회복 에이전트 기반 결함 포용 시스템

이화민[†] · 정순영[††] · 유헌창[††]

요 약

본 논문에서는 기존의 분산 컴퓨팅 시스템의 결함 포용 기법에 멀티 에이전트의 개념을 도입하여 운영체제에 독립적인 에이전트를 이용한 회복기법을 제안한다. 기존의 복귀회복기법들은 운영체제의 관리하에서 결함 발생 시 회복에 관련된 동기화를 수행하는데, 이는 전체 분산 컴퓨팅 시스템의 성능을 저하시키는 원인이 되었다. 이러한 문제점을 해결하기 위해 본 논문에서는 프로세스의 회복을 담당할 회복 에이전트, 결함 포용 규칙과 정보를 유지·관리하는 정보 에이전트, 전체 에이전트간의 통신 기능을 담당할 조정 에이전트를 정의 및 설계하고 회복 에이전트를 이용한 회복 알고리즘을 제안한다. 그리고 코바 환경에서 자바와 에이전트 통신 언어를 이용하여 제안한 회복 알고리즘을 실험하였다. 분산 컴퓨팅 시스템에서 회복 에이전트의 도입은 결함 발생 프로세스의 결함 회복 작업을 어플리케이션 계층과 독립적인 별도의 계층으로 계층화하여 결함 포용을 위한 메카니즘의 이식성 증대 및 확장성 증대를 가져온다.

Fault Tolerant System based on Recovery Agents

Hwa-Min Lee[†] · Soon-Young Jung[††] · Heon-Chang Yu[††]

ABSTRACT

This paper proposes a new approach to rollback-recovery using multi-agent in distributed computing system. Previous rollback-recovery protocols are dependent on inherent communication and operating system, which causes a decline of computing performance in distributed computing system. By using multi-agent, we propose rollback-recovery protocol that is independent on operating system. We define three kinds of agent. One is a recovery agent that performs rollback-recovery protocol after a failure. Other is an information agent that constructs domain knowledge as a rule of fault tolerance and information during failure-free operation. The other is facilitator agent that controls the efficient communication between agents. Also we propose rollback-recovery protocol using multi-agent and simulated the proposed rollback-recovery protocol using JAVA and agent communication language in CORBA environment.

1. 서 론

분산 컴퓨팅 시스템은 통신 네트워크에 의해서 서로 연결된 자율적인 프로세스들의 집합체로서, 각 노드의 결함 발생 가능성과 노드를 상호 연결하는 통신 수단에서의 결함 발생 가능성까지 존재하기에 결함 발생 확률이 단일 시스템에서의 결함 발생 확률보다 월등히 크다. 또한 분산 컴퓨팅 시스템은 각 프로세스간 공유메모리가 없기 때문에 장시간 수행되는 응용프로그램의 경우, 임의의 프로세스에서 결함이 발생하면 프로세스의 의존성으로 인해 전체 응용프로그램이 원하는 결과를 산출해내지 못하는 잠재적인 위험이 존재

† 정 회 원: 고려대학교 컴퓨터교육과 박사과정
†† 종신회원: 고려대학교 컴퓨터교육과 교수

논문접수: 2002년 3월 25일, 심사완료: 2002년 4월 19일
* 본 논문은 2001년 고려대학교의 학술연구비에 의하여 지원되었음

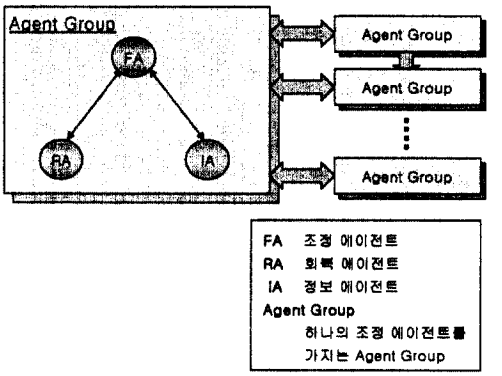
한다[1]. 이처럼 분산 컴퓨팅 시스템은 단일 시스템보다 결함에 대해 민감하기 때문에 기존의 많은 연구들에서 분산 시스템에서 결함이 발생할 경우 이를 해결하기 위한 많은 복귀회복기법들이 연구되었다[2][3].

