결함허용 실시간 소프트웨어 설계를 위한 명세언어
A Specification Language for Fault-Tolerance Real-time Software Design

김 정 술 · 강 병 욱
(영남대학교 컴퓨터 공학과)
Jung-Sool Kim · Byoung-Ug Kang
Department of Computer Engineering, Yeungnam University

요 약

이 논문에서 우리는 결함허용 실시간 소프트웨어 설계를 위한 명세언어를 제안한다. 특히, 현재 가장 인기있는 소프트웨어 기법인 후향 오류 복구를 위한 명세언어로 N-modular redundancy나 voted-process pairs등에도 사용가능하다. 지금까지의 명세 언어로서는 시스템의 정상 개발 차원에서의 명세만 가능했다. 그래서 본 논문에서는 시스템의 오류시에도 복구 가능한 논리 전달을 위한 명세를 제공한다. 복잡함을 피하기 위해 객체단위로 시스템을 이끌며, 명세서 작성시 주요한 부시스템 단위로 이 방법을 적용하면 명세 기술에 따른 오버헤드를 감소시킬 수 있다.

Ⅰ. 서 론

결함허용 시스템이란 하드웨어 오동작, 소프트웨어 에러 또는 정보 오염이 일어날지라도 주어진 임무를 올바르게 수행할 수 있는 시스템을 일컫는다. 최근 결함허용 시스템에 대한 중요성이 트랜잭션 처리, 실시간 제어, 그리고 인간의 안전에 관련된 응용분야에서 급속히 증대하고 있다(김문회,1993, Kim,K.H, 1992).

예를 들면, 증권 거래소의 컴퓨터 시스템이 수 십번이나 고장을 일으켰는데 매번 장시간동안 증권 거래가 중지되었고 전화국의 장거리 전화교환 시스템이 작은 결함으로 인해 고장을 일으켜 수십만 통화의 장거리 전화가 연결되지 못해 도시 전체에 극도의 혼란을 야기 시켰었다. 이와 같은 예에서 볼 수 있듯이 이러한 시스템들이 적절한 결함허용 능력이 있었더라면 극도의 혼란을 피하고 그로 인한 막대한 경제적 손실을 미연

에 방지할 수 있었을 것이다.

결함허용 시스템에는 4가지 기본 기능이 기본적으로 고려되어야 한다. 결함 감지, 결함 발생 장소 검출, 결합 파급효과 방지 그리고, 결합으로부터의 복구 및 시스템 재구성등이 그것이다. 결함이 발생되면 적절한 대응 조치가 취해질 수 있도록 반드시 감지되어야 하며 결함이 감지되면 그 원인이 무엇이고 발생장소가 어디인지를 찾아서 결함이 시스템의 다른 부분의 동작에 나쁜 영향을 미치지 못하도록 제한시켜야한다. 그 다음 결함이 있는 부분을 여분의 결함이 없는 부품으로 교체하거나 시스템을 재구성함으로써 시스템이 발생된 결함으로부터 복구된다.(김문회,1993, Burns,1989)

결함허용 기법은 시스템내에서 타스크들 각자에 부여된 시간 제한을 충족시키며 수행되는 실시간 시스템과 많은 관련이 있으며 특히 주어진 시간 제한이 타스크 수행 결과의 유효성에 결정적인 영향을 주는 하드 실시간 시스템과 깊은 관련이 있다. 하드 실시간 시스템은 정확한 시스템의 동작이 요구되어 결함허용 구조의 필요성이 절대적으로 요구된다. 결함허용은 하드웨어와 소프트웨어의 신뢰도를 향상시키는 기법의 하나로서, 하드웨어와 소프트웨어의 설계시에 함유되어 있을 수 있는 결함요인과 실행 중 외부적인 요인에 의해 발생되는 결함에 의해 발생되는 오류로 인한 피해를 방지하는데에 그 목적을 두고 있다. 그러나 소프트웨어는 주로 설계시에 프로그램에 함유될 수 있는 오류를 방지하는데 주안점을 두고있고 하드웨어는 실행시간에 발생될 수 있는 외부적 요인에 의한 오류방지가 대부분이다.

