실시간객체 기반 결함허용 시스템 모델링

임 형 택† · 양 승 민††

요 약

내장 실시간 시스템의 경우 시스템의 오류로 인한 비정상적인 운행이 커다란 경제적 피해나 인명피해를 초래할 수 있으므로 시스템의 고 신뢰도 보장은 필수적이다. 이에 많은 결함허용 방법들이 연구되었는데 이들은 대부분 단위 객체에서 발생하는 오류만 감지하고 단위 객체 수준에서 오류를 처리하는 객체 수준 결함허용 방법이다. 그런데, 내장 실시간 시스템이 점점 더 복잡해지고 커짐에 따라 객체 수준 결함허용 방법만으로는 감지하거나 처리할 수 없는 결함과 오류가 존재한다. 따라서, 여러 객체들의 상태에 대한 분석을 통해서 시스템의 상태가 정상인지 비정상인지를 판단하고 오류가 발생했을 때에는 오류의 원인이 되는 결함을 찾은 다음 알맞은 복구 및 재배치 방법을 결정하는 시스템 수준 결함허용이 필요하다.

본 논문에서는 객체 수준 결함허용 기능을 제공하는 RobustRTO(Robust Real-Time Object)와 시스템 수준 결함허용 기능을 제공하는 RMO(Region Monitor real-time Object)를 제안하고 이들을 이용하여 객체 수준 결함허용 기능뿐만 아니라 시스템 수준 결함허용 기능도 갖는 고 신뢰도의 결함허용 시스템을 모델링할 수 있음을 보인다. 본 논문은 객체 모델의 장점과 실시간성 표현이 용이한 실시간객체(RTO 또는 Real-Time Object) 모델에 기반한다.

Fault Tolerant System Modeling based on Real-Time Object

Hyung-Taek Lim† · Seung-Min Yang††

ABSTRACT

It is essential to guarantee high reliability of embedded real-time systems since the failure of such systems may result in large financial damage or threaten human life. Though many researches have devoted to fault tolerant mechanisms, most of them are object-level fault tolerant mechanisms that can detect errors occurred in a single object and treat the errors in object level. As embedded real-time systems become more complex and larger, there exist faults that cannot be detected by or tolerated with object-level fault tolerance. Hence, system-level fault tolerance is needed. System-level fault tolerance examines the status of a system whether the system is normal or not by analyzing the status of objects. When an error is detected it should be capable of locating the fault and performing an appropriate recovery and reconfiguration action.

In this paper, we propose RobustRTO (Robust Real-Time Object) that provides object level fault tolerance capability and RMO (Region Monitor real-time Object) that offers system-level fault tolerance capability. Then we show how highly dependable fault tolerant systems can be modeled by RobustRTO and RMO. The model is presented based on real-time objects.

※ 본 연구는 1998년 숭실대학교 교내학술연구비 지원으로 수행되었습니다.
† 준 회 원 : 숭실대학교 대학원 컴퓨터학과
†† 정 회 원 : 숭실대학교 정보과학대학 컴퓨터학부 교수
논문접수 : 1999년 8월 24일, 심사완료 : 1999년 7월 6일

1. 서 론

내장 실시간 시스템의 응용은 매우 다양하며 신뢰도에 대한 요구사항 또한 다양하다. 많은 내장 실시간 시스템의 경우 시스템의 오류로 인한 비정상적인 운행이 커다란 경제적 피해나 인명 피해를 초래할 수 있으므로 시스템의 고 신뢰도 보장은 필수적이다.

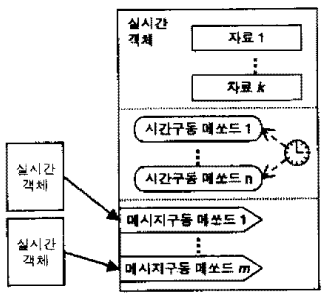
지난 30여년간 결함허용 방법에 대한 많은 연구가 있었지만 이들은 대부분 객체 수준에서만 결함을 처리한다. 객체 수준 결함허용이란 객체 내에서 발생하는 결함을 객체 내부에서 처리함으로써 객체 외부에서 인지할 수 없도록 하는 것이다. 즉, 어떤 객체에서 오류가 감지되었을 때 오류의 원인인 결함도 같은 객체에 있다고 가정하고 결함을 처리한다. 여기서 객체란 결함허용을 위하여 복제되거나 교체될 수 있는 최소 단위를 말하며 그 크기는 설계자에 의해 결정된다. 예를 들면, 노드에서 결함이 발생했을 때 노드 전체를 다른 노드로 교체하는 경우 노드가 객체가 된다.

그런데, 내장 실시간 시스템이 점점 더 복잡해지고 커짐에 따라서 운행 중인 내장 실시간 시스템에서는 다양한 오류가 발생하고 그 원인도 다양하므로 객체 수준 결함허용만으로는 고 신뢰도를 만족시키기 어렵다. 왜냐하면, 한 객체에 대한 감시로는 감지할 수 없고 여러 객체들의 상태에 대한 비교와 분석을 통해서만 감지할 수 있는 오류나 다른 객체로부터 전파되어 발생하는 오류가 존재하기 때문이다. 따라서, 고 신뢰도의 내장 실시간 시스템을 구축하기 위해서는 객체 수준 결함허용 기능과 함께 시스템 수준 결함허용 기능도 필요하다. 시스템 수준 결함허용이란 여러 객체들의 상태에 대한 분석을 통해서 시스템의 상태가 정상인지 비정상인지를 판단하고 오류가 발생했을 때에는 오류의 원인이 되는 결함을 찾은 다음 알맞은 복구 및 재배치 방법을 결정하는 것이다.

본 논문에서는 객체 수준 결함허용 기능을 제공하는 RobustRTO(Robust Real-Time Object)와 시스템 수준 결함허용 기능을 제공하는 RMO(Region Monitor real-time Object)를 제안하고 이들을 이용하여 객체 수준 결함허용 기능뿐만 아니라 시스템 수준 결함허용 기능도 갖는 고 신뢰도의 결함허용 시스템을 모델링한다.

본 논문은 실시간객체 모델에 기반한다. 실시간객체 모델은 객체에 실시간성을 표현할 수 있게 한 모델로써 객체 모델의 장점을 가지며 실시간성 표현이 용이하다. 실시간객체 모델에서 하나의 시스템은 여러 개의 하드웨어 및 소프트웨어 실시간객체들로 표현된다. 대부분의 결함허용 시스템은 내장 실시간 시스템이므로 실시간객체 모델은 결함허용 설계에도 적합하다. RobustRTO와 RMO도 실시간객체이므로 응용 설계와 결함허용 설계 모두를 실시간객체 모델로 할 수 있으므로 일관성 있는 설계를 할 수 있다. 또한, 객체 수준 결함허용과 시스템 수준 결함허용에 대한 설계가 명확히 분리된다는 점과 시스템 구성과는 독립적으로 모델링이 가능하다는 장점도 있다.

