특 집

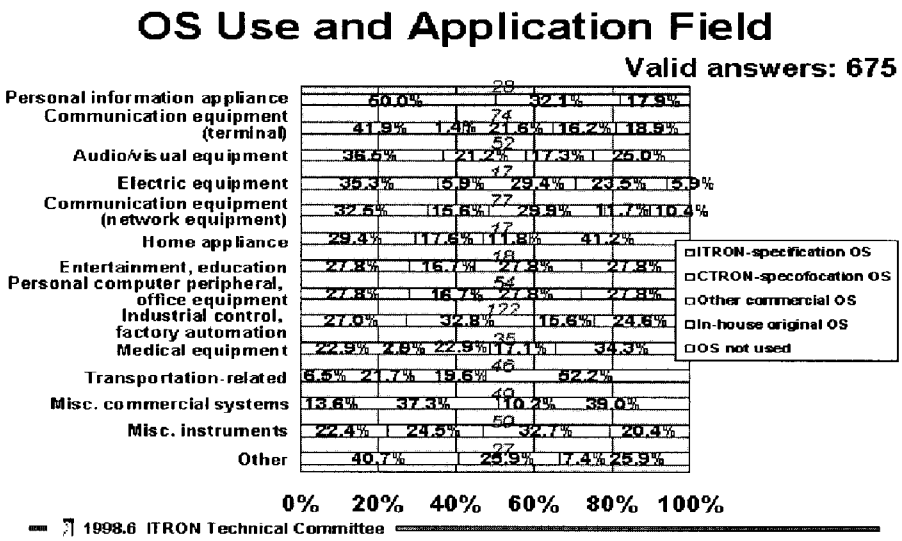
정보가전을 위한 실시간 운영체제 및 미들웨어

홍 성 수*

●목 차●

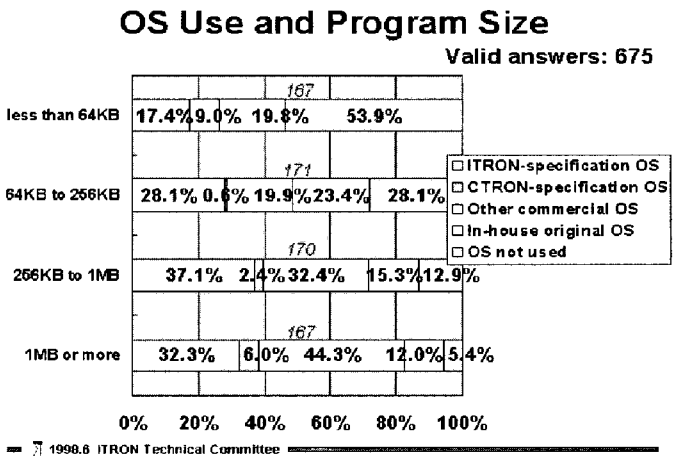
1. 서 론

정보 기술, 가전 기술, 그리고 원격 통신(telecommunication) 기술 같은 전혀 다른 기술이 융합됨에 따라, 정보가전 기기는 세계 가전 시장에서 새로운 컴퓨팅 플랫폼으로서 빠른 속도로 떠오르고 있다. 정보가전 기기는 개인용 컴퓨터와 공학용 워크스테이션과 같은 범용 시스템과 여러 면에서 다르다. 정보가전 기기는 종종 매우 높은 이동성을 가지는 내장형 시스템으로 개발된다. 이는 정보가전 기기가 매우 제한된 하드웨어 자원을 이용하면서, 전력 소모 면에서도 효율적인 장치로서 설계되고 구현



(그림 1) 응용 분야별 운영체제 사용정도

* 서울대학교 전기공학부 조교수



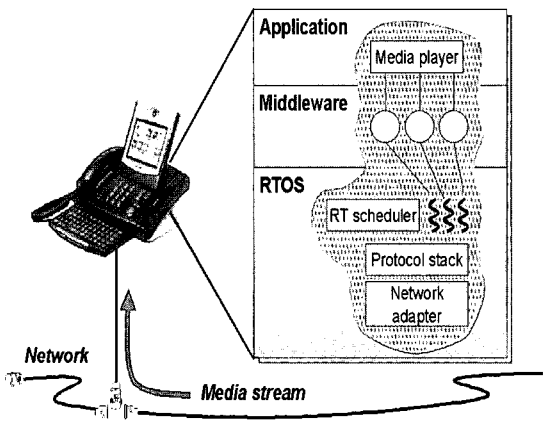
(그림 2) 프로그램 크기별 운영체제 사용 경향

되어야 함을 의미한다. 또한 정보가전 기기는 도처에 산재한 (ubiquitous) 컴퓨팅 플랫폼에 적용된다. 이는 유선 백본과 무선 임시 연결을 이용해서, 매우 복잡하고 동적으로 변화하는 네트워크를 형성함을 의미한다.

따라서 내장형 시스템 개발자들은 위와 같은 기술적인 요구를 만족시키기 위한 어려운 설계 문제에 직면한다. 개발자들은 종종 하드웨어와 소프트웨어를 철저하고 집중적으로 최적화해야 하거나, 다양하고 동적인 네트워크 상에서의 복잡한 분산 프로그래밍을 해야한다. 이는 프로그래머가 이기종성과 재구성성 (reconfigurability)을 처리할 수 있도록 내장형 시스템을 매우 복잡하고 정교하게 만들것을 요구한다. 따라서, 실시간 운영체제 및 잘 정의된 네트워크 프로토콜, 그리고 컴포넌트 기반 미들웨어 시스템 없이 내장형 시스템을 설계하는 것은 불가능하지는 않다 할지라도 매우 어려운 문제가 된다.