기존의 결함 포용 기법에 관련된 검사점기반 회복 기법들과 메시지 로그기반 회복기법들의 대부분은 고장-멈춤(fail-stop) 모델을 채용함으로써 결함이 발생한 프로세스에 대해 운영체제의 관리하에 모든 회복 작업이 수행되어 왔다. 즉, 결함이 발생한 프로세스와 의존 관계에 있는 모든 프로세스들은 각각의 운영체제의 관리하에서 결함 발생 프로세스의 회복에 관련된 동기화를 수행해야 했으며, 이러한 회복 기법은 전체 분산 컴퓨팅 시스템의 성능을 저하시키는 원인이 되었다.

이에 본 논문에서는 분산 컴퓨팅 시스템의 결함 포용 기법에 멀티 에이전트의 개념을 도입하여 운영체제로부터 독립적인 회복기법을 제시한다. 즉, 회복 에이전트는 분산 컴퓨팅 시스템에서 운영체제의 시스템 영역에서 수행되는 결함 발생 프로세스의 결함 회복 작업을 사용자 영역에서 수행되는 서버가 담당하도록 하여 어플리케이션 계층과 독립적인 별도의 계층으로 계층화한다. 이와 같은 계층화를 통해 모듈화가 이루어질 경우 분산 컴퓨팅 시스템에서 결함 포용을 위한 메카니즘의 개방성이 증대된다. 또한 이는 분산 컴퓨팅 시스템에서 결함 포용 기능이 계층화되어 모듈화가 이루어지면 현재 상용 분산 컴퓨팅 시스템을 비롯한 여러 분산 컴퓨팅 시스템에 결함 포용 기능의 탑재가 쉬워져 이식성의 증대 및 확장성 증대를 가져온다. 본 논문에서 제안한 회복 에이전트를 이용한 회복기법은 독립 검사점(independent checkpointing) 기법과 비관적 메시지 로깅(pessimistic message logging) 기법을 사용한다[2][4].

2. 시스템 모델 및 구성 요소

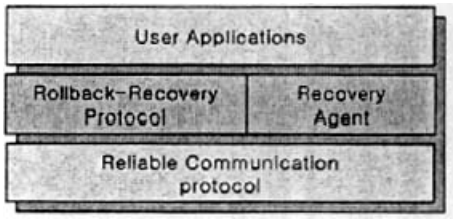
시스템 모델은 응용프로그램을 수행하는 프로세스들과 회복·조정·정보 에이전트 그리고 이들간의 통신 채널로 구성되는 분산 컴퓨팅 시스템 환경이다. 여기서 말하는 분산 컴퓨팅 시스템 환경은 공유 메모리와 같은 기능을 사용하지 않고 오직 메시지만으로 프로세스들과 에이전트들이 통신을 하는 메시지 전달 시스템을 기반으로 한다. 통신 환경은 통신 네트워크가 분할되지 않는다고 가정하며 프로세스간 통신은 신뢰성을 보장하여 메시지를 선입선출 방식으로 전달한다고 가정한다. 또한 프로세스의 고장에 대해서는 고장-멈춤 모델에 따라 프로세스는 휘발성 메모리에 저장된 상태만을 잃어버리고, 실행을 중지한다고 가정한다. 본 논문에서 제안하는 멀티 에이전트를 이용한 결함 포용 시스템의 전체 구성도는 <그림 1>과 같다.



<그림1> 결함 포용 시스템의 구성도

본 논문에서 제안하는 결함 포용 시스템은 회복·조정·정보 에이전트로 구성된다. 회복 에이전트는 분산 컴퓨팅 시스템에서 결함 발생시 자신의 지식 정보와 다른 회복 에이전트와의 협력을 통해 자치적으로 회복 과정을 수행하는 에이전트이다. 정보 에이전트는 회복 에이전트가 결함 발생시 회복 과정 수행을 위해 필요한 프로세스에서 발생한 정보들을 영역지식으로 구축해주는 일을 담당하는 에이전트이다. 그리고 본 논문에서는 에이전트들의 효율적인 통신을 도울 수 있도록 특별한 조정 에이전트를 정의하여 에이전트 연방 시스템을 구축한다.