실시간객체 모델에 대한 연구로는 TMO(Time-triggered Message-triggered Object) 모델[5]과 dRTO (dependable Real-Time Object) 모델[6]이 있는데, 본 논문은 dRTO 모델에 기반한다. dRTO 모델은 신뢰도 높은 내장 실시간 시스템의 효율적인 구축을 위하여 객체지향과 실시간성, 그리고 신뢰성의 세 가지 기본 개념을 단일 모델에 통합한 것이다. dRTO 모델에서는 내장 실시간 시스템의 모든 하드웨어 및 소프트웨어가 실시간객체로 추상화된다. (그림 1)처럼 하나의 실시간 객체는 자료와 실시간메쏘드로 구성된다. 실시간메쏘드는 명세된 시간에 따라 구동되는 시간구동 메쏘드(TM 또는 Time-triggered Method)와 메시지에 의하여 구동되는 메시지구동 메쏘드(MM 또는 Message-triggered Method)로 구분된다. 실시간객체들 간의 통신은 메시지구동 메쏘드의 호출을 통해서 이루어지며 메시지구동 메쏘드가 호출될 때마다 실시간쓰레드가 생성된다. 메시지구동 메쏘드를 호출하는 방식에는 막힘 호출(blocking call)과 안막힘 호출(nonblocking call)의 두 가지가 있다. 실시간메쏘드는 실시간 처리 특성에 따라 긴급, 경성, 연성의 세 실시간 그룹으로 나뉜다. 시간구동 메쏘드는 항상 경성이며 설계시 정해진 실시간 특성에 따라 수행된다. 메시지구동 메쏘드의



(그림 1) dRTO 모델의 실시간객체

호출에 의해 생성된 실시간쓰레드는 긴급, 정상, 연성 중 하나의 실시간 그룹을 가지며 실시간쓰레드의 실시간 그룹과 실시간 특성은 호출자에 의해 결정된다.

본 논문에서는 RobustRTO와 RMO의 개념과 역할을 제시한 후 이들을 이용하여 고 신뢰도의 결함허용 시스템을 모델링할 수 있음을 보인다. 2장에서 관련 연구를 소개한 후 3장에서RobustRTO와 RMO의 개념과 역할을 제시한다. 4장에서는 RobustRTO와 RMO를 모델링하는 방법을 설명하고 5장에서 RobustRTO와 RMO를 모델링한 예를 보인다. 6장에서 본 논문의 결론과 향후 과제를 설명한 후 마친다.

2. 관련 연구

객체 수준 결함허용 방법들은 복제(replica)를 이용한다. 즉, 원본객체에서 결함이 발생하더라도 여분의 정상적인 복제객체가 대신 수행하게 하는 것이다. 객체 수준 결함허용 방법은 결함을 처리하는 방식에 따라서 결함 숨기기(fault masking) 방법, 결함 감지 및 복구(fault detection & recovery) 방법, 합성(hybrid) 방법으로 나뉘며 처리하는 결함의 특성에 따라서 하드웨어 결함허용 방법과 소프트웨어 결함허용 방법으로 구분된다. 하드웨어 결함허용 방법에서는 동일한 복제객체를 여러 개 두는 것으로 충분하지만, 소프트웨어 결함허용 방법에서는 반드시 복제객체가 서로 다른 버전으로 설계 및 구현되어야 한다.

결함 숨기기 방법은 세 개 이상의 복제객체가 동일한 작업을 한 후 그 결과값을 투표하여 다수에 해당하는 값을 선출하는 것이다. 하드웨어 결함 숨기기 방법으로는 TMR(Triple Modular Redundancy)이 있고, 소프트웨어 결함허용 방법으로는 N 버전 프로그래밍(NVP 또는 N-Version Programming)[2]이 있다.

결함 감지 및 복구 방법은 두 개 이상의 복제객체를 사용하며 능동적으로 결함을 감지한 후 결함 있는 복제객체를 정상적인 복제객체로 교체한다. 하드웨어 결함 감지 및 복구 방법으로는 standby sparing, self-checking pair가 있고, 소프트웨어 결함 감지 및 복구 방법으로는 복구블럭(RB 또는 Recovery Block)[1]과 N 자가검사 프로그래밍(NSCP 또는 N Self Checking Programming)[3]이 있다. 결함 감지 및 복구 방법에서 복제객체들은 랭크(rank)를 갖으며 복제객체 중 하나가 주복제객체(primary 또는 active)가 되고, 나머지 복제객체는 부복제객체(backup 또는 standby)가 된다. 시스템이 처음 시작할 때에 랭크가 1인 복제객체가 주복제객체의 역할을 한다. 랭크가 i인 주복제객체에서 결함이 발생하면 랭크가 i+1인 부복제객체가 주복제객체의 역할을 수행한다. Standby sparing과 복구블럭에서 주복제객체와 부복제객체들이 모두 동시에 수행되는 경우를 각각 hot standby sparing과 병렬 복구블럭이라고 하고 주복제객체에서 결함이 감지된 후에 다음 부복제객체가 수행을 시작하는 경우를 각각 cold standby sparing과 순차 복구블럭이라 한다. 복구블럭은 복제객체의 결과값에 대한 수용검사(AT 또는 Acceptance Test)를 통하여 결함을 감지한다. Self-checking pair와 NSCP에서는 두 복제객체가 한 쌍이 되며 결함 감지를 위하여 두 복제객체의 결과값을 비교한다. NSCP에서 한 쌍은 서로 다른 버전의 복제객체로 구성된다.

소프트웨어 복제객체는 관련된 하드웨어에 결함이 발생하면 정상적으로 수행될 수 없다. 따라서, 소프트웨어 복제객체가 하드웨어 결함의 영향을 받지 않도록 소프트웨어 복제객체들을 서로 다른 노드에 분산시킬 수 있다. 예를 들면, N 버전 프로그래밍에서 각 복제객체는 서로 다른 노드에 분산될 수 있다.

분산복구블럭(DRB 또는 Distributed Recovery Block)[4]은 하드웨어 결함과 소프트웨어 결함을 모두 허용하기 위한 방법으로써 두 개의 노드와 두 가지 버전으로 구성된 네 개의 소프트웨어 복제객체를 이용한다. 각 노드에는 서로 다른 버전의 소프트웨어 복제객체가 존재하는데, 주노드에는 첫 번째 버전이 주복제객체가 되고, 부노드에서는 두 번째 버전이 주복제객체가 된다. 네 개의 소프트웨어 복제객체는 모두 병렬적으로 수행되며 정상적인 경우 주노드의 주복제객체가 결과값을 전송한다. 주노드의 주복제객체에서 결함이 감지된 경우 부노드의 주복제객체가 결과를 전송하고 부노드와 주노드는 역할을 바꾼다.

합성 방법인 NMR with spare의 경우에는 n개의 복제객체가 투표를 통해 올바른 값을 선출하며 n개의 복제객체 중 결함이 감지된 복제객체를 대체하기 위하여 m개의 예비 복제객체가 존재한다. 즉, NMR with spare는 총 $n+m$개의 복제객체로 구성된다.

근래에는 재사용성이 좋고 유연한 결함허용 시스템을 개발할 수 있도록 결함허용 시스템의 설계에 객체지향 개념을 도입한 연구들이 진행되고 있다. [9]에서는 분산복구블럭에 기반한 PSTR(Primary-Shadow

TMO Replication) 방법을 실시간 객체 모델인 TMO를 이용하여 모델링하였다. metaobject[11]는 reflection 개념[10][12]에 근거한 것으로써 메타 수준의 객체인 metaobject는 베이스 수준 객체의 행위를 관찰하고 조절할 수 있다. 순수한 시스템의 기능은 베이스 수준 객체에서 구현되고, 결함허용 기법과 같이 시스템의 비기능적인 부분은 metaobject에서 구현된다. 따라서, metaobject의 결함허용 설계는 베이스 수준 객체의 설계에 영향을 미치지 않고 다양한 객체 수준 결함허용 기능이 metaobject에 구현될 수 있다. 그런데, [9]와 [11]은 객체 수준 결함허용 방법만을 모델링한 것이고 시스템 수준 결함허용 방법을 모델링할 수 있는 방법은 제시되지 않았다.

