# 실시간 운영체제에서 Data Distribution Service 설계 및 구현

정근재°, 이철훈
충남대학교 컴퓨터공학과,
{gjjeong°, clee}@cnu.ac.kr

# Design and Implementation of Data Distribution Service based on Real-Time Operating System

Gun-Jae Jeong°*, Cheol-Hoon Lee
Dept. of Computer Engineering, Chung-Nam National University

요 약

임베디드 시스템의 발달로 인해 기존의 컴퓨팅 패러다임(Paradigm)이 모바일이나 웨어러블 컴퓨팅 등 임베디드 환경으로 급격하게 변화하고 있다. 이렇게 컴퓨팅 패러다임이 변화해도 정보 서비스에 대한 기술이 여전히 필요하다. 네트웍 환경에서 많이 사용하고 있는 정보 서비스 기술중의 하나인 Data Distribution Service(DDS)는 간단한 통신 메커니즘을 기반으로 하면서도 높은 성능으로 정보 서비스를 제공할 수 있다. 따라서, 본 논문에서는 실시간 운영체제를 사용하는 내장형 시스템에 Data Distribution Service(DDS)를 적용하여 데이터의 수집과 전송을 효율적으로 사용하게 하였다.

## 1. 서 론

임베디드 시스템은 전용 동작을 수행하거나 특정 임베디드 소프트웨어 응용프로그램과 함께 사용되도록 디자인된 컴퓨터 시스템 또는 컴퓨팅 장치를 의미한다[1]. 이러한 임베디드 시스템에 사용하는 운영체제가 실시간 운영체제(RTOS : Real-Time Operating System)이다. 미래의 정보 사회의 특징은 멀티미디어화, 지능화, 개인화 등으로 규정할 수 있다. 특히 정보 서비스가 많이 보급되고 사용자의 수가 증가하면서, 통신망을 통한 서비스를 필요로 하게 되었다. 향후 정보 서비스의 사용자는 급증할 것이며 따라서 동시에 많은 수의 사용자에게 효율적으로 정보를 서비스하기 위한 방법이 강구되어야 한다. 이를 위한 정보 서비스를 제공하는 방법중의 하나가 DDS 이다. DDS 는 지역적으로 떨어져 있으면서 통신망으로 연결된 다수의 사용자를 대상으로 동시에 정보를 전달하거나 또는 수집하는 기능을 제공한다. 이러한 DDS 를 임베디드 시스템과 연동하여 사용할 경우 임베디드 시스템에서 데이터를 수집하여 전송하고, 반대로 수집된 정보를 임베디드 시스템 사용자에게 보다 쉽게 전송해줄 수 있다.

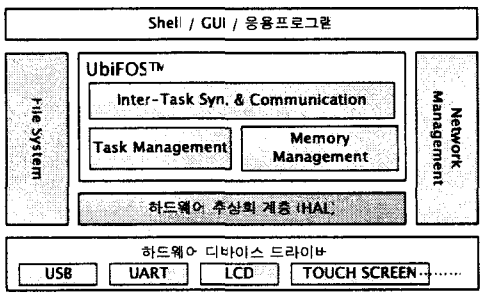
본 논문에서는 실시간 운영체제인 UbiFOS™를 기반으로 DDS를 설계 및 구현 하였다.

본 논문의 구성은 2 장에서 실시간 운영체제 및 DDS 의 전체적인 구성을, 3 장에서는 DDS 를 이용한 데이터 사용의 설계 및 구현 내용을 다룬다. 4 장에서는 테스트 환경 및 결과를 기술하고, 마지막으로 5 장에서는 결론 및 향후 연구 과제에 대해서 기술한다.

## 2. 관련 연구

### 2.1 UbiFOS™

실시간 운영체제인 UbiFOS™은 우선순위 기반의 선점형 스케줄러를 제공하여 태스크의 우선순위를 0 부터 255 까지 256 단계로 구분하고 있다. 그리고 실행 준비된 태스크들 중에서 정해진 시간 내에 가장 높은 우선순위의 태스크를 찾기 위해 별도의 테이블과 리스트를 관리하며, 같은 우선순위의 태스크 사이에서는 선입선출(FIFO) 방법과 Round-Robin 방법을 지원한다.