기존 연구에서는 회복 기법이 사용자 애플리케이션과 신뢰성 있는 통신 기법 사이에 위치하였는데 회복 에이전트를 도입한 경우 회복 기법과 프로세스간 통신은 <그림 2>와 같이 변화된다.



<그림 2> 변화된 3-계층 구조

수정된 회복기법에 따라 기존의 비관적 메시지 로깅에서는 통신 프로토콜을 통해 프로세스에게 메시지가 전달되고 프로세스가 수신한 메시지를 안정된 저장소에 로깅을 했지만 회복 에이전트를 이용한 회복기법에서는 프로세스에게 메시지가 전달되기 전 메시지는 조정 에이전트에게 먼저 전달되고 정보 에이전트에 의해 안정된 저장소에 로그된다.

3. 에이전트의 통신 언어

3.1. 복귀회복기법을 위한 온톨로지

본 논문에서는 분산 컴퓨팅 시스템에서 결함포용을 위해 사용하는 어휘들을 Rollback_Recovery라 이름하여 에이전트들의 온톨로지로 새롭게 정의한다. 정의된 온톨로지는 <표 1>과 같다.

<표1> Rollback_Recovery 온톨로지

어휘	의미
checkpoint	프로세스가 검사점을 취한다.
send	다른 프로세스에게 메시지를 전송한다.
receive	다른 프로세스로부터 메시지를 수신한다.
before	앞의 이벤트가 뒤의 이벤트보다 선행된다.
log	수신한 메시지를 로그한다.
orphan	해당 메시지가 고아메시지이다.
recovery	결함 발생 후 회복을 수행한다.
rollback	프로세스가 해당 사건까지 복귀를 한다.
lastckp	프로세스가 취한 마지막 검사점을 나타낸다.
eventlist	마지막 검사점 이후 발생한 이벤트의 리스트
broadcast	이벤트 리스트를 모든 조정 에이전트에게 전송
replay	메시지 전송을 재수행 한다.

3.2. KIF를 이용한 영역 지식 구축

본 논문에서 결함 포용을 제공하는 에이전트의 영역지식을 위해 정의한 KIF의 BNF 문법은 <표 2>와 같다.

<표2>의 Rollback_Recovery 온톨로지와 <표 3>의 BNF 문법에 따라 정의한 KIF 예를 살펴보면 다음과 같다.

```
(checkpoint p1 c2)
(send p2 m1 )
```

위 두 문장은 각각 프로세스 p1이 검사점 c2를 취했다는 정보와 프로세스 p2에게 메시지 m1를

<표 2> KIF의 BNF 문법

```
<special> ::= " | # | ' | ( | ) | , | \ | ^ | `
<normal> ::= <stringchar>*" | #<digit><digit>*"<ascii>*
<word> ::= <normal> | <word>*
<expression> ::= <word> | (<expression>*)
<string> ::= empty | "<normal>
<indvar> ::= <?word>
<seqvar> ::= <@word>
<sentop> ::= = | /= | not | and | or | => | <= | <=> | forall | exists
<variable> ::= <indvar> | <seqvar>
<constant> ::= <word> - <variable> - <operator>
<logterm> ::= (if <sentence> <term> [<term>]) | (cond (<sentence>
              <term>) ··· (<sentence> <term>))
<quoterm> ::= (quote <expression>)
<term> ::= <indvar> | <constant> | <string> | <funconst> |
           <logterm> | <quoterm>
<sentence> ::= <logconst> | <equation> | <inequality> | <relsent> |
               <logsent> | <quantsent>
<equation> ::= (= <term> <term>)
<inequality> ::= (/= <term> <term>)
<relsent> ::= (<relconst> <term>* [<seqvar>]) | (<funconst>
              <term>* <term>)
<logsent> ::= (not <sentence>) | (and <sentence>*) | (or
              <sentence>*) | (=> <sentence>* <sentence>) | (<=
              <sentence> <sentence>*) | (<=> <sentence>
              <sentence>)
<quantsent> ::= (forall <indvar> <sentence>) | (forall (<indvar>*)
                <sentence>) | (exists <indvar> <sentence>) |
                (exists (<indvar>* <sentence>)
```

전송하였다는 정보를 나타낸다. 위 사건의 선후관계는 복귀회복을 위한 온톨로지의 before 어휘를 이용하여 다음과 같이 표현된다.

```
(before (checkpoint p1 c2) (send p2 m1 ))
```

3.3. KQML를 이용한 에이전트간 통신

본 논문에서 사용하는 KQML의 BNF 문법은 <표 3>과 같다.