3. RobustRTO와 RMO

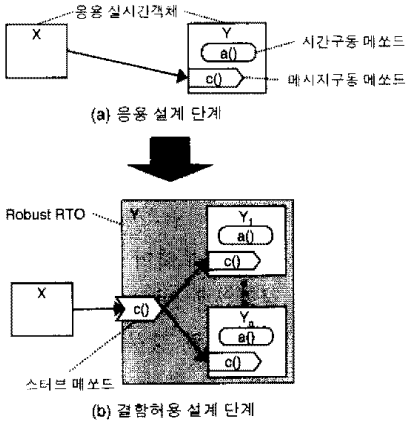
본 장에서는 RobustRTO와 RMO의 개념과 역할에 대하여 설명한다.

3.1 RobustRTO

RobustRTO는 객체 수준 결함허용 능력을 갖는 실시간객체로써 단위 실시간객체 내에서 발생하는 결함을 처리한다. 즉, 응용 실시간객체가 객체 수준 결함허용 능력을 갖게 해준다. RobustRTO는 결함허용과 관련된 모든 정보를 자신 안에 캡슐화함으로써 응용 실시간객체의 설계와 결함허용 설계를 분리한다. 따라서, 결함허용을 위하여 응용 실시간객체가 복제된 경우에도 기존 응용 실시간객체의 인터페이스를 그대로 이용할 수 있도록 해주며 결함허용 구성에 따라 응용 실시간객체의 설계가 변경되지 않게 해준다. 복제객체들은 RobustRTO 안에 물리적으로 포함된 것이 아니고 논리적으로 캡슐화된 것이므로 복제객체들은 서로 다른 노드에 분산되어 수행될 수 있다.

결함허용 구성의 캡슐화를 위하여 RobustRTO의 이름은 캡슐화된 응용 실시간객체의 이름이 된다. 그리고, RobustRTO는 캡슐화된 응용 실시간객체의 메시지구동 메쏘드와 이름이 동일한 메시지구동 메쏘드를 포함하는데 이를 스터브 메쏘드(stub method)라고 한다. 스터브 메쏘드란 외부에서 결함허용 구성에 상관없이 투명하게 RobustRTO 안에 캡슐화된 복제객체들을 접근할 수 있게 해주는 메시지구동 메쏘드이다.

(그림 2)는 응용 실시간객체 Y가 결함허용 능력을 갖게 하기 위하여 RobustRTO를 구성하는 과정을 보여준다. (그림 2) (b)와 같이 응용 실시간객체 Y는 복제되면서 Y_1, Y_2, ..., Y_n(n은 복제된 실시간객체의 개수이다)의 이름을 갖게 되고 RobustRTO의 이름은 Y가 된다. RobustRTO는 복제객체들의 메시지구동 메쏘드인 c()와 이름이 동일한 스터브 메쏘드를 갖는다. (그림 2) (b)에서 X가 Y.c()를 호출하면 RobustRTO Y의 스터브 메쏘드 c()가 실행된다. 스터브 메쏘드 Y.c()는 RobustRTO의 결함허용 구성 방식에 따라서 복제객체들에 있는 자신과 동일한 이름을 지닌 메쏘드(예를 들면, (그림 2) (b)에서 Y_1.c()나 Y_n.c())를 호출한 후 올바른 결과값을 X에게 돌려준다. 이와 같이 RobustRTO가 결함허용 구성을 캡슐화하므로써 응용 설계는 결함허용 구성에 영향을 받지 않는다.



(그림 2) RobustRTO의 구성 과정

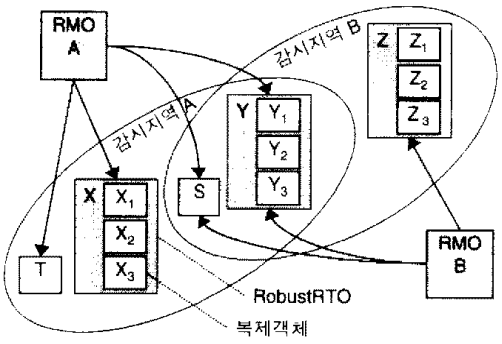
3.2 RMO

RMO는 시스템 수준 결함허용을 수행하는 실시간객체이다. 시스템 수준 결함허용을 위하여 설계 시에 물리적 또는 논리적으로 연관이 있는 실시간객체들의 집합인 감시지역(region)을 설정한다. 각 감시지역마다 하나의 RMO가 정의되며, RMO는 감시지역에 소속된 실시간객체들에 대하여 시스템 수준 결함허용을 수행한다.

3.2.1 감시지역

감시지역은 시스템 수준 결함허용을 위하여 설계 시에 물리적 또는 논리적으로 연관이 있는 실시간객체들로 구성되며, 감시지역을 구성하는 목적에 따라서 오류를 감지하기 위한 것과, 오류가 발생했을 때 결함을

지닌 실시간객체를 찾기 위한 것의 두 가지가 있다. 감시지역을 구성하는 실시간객체들을 감시지역의 멤버 객체라고 한다. 감시지역의 멤버객체로는 응용 실시간 객체((그림 3)에서 T와 S)나 RobustRTO((그림 3)에서 X, Y, Z)가 될 수 있다. RobustRTO 안에 캡슐화된 복제객체 중에서 일부만이 감시지역의 멤버객체가 될 수는 없다. 예를 들면, (그림 3)에서 Z_1과 Z_2만 감시지역 B에 소속되고 Z_3은 감시지역 B에 소속되지 않는 경우는 잘못된 것이다. 감시지역은 서로 겹칠 수 있다. 따라서, 한 감시지역의 멤버객체는 동시에 다른 감시지역에도 소속될 수 있다((그림 3)에서 S와 Y). 감시지역이 구성되면 시스템 수준 결함허용을 수행할 RMO가 정의된다.



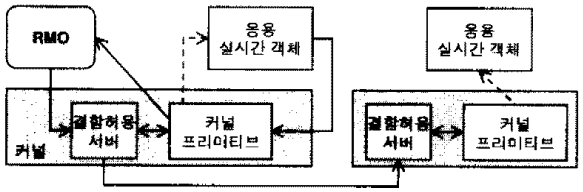
(그림 3) 감시지역의 예

3.2.2 RMO의 기능

RMO의 기능은 감시, 진단, 복구, 기록, 비상사태 처리의 다섯 가지로 나뉜다. **1) 감시 기능** : RMO는 실시간객체의 소속 자료와 예외, 그리고 실시간 메시지를 감시한다. **2) 분석 및 진단 기능** : 감시를 통해 얻어진 정보를 분석하여 오류나 비정상적인 상태를 감지한다. 오류가 감지되면 관련된 실시간객체들을 진단하여 결함을 갖고 있는 실시간객체를 찾고 알맞은 복구 및 재배치 방법을 결정한다. **3) 복구 및 재배치 기능** : RMO는 결함을 지닌 실시간객체들을 제어하여 이들을 복구 또는 재배치한다. 실행 중인 실시간객체를 다른 노드로 이주시키거나 hot standby 방식으로 이중화되어 있는 경우 백업 노드와 주 노드를 교체한다. **4) 비상사태 처리 기능** : 복구나 재배치가 불가능하여 시스템의 정상적인 운행을 보장할 수 없는 경우 RMO는 시스템을 안전하게 정지시키기 위한 비상사태 처리를 수행한다. **5) 기록 기능** : RMO는 감시, 진단 및 복구 결과 등을 기록한다.