내장형 시스템 소프트웨어는 이미 전자장비와 사무용 기기 등에서 광범위하게 사용되고 있는 기반 기술이다. (그림 1)은 일본 ITRON 협회 [1]에서 98년 내장형 시스템 엔지니어들을 대상으로 조사한 것으로서, 각종 내장형 시스템의 평균 75% 이상이 이미 운영체제를 사용하고 있음을 알 수 있다. 특히 이러한 비중은 응용 프로그램의 역할이 큰 첨단 제품에서 더욱 높아진다. (그림 2)를 보면 프로그램 크기가 1MB 이상인 제품에서는 불과 5.4%만이 운영체제 없이 개발되고 있다는 사실을 알 수 있다. 이것은 정보가전 기기와 같이 복잡한 시스템에서는 운영체제와 미들웨어로 구성된 체계화된 시스템 소프트웨어의 도움 없이는 제한된 시간 내에 양질의 제품을 개발해낼 수 없기 때문이다.

(그림 3)은 전형적인 내장형 기기인 웹이 가능한



(그림 3) 전형적인 정보가전 내장형 시스템의 소프트웨어 구조

비디오폰의 내부 소프트웨어 구조를 보여준다. 소프트웨어의 계층의 최 하단인 실시간 운영체제는 입/출력 기기와 하드웨어 컴포넌트를 추상화하고, 우선 순위 기반 선점 멀티태스킹과 태스크 동기화 같은 실시간 스케줄링 기능을 제공한다. 실시간 운영체제는 또한 TCP/IP 프로토콜 같은 표준 프로토콜 스택 구현도 포함한다. 응용 프로그램과 실시간 운영체제 사이에 위치하는 미들웨어는 네트워크에 연결된 내장형 시스템에게, 추상화된 소프트웨어 버스를 제공하여 분산 플러그 앤 플레이 (plug and play)를 가능하게 한다. 이것은 소비자가 네트워크 주소 등록 없이 정보가전 기기를 네트워크 상에 붙일 수 있게 하고, 디바이스 드라이버를 다운로드 하지 않고도 네트워크 상의 다른 서비스들을 사용할 수 있게 한다.

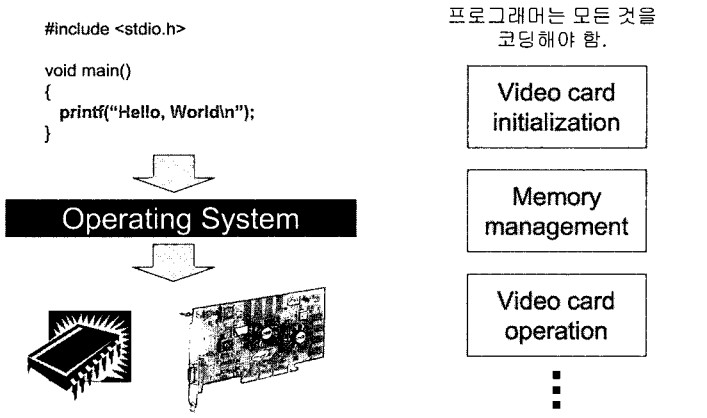
본 논문에서는 실시간 운영체제의 본질 적인 면을 설명하고, 실시간 운영체제를 전통적인 운영체제와 명백히 구분하기 위한 특징들을 설명한다. 그리고, 정보가전 내장형 시스템에 널리 사용되고 있는 미들웨어 기술을 개관한다.

2. 실시간 운영체제

실시간 운영체제가 내장형 시스템 개발자에게 제공하는 주된 기능은 (1) 하드웨어 특성들에 대한 추상화, (2) 멀티태스킹, (3) 예측 가능한 문맥 교환 등이다.

2.1 하드웨어 특성의 추상화

하드웨어 특성의 추상화는 범용 운영체제를 포함하여 모든 운영체제 시스템에 공통적인 기본적인 운영체제의 기능이다. 실시간 운영체제를 사용하지 않고 내장형 시스템의 소프트웨어를 개발하려면, 프로그래머는 응용 프로그램이 관련된 모든 하드웨어의 관리에 대한 코딩을 해주어야 한다. (그림 4)는 일례로, "Hello, World." 라는 간단한 문장을 디스플레이 화면에 보여주고자 할 때, 프로그래머가 어떠한 프로그래밍을 해야 하는 지를 보여준다. (그림 4) (b)는 실시간 운영체제 없이 프로그래밍하는 경우로. 프로그래머는 비디오 카드 초기화, 메모리 관리, 비디오 카드 동작의 모든 작업들에 대하여 코딩을 해주어야 한다. 실시간 운영체제를 사용하면 (그림 4) (a)에서와 같이 사용자는 고급 언어로 간단하게 코딩을 할 수 있다. 정보가전 기기의 소프트웨어는 복잡하고 고기능성을 요구하기 때문에 (그림 4) (b)와 같이 실시간 운영체제 없이 내장형 시스템의 소프트웨어를 개발, 유지, 보수, 관리하는 것은 불가능에 가깝다.