<표 3> KQML의 BNF 문법

```
<performative> ::= (<word> {<whitespace> :<word>
<whitespace> <expression>)*)
<expression> ::= <word> | <quotation> | <string> | (<word> {
<whitespace> <expression>)*)
<word> ::= <character><character>*
<character> ::= <alphabetic> | <numeric> | <special>
<special> ::= < | > | = | + | - | * | / | & | ^ | ~ | _ | @ | $ |
% | : | . | ! | ?
<quotation> ::= '<expression> | '<comma-expression>
<comma-expression> ::= <word> | <quotation> | <string> |
,<comma-expression> (<word> {<whitespace>
<comma-expression>)*)
<string> ::="<stringchar>*" | #<digit><digit>*"<ascii>*
<stringchar> ::= \<ascii> | <ascii> -\-<double-quote>
```

KIF에 대한 <표3>와 KQML에 대한 <표4>에

따른 간단한 KQML의 예를 살펴보자.

```
(ask-one : sender facilitator
         : receiver ra1
         : reply-with id1
         : language KIF
         : ontology Rollback_Recovery
         : content (lastckp p1 ?c))
(tell    : sender ra1
         : receiver facilitator
         : reply-with id5
         : in-reply-to id1
         : language KIF
         : ontology Rollback_Recovery
         : content (checkpoint p1 c2))
```

위의 예에서 ask-one과 tell은 performative로 ask-one의 경우 :content가 질문임을 나타내고 암시적으로 tell performative 형태의 답변이 오기를 기다린다. 위 대화의 내용은 facilitator 에이전트가 회복에이전트 ra1에게 프로세스 p1이 어떤 메시지를 받은 후 검사점 c2를 취한 것이냐 물음에 대해 회복에이전트 ra1이 facilitator 에이전트에게 메시지 m1을 받은 후라고 답을 하는 것이다. 위의 예에 나타난 parameter들을 살펴보면 :sender와 :receiver는 메시지 발신자와 수신자를 의미하고, ask-one의 reply-with와 tell의 in-reply-to는 서로 짝을 이루어 질문에 대한 답변을 id1이라는 값으로 해주기를 요청하는 것이다. :language는 :content에를 표현하는 언어가 KIF라는 것을 밝히는 것이고 :ontology는 content에 쓰인 어휘의 정의를 담고 있는 것으로 여러 에이전트간에 쓰이는 어휘를 공유하기 위한 것이다.

4. 에이전트를 이용한 회복 알고리즘

회복·조정·정보 에이전트는 프로세스가 생성될 때 함께 생성되고 프로세스가 작동하는 동안 수행한다는 가정을 한다. 에이전트들의 작동은 크게 정상 수행 시와 결함 발생 시 두 가지 경우로 구분할 수 있다.

4.1. 자료구조

N개의 프로세스 $p_i(i=1, 2, \ldots, N)$로 구성된 시스템에서 프로세스 p_i의 회복 에이전트는 RA_i로, 정보 에이전트는 IA_i로, 조정 에이전트는 FA_i로 표기한다. 알고리즘에서 사용되는 함수들과 자료구조는 <그림3>과 같다.