3.2.3 RMO의 권한

위에서 설명한 RMO의 다섯 가지 기능을 지원하기 위하여 RMO는 실시간 메시지와 실시간객체의 소속 자료, 예외, 그리고 실시간쓰레드를 감시하고 제어할 수 있는 권한을 갖는다. 이와 같은 권한은 커널과 결함허용 서버의 지원에 의해 이루어진다. 결함허용 서버는 RMO의 권한이 수행되는데 필요한 정보와 기능을 제공할 뿐만 아니라, 원격 노드의 커널 프리미티브를 호출할 수 있게 해준다. RMO는 소속 자료나 실시간쓰레드에 접근하기 위하여 결함허용 서버를 호출한다. 결함허용 서버는 소속 자료나 실시간쓰레드가 같은 노드에 존재하면 직접 커널 프리미티브를 호출하고 원격 노드에 존재하는 경우 원격 노드의 결함허용 서버에게 요청을 전달한다. 그러면, 원격 노드의 결함허용 서버가 커널 프리미티브를 호출한다. 응용 실시간 객체에서 발생한 실시간 메시지와 예외는 커널에 의해 감지되며 커널은 이들을 감시 중인 RMO에게 알려준다.

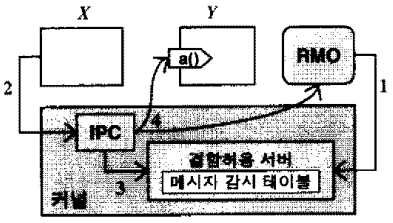


(그림 4) RMO의 시스템 수준 결함허용에서의 기본 구조

(1) 실시간 메시지 감시

RMO는 응용 실시간객체의 메시지구동 메쏘드를 호출할 때 발생하는 실시간 메시지를 감시한다. 실시간 메시지가 발생하면 이는 피호출자 뿐만 아니라 RMO에게도 전달되어야 한다. 실시간 메시지는 커널에 의하여 피호출자에게 전달되므로 RMO에게 실시간 메시지를 전달해주는 것도 커널이 담당한다. 결함허용 서버의 *실시간 메시지 감시 테이블*에는 노드에서 발생한 실시간 메시지 중에서 RMO에게도 전달해야 하는 실시간 메시지들이 기록되어 있다. 실시간 메시지 감시 테이블은 실시간 메시지의 1) 호출자와 2) 피호출자, 그리고 3) 실시간 메시지를 감시하는 RMO들의 리스트로 구성된다. (그림 5)는 실시간 메시지를 감시하는 구조를 보여준다. 1) RMO는 수행을 시작할 때 자신이

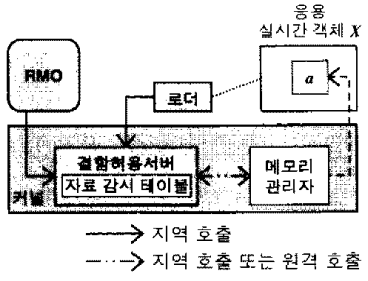
감시해야 하는 실시간 메시지 [X, Y.a()]를 호출자 X가 수행될 노드에 존재하는 결함허용 서버의 실시간 메시지 감시 테이블에 등록한다. 2) 실시간 메시지가 발생하면 (응용 실시간객체 X가 응용 실시간객체 Y의 메시지구동 메쏘드 a()를 호출), 3) 커널은 실시간 메시지 감시 테이블에서 실시간 메시지를 감시하는 RMO를 알아낸다. 4) 그런 다음 응용 실시간객체 Y와 RMO에게 실시간 메시지를 전송한다.



(그림 5) 실시간 메시지의 감시

(2) 소속 자료에 대한 접근

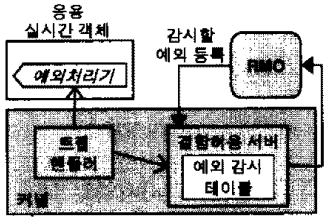
RMO는 감시지역에 소속된 실시간객체의 상태를 주기적으로 비교함으로써 시스템의 비정상적인 상태를 감시한다. 이를 위하여 RMO는 실시간객체의 소속 자료를 감시할 수 있는 권한을 갖는다. RMO는 메모리 관리자를 통해서 실시간객체의 소속 자료에 접근한다 (그림 6). 결함허용 서버에는 같은 노드에 존재하는 실시간 객체의 소속 자료의 1) 이름과 2) 가상 주소로 구성되는 *자료 감시 테이블*이 존재한다. 로더는 응용 실시간객체를 적재할 때 소속 자료들에 대한 가상 주소를 자료 감시 테이블에 기록한다. 이를 위하여 응용 실시간객체의 실행 파일에는 소속 자료들에 대한 가상 주소 리스트가 있어야 하고 로더는 이를 인식할 수 있어야 한다. RMO는 자신이 접근하려고 하는 자료(그림 6에서 *X.a*)를 결함허용 서버에게 알려준다. 원격 노드에 있는 응용 실시간객체의 소속자료를 접근하는 경우 결함허용 서버는 원격 노드에 있는 결함허용 서버에게 소속 자료를 알려준다. 결함허용 서버는 소속 자료의 가상 주소를 자료 감시 테이블에서 알아낸 후 이를 메모리 관리자에게 전달한다. 메모리 관리자는 결함허용 서버에게 소속 자료의 값을 리턴하거나 소속 자료의 값을 갱신한다.



(그림 6) 소속 자료 감시

(3) 예외 감시

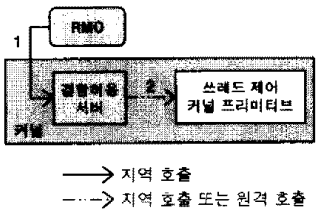
RMO는 응용 실시간객체의 수행 중에 발생하는 예외를 감시한다. 트랩핸들러는 예외를 감지한 후 예외처리기를 찾아서 실행시켜 준다. RMO가 예외를 감시할 수 있도록 트랩핸들러는 발생한 예외를 결함허용 서버에게 알려준다. 결함허용 서버는 *예외 감시 테이블*을 참조하여 예외를 전달할 RMO를 알아낸 후 예외를 RMO들에게 전송한다. 예외 감시 테이블은 응용 실시간객체의 메쏘드와 예외, 그리고 RMO들의 리스트로 구성된다. RMO는 수행을 시작할 때 먼저 감시하려는 예외를 결함허용 서버에 등록한다.



(그림 7) 시스템 예외 감시

(4) 실시간쓰레드 제어

RMO는 실시간쓰레드에서 오류가 발생했을 때 실시간쓰레드를 제어함으로써 오류의 전파를 막거나 오류를 복구한다. 이를 위하여 RMO는 실행 중인 실시간쓰레드를 일시정지, 재시작, 또는 종료시킬 수 있다. RMO가 결함허용 서버에게 쓰레드 제어를 요청하면 결함허용 서버는 쓰레드가 수행 중인 노드를 찾아낸 후 해당하는 노드의 쓰레드 제어 커널 프리미티브를 호출한다.



