한국정밀공학회지 제 23 권 제 9 호 (2006 년 9 월)
Journal of the Korean Society for Precision Engineering, Vol. 23, No. 9, September 2006

◆특집◆ 자동차 제어 및 전자화 기술

OSEK와 AUTOSAR를 중심으로 본 차량용 OS와 미들웨어 기술 동향

홍성수*, 박지용*, 유우석*

Technology Trends in Automotive OS and Middleware: OSEK and AUTOSAR

Seongsoo Hong*, Jiyong Park* and Wooseok Yoo*

Key Words: Automobile (자동차), Operating System (운영체제), Middleware (미들웨어), OSEK (오섹), AUTOSAR (오토사)

1. 서 론

최근 자동차 산업계는 급격한 환경적 변화를 겪고 있다. 기업간의 기술 경쟁이 첨예화되고 있고, 이에 따라 기업간의 합종연횡이 심화되고 있다. 한편 세계적으로 각국의 환경 오염 규제가 강화되고 있으며, 차량 안전 기준도 점차 엄격해지고 있다. 또한 이와는 별도로 소비자들은 최근까지 급격히 발전되어 온 IT 기술을 자동차에서도 향유하기를 원하고 있다

이러한 변화들의 결과로 자동차 회사는 단지 기계 장치를 제조하는 것을 넘어 높은 수준의 전자 장치와 소프트웨어까지 생산하여야 하는 상황에 이르게 되었다. 이 변화들을 좀 더 정리하면 (1) 전자 장치의 증가, (2) 소프트웨어의 복잡도의 증가, (3) 표준화의 필요성의 증가의 세 단계로 요약할 수 있다. 첫째, 앞서 언급했듯이 경쟁력을 높이고 각종 규제에 대응하기 위해 자동차에 장착되는 전자 장치가 증가하고 있다. 분석에 따르면 2010 년까지 자동차 분야 혁신의 90%가 전장 분야에서 창출될 전망이다.[1] 지금도 엔진, 자동 변속기, 브레이크, 에어백 등에 전자 장치가 널리 사용되고 있으며 자동 주차, 나이트 비전, drive by wire 등의 다양한 기능들이 개발되고 있다.

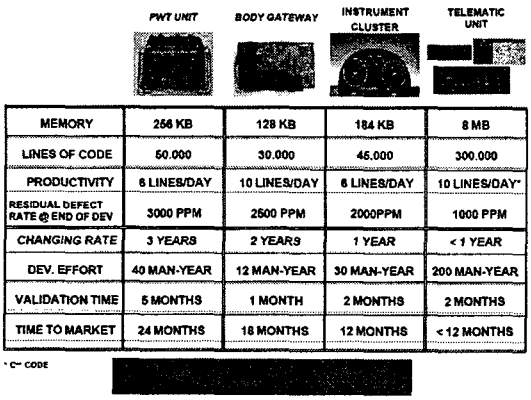
이렇게 전장품의 비율이 증가하면서 ECU 의 개수 또한 계속 증가하고 있다. 이에 따라 파워 트레인, 샤시 등 대부분의 모듈에서 ECU 가 이용되고 있다. 이들은 CAN, LIN, TTP, FlexRay 등의 다양한 자동차용 네트워크 프로토콜을 통해 연결되어 분산 시스템을 이루고 있다. 2003 년 메르세데스 벤츠의 S 클래스의 예를 들면 50 개 이상의 ECU 들이 3 개의 네트워크로 연결되어 150 여 종류의 메시지를 통해 600 여 개의 신호를 주고받고 있다.[2]

둘째, 전자 장치의 비중이 증가하면서 이를 제어하기 위한 소프트웨어의 양과 복잡도가 함께 증가하고 있다. 2010 년까지 전장 분야의 혁신의 80%가 소프트웨어의 영역이 될 것으로 예측되고 있다.[1] 앞서 언급한 벤츠 자동차의 경우 이미 5 백만 라인 이상의 코드가 사용되고 있다. Fig. 1은 파워 트레인과 같은 자동차의 주요 모듈들의 소프트

*서울대학교 전기컴퓨터공학부 대학원 실시간 운영체제 연구실
Tel. 02-880-8370. Fax. 02-882-4656
Email sshong@redwood.snu.ac.kr

웨어 크기와 이들을 개발하는데 소요되는 비용을 보여주고 있다.

소프트웨어의 LOC (lines of code)가 증가하면 이보다 더 빠른 속도로 시스템의 복잡도가 증가한다. 여기서 문제는 복잡도가 증가할 수록 품질은 저하되기 쉽다는 것이다. 실제로 소프트웨어의 복잡도 증가로 인한 자동차 결함 사례가 속출하고 있다. 대표적인 예로 Toyota 사의 하이브리드 자동차인 Prius 는 소프트웨어의 사소한 오류 때문에 가솔린 엔진이 정지하는 문제를 가지고 있었다. 이를 해결하기 위해 Toyota 사는 총 75,000 대를 리콜하여 새로운 소프트웨어를 설치하여야 했다. 또한 BMW 사의 745i 도 엔진 밸브의 타이밍을 조정하는 두 ECU 사이의 동기화 결함으로 주행 중 엔진이 꺼지는 문제가 있었다. 이로 인해 총 5,470 대가 리콜된 바 있다.