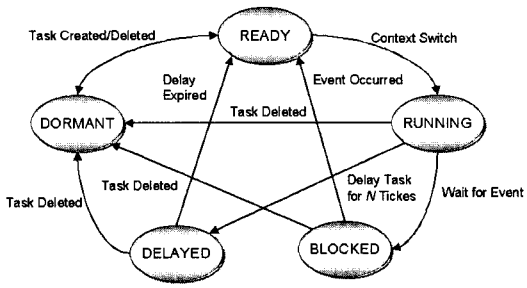


(a) RTOS를 사용하는 경우 (b) RTOS를 사용하지 않는 경우

(그림 4) 내장형 시스템 설계에 있어서의 RTOS의 필요성

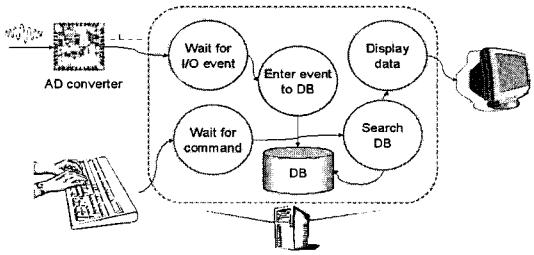
2.2 멀티태스킹 (multi-tasking)

태스크는 응용 프로그램의 실행 가능한 기본 단위를 나타내는 설계 단계의 단위이다. 태스크는 수행을 중단했다가 다시 시작할 수 있도록, 메모리에 런타임 문맥을 가지며 생성 후 (그림 5)와 같은 상태 천이를 할 수 있다. 태스크는 통상 응용 프로그램의 독립적인 수행 컴포넌트를 나타낸다. 내장형 시스템의 소프트웨어는 종종 상호 작용하는 태스크들의 그룹으로써 구조화된다. 멀티태스킹이란 하나 이상의 태스크들을 CPU 상에서 스케줄링하고 전환하는 과정을 말한다. 멀티태스킹을 통하여 응용 프로그램의 동시성을 증가시킬 수 있으며, 이는 CPU의 이용률을 증가시키는 데에도 도움이 된다.



(그림 5) 태스크의 상태 천이도

(그림 6)은 데이터 획득 시스템의 멀티태스킹 예를 보여준다. 아날로그 신호로 발생되는 데이터는 AD Converter를 통하여 디지털 신호로 변환된다. "Wait for I/O event" 태스크는 이를 주기적으로 받아들이고 "Enter event to DB" 태스크에게 데이터를 전달한다. "Enter event to DB" 태스크는 받아들인 데이터를 DB에 저장한다. "Wait for Command" 태스크는 사용자의 입력을 주기적으로 검사하거나, idle 상태에 있다가 인터럽트에 의하여 깨어나 사용자의 입력을 받아들인다. 사용자의 입력이 있으면 "Search DB" 태스크에게 사용자의 입력을 전달하고, "Search DB" 태스크는 DB를 검색한 후 결과를 "Display data"에게 넘겨준다. "Display Data"는 결과 정보를 디스플레이 화면에 나타낸다. 이와 같이



(그림 6) 데이터 획득 시스템에서의 멀티태스킹

이 예에서는 5 개의 독립적으로 수행 가능한 태스크가 상호 작용하면서 하나의 CPU 위에서 각각의 문맥을 가지고 수행된다.

개발자는 DARTS [2]와 같은 잘 알려진 설계 방법론들을 이용하여 주어진 요구사항을 주의 깊게 분석함으로써 시스템 설계 단계에서 태스크들을 추출해낼 수 있다. 그러므로 만약 실시간 운영체제가 직접적으로 멀티태스킹을 지원한다면 개발자는 쉽게 태스크 기반 설계를 적은 노력으로 실시간 구현으로 변환할 수 있다.

내장형 시스템 소프트웨어를 구조화하는 가장 간단한 방법은 무한 폴링 루프를 이용하는 것이다 [3]. (그림 7)은 (그림 6)의 데이터 획득 시스템에 대한 코드의 예로, 차례로 입력 이벤트를 받아들이고 (check_inputs()에서 키보드 입력을, check_digital_interfaces()에서 디지털 신호를 받아들임), 처리 (application())하는 무한 폴링 루프의 전형적인 코드 구조이다. 무한 폴링 루프는 간단한 내장형 응용 프로그램의 경우 효과적인 구조 도구이다. 특히 이들이 실시간 운영체제의 멀티태스킹 능력에 의존

```
Main ( )
{
    for (;;)  {
        check_inputs( );
        check_digital_interfaces( );
        application( );
    }
}
```

(그림 7) 무한 폴링 루프의 전형적 코드 구조

```
Main ( )
{
    i = 0 ;
    for (;;) {
        check_inputs( ) ;
        check_digital_interfaces( ) ;
        switch (i) {
            case 0 : application0( ); break;
            case 1 : application1( ); break;
            case 2 : application2( ); break;
            case 3 : i = 0; break;
        }
        i++;
    }
}
```