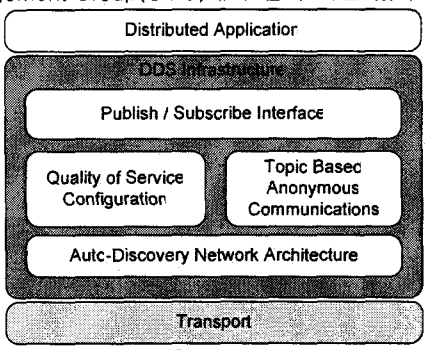

[그림 2-1] UbiFOS™ 전체 구성도

* 본 연구는 정보통신부의 선도기반기술개발사업의 지원으로 수행되었습니다.

2.2 Data Distribution Service(DDS)

DDS 는 분산된 어플리케이션이 통신 메커니즘인 Data-Centric Publish-Subscribe(DCPS)를 사용할 수 있는 API 표준이다. DDS 의 이점은 간단한 Publish-Subscribe 커뮤니케이션 패러다임(Paradigm)을 기반으로 하며, 따라서 정보 서비스를 할 경우 오버헤드가 적어 높은 성능을 요구하는 시스템에서 사용 되어 진다. 또한 정보 전달의 시간 결정성을 보장하고 동적으로 확장, 축소가 가능하다.

DDS 는 통신하기 위한 어플리케이션들이 서로 다른 타입을 가지고 있더라도 통신을 가능하게 하는 내부구조 계층을 제공한다. DDS 의 규격은 Object Management Group(OMG)에서 관리 하고 있다 [3].

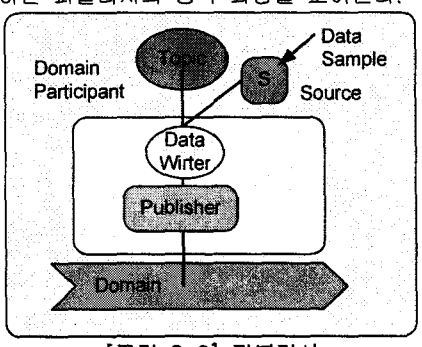



[그림 2-1] DDS 내부구조 계층

2.2.1 퍼블리셔(Publisher)

퍼블리셔는 데이터를 송신하는 역할을 하고, 서브스크라이버와 통신하기 위해 데이터의 이름과 목적지만을 필요로 하고 서브스크라이버에 대한 어떠한 정보도 필요로 하지 않는다.

[그림 2-2]는 데이터를 Data Write 를 통해 퍼블리셔에 전달하고, 전달받은 데이터를 UDP 를 통해 멀티캐스트하는 퍼블리셔의 동작 과정을 보여준다.

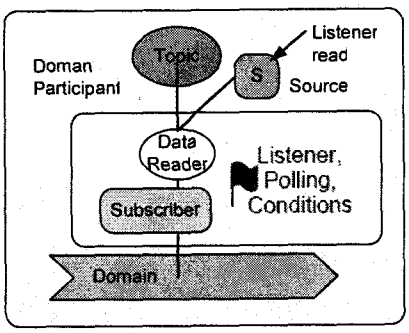



[그림 2-2] 퍼블리셔

2.2.2 서브스크라이버(Subscriber)

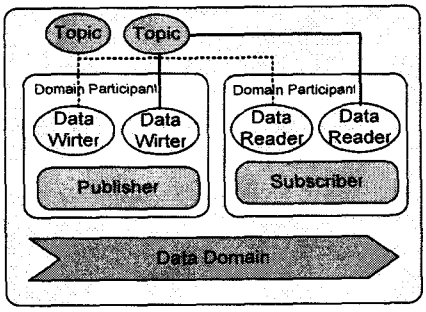
서브스크라이버는 데이터를 수신하는 역할을 한다.

