실시간 객체 모델 dRTO
(Real-Time Object Model dRTO)

이 신 † 손 혁 수 † 양 승 민 ††
(Sheen Lee) (Hyuk-Su Son) (Seung-Min Yang)

요 약 내장 실시간 시스템은 그 응용 분야가 매우 다양하며 그에 따라 실시간 처리와 신뢰도 보장에 대한 요구 또한 다양하다. 이러한 내장 실시간 시스템의 효율적인 개발과 실시간성 및 신뢰도 보장을 위해서는 적절한 실시간 모델이 필요하다. 최근 들어 객체지향 모델과 실시간 시스템을 접목시키는 실시간 객체 모델에 대한 연구가 활발하다. 본 논문에서 제안하는 dRTO (dependable Real-Time Object) 모델은 내장 실시간 시스템의 다양한 요구사항을 지원할 수 있도록 객체지향 개념, 실시간 개념, 신뢰성 개념 등을 단일 모델에 수용한다. 그리고 이 3 가지 기본 개념을 지원하기 위해서 5 가지 원시 클래스를 제공한다. dRTO 모델의 특징은 다음과 같다. 첫째, 객체의 시간 제약사항은 물론 객체 간의 상호작용과 관련된 시간 제약을 효율적으로 모델링하고 구현할 수 있도록 해준다. 둘째, 내장 시스템을 구성하는 하드웨어, 응용 소프트웨어, 커널 등을 하나의 틀 안에서 모델링할 수 있다. 셋째, 결함 감지 및 처리에 관해 명시적으로 표현할 수 있다.

Abstract The application areas of embedded real-time systems are very wide and so are the requirements for real-time processing and reliability of the systems. To develop embedded real-time systems effectively with its real-time and reliability properties guaranteed, an appropriate real-time model is needed. Recently, the research on real-time object-oriented model is active, which graft the concept of object-orientation on real-time systems modeling and development. In this paper, we propose dRTO (dependable Real-Time Object) model, with 5 primitive classes. These allow designers to effectively model the characteristics of real-time systems, i.e., object-orientation, real-time-ness and dependability. The dRTO model has three main features. First, it is able to model and implement the timing constraints imposed on real-time objects as well as interactions among the objects. Second, hardware and software components (including kernel) of embedded systems can be modeled in one frame. Third, it is able to represent fault detection and recovery mechanisms explicitly.

1. 서 론

내장 실시간 시스템은 그 응용 분야가 매우 다양하다. 소형 가전제품에서부터 군사무기나 로봇제어 등 그 규모나 복잡도가 다르고 그에 따라 실시간 처리와 신뢰도에 대한 요구가 다르다. 따라서 내장 실시간 시스템의 효율적인 개발을 위해서는 내장 실시간 시스템의 다양한 요구사항을 잘 반영할 수 있는 시스템 모델이 필요하다.

그 동안 실시간 시스템과 관련된 연구는 대개 태스크 스케줄링이나 시간 모델에 관련된 것이 많았다. 그러나 이런 연구는 시간성 검증에 관한 이론적인 연구에 치우쳤고 실제 시스템 개발에는 도움을 주지 못했다. 또한 많은 실시간 시스템들이 분산 환경에서 작동하며 태스크들간 그리고 외부 세계와 상호작용이 많이 이루어지고 있으나, 이에 대한 모델링 방법이나 검증에 관한 연구는 부족하다.

최근에 와서 이를 해결하기 위한 방법으로 실시간 객체 모델에 대한 연구가 활발하다. [1][3][5][7][10][12][16]. 대표적인 것으로는 TMO (Time-triggered Message-triggered Object) 모델[10][11], ROOM

· 본 연구는 정보통신연구진흥원 대학기초연구지원(과제번호: C1-1999-1177-00)
† 비 회 원 : 숭실대학교 컴퓨터학과
tactlee@tactlee.pe.kr
neoson@hitel.net
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
yang@computing.soongsil.ac.kr
논문접수 : 1998년 9월 10일
심사완료 : 2000년 1월 6일

(Real-time Object-Oriented Modeling) [2][3], Real-Time UML (Unified Modeling Language) [1][13] 등이 있다. 이는 객체지향 기법의 여러 가지 장점들, 즉 상속성, 재사용성, 다형성 등을 실시간 시스템에 접목시키고자 하는 것이다. 또한 객체의 독립적인 속성으로 인해 실시간 객체 모델은 결함 허용 시스템을 모델링하기에 매우 적합하다. 그러나 기존 실시간 객체 모델들은 내장 실시간 시스템의 다양한 실시간 특성과 결함 처리 기법 등을 하나의 틀 안에서 모델링하기에는 부족한 면이 있다.

본 논문에서는 실시간 시스템의 다양한 요구사항을 실용적으로 모델링할 수 있는 dRTO (dependable Real-Time Object) 모델을 제시한다. dRTO 모델은 객체지향 개념, 실시간 개념, 신뢰성 개념 등을 단일 모델에 수용한다. 그리고 이 3 가지 기본 개념을 지원하기 위해서 5 가지 원시 클래스(primitive class)를 지원한다. dRTO 모델의 특징은 다음과 같다. 첫째, 객체의 시간 제약사항은 물론 객체간의 상호작용과 관련된 시간 제약을 효율적으로 모델링하고 구현할 수 있도록 해준다. 이를 위해 실시간 메쏘드뿐만 아니라 실시간 자료와 실시간 메시지에 대한 명세가 가능하며 실행 모델도 제시한다. 둘째, 내장 시스템을 구성하는 하드웨어, 응용소프트웨어, 커널 등을 하나의 틀 안에서 모델링이 가능하다. 이는 시스템 구축을 용이하게 해줄 뿐만 아니라 성능 분석에도 도움을 준다. 셋째, 결함 감지 및 처리에 관해 명시적으로 표현할 수 있도록 해준다. 이는 내장 실시간 시스템이 대부분 고 신뢰도 운영을 위한 결함 허용이 필수적이므로 실시간 객체 모델이 꼭 갖추어야 하는 기능이다.

본 논문의 2 장에서는 대표적인 기존 실시간 객체 모델에 대해 알아보고 장단점을 논한다. 3 장에서는 dRTO 모델의 3 가지 기본 개념과 5 가지 원시 클래스에 대해 설명한다. 4 장에서는 내장 실시간 시스템인 무인자동차 시스템 프로토타입의 일부를 dRTO 모델을 이용하여 모델링 한다. 5 장에서는 결론 및 향후 과제에 대해 논한다.

2. 관련 연구

2 장에서는 실시간 소프트웨어의 개발을 목표로 하는 여러 가지 모델링 방법 중 객체지향 개념을 기본 바탕으로 하는 모델링 방법들의 특징을 간략하게 고찰하고, 이들의 장단점을 알아본다.

2.1 TMO 모델

TMO[10][11] 모델은 K. H. Kim과 H. Kopetz가 제안한 실시간 객체 모델이다. TMO 모델은 연성 실시간 시스템에서부터 경성 실시간 시스템에 이르기까지 폭 넓게 적용 가능하며, TMO라 불리는 실시간 객체들에 대하여 설계 시 실시간성 보장을 목적으로 한다.

TMO는 실시간 자료인 객체 자료 저장소, 실시간 메쏘드인 시간 구동 메쏘드 (time-triggered method 또는 TM), 메시지 구동 메쏘드 (message-triggered method 또는 MM) 등으로 구성된다(그림 1). 실시간 자료는 시간이 경과함에 따라서 유효성을 상실할 수 있다. 따라서 자료마다 유효한 값을 갖는 기간을 설정함으로써 유효 기간이 지난 값을 참조하는 것을 막는데, 이때 각각의 자료에 주어지는 기간을 최대 유효 기간이라고 한다. 객체 자료 저장소는 최대 유효 기간을 가지는 실시간 자료들의 집합이다.

TMO 모델에서는 메쏘드들을 구동 특성에 따라 TM과 MM으로 구분한다. TM은 실시간 클럭에 의해서 주기적으로 수행되는 메쏘드이다. TM의 수행 시간, 주기, 마감시간 등은 자동 활성화 조건(autonomous activation condition)을 통해서 명세한다. MM은 클라이언트의 요청에 의해서 구동되는 메쏘드로서 다른 TM 또는 MM들이 필요로 하는 서비스를 제공한다. MM에는 서비스 보장 시간이 명세된다.