본 논문에서는 이러한 결함허용을 관점으로하여 실시간 소프트웨어의 설계시에 발생될 수 있는 오류에 대한 근원적인 대책 방법의 하나로서 시스템의 요구 분석단계에서 정확한 논리표현의 전달수단으로 사용하는 명세 언어를 다룬다. 본 논문의 구성은 다음과 같다. 2장에서는 실시간 소프트웨어의 결함 허용 기법과 명세 언어에 대해서 살펴보고 3장에서는 결함 허용을 위한 명세 언어를 제시한다. 이어 4장에서는 본 논문에서 제안한 명세언어로서 간단하게 후향 오류 복구기법에 적용하며 5장에서 결론을 내린다.

II. 결함 허용 기법과 명세언어

2.1 신뢰성 향상기법

소프트웨어의 결함은 실시간 시스템의 고장 원인이 될 수 있다. 소프트웨어의 결함원

인은 대부분 부적합한 명세와, 또한 소프트웨어 구성요소의 설계 오류에 의한 결함으로 나누어진다.(N,Leverson, 1986) 소프트웨어의 신뢰성 향상기법에는 서론에서 언급한 결함방지와 결함허용이 있다. 여기서 다시 결함방지는 결함회피와 결함제거로 나누어진다. 결함회피란 시스템을 구성하는 동안 결함 요소가 시스템에 포함되는 것을 방지하는 것으로 정확한 요구명세의 작성과 증명된 설계방법의 이용 및 모듈화, 자료 추상화를 지원하는 프로그래밍언어의 사용을 통하여 소프트웨어의 신뢰도를 향상시킬 수 있다. 그러나 이러한 결함회피의 과정을 거친 이후에도 소프트웨어 구성요소 상의 결함이 전무함을 보장할 수는 없다. 결함제거는 설계과정에서 발생될 수 있는 설계오류를 발견하고 이를 제거하는 과정으로서 사용되는 기법은 설계검토, 프로그램 검증, 프로그램 검사 등이 있다. 이러한 모든 기법을 적용한 경우에도 시스템에 잔존하는 잠재적인 결함을 모두 제거한다는 것은 불가능하다. 결함허용은 이러한 잠재결함에 의하여 시스템의 동작 중 오류가 발생되는 것을 막는 기법으로 응용 프로그램의 특성에 따라 결함허용의 정도를 달리 하기도 한다. 즉, 일반적으로 요구되는 결함허용의 정도가 높을수록 적용되는 결함허용의 기법이 복잡해지고 필요한 여분의 부품의 양도 증가하게 된다. 그 분류는 다음과 같다.(이홍규,1993)

- 완전 결함허용 : 이는 시스템이 오류 발생 이후에도 심각한 기능적 손실이나 성능상의 손상이 없이 일정 시간동안 지속적으로 동작 가능한 정도를 말하는데, 사실 구현이 대단이 어렵다.
- 단계적 성능저하 : 이는 시스템이 결함발생 이후 수용가능한 정도의 기능적, 성능적 손실을 허용하여 계속 운영이 가능한 정도를 의미하며 대부분의 시스템이 여기에 해당한다고 볼 수 있다.
- 고장안전 : 고장발생 이후 시스템의 동작은 정지하나 무결성은 유지하는 정도를 말하며 실시간의 성질과는 부합하기 어렵다.