Data Structure

- a_i : 조정·정보·회복 에이전트를 통칭
- R : 조정 에이전트에 등록된 에이전트 집합
- $ProF$: 에이전트 프로파일 리스트
- $ProF_i$: 임의의 에이전트 a_i의 프로파일
- DK : 정보 에이전트의 영역지식
- C_i : 프로세스가 i 번째 취한 검사점
- m_i : 프로세스가 i 번째 수신하거나 송신한 메시지
- L_i : 프로세스가 i 번째 취한 로그
- *lastckp* : 결함발생 전 마지막으로 취한 검사점
- *eventlist* : 결함발생 전 마지막으로 취한 검사점 이후의 이벤트 리스트
- *rollbackpoint* : 결함발생 후 회복을 위해 복귀해야 하는 지점

<그림3> 자료구조

4.2. 정상 수행 시 알고리즘

정상 수행 시 알고리즘은 멀티 에이전트의 생성과 등록이 이루어지는 초기화 과정과 프로세스에서 발생한 이벤트 정보를 영역 지식으로 구축하는 과정으로 구성된다.

정상 수행 시 조정·정보·회복 에이전트는 프로세스가 생성될 때 함께 생성되고, 정보 에이전트와 회복 에이전트는 생성과 함께 조정 에이전트에게 자신들의 등록 요청을 한다. 조정 에이전트는 정보 에이전트와 회복 에이전트의 등록 요청 메시지가 오면 자신이 관리하는 에이전트 프로파일에 존재 여부를 확인한 뒤 등록되지 않은 에이전트라면 에이전트 프로파일에 추가한다.

초기화 과정이 완료되면 프로세스에서 발생하는 이벤트 정보를 영역 지식으로 구축하는 작업이 수행된다. 프로세스에서 이벤트가 발생할 때마다 조정 에이전트는 발생한 이벤트 정보를 정보 에이전트에게 전송한다. 프로세스의 이벤트 정보를 받은 정보 에이전트는 받은 정보를 KIF를 이용하여 자신의 영역 지식으로 추가한다. 또한 두 개 이상의 이벤트가 발생하면 이들의 선후관계 또한 영역지식으로 구축한다.

정상 수행 시 조정·정보 에이전트의 작동 알고리즘은 <그림4>와 같다.

```
Facilitator Agent FAi
Created with process pi;
  R ← ∅;
  ProF ← ∅;
  if receive register message from ai then
       if (ai ∉ R) then
          R ← R ∪ {ai};
          ProF ← ProF ∪ {ProFi};
       fi;
       send ack message to ai;
  fi;
 Do
   if receive system mesage from process pi then
     translate system message to KQML;
     send addDK_request message to IAi;
   fi;
 oD;

Information Agent IAi
Created with process pi;
 Do
   send register message to FAi;
   receive ack message from FAi;
   if receive addDK_request message from FAi then
     translate addDK_request message to KIF;
     add KIF to DK;
   fi;
 oD;
```

<그림4> 정상 수행 시 작동 알고리즘

4.3. 결함 발생 시 알고리즘

결함 발생 시 결함 발생 후 프로세스가 회복되면 조정 에이전트가 복귀회복 작업을 시작한다. 조정 에이전트는 회복 에이전트에게 복귀회복의 수행을 요청하고 회복 에이전트는 정보 에이전트에게 자신의 프로세스의 마지막 검사점 정보를 요청한다. 정보 에이전트는 영역 지식에서 프로세스의 마지막 검사점 정보를 추출해내어 회복 에이전트에게 알려준다. 마지막 검사점 정보를 받은 회복 에이전트는 정보 에이전트에게 마지막 검사점 이후에 프로세스에서 발생한 이벤트들의 리스트를 요청한다. 정보 에이전트는 영역 지식에서 마지막 검사점 이후에 발생한 모든 이벤트들을 추출하여 이벤트 리스트를 회복 에이전트에게 전송한다. 회복 에이전트는 조정 에이전트를 통해 프로세스를 마지막 검사점까지 복귀시키고 다른 조정 에이전트들에게 이벤트 리스트를 전송하여 각 조정 에이전트의 프로세스가 이벤트 리스트에 있을 경우 해당 이벤트의 재수행을 요청한다.