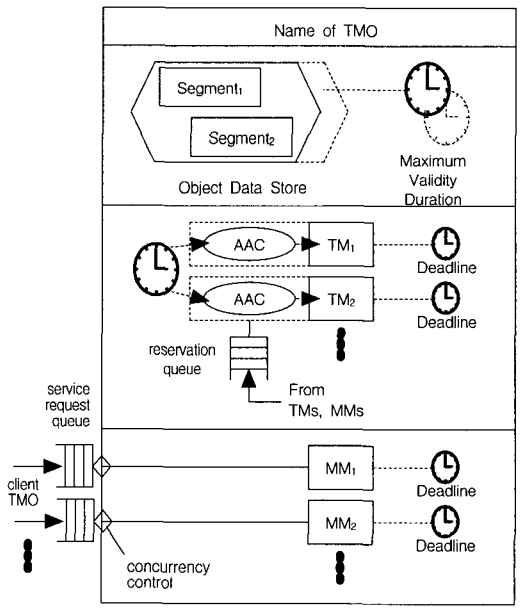


그림 1 TMO 모델

또한 TMO 모델에서는 기본 병행성 제약사항(basic concurrency constraints)을 두어 TM과 MM이 서로 공유하고 있는 자료에 대해서 충돌이 발생할 경우 TM에 우선권을 준다. 즉 이 경우 MM의 실행 시간이 TM과 겹쳐질 것이 예상 된다면 MM의 실행을 뒤로 미루고 TM을 실행한다. 이는 TM의 실행 시간을 설계 시 보장하여 예측성을 높이기 위함이다.

2.2 ROOM

ROOM[2][3]은 Bran Selic, Paul T. Ward, Garth Gullekson 등에 의해 제안된 분산 실시간 객체 모델이다. ROOM은 실시간 시스템이 갖고 있는 실세계의 비동기성, 병행성, 분산성 등 시스템 모델링의 기본 요구조건을 수용하고자 하였다. 또한 모델링 개념과 구현 개념간의 연계를 쉽게 하여 실시간 소프트웨어 개발의 생산성 향상에 목적을 두고 있다.

ROOM에서는 동시에 수행되는 활동적인 객체를 액터(actor)로 나타낸다. 액터는 자신의 행동을 나타내는 행위와 외부 인터페이스인 포트를 갖는다(그림 2). 행위는 외부 사건, 즉 메시지의 전달에 대한 액터 내부의 동작을 말한다. 행위는 확장 유한-상태기계(finite state automata) 형태인 상태 차트(state charts)로 나타낸다. 시스템의 동작 시나리오를 표현하기 위하여 수직으로는 시간 축을 나타내고, 수평으로는 시스템 내부의 객체들이 주고받는 메시지를 나타내는 MSC(Message Sequence Chart)를 이용한다. 액터 내부의 행동을 나타내는 행위와는 달리 MSC는 액터간의 행동을 나타낸다.

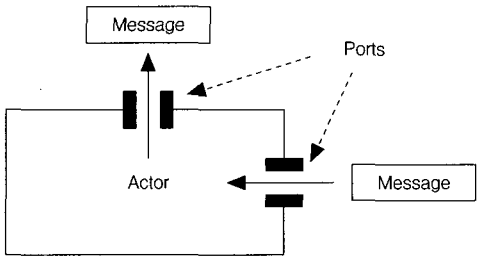


그림 2 액터 모델

ROOM은 액터에 통신과 관련된 포트와 메시지를 정확히 표현함으로써 분산 실시간 시스템에 적합하며, 메시지에 의한 액터 내부의 행위와 액터간의 상호 작용을 잘 나타낸다. 그러나 시간 처리 사건들이 모두 메시지로 변화되어 처리가 되어야 하기 때문에, 모델 수준에서 시간 명세에 대한 표현이 부족하다.

2.3 Real-Time UML

UML[1][13]은 Booch 방법론[4], James Rumbaugh의 OMT(Object Modeling Technique)[9], Jacobson의 OOSE 방법론[6] 등을 연합하여 만든 모델링 개념의 공통 집합 이론으로 객체지향적 분석 및 설계 방법론의 표준 지정을 목적으로 한다. Real-Time UML은 UML을 실시간 시스템에 적용하기 위해 실시간 특성을 추가하여 확장한 모델이다.

UML의 특징은 다양한 방법으로 객체를 모델링 한다는 것이다. UML은 객체와 객체 사이의 정적 및 동적인 관계, 각 객체의 내부 상태, 하드웨어와 소프트웨어 객체에 대한 구분, 재사용 단위로서의 패키지 등을 여러 가지 다이어그램들로 서로 다른 시각에서 표현할 수 있다. 클래스 다이어그램은 객체의 정적인 상태를 나타낸다. 클래스 다이어그램에서는 객체의 각 자료와 메쏘드에 대한 허가 권한을 private, protected, public 등으로 표시하며, 객체와 객체 사이의 정적인 상속과 포함(aggregation 및 composition) 관계를 나타낸다. 순차(sequence) 다이어그램과 상태 다이어그램은 객체와 객체, 객체 내부의 동적인 상황을 나타낸다. 순차 다이어그램의 수평 축에는 객체가 나열되고, 시간의 흐름을 나타내는 수직 축에는 객체간의 메시지 전달 형태를 표시한다(그림 3). 수직 축에 표시된 하나의 메시지 또는 메시지와 메시지 사이의 간격에는 시간 제약 사항을 명시한다.

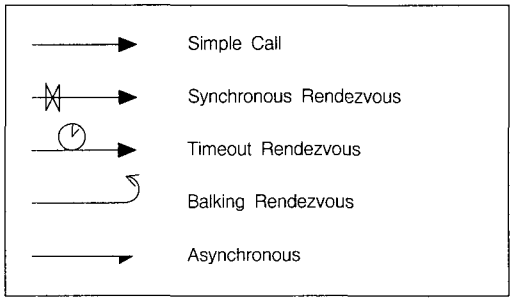


그림 3 메시지 동기화

Real-Time UML은 다양한 방법으로 객체를 모델링 해서 서로 비교 검토하기 때문에, 객체를 한 가지 방법으로 표현함으로써 생길 수 있는 실수를 줄여준다. 또한 기존 UML을 바탕으로 확장했기 때문에, 표현 방법이 풍부하고 구현이 용이하다. 그러나 실시간 메시지에 대한 명세에 비해 실시간 메쏘드에 대한 명세가 부족하며, 이 둘을 연계할 방법이 없다.

이상에서 언급한 3 가지 실시간 객체 모델 중에서 실시간 제약 사항을 명시하는 데는 TMO 모델이 가장 우

수하다. ROOM은 시간 처리 부분을 메시지로 변환하여 처리해야 하기 때문에, 모델 수준에서 시간 명세에 대한 표현이 부족하다. Real-Time UML은 실시간 메쏘드에 대한 명세가 부족하다. 그리고 ROOM과 Real-Time UML 모두 실시간 자료에 대한 명시적인 표현이 없다. TMO 모델은 실시간 메쏘드를 구동 특성에 따라 TM과 MM으로 나누고, 각각 실시간 특성을 표현할 수 있다. 또한 실시간 자료에 대한 명세도 가능하다. 그러나 MM의 우선권을 제약함으로써 긴급한 MM의 처리가 불가능하다. 특히 호출되는 MM에게 호출자가 실시간 특성을 상속시키는 방법이 없다. 또한 3가지 실시간 객체 모델 모두 내장 실시간 시스템의 중요한 특성 중 하나인 결함 허용과 관련하여 명시적으로 지원하는 것이 없다.

본 논문에서 TMO 모델의 단점을 보완하고 확장한 dRTO 모델을 제시한다. dRTO 모델에서는 기본 병행성 제약사항을 없앰으로써 보다 폭 넓은 실시간 시스템 모델링을 가능하게 하였다. 그리고 실시간 메시지를 이용하여 호출되는 MM에게 실시간 특성을 상속시킬 수 있게 하였다. 또한 결함 허용 시스템 구축을 위하여 객체 수준과 시스템 수준에서 결함 감지와 처리를 명시할 수 있는 틀을 제공한다.