(그림 8) 실시간쓰레드 제어

4. RobustRTO와 RMO의 모델링

4.1 RobustRTO의 모델링

RobustRTO는 다양한 객체 수준 결함허용 기능을 갖는다. 따라서, RobustRTO를 이용하여 다양한 객체 수준 결함허용 방법들을 모델링할 수 있어야 한다. 본 절에서는 RobustRTO의 모델링에서 표현되어야 할 요소들을 설명한 후 RobustRTO의 모델링을 위한 명세틀인 RobustRTOspec을 제시한다.

4.1.1 RobustRTO의 모델링 요소

RobustRTO의 모델링에서는 다음과 같은 결함허용 구성 요소들이 표현되어야 한다.

가) RobustRTO 안에 캡슐화된 복제객체의 수와 버전의 수, 그리고 예비 복제객체의 수를 명세한다. 결함허용 방법 중에서 NSCP의 경우 4개의 복제객체가 3개의 버전으로 구성되고, 분산복구블럭의 경우 4개의 복제객체가 2개의 버전으로 구성되기도 한다. 결함숨기기와 결함감지 및 복구 기능이 혼합된 NMR with spare의 경우에는 예비 복제객체가 존재한다.

나) 복제객체들이 소프트웨어인지 하드웨어인지 여부가 표현되어야 한다.

다) 복제 방식이 표현되어야 한다. 복제객체들이 완전히 동일한 복제인 경우 동일복제라 하며 다른 버전의 복제일 경우 버전복제라 한다.

라) 결함 감지 및 복구 방법의 경우 복제객체나 복제객체쌍, 또는 노드에 랭크가 표시된다.

마) 복제객체들이 소프트웨어인 경우 이들이 어느 노드에 분산되어 수행될 것인지 표현되어야 한다.

바) 결함감지 및 복구 방법의 경우 병렬 실행이나 순차 실행 여부가 표현되어야 한다. 병렬 실행이란 모든 복제객체들이 동시에 실행되는 경우이고, 순차 실행이란 랭크가 i인 복제객체에서 결함이 감지되었을 때에만 랭크가 i+1인 복제객체가 실행되는 경우이다.

사) 스터브 메쏘드가 결함을 감지하고 처리하는 방식이 표현되어야 한다. 결함숨기기 방법에서는 투표를 통하여 다수의 결과값을 RobustRTO의 결과값으로 선택함으로써 결함을 처리한다. 결함감지 및 복구 방법에서는 수용검사나 비교를 통하여 결함을 감지한다. 수용검사에 성공하거나 비교에서 일치하는 복제객체의 결과가 RobustRTO의 결과로 선택된다.

4.1.2 그래픽 표기법

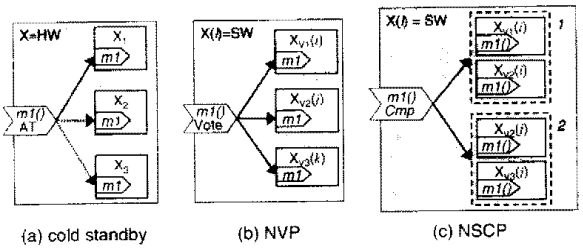
실시간객체(RobustRTO나 복제객체)와 실시간메쏘드는 (그림 1)과 같이 표현한다. 캡슐화된 복제객체들은 (그림 2)와 같이 RobustRTO 안에 그리고, 스터브 메쏘드는 한쪽이 오목한 이정표 모양의 도형으로 표현한다. 복제객체가 하드웨어인지 소프트웨어인지는 RobustRTO의 이름 옆에 X=(SW|HW)와 같은 형식으로 표현한다. 복제 객체의 이름은 동일복제된 경우 X_1, X_2, ..., X_n과 같이 응용 실시간객체의 이름에 아래 첨자 번호를 붙여서 나타내고((그림 9) (a)), 버전복제의 경우 X_{V1}, X_{V2}, ..., X_{Vn}과 같이 아래 첨자 번호 앞에 'V'를 붙인다((그림 9) (b), (c)).

결함 처리 방식이 투표인 경우는 "Vote", 수용검사인 경우는 "AT", 그리고 비교인 경우는 "Cmp"라고 스터브 메쏘드에 표시한다. 비교를 통하여 결함을 감지하는 경우 (그림 9) (c)처럼 각 복제객체 쌍을 점선으로 된 직사각형으로 표시한다.

랭크 r은 X_{Vr}과 같이 복제객체의 이름의 오른쪽에 위첨자를 사용하여 표시한다. 복제객체쌍의 랭크는 (그림 9) (c)의 NSCP처럼 점선으로 된 직사각형의 오른쪽 위에 표시한다. 즉, 주복제객체쌍 (X_{V1}, X_{V2})의 결과값이 불일치하면 부복제객체쌍 (X_{V2}, X_{V3})이 주복제객체쌍이 되어 결과값을 전송한다.

실시간객체가 수행될 노드는 실시간객체 이름 옆의 괄호 안에 알파벳으로 명세한다. (그림 9) (b)의 경우 RobustRTO X와 캡슐화된 세 복제객체들은 모두 서로 다른 노드에서 수행된다는 것을 나타낸다. (그림 9) (c)에서는 노드 i에서 X_{V1}과 X_{V2}가 수행되고, 노드 j에서 X_{V2}와 X_{V3}가 수행됨을 나타낸다. 노드에 랭크가 있는 경우 (i)'처럼 위첨자로 표시한다.

복제객체들이 순차 수행되는 경우는 (그림 9) (a)처럼 스터브 메쏘드에서 주복제객체로 향하는 화살표만 실선으로 표시하고, 부복제객체로 향하는 화살표는 점선으로 표시한다. 즉, (그림 9) (a)에서 부복제객체인 X_2와 X_3는 X_1이 정상적으로 수행되는 경우에는 동작하지 않고 있다가 X_1에서 결함이 감지되었을 때에 X_2가 수행을 시작한다. 복제객체들이 병렬 수행되는 경우에는 (그림 9)의 (b)와 (c)처럼 스터브 메쏘드에서 동시에 수행될 복제객체 또는 복제객체 쌍으로 화살표를 그린다.



(그림 9) 그래픽 표기법으로 표현된 RobustRTO의 예

4.1.3 RobustRTOspec

RobustRTOspec은 RobustRTO의 모델링을 위한 명세틀로써 응용 실시간객체에 대한 명세나 설계가 완료된 후에 응용 실시간객체에 대한 결함허용 설계 시 작성된다. RobustRTOspec으로부터 dRTO 형태[1]의 RobustRTO가 만들어진다. (그림 10)과 같이 RobustRTOspec은 replication 부분, fault_handling 부분, stub 부분, comment 부분의 네 부분으로 나뉜다.

```
RobustRTOspec name {
replication:
  l replica [, m version ] [, n spare];
 [ HW | SW ] <structure>;
fault_handling:
  ( <Vote> | <AT> | <Cmp> ) ;
stub:
  <stub-list>
   stub() {
   . . . .
   }
Comment:
}
```

(그림 10) RobustRTOspec

(1) Replication 부분

Replication 부분은 복제객체의 특성과 구성에 관련된 것들이 명세된다. 먼저 복제객체의 개수(*l* **replica**), 버전의 개수(*m* **version**), 예비복제객체의 개수(*n* **spare**)가 명세된다. 버전과 예비복제객체의 개수는 생략될 수 있다. 버전의 개수가 생략되면 *m*=1이 되며 예비복제객체의 개수가 생략되면 *n*=0이 된다. 다음으로 복제객체들이 하드웨어인지 소프트웨어인지를 나타낼 수 있다.