2.2 소프트웨어 결함허용 기법

소프트웨어 결함허용 기법은 소프트웨어 모듈의 중복이나 재수행(rollback & retry) 또는 이 두가지 방식을 혼용하여 쓰고있다. 대표적인 기법들은 다음과 같다.(Hecht, 1976)

(1) Check pointing 기법 : 소프트웨어 실행중 검사 시점을 설정하여 오류발생 유무

를 검사하여 이상이 없으면 계속 실행하고 이상이 감지되면 그 이전의 검사시점으로 되돌아가 재수행하는 방식이다.(Johnson,1989)

(2) Recovery block(RB) 기법 : 재수행에 근거하여 검사 시점에서 오류가 감지되면 지정된 이전 시점으로 되돌아가 같은 기능을 가진 다른 소프트웨어 모듈을 실행하는 방법으로 단일 프로세스내에 적용될수 있다.(Randell, 1975)

(3) Conversation 기법 : RB의 확장으로 여러개의 프로세스들간의 상호 정보 교환을 목적으로하는 기법으로 재수행에 기초한다.(Randell, 1975)

(4) N Self-checking programming 기법 : 두 개 이상의 자가진단 부품이 실행되면서 그 중의 한 실행부품이 주어진 기능을 수행하고 대기부품은 여분으로 대기 상태에 있다가 자가진단에 의해 실행부품에 결함이 발견되면 대기부품중의 하나가 새로운 실행부품이 되어 주어진 기능을 수행하는 기법이다.(Laprie, 1990)

(5) N-version programming 기법 : 이는 하드웨어 결함허용 기법인 TMR(Triple modular redundancy)과 유사한 기법으로 N개의 독립적인 소프트웨어 모듈이 수행한 결과를 서로 비교하여 다수의 동일한 결과를 채택하는 기법이다.(Kim,K.H, 1992)

이러한 방법들은 일반적인 명세로부터 개발되어지므로 명세서에서의 실수를 검출하지 못한다. 그래서 이러한 많은 문제점들을 극복하기 위한 궁극적인 접근방법은 소프트웨어 설계시에 확정된 설계 규칙과 방법들을 사용하는 것이다. 이러한 접근방법은 결국 결함을 피할 수 있어 신뢰성 있는 소프트웨어를 설계할 수 있을 것이다. 즉, 소프트웨어가 맨 처음에 명세서에 기술된 것 같이 정확하고 완전하게 설계되어 진다면 소프트웨어를 위한 결함허용 기법들은 불필요하게 될 것이다. 이 논문에서 제시하고자 하는 것이 바로 이것이다. 올바른 제어 논리의 표현을 위한 시스템 차원의 명세와 타스크 및 통신모듈 의 명세를 제시한다. 통신모듈은 동기와 비동기로 분리하여 표현가능하다.

2.3 명세 언어

실시간 시스템을 위한 명세 언어들은 크게 세가지로 분류된다.

첫째, 실시간 프로그래밍 언어

둘째, 기호 언어

셋째, 논리 및 수학적 명세언어

위에서 분류된 언어들 중 실시간 분석, 설계에 많이 쓰이는 언어는 논리와 수학적 명세언어이다. 그런데 이들 언어들은 대개가 논리적이며 정확성에 기초를 두고 있어서 이해하기가 여간 힘들지 않다. 그래서 본 논문에서는 요구 명세서를 설계자가, 설계명세서를 프로그래머가 쉽게 이해할 수 있도록 함수 형태의 표현에 그 관점을 맞추고 있다.

III. 제안된 명세 방법

3.1 시스템 묘사 구조

기본적인 시스템또는 부시스템을 위한 구조는 아래와 같다.

SYSTEM < *system_name*>
--시스템 or 부시스템의 이름
OBJECT
object 의 선언
-- 타스크와 통신채널list
STRUCT
object 연결 구조
-- object 연결list
RECONFIG
조건 천이 pairs
-- 조건 구조변경list
ENDSYSTEM

결함허용 구조에서는 결함허용 천이가 본 논문에서 대상이 되는 타스크나 통신모듈 각각에 국한되는 것이 아니기 때문에 천이가 일어나는 대상들 전체를 묶어 하나의 시스템 또는 부시스템 단위로 본다. 시스템의 내부 성분들은 객체단위로 정의했고(타스크는 활성객체, 통신 모듈은 수동객체) 객체들간의 결합은 STRUCTure로 설명된다. 조건이 매칭되어(즉, 결함허용) 변경된 흐름구조는 RECONFIGure로 설명된다. 활성객체인 타스크의 묘사는 다음과 같다.

```
TASK <task_name>
    PORT (<direction>,<type_name>)
      INPUT <module_name>
      OUTPUT <module_name>
      OUTSIG (<sig_name>,<destination_task>)
    INSIG (<sig_name>,<source_task>)
      PRIORITY <priority_level>
      PROCEDURE -- function bodies
ENDTASK
```