3. dRTO 모델

dRTO 모델의 목표는 내장 실시간 시스템의 다양한 요구사항을 하나의 틀 안에서 묘사하는 것이다.

내장 실시간 시스템은 하드웨어와 소프트웨어(운영체제 포함)가 하나의 덩어리로 시스템을 작동시킨다. 때문에 하드웨어 구성이 조금만 바뀌어도 시스템의 구성이 바뀌어야 하고, 소프트웨어의 재사용도와 확장성이 떨어진다. 이것은 내장 실시간 시스템의 개발을 어렵게 하는 주된 원인이 된다. 또한 내장 실시간 시스템은 응용 분야의 특성상 경성 실시간성과 높은 신뢰도를 요구하는 경우가 많다. 그러므로 내장 실시간 시스템을 묘사하기 위해서는 다양한 시간 제약사항과 신뢰도를 높이기 위한 여러 종류의 결함 감지 및 처리 방법에 대해 명시적으로 표현할 수 있어야 한다.

이러한 요구사항들을 표현할 수 있는 틀을 제공하고자 dRTO 모델은 객체지향 개념, 실시간 개념, 신뢰성 개념 등을 단일 모델에 수용한다(그림 4). 그리고 이 3가지 기본 개념에 따라 내장 실시간 시스템을 쉽게 모델링할 수 있도록 MVD(Maximum Validity Duration) 클래스, HKT(History Keeping with Time) 클래스, HKS(History Keeping with Size) 클래스, Group 클래스, Monitor 클래스 등의 원시 클래스들을 제공한다.

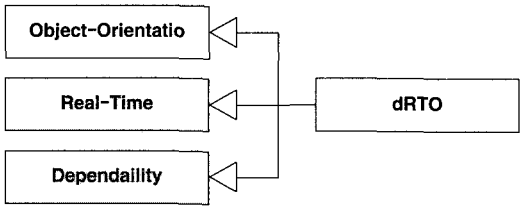


그림 4 dRTO 모델의 개념적 구성

3.1 dRTO 모델의 기본 개념

3.1.1 객체지향 개념

dRTO 모델에서 내장 실시간 시스템의 모든 하드웨어 및 소프트웨어는 RTO(Real-Time Object)라 불리는 객체로 추상화된다. 이렇게 추상화된 객체는 시스템의 복잡도를 줄이고, 소프트웨어의 재사용도를 높이며, 뛰어난 확장성을 제공한다.

한편 내장 실시간 시스템을 구현하는 경우, 시스템의 성능을 높이기 위해 설계와는 다른 구현을 하는 경우가 있다. 특히 객체지향 시스템의 경우, 설계와 구현의 틈이 크게 벌어지는 경우가 많다. 그러나 설계와 구현의 틈이 커지면 구현 후 유지 보수 및 확장이 힘들다. dRTO 모델에서는 설계와 구현의 틈을 줄이고자 객체지향 개념에 내포 클래스(nested class), 내부(inner) 접근자 등을 제공한다.

3.1.1.1 RTO

dRTO 모델에서 내장 실시간 시스템은 RTO의 집합으로 구성된다. TMO 모델의 TMO와 마찬가지로 dRTO 모델의 RTO는 자료와 MM 및 TM들로 구성된다. 그러나 TMO 모델과 달리 dRTO 모델에서는 MM 호출 시 우선순위 및 시간 제약사항을 명시할 수 있다. 그림 5는 RTO의 구성을 보여준다.

3.1.1.2 내포 클래스

내장 실시간 시스템에서 공동작업을 하는 RTO들 사이의 빈번한 메시지 교환은 시스템의 부하를 높인다. 설계 시 RTO들 사이의 자료 공유를 명시하고 공유 규칙에 따라 자료를 공유한다면, 시스템의 부하를 크게 줄일 수 있다. dRTO 모델은 내포 클래스 개념을 통해 공동작업을 하는 RTO들 사이의 명시적인 자료 공유를 허용한다.

내포 클래스란 클래스 내부에서 새로 선언된 클래스를 말한다. 반대로 내포 클래스가 내부에 선언되어 있는 클래스를 내포하는 클래스라고 부른다. C++ 언어에서도 내포 클래스 개념을 제공한다. C++ 언어에서는 내포 클

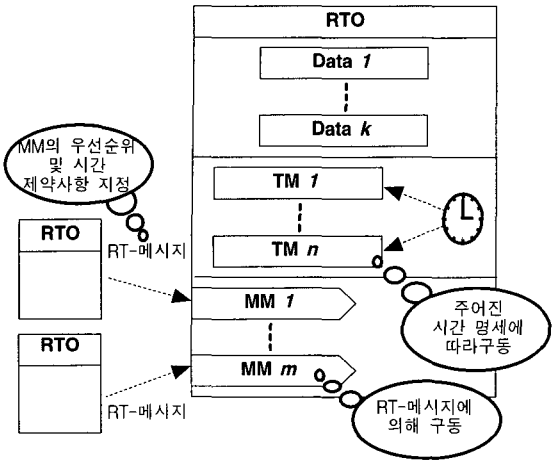


그림 5 RTO의 구성

래스에 의해 만들어진 객체의 소속 함수는 내포하는 클래스의 소속 자료에 접근할 수 없다. 그러나 dRTO 모델에서는 내포 클래스에 의해 만들어진 RTO의 RT-메쏘드가 내포하는 클래스의 소속 자료에 접근하는 것을 허용한다. dRTO 모델의 내포 클래스 개념은 파스칼 언어의 내포 프로시저(nested procedure) 개념이나 BETA 언어[14]의 내포 패턴(nested pattern) 개념과 비슷하다.

그림 6은 dRTO 모델의 내포 클래스 개념을 이용해서 운영체제의 커널을 구성한 예이다. 내포 클래스에 의해 실체화된 Manager_RTO와 Scheduler_RTO는 시스템의 부하를 줄이기 위해 이들을 내포하는 Kernel_RTO의 자료인 RTO_Information_Block을 공유할 수 있다.

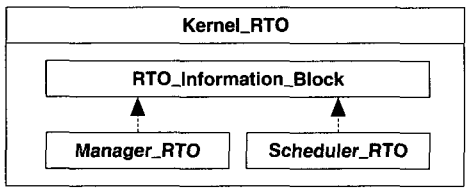
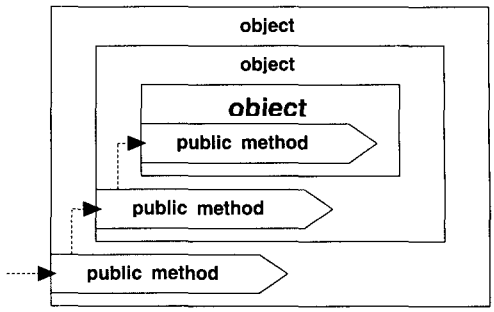


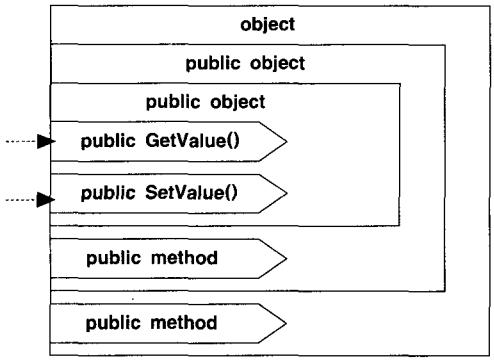
그림 6 내포 클래스 개념을 이용한 운영체제 커널의 구성 예

3.1.1.3 내부 접근자

C++ 언어는 범용(public), 전용(private), 보호(protected) 등의 3 가지 접근자를 제공한다. 전용 및 보호 접근자로 선언된 소속 함수는 객체 내부에서의 접근만을 허용하는데 반해, 범용 접근자로 선언된 소속 함수는



(a) 연속된 소속 함수 호출을 통한 내부 객체로의 접근



(b) 객체를 범용 접근자로 선언하여 접근

그림 7 C++ 언어에서 객체 접근자를 사용하는 예

객체 외부에서의 접근도 허용한다. 만일 클래스 내부에서 선언된 객체를 접근하려면 여러 번의 소속 함수 호출이 필요하다(그림 7 (a)). 이러한 접근 비용을 줄이기 위해서 C++ 언어는 객체를 범용 접근자로 선언하여 접근하는 방법을 제공한다(그림 7 (b)). 그러나 이 경우 객체에서 범용 접근자로 선언된 모든 소속 함수가 외부에 공개되는 문제점이 있다. 예를 들어 그림 7 (b)에서 GetValue()는 외부에 공개해도 좋겠지만, SetValue()를 외부에 공개하면 문제가 발생할 수 있다.

dRTO 모델은 이 문제를 해결하기 위해서 내부 접근자를 제공한다. 내부 접근자로 선언된 자료나 RT-메쏘드는 부모 RTO의 한 수준 위의 RTO에게만 공개된다. 즉 내부 접근자로 선언된 자료나 RT-메쏘드는 이것을 포함하고 있는 RTO의 형제 RTO와 부모 RTO에게만 공개되고, 부모 RTO의 형제들이나 부모 RTO의 조상들에게는 공개되지 않는다(그림 8).