결함 발생시 조정 에이전트의 작동 알고리즘은 <그림5>와 <그림 6>과 같다.

```
Facilitator Agent FAi
 Do
   if receive recovery_request message from FAi then
     send lastckp_request message to IAi;
     if receive lastckp_reply message from IAi then
        send loglist_request message to IAi;
        if receive loglist_reply message to RAi then
          decide rollback_point;
          send rollback_request to pi;
          send replay_request to Fi;
          receive ack message to pi;
        fi;
     fi;
   fi
 oD;

Information Agent IAi
Do
  if receive lastckp_request message to IAi then
     find last_checkpoint from DK;
     translate last_checkpoint KIF to KQML;
     send lastckp_reply message to RAi;
  fi;
  if receive loglist_request message to IAi then
    find loglist after last_checkpoint from DK;
    translate loglist KIF to KQML;
    send loglist_reply message to RAi;
  fi;
 oD;
```

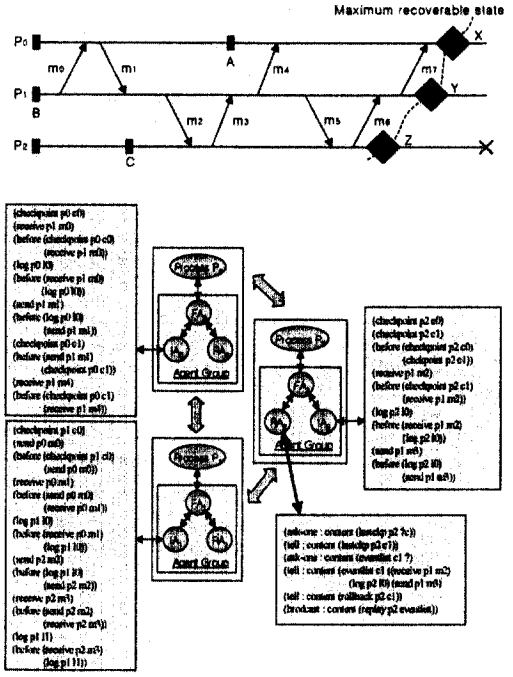
<그림5> 결함 발생 시 작동 알고리즘(1)

```
Recovery Agent RAi
 Do
   if receive recovery_request message from FAi then
      send lastckp_request message to FAi;
      wait lastckp_reply message from FAi;
      if receive lastckp_reply message from FAi then
         send loglist_request message to FAi;
         wait loglist_reply message to FAi;
      if receive loglist_reply message to FAi then
        decision rollback_point;
        send rollback_request to FAi;
        send replay_request to FAi;
        wait ack_message to FAi;
        receive ack_message to FAi;
      fi;
      fi;
   fi
 oD;
```

<그림6> 결함 발생 시 작동 알고리즘(2)

4.4. 제안된 복귀회복 알고리즘의 적용 예

<그림 7>은 멀티 에이전트를 이용한 회복 알고리즘의 적용 예이다. 전체 시스템은 프로세스 p_0, p_1, p_2로 구성되고 프로세스 p_2에서 결함이 발생한 경우이다.



<그림 7> 제안된 회복 알고리즘의 적용 예

5. 실 험

5.1. 실험 환경

본 논문에서는 에이전트들간의 통신 환경은 코바를 이용하고, 각각의 에이전트는 프로세스로 설계하여 에이전트를 이용한 복귀회복기법의 수행을 실험하였다. 실험을 수행하는 분산 컴퓨팅 시스템은 세 개의 노드로 구성되며, 각 노드의 시스템 환경은 <표4>와 같다.

<표4> 각 노드의 시스템 환경

구분	P0	P1	P2
IP	163.152.91.82	163.152.91.89	163.152.91.90
OS	Windows 2000	Windows 2000	Windows 2000
CPU	Intel P-II 400	Intel P-III 933	Intel P-III 933
RAM	256MB	192MB	192MB
HDD	13GB	20GB	20GB

<표5>는 본 논문의 실험에서 사용한 클래스들이다.