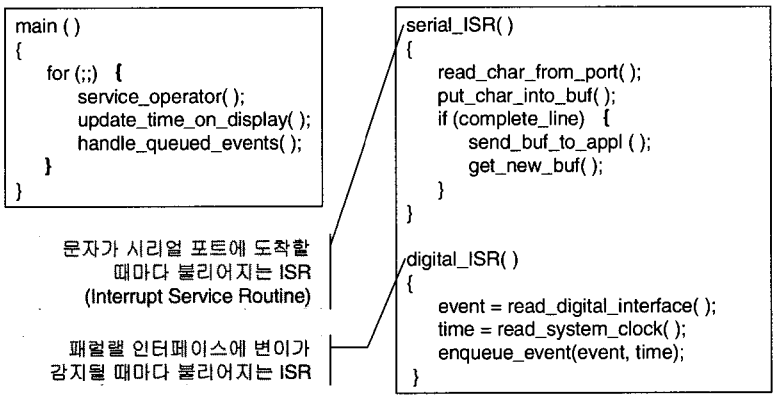
(그림 8) 시간성이 중시되는 이벤트를 위하여 쪼개어진 코드

하지 않는 다는 점에서 매우 효율적일 수 있다. 그러나 다음과 같은 이유에서 복잡한 내장형 응용 프로그램의 구현에는 적당하지 않다.

- 많은 프로세서 이용률이 폴링 루프에서 불필요하게 낭비된다. 이는 폴링 루프의 주기가 모든 입력 이벤트들의 주기 중에서 가장 작은 값으로 결정되기 때문이다. 주 폴링 루프는 쓸데없이 자주 도착하지 않는 이벤트들을 검사한다.
- 폴링 루프에서 한 이벤트 (결과적으로 각 모든 이벤트)의 최대 응답 시간은 폴링 루프의 총 수행 시간으로부터 결정된다. (그림 7)에서 이벤트들의 최대 응답 시간은 입력을 체크하는 함수 (check_inputs()와 check_digital_interfaces ())의 실행시간을 무시할 수 있다고 가정하면 application()함수의 최악 실행 시간이 된다.

만약 실시간 (time-critical) 이벤트가 매우 짧은 반응시간을 요구한다면 무한 폴링 루프의 주요부는 입력 이벤트들의 도착을 더 자주 확인할 수 있도록 (그림 8)에 보여진 것처럼 분해되어야만 한다. 프로그램 코드가 암묵적인 시간 가정을 포함하고 있기 때문에 실제로 이러한 변환은 코드를 매우 알아보기 힘들기 만들기 때문에 소프트웨어 관리가 심각하게 어려워진다. 또한 폴링에 의한 입출력 처리에 의하여 CPU 사이클을 낭비한다. 한편, 코드가 분해된 뒤에서 본래의 알고리즘과 비교해 분해된 코드가 올바른지를 검사해야 한다. 따라서 코드의 분해 작업이 한번에 끝날 수 있는 것이 아니라 반복적으로 이루어져야 하며, 수정될 때마다 시간성/기능성의 정확성에 대한 검사가 필요하다.

이러한 문제들은 무한 폴링 루프의 주요부로부터 실시간 이벤트들을 분리해 냄으로써 부분적으로 해결될 수 있다. 일례로 이들 이벤트들을 하드웨어 인터럽트로 대응시킴에 의해서 해결할 수 있다. (그림 9)는 인터럽트를 사용한 무한 폴링 루프의 구현 예이다. 입력을 체크하는 두 함수 check_



(그림 9) 인터럽트를 사용한 무한 폴링 루프

```
service_operator( )
{
    for(;;)  {
        wait_for_cmd( ) ;
        search_database( );
    }
}
```

```
update_time_on_display( )
{
    for(;;)  {
        wait_one_second( );
        display_time_1( );
    }
}
```

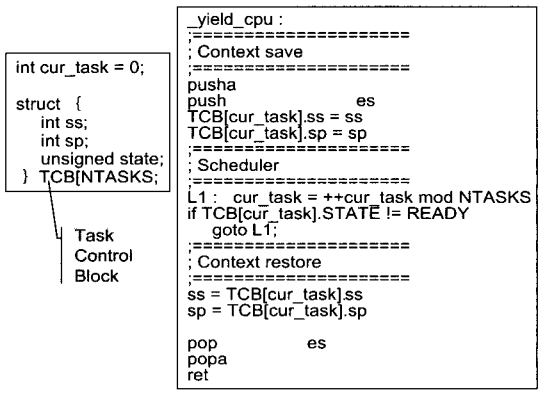
```
handle_queued_events( )
{
    for(;;)  {
        wait_for_event( );
        enter_event_in_database( );
    }
}
```

(그림 10) 멀티태스킹에서의 각 태스크 코드의 예

inputs()와 check_digital_interfaces()를 각각 인터럽트 서비스 루틴 (ISR, Interrupt Service Routine)으로서 serial_ISR()과 digital_ISR()로 구현하였다. 그러나 이러한 접근 방법도 여전히 문제점을 가지고 있다. 이벤트 처리가 과도하다면, 인터럽트가 없이 구현된 무한 폴링 루프와 마찬가지의 문제점을 가지게 된다. 가장 큰 문제점은 많은 실시간 이벤트를 처리할 수는 없다는 점이다. 왜냐하면 하드웨어 우선순위 단계의 수는 많은 내장형 시스템에서 20~30개 이하로 고정되어 있기 때문이다.