내부 접근자는 외부에 공개하는 메쏘드들 중에서도 접근의 제한을 두고자 하는 메쏘드가 있을 경우에 매우 유용하다. 예를 들어 그림 8에서 Sensor_RTO가 수집한 자료들을 보관하는 SensingData_RTO의 경우, 비록

범용 접근자를 통해 외부에 공개되어 있지만, GetValue()만 외부에서 접근할 수 있고, 내부 접근자로 선언된 SetValue()는 Sensor_RTO 내부에서만 접근할 수 있다. 또한 내부 접근자로 선언된 Gathering_RTO는 SensingData_RTO가 범용 접근자로 선언되었다 하더라도 Sensor_RTO 외부에는 공개되지 않는다.

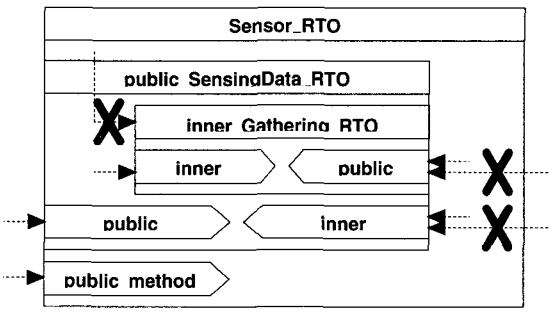


그림 8 내부 접근자

3.1.2 실시간 개념

dRTO 모델은 스케줄링 대상을 실시간 특성에 따라 스케줄링 클래스로 나누고, 객체에 실시간 특성을 부여하기 위해 RT-메쏘드, RT-메시지 등을 제공한다. 또한 TMO 모델의 기본 병행성 제약사항을 따르지 않고, RT-메시지에 의한 MM 구동을 제시하여 다양해지는 실시간 응용에 실시간 객체를 좀 더 유연하게 적용할 수 있도록 한다.

3.1.2.1 스케줄링 클래스

dRTO 모델은 긴급, 경성 실시간, 연성 실시간 등의 스케줄링 클래스를 제공한다. 실시간 스케줄러는 긴급, 경성 실시간, 연성 실시간의 순서로 스케줄링을 고려하고, 각 스케줄링 클래스마다 다른 스케줄링 기법을 적용한다. 긴급 스케줄링 클래스로 지정하는 메쏘드는 매우 긴급한 처리를 필요로 하는 메쏘드로서 시스템에서 가장 높은 우선순위를 갖고 실행된다. 긴급 스케줄링 클래스로 지정하는 메쏘드는 마감시간을 명시할 필요가 없다.

경성 실시간 스케줄링 클래스로 지정하는 메쏘드는 실시간 제약사항을 반드시 만족시켜야 하는 경성 실시간 메쏘드로서 주기, 마감시간 등이 명시되어야 한다. 이 외에도 실시간 스케줄러의 필요에 따라 필요한 요소가 추가될 수 있다. 예를 들어 실시간 스케줄러에 따라서는 같은 마감시간을 갖는 스케줄링 대상 중에서 희생자를 고르기 위해 중요도를 명시할 수도 있다.

연성 실시간 메쏘드는 연성 실시간 스케줄링 클래스로 지정한다. 연성 실시간 스케줄링 클래스로 지정하는 메쏘드에는 우선순위 또는 우선순위를 계산하기 위한 요소들이 명시된다.

3.1.2.2 RT-메쏘드

RT-메쏘드는 RTO 내부에 존재하는 메쏘드로서, 구동 방식에 따라 TM과 MM으로 나뉜다. TM은 명세된 주기마다 능동적으로 구동하고, 다른 RT-메쏘드에 의해 호출되지 않는다. 따라서 TM에게 인자를 전달하기 위해서는 객체 내의 자료를 통해서 간접적으로 전달하거나, TM이 능동적으로 MM을 불러서 인자를 얻어와야 한다. MM은 RT-메쏘드의 호출에 의해서 수동적으로 구동한다. RTO들 사이의 메시지 전달은 일반 객체 모델과 마찬가지로 MM 호출을 통해서 이루어진다.

TM은 자신의 실시간 특성인 자동 활성화 조건을 갖는다. 자동 활성화 조건은 TM의 실행 주기, 예상 최대 수행시간, 마감시간 등으로 구성된다. 자동 활성화 조건에는 스케줄링 클래스가 명시되지 않는다. 왜냐하면 TM은 기본적으로 경성 실시간 스케줄링 클래스이기 때문이다.

MM은 예상 최대수행시간을 기본적으로 갖고 있고, MM의 실행과 관련된 호출 방식(동기 호출 또는 비동기 호출), 전달인자, 스케줄링 클래스, 마감시간 등은 호출자가 지정한다. 즉 TMO 모델에서는 MM 자체에 서비스 보장 시간을 명세하도록 한 것과는 달리, dRTO 모델에서는 호출자가 실시간 메시지를 이용하여 MM (명확히 말하자면 MM이 실체화된 쓰레드)의 실시간 제약사항을 결정한다.

TM의 실시간 특성은 설계 시 결정된다. 반면 MM의 실시간 특성은 실행 시 호출자에 의해 결정된다. MM을 호출하는 방식에는 동기 호출과 비동기 호출의 두 가지가 있다. 동기 호출 방식의 경우, 호출되는 MM은 호출자의 시간 연장선상에서 실행된다. 그러므로 동기적으로 호출되는 MM은 호출자의 실시간 특성(스케줄링 클래스, 마감시간 등)을 상속받는다. 예를 들어 호출자의 스케줄링 클래스가 경성 실시간이라면 동기적으로 호출되는 MM의 스케줄링 클래스도 경성 실시간이고, 호출자의 스케줄링 클래스가 연성 실시간이라면 동기적으로 호출되는 MM의 스케줄링 클래스도 연성 실시간이다.

비동기 호출 방식의 경우, 호출되는 MM과 호출자는 비동기적으로 실행된다. 그러므로 비동기적으로 호출되는 MM의 실시간 특성은 호출자가 결정한다. 예를 들어 호출자의 스케줄링 클래스가 경성 실시간이라 하더라도 비동기적으로 호출되는 MM의 스케줄링 클래스는 연성

실시간으로 지정하여 구동시킬 수 있다.

결국 TMO 모델에서는 MM의 실시간 특성을 MM 자체에 정적으로 지정하지만, dRTO 모델에서는 호출자가 서비스를 원하는 긴급도에 따라 MM의 실시간 특성을 결정한다.

3.1.2.3 RT-메시지

RTO들 사이에 실시간 정보를 가지고 전달되는 메시지를 RT-메시지라 한다. RT-메시지는 그림 9처럼 세 가지 특성을 갖는다.

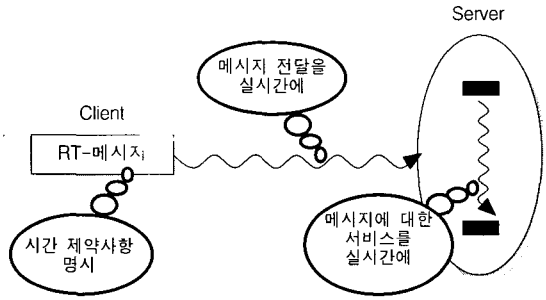


그림 9 RT-메시지의 특성

(1) RT-메시지는 실시간 정보를 갖는다. RT-메시지에는 MM의 호출 방식과 전달인자, 실시간 그룹, 마감시간 등이 명시된다. MM의 동기 호출은 RT-메시지의 동기 전송을 의미하고, 비동기 호출은 비동기 전송을 의미한다.