<표5> 실험에서 사용한 클래스

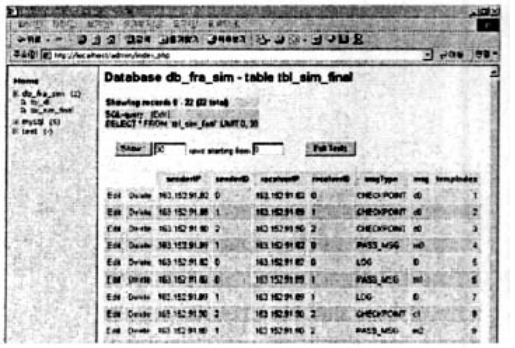
component	class name
simulator	manSim
	ProcessServer
process	ProcessImpl
	FAServer
Facilitator Agent	FAImpl
	RAServer
Recovery Agent	RAImpl
	IAServer
Information Agent	IAImpl
	ManDK
Communication	CommPathWithAgent
	CommPathForPrrocess

실험은 크게 정상 수행 시 에이전트의 초기화 작업 및 프로세스에서 발생한 이벤트 정보를 영역지식으로 구축하는 부분과 결함 발생 시 복귀회복과정을 수행하는 두 부분으로 나누어 수행하였다. 실험은 정상 수행 시 프로세스가 발생한 이벤트 정보를 조정 에이전트와 정보 에이전트의 협력을 통해 KIF로 영역 지식으로 정확히 구축하고 결함 발생 시 회복 에이전트가 정보·조정 에이전트와의 협력을 통해 영역지식에서 프로세스의 마지막 검사점을 찾아내고 그 이후 발생한 이벤트 리스트를 추출하여 복귀회복과정을 수행하는 것을 목표로 하였다.

5.2. 실험 결과 및 분석

본 실험을 통해서 멀티 에이전트를 이용하여 기존의 프로세스가 수행하던 복귀회복과정이 정상적으로 수행됨을 보인다.

<그림8>은 실험 시나리오를 입력하는 화면이다.



<그림 8> 시나리오 입력 화면

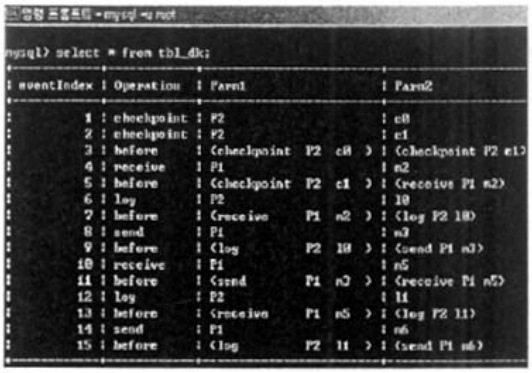
<그림9>는 정보 에이전트와 회복 에이전트가 조정 에이전트에 등록하는 과정을 나타낸다.

<그림 9> 에이전트들의 등록 화면

<그림10>은 정상 수행 시 정보 에이전트가 KQML을 이용하여 조정 에이전트로부터 프로세스의 이벤트 정보를 송·수신하는 모습이다.

<그림10> 정상 수행 시 에이전트의 통신

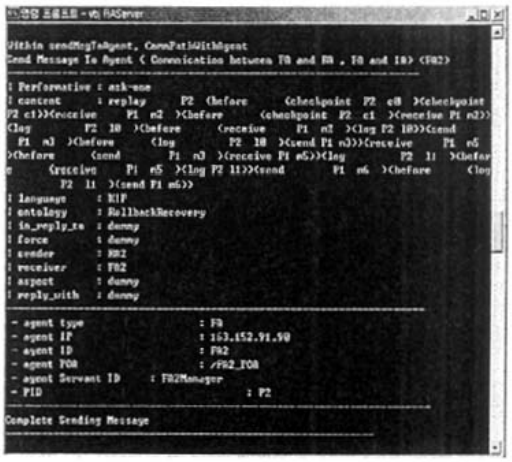
<그림11>은 정상 수행 시 프로세스 p2의 정보 에이전트가 프로세스 p2에서 발생한 이벤트 정보를 KIF를 이용하여 영역지식으로 구축하여 만든 데이터베이스이다.



<그림 11> 정상 수행 시 구축된 영역지식

<그림12>는 회복 에이전트가 정보 에이전트로 받은 마지막 검사점 이후의 이벤트 리스를 가지고 다른 프로세스들에게 재수행을 요청하는 메시지를 전송하는 과정을 나타낸다.