[그림 2-3]는 UDP 멀티캐스트를 통해 전달 받은 데이터를 서브스크라이버에서 받아들여 Data Reader 를 통해 데이터를 읽어 들이는 서브스크라이버의 동작 과정을 보여준다.




[그림 2-3] 서브스크라이버

[그림 2-4]는 서로 다른 토픽을 퍼블리셔를 통해 서브스크라이버로 전달하는 동작 과정을 나타내고 있다. 여기서 토픽은 퍼블리셔와 서브스크라이버사이에 기본적인 연결 포인트를 제공하고, 데이터 타입이나 전송되는 정보의 종류 등을 나타낸다. 각각의 토픽마다 Data Writer 와 Data Reader 가 존재하고 퍼블리셔의 토픽은 서브스크라이버의 토픽과 매치된다.




[그림 2-4] DDS 동작 과정

## 3. 실시간 운영체제에서의 DDS 구현

3.1 퍼블리셔 구현

퍼블리셔는 리눅스 서버에서 동작하면서 각기 다른 토픽의 데이터를 퍼블리쉬하고 토픽은 그림이미지와 정보전달용 텍스트로 하였다.

[그림 3-1]은 토픽으로 사용되는 데이터 구조체를 보여준다. imagebits 는 토픽의 그림이미지가 배열로 변환되어 저장되는 변수이고, text 는 토픽의 정보 문자열이 저장되는 변수, name 은 토픽의 이름, destination 은 토픽의 목적지를 나타낸다.

```
Typedef struct data {
        unsigned char imagebits[145*100];
        char text[15];
        char name[15];
        char destination[15];
}MK_DATA;
```

[그림 3-1] 토픽 구조체

[그림 3-2]의 Data Writer 는 토픽 구조체에 전달하려는 데이터 값을 기록하는 역할을 한다.

```
MK_DATA DataWriter(char text[], unsigned char
imagebits1[], char name[], char destination[])
{
    MK_DATA Topic;
    strcpy(Topic.imagebits, imagebits1);
    strcpy(Topic.text, text);
    strcpy(Topic.name, name);
    strcpy(Topic.destination, destination);
    return Topic;
}
```

[그림 3-2] Data Writer

[그림 3-3]은 토픽을 생성하고 퍼블리쉬하는 리눅스 서버에서의 동작과정을 보여준다. DataWriter()를 통해 토픽을 생성하고 소켓을 통해 전송한다. 이때 전송 도중 토픽이 분실될 경우를 대비하여 2 번 반복하여 전송한다.

```
// 토픽형 변수 선언
MK_DATA temp;
// text 와 imagebits1 을 가지고 토픽형 temp 를 생성
temp = DataWriter("text", imagebits1, "imfo",
                  "168.188.46.173");
memset((char*)&Saddr, (char)0, sizeof(Saddr));
Saddr.sin_family = AF_INET;
Saddr.sin_port = htons(TEST_PORT);
memcpy( (void *)&Saddr.addr.s_addr,
        (void *)Topic.destination, 4);
// 소켓을 생성한다.
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
//send topic
for(k=0 ;k<2 ;k++ ) {
    printf("Send Packet₩n");
    sendto(sockfd, temp.text, strlen(temp.text), 0,
           (struct sockaddr *)&Sddr, sizeof(Sddr));
    sleep(2);
    sendto(sockfd,temp.imagebits,
        strlen(temp.imagebits), 0, (struct sockaddr *)
        &Saddr, sizeof(Saddr));
    sleep(10);
    for(j=0;j<20000000;j++);
}
```

[그림 3-3] 퍼블리셔 동작

### 3.2 서브스크라이버 구현

서브스크라이버는 임베디드 플랫폼에서 동작하고 퍼블리쉬된 데이터를 받아 Consumer 에서 이용 가능하게 한다. 토픽을 Listen 하고 데이터 리더를 통해 데이터를 읽어오는 태스크(UDPReceive_Task)와 토픽을 LCD 창에 출력하는 태스크(GUI_Task )로 동작한다.