(2) RT-메시지의 전달은 실시간에 이루어진다. RT-메시지는 메시지 안에 명시된 마감시간에 따라 전송 매체를 통해 전달되고, 전달된 RT-메시지는 이것을 처리할 MM을 활성화시킨다.

(3) RT-메시지에 대한 서비스는 실시간에 이루어진다. 실시간 스케줄러는 RT-메시지를 처리하는 MM이 마감시간 안에 작업을 마칠 수 있도록 스케줄링 한다.

3.1.3 신뢰성 개념

dRTO 모델은 신뢰도 높은 시스템 구축을 위해 단위 객체 수준과 시스템 수준에서의 결함 허용 방법을 제시한다. 단위 객체는 결함 허용의 기본 단위로서 자체적으로 결함 처리 능력을 갖고 있다. 즉 RTO 내에서 발생한 결함은 RTO 내에서 처리하는 것을 원칙으로 한다. 그러나 단위 RTO의 결함 허용만으로는 시스템 수준의 결함 허용을 만족시키지 못한다.

dRTO 모델은 단위 RTO가 결함 허용 기능을 가질 수 있도록 객체 수준의 복제(*multiple-copy* 또는 multiple-version)와 메쏘드 수준의 복제를 허용한다. 그림 10은 RTO를 N-버전으로 구성한 예이다. object의 a()가 호출되면, a()는 group object의 SyncMCast()를 통해서 각 버전의 a()를 동기 멀티캐스팅한다. SyncMCast()는 각 버전의 a()로부터 받은 되돌림 값(return value)을 배열로 만들어 SyncMCast()를 호출한 a()에게 돌려준다. a()는 SyncMCast()로부터 받은 되돌림 값의 배열에서 타당한 값을 뽑아 호출자에게 되돌려준다.

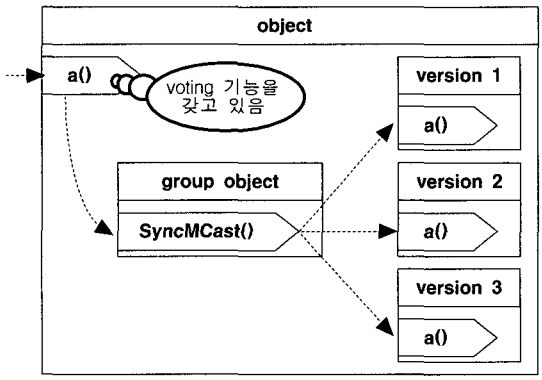


그림 10 RTO를 N-버전으로 구성한 예

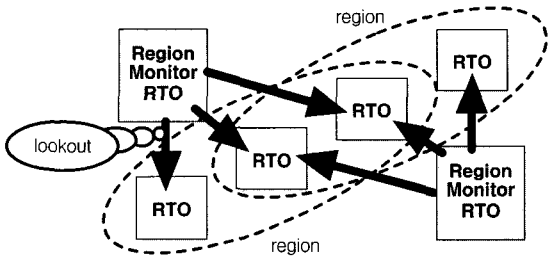


그림 11 지역 모니터 RTO

시스템 수준의 결함 허용을 위해서 감시 지역(region) 개념을 제공한다. 감시 지역이란 물리적 또는 논리적으로 연관이 있는 RTO들의 집합이다. 즉 단위 객체의 결함 허용만으로는 판단할 수 없는 시스템 수준의 이상 행동을 감지하기 위해 감시 및 진단의 대상이 되는 객체들의 집합을 말한다. 감시 지역 모니터 RTO는 영역 내의 RTO들을 감시 및 진단, 복구하는 역할을 하는 RTO이다(그림 11). dRTO 모델은 감시 지역 모니터 RTO를 생성하기 위해 Monitor 클래스를 제공한다.

3.2 dRTO 모델이 제공하는 원시 클래스

3.2.1 MVD(Maximum Validity Duration) 클래스

일반 객체 모델에서 객체 내부의 자료는 유효 기간의 제한이 없다. 그러나 실시간 시스템에는 시간의 흐름에 따라 자료의 유효성이 변할 수 있다. 예를 들어 일정 시간이 지나면 무효해지는 자료가 있다. dRTO 모델은 자료의 가치에 대한 유효 기간을 부여하기 위해 MVD 클래스를 제공한다. MVD 클래스에 의해 생성되는 객체는 최대 유효 기간을 갖는다. 즉 MVD 클래스의 자료는 최대 유효 기간 내에서만 자료로서 가치를 갖는다. 또한 유효 기간이 별도로 지정되지 않는 비 실시간 자료는 유효 기간이 무한대인 것으로 볼 수 있다.

그림 12는 MVD 클래스의 구성을 보여준다. MVD 클래스는 GetValue(), SetValue(), GetMVD(), SetMVD(), IsValid() 등의 MM을 제공한다. GetValue()는 호출자에게 자료를 돌려주고, SetValue()는 호출자가 지정한 값으로 자료를 설정한다. GetMVD()는 호출자에게 현재 설정되어있는 최대 유효 기간을 돌려주고, SetMVD()는 호출자가 지정한 값으로 최대 유효 기간을 설정한다. 호출자는 IsValid()를 통해서 자료의 유효성 여부를 판단할 수 있다.

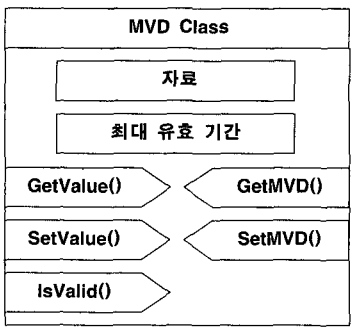


그림 12 MVD 클래스

3.2.2 HKT(History Keeping with Time) 클래스

실시간 시스템의 경우 어떤 결정을 내리기 위해 입력이 되는 자료들을 일정시간 보관할 필요가 있다. 예를 들어 RTO를 감시하는 모니터 RTO가 있다고 하자. 모니터 RTO가 감시 대상 RTO로부터 자료를 수집하는 방법에는 두 가지가 있다. 하나는 모니터 RTO는 매 주기마다 감시 대상 RTO에게 자료를 요구하는 것이다. 이 방법은 감시 대상이 많을 경우, 모니터 RTO가 감시 대상 RTO에게 자료를 요구하는 시간 비용이 많이 든다. 다른 하나는 감시 대상 RTO가 능동적으로 모니터 RTO에게 자료를 보내는 것이다. 이 방법은 감시 대상 RTO로부터 자료를 수집하는 비용을 줄일 수 있는 반면, 모니터 RTO는 자료들의 기록을 유지하기 위한 자료구조가 필요하다. dRTO 모델은 일정 기간 동안의 자료를 기록하기 위해 HKT 클래스를 제공한다. HKT 클래스는 자료 유지 기간과 자료를 유지하기 위한 큐(queue) 자료구조를 갖는다(그림 13). 자료 유지 기간은 상대 시간으로써 현재 시간을 기준으로 과거 얼마만큼의 시간을 말한다. HKT 클래스에 의해 생성되는 객체는 설정된 자료 유지 기간만큼의 자료만을 큐 자료구조에 보관한다.

그림 13은 HKT 클래스의 구성을 보여준다. HKT 클래스는 GetFirstValue(), GetLastValue(), CopyFirstValue(), CopyLastValue(), SetValue(), GetDuration(), SetDuration(), NumOfElem(), Clear() 등의 MM을 제공한다. GetFirstValue()와 GetLastValue()는 큐에 가장 먼저 입력된 자료와 가장 나중에 입력된 자료를 큐에서 꺼내어 돌려준다. SetValue()는 큐에서 가장 나중에 입력된 자료 뒤로 새 자료를 넣는다. 즉 GetFirstValue()와 SetValue()를 같이 사용하면 FIFO(first-in first-out)의 효과를 얻을 수 있고, GetLastValue()와 SetValue()를 같이 사용하면 LIFO (last-in first-out)의 효과를 얻을 수 있다. CopyFirstValue()와 CopyLastValue()는 GetFirstValue(), GetLastValue()와 같은 값을 돌려주지만, 큐에서 자료를 복사만 하고 꺼내지는 않는다. 즉 CopyFirstValue()와 CopyLastValue()를 호출하더라도 큐가 보관하고 있는 자료의 개수는 변하지 않는다. GetDuration()은 설정된 자료 유지 기간을 돌려주고, SetDuration()은 자료 유지 기간을 새로 설정한다. NumOfElem()은 큐에 보관되어 있는 자료의 개수를 돌려준다. Clear()는 큐를 비운다.

