t2)은 다음과 같다

훅킹 방법에 의해서 사건(event)이 발생하면 프로세스의 수에 관계 없이 1번 발생한다.

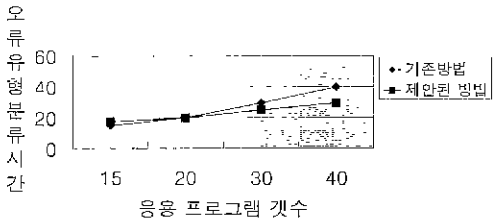
$$tat\text{-}t2 = t + 1$$

pr-cnt1 >= 1이므로 제안된 방법이 더 효율적이다 프로세스의 수가 많아질수록 제안된 방식이 더 효율적이다

(그림 3)은 응용 프로그램 개수와 오류 감지 수행 시간과의 관계를 나타낸 것이다. 만일 프로세스 간의 메시지가 전달될 때 걸리는 시간을 0.01초라고 하면 한번 폴링하는 시간은 0.02초가 된다. 응용 프로그램의 개수를 10, 20, 30, 40개라고 하면 기존 방법의 오류 감지 수행 시간은 각각 0.2, 0.4, 0.6, 0.8이 된다 훅 킹을 사용한 제안된 방법은 프로그램 개수에 관계없이 각각 0.01씩 된다. 프로세스의 수가 많아질수록 제안된 방식이 더 효율적임을 알 수 있다



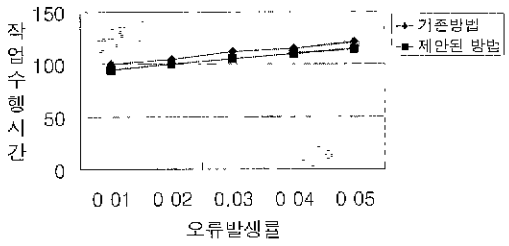
(그림 3) 응응 프르그램의 개수와 오류 감지 수행 시간과의 관계



(그림 4) 응용 프로그램의 개수와 오류 유형 분류 시간과의 관계

(그림 4)는 응용 프로그램 개수와 오류 유형 분류 시간과의 관계를 나타낸 것이다 만일 프로세스 간의 메시지가 전달될 때 걸리는 시간을 0.5초라고 하면 한번 폴링하는 시간은 1초가 된다 세션과 관련된 응용 프로그램의 개수를 10개, 세션과 무관한 응용 프로그램의 개수를 5, 10, 20, 30개라고 하면 전체 응용 프로

그림의 개수는 각각 15, 20, 30, 40개가 된다. 기존 방법의 오류 분류 수행 시간은 각각 15,20,30,40이 되고 제안된 방법의 오류 유형 분류 수행 시간은 각각 17.5, 20, 25, 30이 된다. 세션에 관련이 없는 응용 프로그램의 개수가 두레에 관련이 있는 응용 프로그램의 개수보다 적은 경우에는 기존 방법이 효율이 높다



(그림 5) 오류 발생률과 작업 수행 시간

(그림 5)는 기존 방식과 제안된 방식의 오류 발생률의 변화에 의한 작업 시간의 변화를 나타낸 것이다.

체크 포인트를 사용한 시스템의 한 체크 포인트 간격에서의 수행 시간의 기대 값은 다음 식과 같다.

$$\Gamma = (1/\lambda) \times e^{\lambda R} \times (e^{\lambda(T_c + C)} - 1)$$

$$T_{opt} = (2C/\lambda)^{1/2}$$

여기서

Γ는 전체 수행시간, R은 귀환(roll back) 시간, C는 체크포인팅시간, T_c는 체크포인트 간격, T_{opt}는 최적의 체크포인트 간격을 나타낸다.

작업의 유효 수행 시간을 100으로 하여 오류 발생률(λ)을 0.01에서 0.05까지 변화시키면서 시뮬레이션하면 (그림 5)와 같은 결과를 얻을 수 있다.

제안된 논문의 장점은 세션 개설시 잘못된 메시지를 주고 받을 수 있을 때 발생하는 도미노효과를 고려한 검사점 설정 및 복구 알고리즘에 대한 연구물 제시한 점이다 또한 기존 방식인 정기적 검사점 설정 방법은 순수체크 포인트에 걸리는 시간은 작지만 롤백 거리가 크다 제안된 방식은 롤백 클래스의 개념을 도입하여 롤백 거리를 줄이는 역할을 하였다.

본 논문의 단점은 스택을 사용함으로써 메모리 사용량이 증가하여 오버헤드가 있다는 점이다

5. 결 론

본 논문에서 제안하는 방식은 두레라는 멀티미디어 협동 작업 환경에서 세션을 잘 유지하고 복구할 수 있는 오류 감지 및 복구 시스템을 제안하였다. 본 논문에서 제안한 EDR은 오류 감지, 검사점 설정 및 복구 기능을 갖고 있는 에이전트이다. 세션에서 발생한 오류를 훅킹 방법을 사용하여 감지한다. 이때 세션에서 발생한 사건만 오류 감지를 수행한다. 오류가 감지되면 그 유형에 따라 도미노 효과를 고려한 검사점 설정 및 자동으로 복구 시켜주는 에이전트이다. 본 논문의 특징은 세션에서 발생하는 프로세스간의 메시지 교환을 고려하여 오류 복구시 도미노 효과를 제거하였다. 향 후 연구 과제는 다중 세션이 활성화되어 있는 경우에서의 오류 감지 및 복구 시스템에 대한 연구이다. 특히, 세션 개설시 프로세스간의 메시지 교환이 일어날 때 도미노 효과를 제거하기 위하여 필요한 메시지 헤더의 크기를 줄이는 방법 등에 대한 연구가 필요하다.