따라서 내장형 시스템을 위한 태스크 기반 설계 방법과 멀티태스킹 실시간 커널이 필요하다. (그림 10)은 (그림 6)의 시스템이 멀티태스킹 실시간 커널에서 구현될 때 수행되는 각 태스크 (일부 태스크) 코드에 대한 예를 보여준다. 멀티태스킹은 수행 중 태스크의 문맥을 저장하고 복구할 수 있는 기능이 지원되어야 가능하다. 태스크의 수행 중 문맥은 태스크 코드, 데이터 세그먼트, 스택, CPU 레지스터 등으로 구성된다. 태스크의 문맥 전환은 멀티태스킹 커널이 다른 태스크를 수행하겠다고 결정할 때 일어나며, 커널이 현재 수행 중인 태스크의 문맥을 그 태스크의 스택에 저장하고, 새로 수행시켜야 할 태스크의 문맥을 그 태스크의 스택으로부터 복구함으로써 간단하게 이루어진다. (그림 11)은 태스크의 문맥을 저장하기 위한 TCB (Task Control Block)의 자료 구조와 태스크의 문맥 전환을 위한 시스템 콜인 yield_cpu()의 전형적인 코드 구조를 보여준다.

태스크는 중요도에 따라 우선 순위를 가질 수 있는데, 가장 높은 우선 순위를 가진 태스크가 항상 ready 큐에서 선택되어 CPU의 제어권을 받아야 한다. 멀티태스킹 커널은 선점형 커널과 비선점형 커널의 두 가지 종류로 분류할 수 있다. 완전 선점형 커널에서는 만약 현재 수행중의 태스크의 우선 순위보다 더 높은 우선 순위의 태스크가 시스템에 도착하면, 현재 수행중인 태스크는 선점되고, 높은 우선 순위의 태스크가 즉각적으로 CPU의 제어권을 받게된다. 비선점형 커널에서는 현재 수행중인 태스크가 수행을 종료할 때까지 문맥 전환이 이루어지지 않는다. 선점형 커널은 시스템 응답 시간이 중요할 때 사용되며, 대부분의 상용 실시간 운영체제에서 사용된다.



(그림 11) 문맥 전환 코드

2.3 예측 가능한 문맥 전환

내장형 시스템 개발자 간에는 사용 실시간 운영체제의 기능에 관하여 뚜렷한 합의가 있다. 그러나 실질적으로, "실시간 운영체제"의 용어는 다소 모

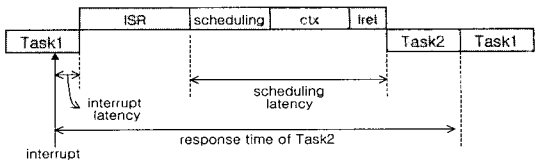
호한 방식으로 사용되고 있다. 여기에서는 실시간 운영체제의 기본적인 구조를 정의하는 네 가지 필요 조건들을 검토한다.

비록 모든 내장형 시스템들이 실시간 시스템이 아니지만 많은 내장형 시스템이 실시간 요구 사항을 가지고 있다. 실시간 시스템은 "시간에 맞게" 올바른 결과를 전달해야 하는 시스템으로 정의된다. 제 시간에 맞추어 나오지 않은 결과는 잘못된 결과가 시스템에 해를 주는 만큼 악영향을 끼친다. 따라서, 시스템이 사용되기 전에 내장형 시스템 개발자들이 시스템을 분석할 수 있는 것이 매우 중요하다. 그 결과로 내장형 시스템들은 정적 스케줄 분석을 포함한 다양한 정적 분석에 대하여 기능적, 시간적 행동 양식이 충분히 결정적이고 예측 가능하도록 설계되어야 한다.

이러한 요구 사항들을 만족시키기 위하여 실시간 운영체제들은 내장형 시스템 개발자들에게 결정적으로 분석할 수 있는 시스템 소프트웨어 플랫폼을 제공해야 한다. 구체적으로, 실시간 운영체제들은 실시간 태스크를 처리함에 있어서 제한되지 않는 문맥 교환 지연을 발생시키지 않아야 한다. 이는 실시간 운영체제 시스템이 다음의 네 가지 필요 조건을 만족해야 함을 의미한다.

1. (완전) 선점형 커널을 제공해야 한다.
2. 우선 순위 기반 스케줄링을 제공해야 한다.
3. 인터럽트 지연 시간이 일정 범위 안에 있어야 한다.
4. 스케줄링 지연 시간이 일정 범위 안에 있어야 한다.

처음의 두 조건은 시간 요건이 중대한 태스크가 높은 우선 순위를 할당받고, 불리었을 때, 제한된 지연 시간 안에 처리되어야 함을 의미한다. 만약에 실시간 운영체제가 단지 비선점형 커널을 가지고 있다면, 새롭게 도착한 더 높은 우선 순위의 태스크가 현재 수행 중인 낮은 우선 순위 태스크가 CPU를 내놓을 때까지 지연될 수 있다.