3.2.3 HKS(History Keeping with Size) 클래스

HKS 클래스는 HKT 클래스와 비슷한 구조를 갖는다. 다른 점은 HKT가 자료 유지 기간을 갖는 대신 HKS 클래스는 자료 유지 개수를 갖는다는 것이다. 즉 HKS 클래스에 의해 생성되는 객체는 설정된 자료 유지 개수만큼의 자료만을 보관한다. HKS 클래스는 Group 클래스와 함께 결함 허용 기법을 구사할 때 유용하게 사용된다.

그림 14는 HKS 클래스의 구성을 보여준다. WaitQFull()을 제외한 대부분의 MM은 HKT 클래스와 비슷한 기능을 갖는다. WaitQFull()은 설정된 자료 유지 개수만큼의 자료가 큐에 기록될 때까지 대기하는 기

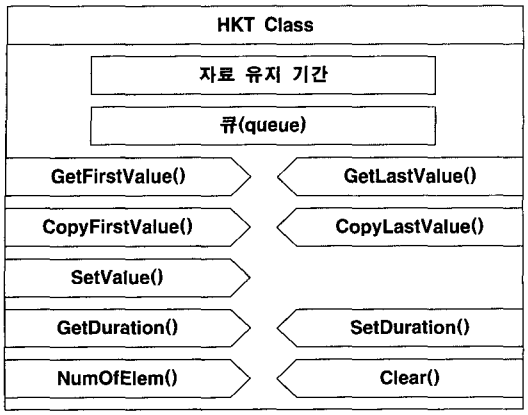


그림 13 HKT 클래스

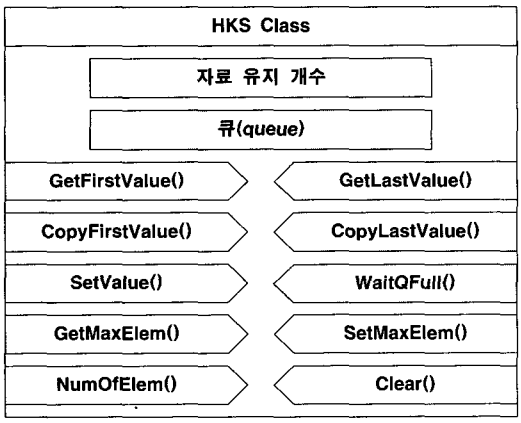


그림 14 HKS 클래스

능을 갖는다.

3.2.4 Group 클래스

dRTO 모델은 메시지의 멀티캐스팅을 지원하기 위해서 Group 클래스를 제공한다. 그림 15는 Group 클래스의 구조를 보여준다.

Group 클래스는 ClearMember(), AddMember(), DelMember(), AsyncMCast(), SyncMCast() 등의 MM을 제공한다. ClearMember()는 Group 클래스의 구성원 테이블을 초기화한다. AddMember()와 DelMember()를 통해서 구성원의 등록 및 탈퇴가 이루어진다. RT-메시지가 전달되는 실제 대상이 RTO의 MM이기 때문에, 구성원은 RTO와 MM의 이름으로 구성된다. AsyncMCast()는 메시지를 구성원들에게 비동기 멀티캐스팅하고, SyncMCast()는 동기 멀티캐스팅한다. dRTO 모델에서 실시간 메시지인 RT-메시지의 전송은 곧 MM의 호출을 의미한다. 즉 AsyncMCast()는 구성원들을 비동기 호출하고, SyncMCast()는 구성원들을 동기 호출한다. AsyncMCast()는 되돌림 값을 받지 않지만, SyncMCast()는 되돌림 값을 받아야 한다. 그러므로 SyncMCast()를 통해서 받은 되돌림 값은 멀티캐스팅된 구성원 수만큼의 배열이다.

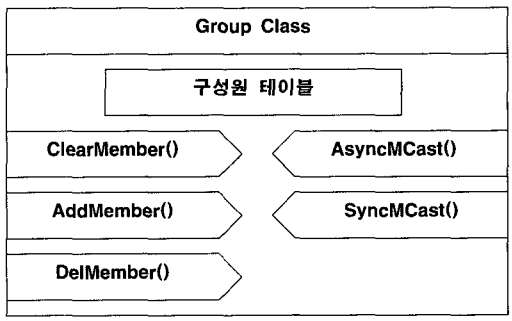


그림 15 Group 클래스

Group 클래스를 통해서 결함 허용 기법을 쉽게 사용할 수 있다. 그림 10은 Group 클래스를 이용하여 N-버전을 구성한 예이다. 그림 10에서 group object의 SyncMCast()는 각 버전의 a()를 동기 멀티캐스팅하고, 되돌림 값을 모아 배열로 만들어 호출자에게 돌려준다.

3.2.5 Monitor 클래스

dRTO 모델은 시스템 수준의 결함 허용을 위해서 감시 지역 개념을 제공한다. 감시 지역은 감시 대상 RTO들로 구성되고, 감시 지역 모니터 RTO는 감시 지역 내의 감시 대상 RTO들을 감시 및 진단, 복구한다(그림 11).

감시 지역 모니터 RTO는 감시 대상 RTO의 상태와 행동을 감시하기 위해서 감시 대상 RTO의 자료, MM의 전달인자, 되돌림 값 등을 얻어야 한다. 감시 대상 RTO의 자료는 감시 대상 RTO가 제공하는 MM을 호출함으로써 얻을 수 있다. MM의 전달인자와 되돌림 값은 MM이 호출될 때마다 자신이 받은 전달인자와 되돌림 값을 감시 지역 모니터 RTO의 MM을 호출함으로써 전달해줄 수 있다. 그러나 이 방법은 감시 대상 RTO가 감시 지역 모니터 RTO에 독립적이지 못하다는 문제가 있다. dRTO 모델은 이 문제를 해결하기 위해서 호출 가로채기(invocation trapping)를 제공한다. 호출 가로채기란 MM에 대한 호출이 있을 경우, MM의 전달인자와 되돌림 값을 지역 모니터 RTO가 가로채서 검사하는 방법이다(그림 16). 호출 가로채기는 하드웨어 객체를 검

사하기 위해서 하드웨어 모듈의 입출력 단자를 검사하는 방법을 소프트웨어 객체에 적용시킨 것이다[8]. 결국 감시 지역 모니터 RTO는 감시 대상 RTO의 내부 자료에 직접 접근하여 감시하는 것이 아니라, 감시 대상 RTO의 입출력 단자에 해당하는 MM을 감시한다.

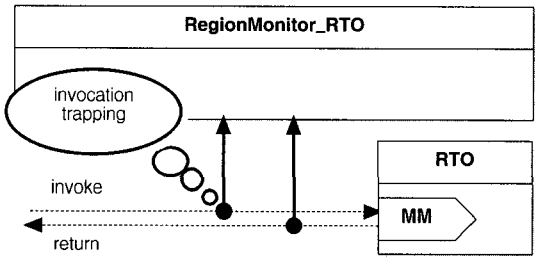


그림 16 호출 가로채기

감시 대상 RTO에서 이상 증세가 발견되면, 감시 지역 모니터 RTO는 감시 대상 RTO에 대한 진단(diagnosis)을 한다. 감시 지역 모니터 RTO는 진단을 통해 결함이 발견된 RTO에 대해 실행 정지 또는 교체 등의 복구 작업을 한다. 최악의 경우에는 시스템의 상태를 비상상태로 전이한다.

dRTO 모델은 감시 지역 모니터 RTO를 지원하기 위해서 Monitor 클래스를 제공한다. 감시 지역 모니터 RTO가 감시 대상 RTO에 대해 감시 및 진단, 복구 등의 기능을 수행하기 위해서는 운영체제 수준의 지원이 필요하다. Monitor 클래스는 운영체제 수준의 지원을 받아 감시 대상 RTO에 대한 감시 및 진단, 복구 등의 권한을 갖는다. 즉 Monitor 클래스에 의해 생성된 RTO는 타 RTO에 대한 감시 및 진단, 복구 등의 기능을 운영체제에게 요구할 수 있다.