[그림 3-4]는 2 개의 태스크를 생성하는 함수이다.

```
// GUI_Task 생성
MK_CreateTask(&Task1, Fun1, 22, "GUI_Task", 1,
              (MK_S8_t *)pAddr, 4096, 0, 0);
// UDPReceive_Task 생성
MK_CreateTask(&Task2, Fun2, 20, "UDP_Task", 1,
              (MK_S8_t *)pAddr, 4096,  0, 0);
```

[그림 3-4] 태스크 생성

[그림 3-5]는 UDPReceive_Task 에서 무한 루프를 돌면서 패킷이 도착하는지 Listen 하면서, 패킷 도착시 Data_Reader()함수를 통해 토픽 구조체로 읽어 들이는 과정이다. 토픽이 비 정상적으로 수신 되었을 경우 다시 패킷이 도착하길 Listen 하고, 토픽이 정상적으로 수신되었을 경우 데이터를 읽어 들이고 GUI_Task 에게 CPU 점유권을 넘겨주는 과정을 보여준다.

```
sockfd = socket(AF_INET, SOCK_DGRAM, 0);
bind(sockfd, (sockaddr *)&Saddr, sizeof(Saddr));

while(1)
{   // 토픽을 수신
    datalen = recvfrom(sockfd, (MK_Void_t *)buff,
                100, 0, (struct sockaddr *) &cliaddr,
                (socklen_t *)&clilen);

    // DATA_Reader 를 호출
    Topic = DATA_Reader((MK_Void_t *)buff )

    // 패킷을 제대로 받지 못했을 경우
    if(datalen <= 0)
    {
        MK_Printf("continue ₩n");
        continue;
     }
     // 패킷을 제대로 받았을 경우
     else {
        flag = MK_InterruptDisable();
        buff[datalen] = '₩0';
        MK_InterruptRestore(flag);

        // 토픽을 받았으므로 출력을 해주기 위해
        // UDPReceive_Task 는 슬립 함으로써
        // GUI_Task 에게 CPU 점유권을 넘긴다.
        MK_SleepTicks(5);
      }
}
```

[그림 3-5] 토픽 수신

[그림 3-6]의 Data Reader 는 전달받은 패킷을 토픽 구조체로 읽어 들이는 역할을 한다. buff[]는 전달받은 패킷을 저장하고 있다.

```
MK_DATA DATA_Reader((MK_Void_t *)buff [])
{
    MK_DATA Topic;
    strcpy(Topic.imagebits, buff[1]);
    strcpy(Topic.text, buff[2]);
    strcpy(Topic.name, buff[3]);
    strcpy(Topic.destination, buff[4]);
    return Topic;
}
```

[그림 3-6] Data Reader

[그림 3-7]은 UDPReceive_Task 에서 받은 토픽을

MBA2440 보드의 LCD 창에 출력하는 과정을 보여준다. 먼저 메인 윈도우를 생성하여 그 안에 Data Reader 를 통해 읽어 들인 토픽 그림 이미지를 출력하고 또 다른 윈도우를 생성하여 토픽 텍스트 문자열을 출력하게 된다.

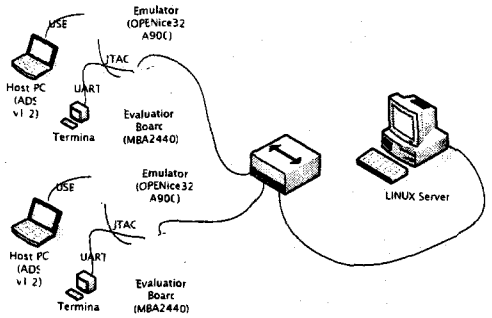
```
// 메인 윈도우를 생성
window1 = GrNewWindow(GR_ROOT_WINDOW_ID,
                      10,10,300,20,1,WHITE,BLACK);
GrMapWindow(window1);
GrText(window1, gc, 10, 15, "Shopping
       Information", -1, MWTF_ASCII);
while ( 1 )
{
  MK_SleepTicks(10);
  // 토픽 그림 이미지 출력
  window2 = GrNewWindow( GR_ROOT_WINDOW_ID,
                    10,40,145,100,1,WHITE,BLACK);
  GrMapWindow(window2);
  GdDrawImage(psd,10,40,&Topic. imagebits);

  window4 = GrNewWindow(GR_ROOT_WINDOW_ID,
                  10,150,300,80,2,LTGRAY,BLACK);
  GrMapWindow(window4);
  // 텍스트 출력부분
  GrText(window4, gc, 10, 15, text, -1,
         MWTF_ASCII);
  MK_SleepTicks(10);
}
```