(그림 12) 문맥 전환 지연 시간의 분해

세 번째, 네 번째 조건들은 시간 조건이 중대한 태스크의 문맥 전환 지연 시간을 한정할 수 있는 데에 도움이 된다. (그림 12)는 문맥 전환 지연 시간을 분해한 모습을 보여준다. (그림 12)에서 태스크 1이 수행중인 동안 태스크 2를 깨우면서 시스템에 인터럽트가 발생하여 태스크 2를 깨우는 상황을 보여주고 있다. 이 때 태스크 2가 태스크 1보다 우선 순위가 높다고 가정하자. 태스크 2가 수행되기 전 스케줄러와 디스패처의 루틴이 들어 있는 인터럽트 서비스 루틴이 수행된다. 인터럽트 발생으로부터 인터럽트 서비스 루틴의 첫 번째 명령 수행까지의 시간 간격을 "인터럽트 지연 시간"이라고 한다. 또한 스케줄러 루틴의 실행 시간을 "스케줄링 지연 시간"이라고 한다. 태스크 2의 응답 시간의 범위를 제한하기 위하여 실시간 커널은 태스크 선점 동안 문맥 전환 지연 시간, 즉 인터럽트 지연 시간과 스케줄링 지연 시간의 범위를 제한해야 한다. 여기에서 인터럽트 서비스 루틴의 수행 시간은 비록 문맥 전환 시간의 부분이라 할지라도 고려하지 않았다. 이는 인터럽트 서비스 루틴들은 응용 태스크 코드처럼 진정한 운영체제 코드가 아니기 때문이다. 인터럽트 지연 시간은 운영체제와 응용 프로그램들에서 인터럽트 불능 코드 영역에 의해 결정된다. 실시간 시스템에서 인터럽트는 최대한 주의해서 억제되어져야 한다. 그렇지 않으면 문맥 전환이 지연 시간의 범위를 제한할 수 없을 수도 있다. 스케줄링 제한시간은 커널의 우선 순위 중재 작용에 의해 결정된다. 상업적인 실시간 운영체제들은

이러한 제한 시간 값을 가지고 공개된다. 여기에서 소개한 실시간 운영체제의 네 조건들은 단지 필요 조건들이다. 상업적인 실시간 운영체제들은 전원 관리 모듈이나 안정적인 파일 시스템과 다양한 디자인 툴 등의 특징들을 가지고 출시되고 있다.

3. 미들웨어

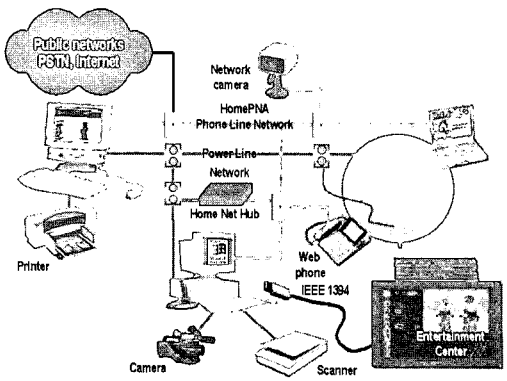
정보가전이라는 말이 뜻하는 바와 같이 정보가전 기기는 인터넷과 홈 네트워크에 접속되기 위하여 설계되고 개발되었다. 최근 많은 홈 네트워크 기술이 나타나고 있으며, 시장 점유를 위해서 서로 경쟁하고 있는 상태이다. 현재 추세로는 가까운 미래에 한 가지 홈 네트워크 기술이 전 홈 네트워크 시장을 독점할 것으로 보이지는 않는다. 따라서 가정 내의 네트워크 인프라는 여러 가지 종류의 네트워크 미디어와 프로토콜이 혼재하게 될 것으로 예상된다. 또한 정보가전 기기의 다양성으로 인해서 가정은 매우 복잡한 분산 환경이 될 것이다. (그림 13)은 HomePNA (Home Phoneline Networking Alliance [4])와 전력선 네트워크, IEEE 1394 및 정보가전 기기들로 구성된 홈 네트워크의 예를 보여주는 그림이다.

홈 네트워크의 복잡성과 이질성은 분산 내장형 시스템 프로그래머에게 중대한 설계 문제를 안겨주게 되었다. 이 문제들은 CORBA [5], DCOM [6]과 Jini [7], UPnP [8]등과 같은 미들웨어 기술들을 통해서 부분적으로 해결이 될 수 있다. 최근 자원이 제한된 내장형 시스템을 위해서 이러한 미들웨어들을 특화하는 방법에 관한 몇 가지 연구 활동이 있었다 [9, 10, 11].

내장형 미들웨어는 네트워크 상에서의 서비스와 이러한 서비스를 사용하는 소프트웨어 구성 요소들 간의 자연스러운 상호작용을 위한 간단한 하부 구조, 또는 소프트웨어 버스를 제공한다. 이는 서비스들이 안정적으로 네트워크에 참가하고 떠날 수 있도록 해준다. 또한 클라이언트에게는 일람된 서비스의 가용성을 보장해준다. 그렇지 못하면 적어도 명확한 오류 조건을 제공해준다. 사용자는 서비스가 제공하는 객체를 사용하여 서비스와 상호작용을 할 수 있다. 이러한 객체는, 사용자가 처음 보는 종류의 서비스와도 통신할 수 있도록, 소프트웨어 컴포넌트로 다운로드된다. 다운로드된 객체는 기존 네트워크에서의 디바이스 드라이버와 같은 것으로, 서비스와의 구체적인 통신 방법을 알고 있다 [7]. 이러한 시나리오를 지원하기 위해서는 내장형 미들웨어는 다음과 같은 특징을 제공하여야 한다.