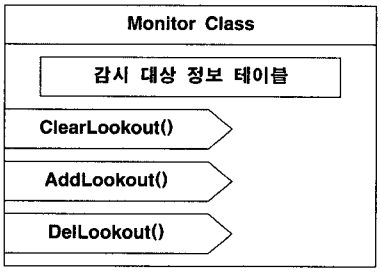


그림 17 Monitor 클래스

그림 17은 Monitor 클래스의 구조를 보여준다. Monitor 클래스는 AddLookout(), DelLookout(), ClearLookout() 등의 MM을 제공한다. AddLookout()은 감시 대상 RTO의 MM과 이것을 감시할 함수를 등록한다.

4. dRTO 모델의 적용 예

본 장에서는 dRTO 모델의 적용 예를 보이기 위해서 무인자동차 시스템(intelligent vehicle system 또는 IVS) 프로토타입(prototype)을 모델링한다. 무인자동차 시스템은 운전자의 도움 없이 자동차를 목적지까지 안전하게 운행하는 내장 실시간 시스템으로 특히 인간의 생명에 치명적인 영향을 미치는 경성 실시간 시스템의 예이다. 무인자동차 시스템은 여러 감지기들로부터 주변 환경에 대한 정보를 읽어 들인 후, 실시간으로 구동기를 제어해서 자동차의 주행 방향 및 속도를 조절해야 한다. 뿐만 아니라 다른 무인자동차 시스템과 혹은 교통 통제 시스템과 통신을 하면서 자동차의 운행에 관련된 정보를 주고받는다.

4.1 무인자동차 시스템 프로토타입의 주행 시나리오

무인자동차 시스템 프로토타입은 그림 18의 모의 환경에서 다음의 시나리오에 따라 주행한다.

1) 자동차가 출발선에서 대기하고 있다.
2) 출발 신호를 받고 주행선을 따라 출발한다.
3) 지정된 최대 속도까지 가속한다.
4) 횡단보도 신호를 받고 감속한다.
5) 횡단보도 정지선에 멈춘다.
6) 초음파 감지기로 횡단보도에 장애물이 없다는 것을 확인하고 출발한다.
7) 곡선 주행선을 따라 주행한다.
8) 빨간 신호등 신호를 받고 감속한다.
9) 신호등 정지선에 멈춘다.
10) 파란 신호를 받고 주행을 계속한다.
11) 급회전 곡선 주행에서 서행한다.
12) 급회전 곡선을 벗어나서 정상 속도로 주행한다.
13) 초음파 감지기로 전방에 장애물이 나타났다는 것을 감지하고 비상 정지한다.
14) 초음파 감지기로 전방에 장애물이 없다는 것을 확인하고 주행을 계속 한다.
15) 노면 요철 신호를 받고 서행한다.
16) 노면 요철 해제 신호를 받고 정상 속도로 주행한다.
17) 목적지 신호를 받고 감속한다.
18) 목적지 정지선에 멈춘다.

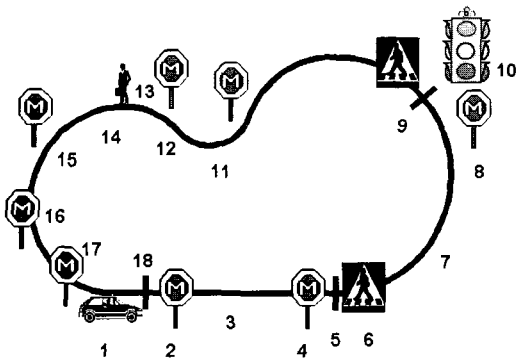


그림 18 무인자동차 시스템 프로토타입의 주행 모의 환경

4.2 무인자동차 시스템 프로토타입의 모델링

그림 19는 무인자동차 시스템 프로토타입을 상위 수준에서 모델링한 것이다. IVS_RTO는 크게 Sensor_RTO, Controller_RTO, Actuator_RTO 등으로 구성된다. Sensor_RTO는 주행선과 장애물을 감지하고, 각종 신호를 판독하고, 자동차의 주행 속도를 측정한다. Controller_RTO는 감지기로부터 받은 자료들을 바탕으로 자동차의 위치와 자세를 파악하여 주행 속도 및 방향을 결정하여 Actuator_RTO에게 제어 명령을 내린다. Actuator_RTO는 Controller_RTO의 명령대로 주행 모터와 조향 모터를 제어한다.

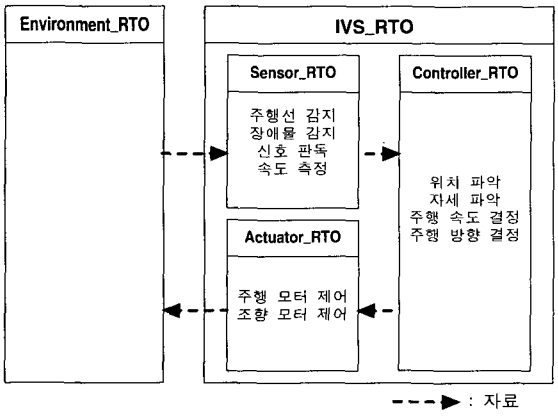


그림 19 IVS_RTO

Sensor_RTO를 좀더 자세히 살펴보면 그림 20과 같다. Sensor_RTO는 LaneSensor_RTO, UltrasonicSensor_RTO, MarkerReader_RTO, Speedometer_RTO, SensingData_RTO, Sensing_TM, ServiceData_RTO, GetValue_MM 등으로 구성된다. LaneSensor_RTO는 주행선을 감지하고, UltrasonicSensor_RTO는 장애물을 감지하고, MarkerReader_RTO는 모의 환경에서 제공하는 각종 신호를 감지하고, Speedometer_RTO는 자동차의 주행 속도를 감지한다. 이들 감지기들은 감지한 자료를 SensingData_RTO에 저장한다. SensingData_RTO는 HKT 클래스로서 감지기들로부터 입력된 감지 자료들을 일정시간 동안 보관한다. Sensing_TM은 주기적으로 SensingData_RTO에 보관된 감지 자료들을 읽어서 외부에 서비스하기 위한 감지 자료로 가공하여 ServiceData_RTO에 저장한다. ServiceData_RTO는 MVD 클래스로서 감지 자료가 입력되어 일정 기간이 지난 후에는 무효해진다. ServiceData_RTO는 GetValue_MM을 통해서 외부에 서비스된다.

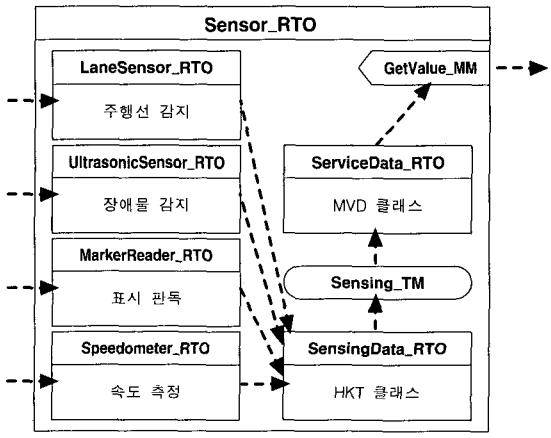


그림 20 Sensor_RTO

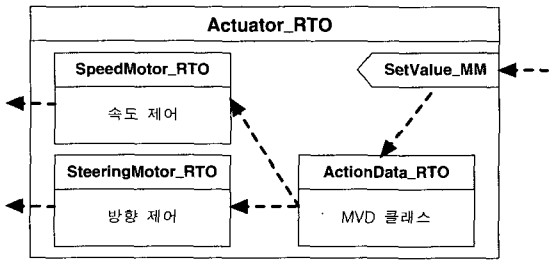


그림 21 Actuator_RTO

그림 21은 Actuator_RTO를 좀더 자세히 본 것이다. Actuator_RTO는 SpeedMotor_RTO, SteeringMotor_RTO, Service_MM, ActionData_RTO 등으로

구성된다. SetValue_MM은 Controller_RTO로부터 제어 명령을 받아 ActionData_RTO에 넣어준다. ActionData_RTO는 MVD 클래스이고, SpeedMotor_RTO와 SteeringMotor_RTO에게 공유된다. SpeedMotor_RTO와 SteeringMotor_RTO는 ActionData_RTO의 제어 명령에 따라 주행 모터와 조향 모터를 제어한다.