실험 결과 정상 수행 시 조정 에이전트와 정보 에이전트를 통해 각 프로세스에서 발생한 모든 이벤트 정보가 정확하게 영역 지식으로 유지되었다. 그리고 결함 발생 시는 회복 에이전트가 조정 에이전트와 정보 에이전트와의 협력을 통해 영역지식에서 자신의 프로세스의 최종 검사점 정



<그림12> 결함 발생 시 회복 과정

보와 그 이후 발생한 이벤트 리스트를 추출해서 결함이 발생한 프로세스에게는 마지막 검사점까지 복귀를 요청하는 메시지와 다른 프로세스들에게는 최종 검사점 이후 발생한 메시지의 재수행을 요청하는 메시지를 발생하였다. 이를 통해 분

산 컴퓨팅 시스템에서 결함이 발생한 경우 에이전트를 이용하여서도 복귀회복 과정을 수행할 수 있음을 확인할 수 있었다.

6. 결 론

본 논문에서는 기존의 분산 컴퓨팅 시스템의 결함 포용 기법에 멀티에이전트의 개념을 도입하여 회복·정보·조정 에이전트를 정의 및 설계하여 운영체제에 독립적인 에이전트를 이용한 회복 기법을 제안하였다. 에이전트를 이용한 회복기법은 결함 발생 프로세스의 결함 회복 작업을 사용자 영역에서 수행되는 서버가 담당하도록 하여 어플리케이션 계층과 독립적인 별도의 계층으로 계층화한다. 이와 같은 계층화는 분산 운영 체제의 핵심 기술이라 할 수 있는 마이크로커널의 구현을 위해 필수적인 사용자 영역에서의 서버의 시스템 기능성 제공에 에이전트를 도입하는 의미있는 시도가 된다. 또한 이는 현재 상용 분산 컴퓨팅 시스템을 비롯한 여러 분산 컴퓨팅 시스템에 결함 포용 기능의 탑재가 쉬어지는 이식성의 증대 및 확장성 증대를 가져온다.

향후 연구과제로는 낙관적 메시지 로깅 기법, 인과적 메시지 로깅 기법을 적용하여 회복 에이전트를 이용한 복귀회복기법을 연구하고, 각 복귀회복기법을 비교하는 성능평가를 수행하고자 한다.

참 고 문 헌

[1] L. Alvisi, "Understanding the message logging paradigm for masking process crashes," *Ph.D. Thesis, Department of Computer Science, Cornell University*, Jan. 1996.

[2] E. N. Elnozahy, D. B. Johnson and Y. M. Wang, "A Survey of Rollback-Recovery Protocols in Message Passing Systems," *CMU Technical Report CMU-CS-99-148*, June 1999.

[3] E. L. Elnozahy, W. Zwanepoel. "Manetho: Transparent rollback-recovery with low overhead, limited rollback and fast output commit," *IEEE Transactions on Computers*, 41(5):526-531, March 1992.

[4] L. Alvisi and K. Marzullo, "Message Logging: Pessimistic, Optimistic, Causal and Optimal," *IEEE Trans. on Software Engineering*, Vol. 24, pp. 149-159, Feb. 1998.

이 화 민

2000 고려대학교 컴퓨터교육과 (이학사)
2002 고려대학교 컴퓨터교육과(교육학석사)
2002~현재 고려대학교 컴퓨터교육과 박사과정
관심분야: 분산시스템, 에이전트, WBI
E-Mail: zelkova@comedu.korea.ac.kr

정 순 영

1990 고려대학교 전산과학과(이학사)
1992 고려대학교 전산과학과(이학석사)
1997 고려대학교 전산과학과 (이학박사)
1997~2000 (주) ECO 연구개발실장
2000~현재 고려대학교 컴퓨터교육과 조교수
관심분야: 데이터베이스, 웹기반 교육, 이동컴퓨팅시스템
E-Mail: jsy@comedu.korea.ac.kr

유 헌 창

1989 고려대학교 전산과학과(이학사)
1991 고려대학교 전산과학과(이학석사)
1994 고려대학교 전산과학과 (이학박사)
1995~1998 서경대학교 컴퓨터공학과 조교수
1998~현재 고려대학교 컴퓨터교육과 부교수
관심분야: 분산처리, 결함포용시스템, 컴퓨터교육
E-Mail: yuhc@comedu.korea.ac.kr