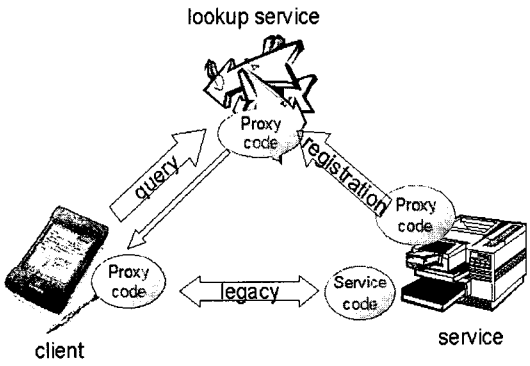
- 컴포넌트 기반 소프트웨어 구조
- 객체 지향 프로그래밍 인터페이스 및 네이밍
- 분산 디렉토리 서비스 및 네임 바인딩
- 오류 검출 및 오류 격리 및 복구
- 가입 기반 그룹 통신
- 객체 이동

"컴포넌트"는 개별적으로 개발되어서 독립적으로 수행될 수 있는 코드 단위를 말하는 소프트웨어 공학 용어이다. 컴포넌트는 시스템 전체에 공표되는, 외부에서 인식 가능한 이름과 인터페이스를 가지고 있다. 인터페이스는 제공할 서비스들과 각 서



(그림 13) 다양한 정보가전 기기들과 이기종의 홈 네트워크 플랫폼

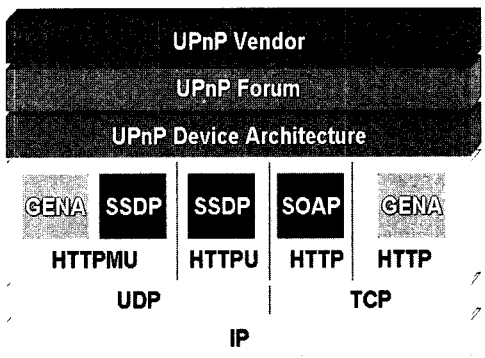
비스의 인수 리스트로 구성된다. 만약 분산 플랫폼이 컴포넌트 기반 미들웨어 기술을 사용하여 구성되어 있다면, 잘 정의된 인터페이스와 공개된 이름을 통해서 네트워크 상의 모든 서비스를 접근할 수 있으므로, 새로운 내장형 기기를 플랫폼에 추가하는 것은 매우 쉬운 일이다. 컴포넌트 기반 소프트웨어 개발과 관리를 지원하기 위해서는 미들웨어 기술은 디렉토리 서버를 제공하여야 한다. 디렉토리 서버는 네트워크 상의 클라이언트가 설계된 서비스들을 찾아 볼 수 있는 디렉토리를 제공한다.



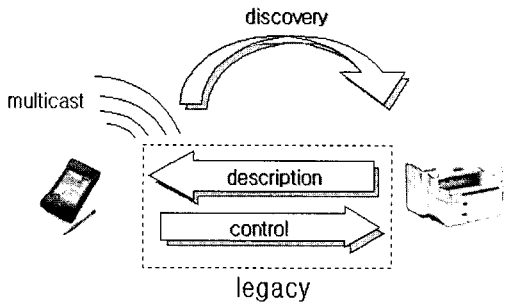
(그림 14) Jini의 분산 플러그 앤 플레이

정보가전 기기는 자주 네트워크에 접속되었다 다시 분리되게 된다. 따라서 미들웨어 시스템은 클라이언트와 서비스 모두에 대해서 동적으로 등록을 할 수 있는 기능 및 다양한 시스템 오류를 처리할 수 있는 서비스를 제공하여야만 한다. (그림 14)는 Jini 시스템의 서비스 등록과 검색 (look up) 과정을 보여준다. 구체적으로, 네트워크 프린터가 네트워크 상에 설치되었을 때 프린팅 서비스와 관련 프록시 코드 모두를 프록시 서버에게 등록할 필요가 있음을 보여 준다. 프린터가 제공하는 프록시 코드는 클라이언트가 프린터에 대해서 상세히 모르더라도 프린터가 제공하는 서비스에 접근할 수 있도록 하는 루틴을 포함하고 있다. 이후에, 그림에서의 PDA와 같은 이동 터미널이 네트워크에 접속이 되었다고 하자. PDA가 프린트 서비스를 원한다면 서비스 디렉토리를 검색 (검색 서비스 이용)하기 위해서 프록시 서버에 접근할 수 있다. PDA는 프록시 서버로부터 프록시 코드를 다운로드 받고, 그 프록시 코드를 통해서 직접 프린터에 접근할 수 있다.