SpeedMotor_RTO와 SteeringMotor_RTO는 주행 모터와 조향 모터에게 PWM(Pulse Width Modulation) 방식으로 펄스를 공급해주는 TM으로 구성된다. PWM은 일정한 주기 내에서 듀티 사이클(duty cycle)이라고 불리는 시간 간격을 두고 전원을 켜고 끔으로써 사각파(square wave)를 발생시키는 방식이다. PWM은 아날로그 신호의 출력, 직류 모터의 속도 제어, 서보(servo)의 위치 제어 등 제어의 여러 분야에서 흔히 사용된다[15]. SpeedMotor_RTO와 SteeringMotor_RTO에서는 PWM 방식으로 각각 주행 모터의 속도와 조향 모터의 위치를 제어한다. PWM 방식은 고해상도의 타이머를 요구하고, 프로세서의 많을 부하를 필요로 한다. 그러므로 PWM 방식으로 펄스를 만들기 위해서는 별도의 마이크로컨트롤러(microcontroller)를 사용하는 것이 타당하다. 그래서 Sensor_RTO와 Actuator_RTO는 각각 제어가 편리한 마이크로컨트롤러를 할당하고, Controller_RTO에는 연산 속도가 빠른 마이크로컨트롤러를 할당한다(그림 22). 그리고 Sensor_RTO와 Controller_RTO 사이, Controller_RTO와 Actuator_RTO 사이에는 통신을 위해 FromSensor_RTO와 ToActuator_RTO를 둔다. FromSensor_RTO와 ToActuator_RTO는 마이크로컨토롤러 사이의 단방향 통신을 지원하기 위한 별도의 하드웨어 장치이다.

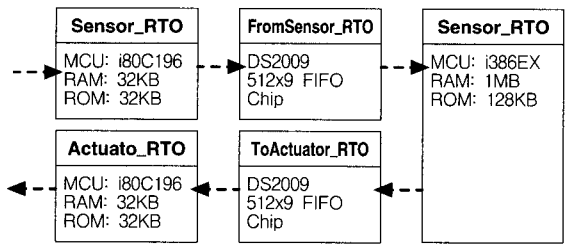


그림 22 무인자동차 시스템 프로토타입의 하드웨어 구성

5. 결론 및 향후 과제

본 논문에서 제시하는 dRTO 모델은 객체지향 개념, 실시간 개념, 신뢰성 개념 등을 통해서 내장 실시간 시스템의 다양한 요구사항들을 표현할 수 있는 틀을 제공한다. 그리고 이 3 가지 기본 개념에 따라 내장 실시간 시스템을 실용적으로 모델링할 수 있도록 MVD 클래스, HKT 클래스, HKS 클래스, Group 클래스, Monitor 클래스 등의 원시 클래스들과 내포 클래스, 내부 접근자 등을 제공한다.

dRTO 모델은 무인자동차 시스템 프로토타입을 모델링한 예에서도 볼 수 있듯이 내장 실시간 시스템을 모델링하기에 매우 적합하다. 그러나 dRTO 모델은 아직 구체적인 명세 및 시간 검증 방법이 없다. 또한 dRTO 모델로 모델링한 것을 구현하기 위해서는 운영체제 수준에서의 지원이 필요하다. 현재 dRTO 모델에 대한 연구는 지원 도구와 실행 환경 부분으로 나뉘어 진행 중이다. dRTO 모델 지원 도구에 대해서는 RTO들 사이의 정적 및 동적 관계에 대한 명세와 시간 검증에 대한 연구가 진행 중이다. dRTO 모델 실행 환경에 대해서는 dRTO 모델을 지원하기 위한 실시간 스케줄러의 자료구조에 대한 연구가 진행 중이고, 운영체제로서 DRAGON (Distributed Real-time Availability Guaranteed Operation and Networking) 커널이 구현 중이며, 기존 운영체제에서 dRTO 모델을 지원하기 위한 시뮬레이션 환경도 구현 중이다.

참 고 문 헌

[1] B. P. Douglass, *Real-Time UML:Developing Efficient Objects for Embedded Systems*, Addison-Wesley, 1998.

[2] B. Selic, G. Gullekson, J. McGee and I. Engelberg, "ROOM:An Object-Oriented Methodology for Developing Real-Time Systems," *CASE'92 Fifth International Workshop on Computer-Aided Software Engineering*, 1992.

[3] B. Selic, P. T. Ward and G. Gullekson, *Real-Time Object-Oriented Modeling*, John Wiley & Sons, 1994.

[4] G. Booch, *Object-Oriented Analysis and Desing With Applications*, Benjamin/Cummings, 1994.

[5] H. R. Callison, "A Time-Sensitive Object Model for Real-Time Systems," *ACM Transactions os Software Engineering and Methodology*, Vol. 4, No. 3, pp. 287-317, 1995.

[6] I. Jacobson, *Object-Oriented Software Engineering:A Use Case Driven Approach*, Addison-Wesley, 1992.

[7] I. Satoh and M. Tokoro, "A Formalism for Real-Time Concurrent Object-Oriented Computing," *In Proceedings of ACM OOPSLA'92*, 1992.

[8] J. Fabre et al., "Implementing Fault Tolerant

Applications using Reflective Object-Oriented Programming," *Proc. of the 25th IEEE International Symposium on Fault-Tolerant Computing (FTCS-25)*, Pasdena (CA), pp. 489-498, June 27-30, 1995.

[9] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy and W. Lorensen, *Object-Oriented Modeling and Desing*, Prentice Hall, 1991.

[10] K. H. (Kane) Kim, "Object Structures for Real-Time Systems and Simulators," *IEEE Computer*, August 1997.

[11] K. H. Kim and H. Kopetz, "A Real-Time Object Model RTO.k and an Experimental Investigation of its Potentials, " *Proc. 1994 IEEE COMPSAC*, pp. 392-402, Nov. 1994.

[12] K. Takashio and M. Tokoro, "DROL:An Object-Oriented Programming Language for Distributed Real-Time Systems," *OOPSLA'92*, pp. 276-294, 1992.

[13] M. Fowler and K. Scott, *UML Distilled:Applying The Standard Object Modeling Language*, Addison-Wesley, 1997.

[14] O. L. Madsen et al., *Object Oriented Programming in the BETA Programming Language*, ISBN 0-201-62430-3, Addison-Wesley, June 1993.

[15] R. Katz and H. Boyet, *The 16-Bit 8096 Programming, Interfacing, Applications*, Microprocessor Training Inc., 1986.

이 신

1994년 2월 숭실대학교 전자계산학과 졸업(학사). 1997년 2월 숭실대학교 전자계산학과 졸업(석사). 1997년 3월 ~ 현재 숭실대학교 컴퓨터학과 박사과정. 관심분야는 실시간 시스템, 운영체제, 프로그래밍 언어 등

손 혁 수

1997년 2월 숭실대학교 전자계산학과 졸업(학사). 1999년 2월 숭실대학교 전자계산학과 졸업(석사). 1997년 3월 ~ 현재 한우테크(주) 부설 기술연구소 근무. 관심분야는 실시간 운영체제, 객체 모델링, 시스템 소프트웨어 등

양 승 민

1978년 2월 서울대학교 전자공학과 졸업(학사). 1983년 4월 미국 Uni. of South Florida 전산학(석사). 1986년 12월 미국 Univ. of South Florida 전산학(박사). 1988년 ~ 1993년 미국 Univ. of Texas at Arlington 조교수. 1993년 ~ 현재 숭실대학교 컴퓨터학과 부교수. 1996년 ~ 1998년 국회도서관 정보처리국장. 관심분야는 실시간 시스템, 운영체제, 결함허용 시스템 등.