[그림 3-7] 토픽 출력

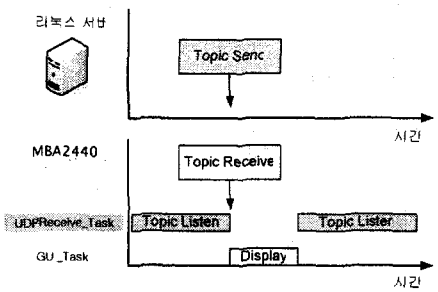
## 4. 테스트 환경 및 결과

본 논문에서 구현한 DDS 는 실시간 운영체제인 UbiFOS™ 를 대상으로 하였으며, 리눅스 서버를 퍼블리셔로, ARM920T 32-bit RICS CPU 가 탑재되어 있는 MBA2440 보드 2 대를 서브스크라이버로 하여 테스트 하였다.



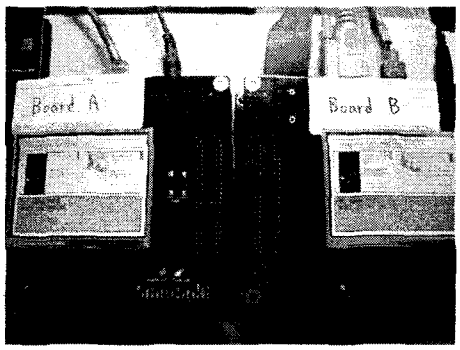

[그림 4-1] 테스트 환경

[그림 4-2]는 프로그램이 동작하는 과정을 그림으로 나타낸 것이다. 리눅스 서버에서 저장되어 있는 토픽을 소켓 생성 후 퍼블리쉬하면 MBA2440 보드의 UDPReceive_Task 에서 패킷을 Listen 하고 있다가 패킷을 수신한다. 받은 패킷을 DataReader() 함수를 통해 토픽 구조체로 변환하고, 그 후 GUI_Task 로 CPU 점유권을 넘긴다. GUI_Task 는 전달받은 토픽을 LCD 창에 출력하고, 출력 후엔 UDPReceive_Task 로 다시 CPU 점유권이 넘어가 패킷을 Listen 하게 된다.




[그림 4-2] 테스트 시나리오

[그림 4-3]을 통해 리눅스 서버에서 그림 이미지와 문자열을 가진 토픽을 퍼블리쉬하면 MBA2440 보드 2 대에 동일한 토픽이 수신되어 서비스 되는 것을 확인할 수 있다.




[그림 4-3] 실험 결과

## 5. 결론 및 향후 연구 과제

본 논문에서 실시간 운영체제에서의 DDS 를 구현하였고, 이를 통해 데이터를 서비스하는 과정을 확인하였다.

차후에는 정보를 수집할 수 있는 센서 보드 등을 사용하여 임베디드 플랫폼을 퍼블리셔로 하고 리눅스 서버를 서브스크라이버로 하여 데이터를 서비스 할 수 있는 DDS 에 관한 연구가 진행되어야 하겠다.

## 참고문헌

[1] Embedded System & RTOS, http://www.aijisystem.com.
[2] David Stepner, et. al., , " Embedded Application Design Using a Real-Time OS " , IEEE, 1999
[3] An Introduction to DDS and Data-Centric Communications http://www.omg.org.