<structure>에서는 복제객체들의 구성을 명세한다.[2] 버전복제된 복제객체는 V1, V2와 같이 대문자 "V" 옆에 숫자를 붙여서 구분한다. 소프트웨어 복제객체들이 분산 수행되는 경우 **distributed** 라는 키워드를 쓴다. 대괄호 [ ] 안에 한 노드에서 수행될 버전이나 동일복제인 경우 복제객체들의 개수를 쓴다. 버전복제인 경우 "[V1, V2]"와 같이 한 노드에서 수행될 버전을 표시하고, 동일복제인 경우 "[3]"과 같이 한 노드에서 수행될 복제객체의 수를 표시한다.

```
<structure> → distributed [ <vnodes> | <inodes> ]
<vnodes> → <vnode> {, <vnode> }
<vnode> → \[ <ver> {, <ver> } \]^rank
<inodes> → \[ number \] {, \[ number \] }
<ver> → V1^rank
```

예를 들면, 분산복구블럭에서 4개의 복제객체가 2개의 버전으로 구성되고 이들이 2개의 노드에 분산된다. 그리고, 각 노드에는 서로 다른 버전의 복제객체 2개가 수행된다. 이 경우 "4 **replica**, 2 **version**; SW **distributed** $[V1^1,V2^2]^1$, $[V2^1,V1^2]^2$;"으로 명세한다. 동일복제된 네 개의 복제객체가 두 개의 노드에서 수행되는 경우 "[2],[2]"와 같이 표현한다. N 버전 프로그래밍 방법에서 3 개의 복제객체들이 모두 서로 다른 노드에 분산되어 있는 경우 "3 **replica**, 3 **version**; SW **distributed**;"와 같이 나타낸다.

(2) Fault_handling 부분

Fault_handling 부분은 결함을 어떻게 처리할 것인지를 나타낸다. 결함 처리 방식으로는 투표, 수용검사, 비교의 세 가지가 있다.

```
<Vote> → Vote;
<AT> → AT (parallel | sequential);
<Cmp> → Cmp [ \(<ver>:<ver>\)^rank {, \( <ver>:
                <ver> \)^rank } ]
          (parallel | sequential)
```

투표 방법에서는 "Vote"라고만 나타낸다. 수용검사와 비교에서는 복제객체들의 병렬 수행과 순차 수행도 명세한다. 병렬 수행은 parallel로, 순차 수행은 sequential로 나타낸다. 예를 들면, hot standby sparing 방법의 경우 "**AT parallel**"로 명세하고, cold standby sparing의 경우에는 "**AT sequential**"로 명세한다. 비교에서 서로 비교해야 할 쌍을 괄호 안에 명세하며 괄호의 오른쪽에 랭크를 표시한다. 예를 들

1) 자료, 메시지구동 메쏘드, 시간구동 메쏘드를 갖는 실시간 객체

2) 대괄호 [ ]는 괄호 안의 내용이 0 또는 1번 나타난다는 의미이고, 중괄호 { }는 중괄호 안의 내용이 0 또는 1 이상의 반복하여 나타날 수 있다는 것을 나타낸다. 그리고, \는 바로 다음 문자가 메타문자가 아니라 일반 문자임을 나타낸다.

인, 3개의 복제객체가 3개의 메쉬으로 구성된 NSCP 방법의 경우 "**Cmp** (V1:V2)[1], (V2:V3)[2] parallel;"과 같이 명세한다.

(3) stub 부분과 comment 부분

stub 부분에는 스터브 메쏘드를 명세한다. 그리고, 모든 스터브 메쏘드에 공통된 중요한 사항들도 명세할 수 있다. Comment 부분에는 결함허용 구성에 대한 제약사항 등을 쓴다.

4.2 RMO의 모델링

RMO의 모델링에서는 RMO가 감시하는 대상들에 대한 명세와 오류 및 진단 조건에 대한 명세가 이루어진다. 이와 같은 명세로부터 dRTO 형태의 RMO가 설계된다. 본 논문에서는 RMO의 모델링 틀인 RMOspec을 제시한다.

4.2.1 RMOspec

RMO의 모델링에서는 RMO가 담당할 감시지역에 어떤 멤버객체들이 있는 지와 RMO가 어떤 실시간 메시지와 예외, 그리고 자료를 감시해야 하는 지가 명세되어야 한다. 뿐만 아니라, 감시지역에 이상이 발생했는지 여부를 판단하기 위한 오류조건, 오류가 발생했을 때 어떤 실시간객체가 결함을 지니고 있는지를 결정하기 위한 진단조건, 결함을 지닌 실시간객체를 찾았을 때에 복구 및 재배치 방법도 RMO 모델링 시에 명세되어야 한다.

RMO는 실시간 메시지와 예외, 그리고 자료에 대한 가장 최근 감시결과뿐만 아니라 그 이전 감시결과들도 보관한다. 따라서, 감시결과를 기록할 RMO의 히스토리에 대해서도 명세한다. 히스토리는 크기에 의해 정의되는 것과 기록유지기간에 의해 정의되는 것의 두 가지가 있다. 크기에 의해 정의되는 히스토리는 기록해야 할 감시결과의 개수가 히스토리의 크기보다 클 경우 오래된 감시결과부터 히스토리에서 삭제한다. 기록유지기간에 의해 정의되는 히스토리에서는 감시결과가 히스토리에 기록될 때 기록유지기간이 설정되며, 기록유지기간이 만기된 감시결과는 히스토리에서 삭제된다.

RMOspec은 RMO를 모델링하는데 필요한 정보를 명세할 수 있게 해주는 틀로써 (그림 11)처럼 7개 부분으로 나뉜다.

```
RMOspec name {
member:
  <RTO> {, <RTO> };
mesg:
  <mesg_list>
excp:
  <excp_list>
data:
  <data-list>
error_cond:
  {<Error> : <boolean_expression> <period>; }
diag_cond:
  {<Fault> : <boolean_expression>; }
recovery:
  {<Fault> : {<statement> } }
}
```

(그림 11) RMOspec

(1) member 부분

member 부분에는 RMO가 담당할 감시지역에 소속된 멤버객체들이 명세된다. 예를 들면, "X, Y, Z;"와 같이 명세한다

(2) mesg 부분

mesg 부분에는 RMO가 감시할 실시간 메시지들이 표현된다. 실시간 메시지는 호출자와 피호출자로 구성된다. 실시간 메시지에 대한 명세 형식은 다음과 같다. 히스토리의 명세 형식은 <history>에 표시되어 있다.

```
<mesg-list> → { <mesg> <history> }
<mesg> → <caller> \> <callee>
<caller> → <RTO> | <RTO.RTMethod>
<callee> → <RTO.RTMethod>
<history> → last n (elements | msec | sec | min | hour)
```