위의 타스크 묘사중에서 결함허용을 위해서 port 성분만 이용한다. 생산자 타스크는 메시지 타입의 데이터 전송을 위해 하나의 출력 port를 가진다. 소비자 타스크는 같은 타입의 데이터 수신을 위해 하나의 입력 port를 가진다. 나머지 요소들은 일반적인 실시간 타스크들을 묘사하기위해 필요한 요소들이다. 물론, 시스템의 충분한 묘사를 위해 port성분과 같이 사용될 수 있다. 기타 부분의 자세한 설명은 (김정술, 1994)를 참고하기 바란다.

수동객체인 통신 채널 묘사는 아래와 같다.

```
MODULE <channel_name>
      PORT (<direction>,<type_name>)
      ATTR <size>,<type_name>
      PRODUCER <task_name>
      CONSUMER <task_name>
      OPERATION <function_name>
      PROCEDURE --function bodies
```

이 모듈의 묘사도 마찬가지다. 먼저 채널은 약결합 통신 모듈과 강결합 통신 모듈로 나누어 정의할 수 있다. 즉, 동기, 비동기 통신을 의미한다. 이 수동모듈의 대상은 메시지 큐에 국한한다. 아울러 port 성분은 생산자 타스크로부터 들어오는 메시지 타입의 데이터를 위해 하나의 입력 port와 소비자 타스크로의 메시지 전송을 위한 하나의 출력 port를 가진다. 기타 성분들의 자세한 설명은 (김정술, 1994)를 참조하기 바란다.

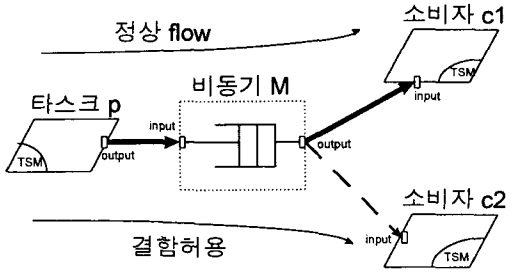
아울러 본 논문에서는 새로운 타입의 모듈을 정의한다. 즉, 상태 천이 관리자 모듈이다. 이 모듈은 후향 에러 복구를 기반으로 하는 소프트웨어 결함허용 복구 기법의 명세시에 필요한 모듈로 시스템 전체의 상태를 관리하는 모듈이다 이 모듈은 내부에 상태

천이 테이블을 은닉하여 시스템이 결함허용을 수행시에 다시 그 전의 상태로 되돌아갈 수 있게 하여 시스템의 일관성을 유지 시켜주는 하나의 상태 감독자 모듈로 생각할 수 있다. 이 모듈은 외부의 사건을 입력으로 하여 상태를 validate 해주는 기능과 내부 액션을 determine 해주는 내부 행위를 수행한다.(김정술, 1994)

IV. 적용 사례

본 장에서는 간단한 결함허용의 사례를 적용한다.(Mario, 1993) 먼저 기본적인 생산자-소비자 타스크 사이의 메시지 전달시 물리적, 또는 논리는 오류가 발생했을시의 시스템의 동적인 재구성을 [그림 1]에서 보인다. 시스템의 명세는 아래와 같다.

```
SYSTEM <producer_consumer>
  OBJECT :
          task <p>;
                  task <c1,c2>;
                  module <message,integer>;
          STRUCT :
              BC : BEGIN
                  flow : p, lmcm, c1;
                  lmcm : p.outport -> c1.inport;
```