참 고 문 헌

[1] 최양희, "멀티미디어 정보 통신 개요", 정보과학회지, 제9권 제3호, pp.5-18, 1991.6

[2] 성미영, 유재홍, "웹 회상회의 시스템을 위한 KQML 기반의 멀티 에이전트 구조", 한국정보처리학회 논문지, 제6권 제12호, pp.3477-3489, 1999.12.

[3] Ralf Steinmetz and Klara Nahrstedt. 'Multimedia : Computing," Communications & Applications. Prentice Hall PTR. p.854, 1995.

[4] Eric Garland and Dave Rowell, "Face-to-Face Collaboration," Byte, Vol 19, No 11, pp.233 242, November, 1994.

[5] Stephen Jabele, Steven Rohall, Ralph L Vinciguerra, "High Performance Infrastructure for visually-Intensive CSCW Applications." Proceedings on CSCW 94, ACM Press, pp.395-403, October 1994.

[6] Hiroaki Higaki, Kenji Shima, Takayuki Tachikawa, Makoto Takizawa, "Checkpoint and Rollback in Asynchronous Distributed Systems," IEEE INFOCOM 97, Proceedings Volume 3.

[7] Philip A. Laplante, "Real-Time Systems Design and Analysis," IEEE Press, 1997, pp 269

[8] Feignbaum, Armand V, Total Quality Control, 3rd ed, New York : McGraw-Hill, 1983.

[9] 허 신, "소프트웨어 결함 허용 기법에 대한 고찰", 한국정보과학회지, 제11권 제3호, pp.32-39, 1993. 6.

[10] 장순주, 임종규, 정구영, 구용완, "분산 시스템에서 결함 허용성을 위한 프로세스 이주 연구", 한국정보과학회지, 가을학술발표 논문집, Vol.21, No2, p.132, 1994.

[11] 김문희, "결함 허용 시스템의 설계 고려사항 및 동향," 한국정보과학회지, 제11권 제3호, pp.7, 1993.

[12] 윤재영, 김학배, "Rollback과 Roll-forward 기법을 사용한 TMR 고장의 시간여분 복구 정책", 한국정보처리학회논문지, 제6권 제1호, pp.216-224, 1999. 1.

[13] Jonson, B W., "Design and Analysis of Fault-Tolerant Digital Systems," Addison Weslev, 1989.

[14] Randell, B., "System Structure for Software fault tolerance," IEEE Trans,on Soft Engr., pp.220-232, June 1975.

[15] S S Yau and R. C. Cheung, "Design of Self-Checking Software," in Proc. 1975 Int Conf Reliable Software, pp.450-457, April 1975.

[16] J J Horing, H. C. Lauer, P M. Mellar-Smith and B. Randell, "A Program Structure for Error Detection and Recovery," Lecture Notes in Computer Science 16, berlin : Springer-Verlag, pp.171-187, 1974.

[17] R. Koo, & S. Toueg, "Checkpointing and Rollback recovery for Distributed Systems," IEEE Trans. Software Eng., Vol. SE-13, No.1, pp.23-31, 1987

[18] 박윤용, 전성익, 조주헌, "분산 트랜잭션 처리 시스템에서 2-단계 확인 프로토콜을 근거로 하는 검사점 설정 및 오류 복구 알고리즘", 한국정보처리학회논문지, 제3권 제2호, pp.327-338, 1996.3.

[19] Hong Va Leong, Divyakant Agrawal, "Using Message semantics to Reduce Rollback in Optimistic message Logging Recovery Schemes," 1063-6927/94, 1994 IEEE, pp.227-234, 1994.

[20] Dhiraj K. Pradhan, Nitin H. Vaidya, "Roll-Forward and Rollback Recovery : Performance Reliability Trade-off." 0363-8928/94, 1994 IEEE, pp.186-195, 1994.

[21] Bernard P Zeigler, "Object-Oriented Simulation with hierarchical, Modular Models," Academic Press, 1990.

[22] N. H. Vaidya, "On checkpoint latency," in Proc. Of the 1995 pacific Rim International Symposium on Fault-Tolerant Systems, pp.60-65, Dec.1995.

[23] J. Young, "A first order approximation to the optimal checkpoint interval," Communication of the ACM, Vol.17, pp.530-531, Sept, 1974.

고 응 남

e-mail : sskcn@hanmail.net

1984년 연세대학교 수학과(이학사)

1991년 숭실대학교 정보과학 대학원(공학석사)

2000년 성균관대학교 정보공학과(공학박사 예정)

1983년~1993년 대우통신 컴퓨터 개발부 선임 연구원

1993년~1997년 동우대학 전자계산과 교수

1997년~현재 신성대학 컴퓨터 게열 교수

관심분야 : 멀티미디어, 결함 허용, 에이전트 등

황 대 준

e-mail : djhwang@yurim.skku.ac.kr

1978년 경북대학교 전자 계산기 공학과(공학사)

1981년 서울대학교 자연과학대학 계산통계학과(이학석사)

1986년 서울대학교 자연과학대학 계산통계학과(이학박사)

1981년~1987년 한남대학교 교수

1990년~1991년 MIT 컴퓨터과학연구소 연구교수

1987년~현재 성균관대학교 정보공학과 교수

관심분야 : 멀티미디어, 병렬 처리, 원격교육 등