실시간객체 X의 메쏘드 $a()$가 실시간객체 Y의 메쏘드 $b()$를 호출할 때 발생하는 실시간 메시지를 감시하고 이 실시간 메시지의 최근 20 개에 대한 정보가 히스토리에 보관되는 경우 "$X.a() > Y.b()$ **last** 20 **elements**"와 같이 표현한다. "$T > S.c()$ **last** 3 **min**"와 같이 실시간 메시지의 호출자는 실시간객체만 명세할 수도 있다

(3) excp 부분

excp 부분에는 RMO가 감시할 예외들이 표현된다. 예외는 1) 실시간객체와 2) 실시간메쏘드, 그리고 3) 예외이름의 세 쌍으로 표현되며 명세 형식은 다음과 같다.

```
<excp-list> → { <excp> <history> }
<excp> → <RTO.RTMethod.Exception>
```

실시간객체 X의 메쏘드 $a()$에서 발생하는 *excp1*을 감시하며 히스토리의 크기를 5로 한 경우 "*X.a().excp1* **last** 5 **elements**"와 같이 표현한다. "*"는 와일드카드로 이용된다. 즉, 실시간객체 X의 모든 메쏘드의 수행 중 발생한 예외 *excp1*을 감시하는 경우에는 "*X.*.excp1* **last** 5 **elements**"와 같이 표현하고 감시지역에 소속된 모든 실시간객체에서 발생한 *excp1*을 감시하는 경우에는 "*.*.*excp1* **last** 10 **elements**"와 같이 표현한다.

(4) data 부분

data 부분에서는 RMO가 감시할 자료들이 표현된다. 실시간 메시지와 예외는 커널이 RMO에게 알려주지만 자료는 RMO가 직접 주기적으로 감시하여야 한다. 따라서, 자료와 함께 자료를 감시하는 주기도 명세된다. 자료에 대한 명세 형식은 다음과 같다.

```
<data-list> → { <data> <period> <history> }
<data> → <RTO.Data>
<period> → every n (msec|sec|min|hour)
```

실시간객체 X의 정수 변수 i를 2초마다 감시하며 히스토리의 자료유지기간을 1분으로 한 경우 "**int** *X.i* **every** 2 **sec last** 1 **min**;"와 같이 명세한다.

(5) error_cond 부분

error_cond 부분에는 RMO가 감시지역에 이상이 발생했는지 여부를 판단하기 위한 오류조건이 명세된다. 오류조건은 오류와 논리수식의 쌍으로 표현되며 논리수식이 만족되면 오류가 발생한 것으로 판단한다.

(6) diag_cond 부분

diag_cond 부분에는 오류가 발생했을 때 어떤 실시간객체가 결함을 지니고 있는지를 결정하기 위한 진단조건이 명세된다. 진단조건은 결함을 지닌 실시간객체와 논리수식의 쌍으로 표현되며 논리수식이 만족된 경우 해당하는 실시간객체에 결함이 있는 것으로 판단한다.

(7) recovery 부분

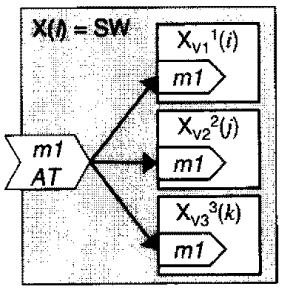
recovery 부분에는 결함을 지닌 실시간객체를 찾았을 때 복구 및 재배치 방법이 명세된다. 결함을 지닌 실시간객체에 대한 복구 방법을 서술식으로 간단히 명세한다.

5. 모델링 예

본 장에서는 복구블럭을 RobustRTO로 모델링한 예와 무인자동차 시스템에서 RMO를 모델링한 예를 보인다.

5.1 RobustRTO의 모델링 예

복구블럭은 여러 개의 수행블럭(try block)을 두고 각 수행블럭의 결과값이 수용 가능한지 검사한다. 주 수행블럭의 결과가 수용 가능하면 주 수행블럭의 결과값이 복구블록의 결과값으로 채택되고 수용검사에서 실패하면 다음 수행블럭의 결과값을 채택한다. (그림 12)는 복구블럭 방법으로 구성된 RobustRTO를 각각 그래픽 표기와 RobustRTOspec으로 모델링한 예이다. RobustRTO는 세 개의 복제객체를 갖고 있으며 이들은 버전복제되었다. 세 개의 복제객체는 동시에 수행되며 모두 서로 다른 노드에 분산된다. 스터브 메쏘드는 수용검사를 통해서 올바른 결과값을 리턴한다.



(a) 그래픽 표기

```
RobustRTOspec X {
replication:
  3 replica, 3 version;
  SW distributed;
fault_handling:
  AT parallel;
stub:
  m1();
Comment:
}
```

(b) RobustRTOspec

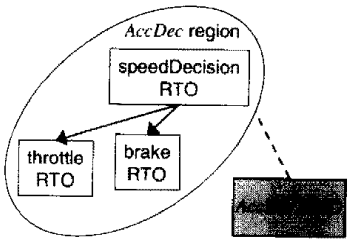
(그림 12) 복구블럭으로 구성된 RobustRTO

5.2 RMO의 예

본 절에서는 무인자동차 시스템에 대한 감시지역과 RMO의 예를 제시한다. 무인자동차 시스템은 여러 종류의 센서와 교통제어센터, GPS 등으로부터 얻은 다양한 정보와 시스템의 상태에 따라서 가속 장치와 감속 장치, 그리고 조향 장치를 이용하여 자동차를 제어함으로써 운전자의 도움 없이 목적지까지 안전하게 주행할 수 있도록 자동차를 제어하는 분산 실시간 시스템이다.

무인자동차 시스템은 다양한 실시간객체들로 이루어지며 시스템 수준 결함허용을 위하여 여러 RobustRTO와 RMO들로 구성된다. 본 절에서는 감속과 가속을 동시에 하려고 하는 오류를 감지하기 위한 *AccDec* 감시지역과 *AccDec* RMO를 보인다. *AccDec* 감시지역에는 speedDecision, throttle, brake의 세 멤버객체가 존

제한다(그림 13). SpeedDecision 실시간객체는 희망하는 속도와 희망하는 속도까지의 가속도를 결정한다. SpeedDecision 실시간객체는 가속도를 throttle과 brake 실시간객체에게 전송한다. 가속도가 양수이면 throttle 실시간객체가 드로틀밸브의 위치값을 계산하여 드로틀 밸브를 제어한다. 가속도가 음수이면 brake 실시간객체는 브레이크의 압력을 계산하여 브레이크를 제어한다. throttle 실시간객체와 brake 실시간객체는 객체 수준 결함허용을 위하여 3개의 복제객체를 지닌 standby spare로 구성된 RobustRTO에 캡슐화되어 있다.