[그림 1] 생산자-소비자의 동적 재구성

```
                ENDBC;
            AC : BEGIN
                flow : p, lmcm, c2;
                lmcm : p.outport -> c2.inport;
                ENDAC;
        RECONFIG :
            ENTER -> BC;
            BC -> AC WHEN signal(c1, 1);
ENDSYSTEM;

TASK <producer p>
        PORT (<output>,<message>);
ENDTASK;

TASK <consumer c1 or c2>
        PORT (<input>,<message>);
ENDTASK;

MODULE <lmcm>
        PORT (<input>,<message>),
             (<output>,<message>);
        ATTR <integer>;
ENDMODULE;
```

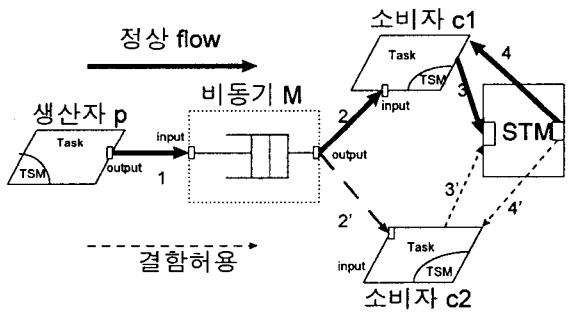
[그림 1]의 설명은 다음과 같다. 생산자 타스크 P가 약 결합 통신모듈(LMCM)을 통하여 소비자 타스크 C1과 정상적인 메시지 송수신을 하는 과정에서 오류가 발생시 동적으로 재구성하는 행위를 명세한 것이다. 객체들은 송수신 타스크와 모듈로 구성되며 성분 STRUCT는 BC(Before Crash)와 AC(After Crash)의 두 흐름으로 묘사되고 BC는 시스템의 오류발생 전의 메시지 전달 흐름방향을 표현하고, AC는 오류발생 후의 시스템이 재구성된 메시지의 전달 경로를 표현한다. RECONFIG는 초기에 어떤 외부에서의 기본 입력 사건을 받아 현재 수행되고 있는 상태를 의미하며 오류 발생시 C1의 내부 제어 모듈(TSM: 타스크 동기 모듈)이 이를 감지하여 재구성을 위한 신호를 발생시킨다. 신호 타입은 1과 -1을 기본적으로 가지는데 1은 정상적인 동작시, -1은 프로세스가 비 정상적으로 종료될시에 발생된다. 여기서 타스크 동기 모듈이 제시되는데 이 모듈은 하나의 병행 타스크에는 내부에 하나의 제어 모듈을 가진다. 즉, C언어에서 main

모듈과 같은 것이다. 다음의 예는 후향 오류 복구(backward error recovery) 기법에 적용가능한 명세 표현이다. 명세 표현은 다음과 같다.

```
SYSTEM <backward error recovery>
  OBJECT :
          task <p>;
                  task <c1,c2>;
                  lmcm <message,integer>;
                  stm <event_type,state_type>;
          STRUCT :
              BC : BEGIN
                  flow : p, lmcm, c1, stm;
                  lmcm : p.outport -> c1.inport;
                  stm : c1.outport -> c1.inport;
                  ENDBC;
              AC : BEGIN
                  flow : p, lmcm, c2, stm;
                  lmcm : p.outport -> c2.inport;
                  stm : c2.outport -> c2.inport;
                  ENDAC;
          RECONFIG :
              ENTER -> BC;
              BC -> AC WHEN signal(c1, -1);
              AC -> EXIT WHEN signal( , 1);
  ENDSYSTEM;

  TASK <producer p>
          PORT (<output>,<message>);
  ENDTASK;

  TASK <consumer c1 or c2>
          PORT (<input>,<message>),
                (<output>,<state_type>),
```

```
                (<input>,<state_type>);
ENDTASK;

MODULE <lmcm>
        PORT (<input>,<message>),
                (<output>,<message>);
        ATTR <integer>;
ENDMODULE;

MODULE <stm>
        PORT (<input>,<event>),
                (<input>,<state>),
                (<oupput>,<state>);
ENDMODULE;
```