(그림 15)는 UPnP [8]시스템의 구조를 보여 준다. UPnP는 기존의 프로토콜 (IP, TCP, UDP)을 그대로 사용하고, 그 위에 HTTP, HTTPMU (HTTP Multicast over UDP), HTTP (HTTP over UDP)를 확장한 GENA (General Event Notification Architecture), SSDP (Simple Service Discovery Protocol), SOAP (Simple Object Access Protocol)을 사용하고 있다. 이를 바탕으로 UPnP 기기를 제작한 업체나 UPnP 포럼은 UPnP 기기 구조가 정의한 형식을 가지고 각각의 기기에 맞도록 특화한 내용으로 만들게 된다. (그림 16)은 UPnP 시스템에서 네트워크 프린터가 네트워크 상에 설치되어 있을 때 어떻게 PDA가 프린터를 사용할 수 있는 지를 보여 준다. PDA는 처음에 멀티캐스트 채널을 통해서 프린터를 발견하고, 프린터는 자신의 주소를 알려준다 (디스커버리). PDA는 프린터 사용법을 요청하고, 프린터는 PDA에 자신의 사용법을 알려준다 (디스크립션). 그러면 PDA는 디스크립션 메시지에서 어떻게 프린터를 사용할 수 있는 지를 알게



(그림 15) UPnP 구조



(그림 16) UPnP의 분산 플러그 앤 플레이

되어서 이를 바탕으로 프린터 서비스를 이용하게 된다 (컨트롤). UPnP는 (그림 14)의 Jini와는 달리 서비스 등록을 바탕으로 검색 (look up) 서비스를 해 주는 역할을 하는 것이 없음을 알 수 있다.

분산 환경에서 플러그 앤 플레이를 자연스럽게 지원하기 위해서, 내장형 미들웨어는 OMG (Object Management Group)의 고장 감내 규약에서와 같이 수동 및 능동 복제 등과 같은 오류 내성 기능을 지원하여야 한다. [11]에서 CAN (Controller Area Network) 기반의 분산 내장형 시스템을 위한 고장 감내 내장형 미들웨어가 제안된 바 있다. 내장형 미들웨어 기술은 곧 성숙하여 홈 네트워크를 포함한 내장형 네트워크의 많은 분야에서 사용될 것이다.

4. 결 론

컴퓨터, 통신, 가전 기술이 융합된 정보가전 기기들이 차세대 정보통신산업의 주력으로 급부상하고 있다. 정보가전 기기를 위한 내장형 시스템의 개발은 하드웨어에 의존한 저수준 최적화와 태스크 기반의 복잡한 시스템 설계, 통신 프로토콜 처리, 분산 네트워크 프로그래밍까지 모두 포함하고 있다. 이에 따라 내장형 시스템 소프트웨어의 위기 (software crisis)에 당면하게 되었다. 최근에 멀티태스킹 실시간 커널 상에서 실행되는 내장형 미들웨어 기술은 이런 내장형 시스템의 문제를 극복할 수 있는 가능한 툴로 인식되고 있다. 이 논문에서는, 실시간 운영체제의 핵심 기능과 실시간 운영체제가 반드시 만족시켜야 하는 네 가지의 필수 조건을 살펴보았다. 그리고 분산 플러그 앤 플레이를 지원하는 내장형 미들웨어 기술을 살펴보았다.

참고문헌

[1] The Real-time Operating system Nucleus (TRON) Architecture, http://tranweb.super-nova.co.jp

[2] H. Comma. A software design method for real-time systems, Communications of ACM, vol.27, no. 7, pp.938-949, Sep. 1984.

[3] J. Ready and D. Howard. Structuring real-time application software Part 1, VMEbus Systems, pp.33-45, Apr. 1991.

[4] HomePNA. Home Phoneline Networking Alliance: Simple, High-speed Ethernet Technology for Home, A white paper. Jun. 1998.

[5] Object Management Group. The Common Object Request Broker: Architecture and specification revision 2.2, Feb. 1998.

[6] N. Brown and C. Kindel. Distributed component object model protocol DCOM/1.0, 1998.

[7] K. Arnold, B. OSullivan, R. Scheifler, J. Waldo, and A. Wollrath. The Jini Specification. Addison Wesley Publishing Co. Jul. 1999.

[8] Universal Plug and Play Forum, http://www.UPnP.org

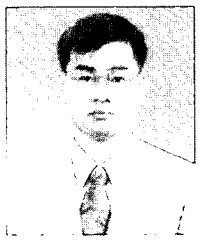
[9] K. Kim, G. Jeon, S. Hong, T.-H. Kim, and S. Kim. Integrating subscription-based and connection-oriented communications into the embedded CORBA for the CAN bus, In IEEE Real-Time

Technology and Applications Symposium, pp. 178-187, Jun. 2000.

[10] Object Management Group. Minimum CORBA joint request submission, OMG document orbos/ 98-08-04 edition, Aug. 1998.

[11] G. Jeon, T.-H. Kim, and S. Hong. A fault-tolerant extension to the embedded CORBA for the CAN bus systems, In ACM SIGPLAN Workshop on Languages, Compilers, and Tools for Embedded Systems, pp.60-69, June 2000.

저자약력



홍 성 수

1986년 공학 학사, 서울대학교 컴퓨터공학과
1988년 공학 석사, 서울대학교 컴퓨터공학과
1988년~1989년 한국전자통신연구소 (연구원)
1994년 전산학 박사, University of Maryland
1994년~1995년 Faculty Research Associate, University of Maryland
1995년 Member of Technical Staff, Silicon Graphics Inc.
1995년~1997년 서울대학교 전기공학부 전임강사
1997년~현재 서울대학교 전기공학부 조교수
관심분야 : 실시간 운영체제, 내장형 미들웨어, 내장형 시스템 설계, 분산 컴퓨터 제어 시스템, 실시간 시스템 설계 방법론, 소프트웨어 공학