(그림 13) Acc 감시지역의 예

AccDec RMO는 100msec마다 speedDecsion.acc와 throttle.thrLoc, 그리고 brake.pres를 감시한다. 그리고, 500 msec마다 acc_error 오류를 감지하기 위하여 오류조건이 만족되는지 검사한다. 오류조건이 만족되면 결함을 지닌 실시간객체를 찾기 위하여 RMO는 감시지역을 진단한다.

```
RMOspec AccDec {
member:
  speedDecision, throttle, brake;
mesg:
  speedDecision > throttle.newacc();
  speedDecision > brake.newacc();
  . . . .
excp:
  speedDecision.*.excp1;
  . . . .
data:
  speedDecision.acc every 100 msec last 20 elements;
  throttle.thr_loc every 100 msec last 5 sec;
  brake.pres every 100 msec last 5 sec;
  . . . .
error_cond:
  acc_error : (brake.pres > 0) and ( throttle.thr_loc > 0)
              every 200 msec;
diag_cond:
  thr_fail : (throttle.thr_loc > 0) and (speedDecision.acc < 0)
  brake_fail : . . . .
  . . . .
recovery:
  thr_fail : change primary and backup in throttle
             throttle.thr_loc = 0;
  . . . .
}
```

acc_error가 감지되면 결함을 지닌 실시간 객체를 찾기 위하여 diag_cond에 명세된 여러 진단조건을 검사한다. "(throttle.thr_loc > 0) and (speedDecision.acc < 0)"이 만족되면 드로틀밸브나 드로틀밸브를 제어하는 throttle 실시간객체에 결함(thr_fail)이 있는 것이다. 진단한 결과 thr_fail 결함을 찾았으면 AccDec RMO는 throttle RobustRTO에게 주복제객체와 부복제객체를 교환하도록 요청하고 throttle.thr_loc의 값을 0(드로틀 밸브를 닫는 것)으로 갱신하게 한다.

6. 결 론

고 신뢰도의 내장 실시간 시스템을 구축하기 위해서는 객체 수준 결함허용 기능과 함께 시스템 수준 결함허용 기능도 필요하다. 본 논문에서는 객체 수준 결함허용 기능을 제공하는 RobustRTO와 시스템 수준 결함허용 기능을 제공하는 RMO를 제안하였고 이들을 이용하여 고 신뢰도의 결함허용 시스템을 모델링하였다. 그리고, 각각 RobustRTO와 RMO를 모델링하기 위한 명세틀인 RobustRTOspec과 RMOspec을 제시하고 이를 이용한 모델링 예를 보였다.

RobustRTO와 RMO를 이용한 실시간객체 기반 결함허용 시스템 모델링의 장점은 다음과 같다. 첫째, 객체 모델의 장점을 가진다. 둘째, 실시간성 표현이 용이하다. 셋째, 응용 설계와 결함허용 설계 모두를 실시간 객체 모델로 할 수 있으므로 일관성 있는 설계가 가능하다. 넷째, 응용에 대한 설계와 결함허용 설계가 독립적이다. RobustRTO는 객체 수준 결함허용과 관련된 모든 정보를 자신 안에 캡슐화하므로 어떤 객체에 구현될 구체적인 결함허용 방법에 상관없이 응용 실시간 객체들을 설계할 수 있다. 다섯째, 시스템 수준 결함허용 방법을 모델링할 수 있으므로 더욱 높은 신뢰도를 갖는 내장 실시간 시스템을 개발할 수 있다. 여섯째, 객체 수준 결함허용과 시스템 수준 결함허용을 명확히 구분하여 표현함으로써 설계 및 구현을 용이하게 해준다. 일곱째, 시스템 구성과는 독립적으로 모델링이 가능하다.

RobustRTO와 RMO의 수행을 지원하는 커널과 결함허용 서버는 매우 중요하므로 커널과 결함허용 서버에 대한 연구가 필요하다. 또한, 상용 실시간 운영체제 상에서도 RobustRTO와 RMO가 수행될 수 있도록 미들웨어 수준에서 RobustRTO와 RMO를 지원하는 방

법에 대해서도 연구할 것이다. dRTO 모델에 의하여 모델링 및 설계된 내장 실시간 시스템의 신뢰도를 측정하고 결함주입 방법에 의하여 시스템을 시험하는 방법을 개발한다. RobustRTO와 RMO를 명세하고 검증하는 방법에 대한 연구도 필요하다.

참 고 문 헌

[1] B. Randell, "System Structure for Software Fault Tolerance," IEEE Transactions on Software Engineering, pp.220-232, Jun. 1975.

[2] A. Avizienis, "The Methodology of N-version Programming," Software Fault Tolerance, Michael R. Lyu, Wiley, pp.23-46,1995.

[3] Jean-Claude Laprie, Jean Arlat, Christian Beounes and Karama Kanoun, "Architectural Issues in Software Fault Tolerance," Software Fault Tolerance, Michael R. Lyu, Wiley, pp.47-80, 1995.

[4] K. H. Kim, "Action-Level Fault Tolerance, Advances in Real-time Systems," Advances in Real-Time Systems, S. Son, Prentice-Hall, pp.415-434, 1995.

[5] K. H. Kim, "Object Structures for Real-Time Systems and Simulators," IEEE Computer, pp.62-70, Aug. 1997.

[6] 이 신, 손혁수, 양승민, "실시간객체 모델 dRTO," 숭실대학교 컴퓨터학부 기술보고서 SUSC TR-98-1, 1998년 9월 7일.

[7] B. W. Johnson, 'Design and Analysis of Fault Tolerant Digital Systems', Addison Wesley, 1989.

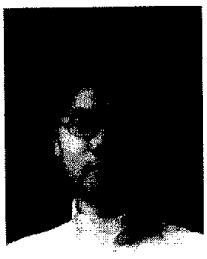
[8] N. Storey, 'Safety-Critical Computer Systems', Addison Wesley, 1996.

[9] K.H. Kim and C. Subbaraman, "An Integration of the Primary-Shadow TMO Replication Scheme with a Supervisor-based Network Surveillance Scheme and its Recovery Time Bound Analysis, " Proc. of 17th Symp. On Reliable Distributed Systems(SRDS'98), pp.168-176, 1998.

[10] J. C. Fabre, V. Nicomette, T. Perennou, R. J. Stroud, and Zhixue Wu, "Implementing Fault Tolerant Applications using Reflective Object-Oriented Programming," 25th IEEE Int'l Symp. on Fault-Tolerant Computing(FTCS'95), pp.489-498, 1995.

[11] J.-C. Fabre and T. Perennou, "A Metaobject Architecture for Fault-Tolerant Distributed Systems: The FRIENDS Approach, "IEEE Trans. on Computers, Jan., pp.78-95, 1998.

[12] J. A. Stankovic and K. Ramamritham, "A Reflective Architecture for Real-Time Operating Systems," Advances in Real-Time Systems, S. Son, Prentice-Hall, pp.23-38, 1995.



임 형 택

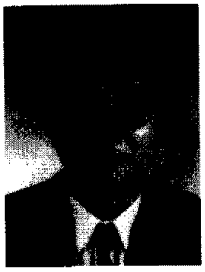
e-mail : htlim@realtime.soongsil.ac.kr

1994년 숭실대학교 전자계산학과 졸업(학사)

1996년 숭실대학교 대학원 컴퓨터학과 졸업(석사)

1996년~현재 숭실대학교 대학원 컴퓨터학과 박사과정

관심분야 : 결함허용, 실시간 시스템, 정형 명세 등



양 승 민

e-mail : yang@computing.soongsil.ac.kr

1978년 2월 서울대학교 전자공학(학사)

1983년 4월 미국 Univ. of South Florida 전산학(석사)

1986년 12월 미국 Univ. of South Florida 전산학(박사)

1988년~1993년 미국 Univ. of Texas at Arlington 조교수

1996년~1998년 국회도서관 정보처리국장

1993년~현재 숭실대학교 컴퓨터학과 부교수

관심분야 : 실시간 시스템, 운영체제, 결함허용 시스템 등