이 예에 대한 설명은 기본적으로 생산자-소비자의 동적 재구성과 같고 단지 상태를 관리하는 상태 관리자 모듈이 추가 되었다. [그림 2]를 참조하면 타스크와 모듈이 포트를 통하여 메시지의 전송 및 상태저장이 수행된다. 이 상태 관리 모듈의 경우는 후향오류 복구를 위하여 들어오는 실패 사건과 현재의 상태를 주기적으로 체크한다. 실패 사건이 들어오면 원하는 기능을 수행하기 위한 대체 알고리즘을 실행하여 가장 최근에 저장된 타스크의 상태로 롤백(rollback)한다. 타스크가 실행을 시작할 때 상태정보가 이



[그림 2] 후향 오류 복구 재구성

용가능한지 STM의 입력 포트를 조사한다. 이 예는 전형적인 checkpoint restart 모델이다. 그리고 STM을 설계시 상태 저장을 위해 물리적 또는 논리적으로도 독립되도록 설계되어야 한다.

V. 결 론

지금까지 우리는 결함허용 소프트웨어를 위한 명세 방법을 제시하였다. 기본적인 시스템 구성 단위를 객체에 두고 접근하여 복잡함을 해결하였고, 실제 소프트웨어의 요구명세나 설계 명세를 작성시에 결함허용을 위한 명세를 중요한 부 시스템들에만 적용하면 명세서 작성시의 오버헤드를 감소시키며, 결함허용을 위한 명세로 사용가능 할 것이다. 본 논문에서는 현재 가장 많이 쓰이고 있는 후향 오류 복구 기법에 초점을 맞추었지만 N modular redundancy 기법이나 voted process pairs 기법의 적용시에도 명세가 가능하다. 실제, 결함허용의 원인들이 하드웨어나 소프트웨어내에 수없이 잠재하고 있기 때문에 다양하고, 정확한 명세기능을 제공해야 하겠지만 현재 인기있는 기법들의 결함허용 명세방법이 전무하기 때문에 기본 기능들만 제공한다. 복잡 다단해져 가는 실시간 환경을 충분히 반영하기 위해서는 앞으로 분산기법을 적용하기 위해 새로운 개념을 도입할 필요가 있고 보다 더 구체적인 논리표현에 연구가 있어야 한다.

참 고 문 헌

[1] 김정술, 강교철 "DARTS방법의 행위명세 언어", 『석사학위논문』 포항공과대학교,1994.

[2] 김문회, "결함허용 시스템의 설계 고려사항 및 동향", 『한국 정보과학회지』 특집, Vol. 11, No. 3, Aug 1993.

[3] 이홍규,이귀영, "소프트웨어 결함 허용 측면에서의 결함 허용 실시간 시스템에 관한 고찰", 『한국 정보과학회지』 특집, Vol. 11, No. 3, Aug 1993.

[4] A. Burns and A. Wellings, Real-time Systems and their Programming Languages, Addison-Wesley Press., 1989, pp. 91-124.

[5] Hecht, H., "Fault-Tolerant Software for Real-time Application", Computing Survey, Dec. (1976), pp. 391-407.

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

[7] Kim, K, H., "Design of Real-Time Fault-Tolerant Computing Stations", Lecture note in the NATO Advanced Science Institute on Real-Time Computing, Sint Maarten, Oct. 1992.

[8] Laprie, J-C, et al., "Definition and Analysis of Hardware and Software-Tolerant Architectures", Computer, July (1990), pp. 39-51.

[9] Mario R. Barbacci, et al., "Durra: a structure description language for developing distributed applications", Software Engineering Journal, March (1993), pp. 83-94.

[10] N. G. Leveson., "Software Safety: Why, What and How," ACM Computing Surveys, Vol. 18, No.2 (1986), pp. 125-163.

[11] Randell, B., "System Structure for Software fault tolerance", IEEE Trans, on Software Eng., June, (1975), pp. 220-232.