MEMORY	256 KB	128 KB	184 KB	8 MB
LINES OF CODE	50,000	30,000	45,000	300,000
PRODUCTIVITY	6 LINES/DAY	10 LINES/DAY	6 LINES/DAY	10 LINES/DAY*
RESIDUAL DEFECT RATE @ END OF DEV	3000 PPM	2500 PPM	2000PPM	1000 PPM
CHANGING RATE	3 YEARS	2 YEARS	1 YEAR	< 1 YEAR
DEV. EFFORT	40 MAN-YEAR	12 MAN-YEAR	30 MAN-YEAR	200 MAN-YEAR
VALIDATION TIME	5 MONTHS	1 MONTH	2 MONTHS	2 MONTHS
TIME TO MARKET	24 MONTHS	18 MONTHS	12 MONTHS	< 12 MONTHS

* C++ CODE

Fig. 1 Development costs of automotive modules

셋째, 이러한 복잡성 문제를 해결하기 위해 자동차용 소프트웨어의 표준화 필요성이 증가하고 있다. 실제로 자동차 업체들은 표준화 단체 등을 통하여 자동차용 소프트웨어 모듈의 작동 환경, 통신 방식, 배포 방식을 단일화하는 작업을 진행하고 있으며 이들의 궁극적 목표는 소프트웨어 모듈의 plug & play 이다.

이와 같은 변화에 대응하기 위해서는 체계화된 런타임 소프트웨어 플랫폼이 반드시 필요하다. 런타임 소프트웨어 플랫폼이란 시스템을 설계하는데 발생하는 위와 같은 복잡한 문제들을 해결해 주고 응용 프로그래머가 프로그램 본연의 기능을 구현하는데 집중할 수 있도록 도와주는 소프트웨어를 말한다. 이러한 런타임 소프트웨어 플랫폼들 중 자동차에 반드시 필요한 것들로 실시간 운영체제와 미들웨어가 있다. 실시간 운영체제는 자동차에서와 같이 시간 제약을 가지는 수 많은 프로그램들이 한 CPU 에서 동시에 안전하게 수행될 수 있도록 해 주는 소프트웨어 플랫폼이다. 미들웨어는 네트워크로 연결된 각 호스트 사이의 통신을 담당하여 분산 시스템을 하나의 단일 시스템으로 간주하고 프로그래밍 할 수 있도록 해 주는 소프트웨어 플랫폼이다. 본 논문에서는 자동차용 실시간 운영체제 표준인 OSEK 과 자동차용 미들웨어의 대표적인 사례인 AUTOSAR 를 각각 소개한다.

2. 차량용 운영체제 – OSEK

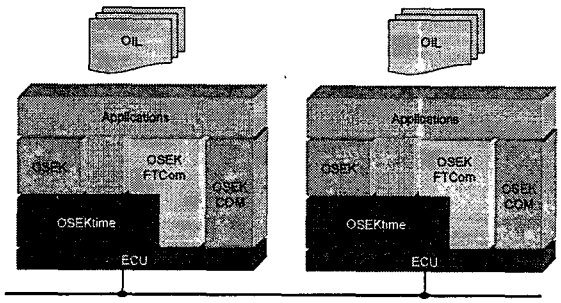


Fig. 2 Relationship of OSEK specifications

OSEK 은 차량용으로 특별히 설계된 실시간 운영체제 표준이다. 정식 명칭은 OSEK/VDX 이다. OSEK 의 개발은 1993 년 OSEK 과 VDX 라는 두 프로젝트가 하나로 연합하면서 시작되었다. 현재 OSEK/VDX 프로젝트는 BMW, Volkswagen, Siemens, Renault 등 유럽계 자동차 회사들의 주도로 진행되고 있다.[3] 본 논문에서는 OSEK/VDX 를 간단히 OSEK 으로 부르도록 한다. 현재 널리 알려진 OSEK 제품으로는 3Soft 사의 ProOSEK 과 Metrowerks 사의 OSEKturbo 등이 있다.

OSEK 표준은 8 개의 세부 표준으로 나누어져 있다. 본 논문에서는 그 중 (1) OSEK OS, (2) OSEK COM, (3) OSEKtime OS, (4) OSEK FTCom, (5) OIL 의 다섯 가지 표준에 대해 살펴본다. 첫째, OSEK OS 는 멀티 태스킹을 지원하는 가장 기본적인 운영환경이다. 여기에서는 태스크의 생성, 이벤트 기반(event triggered)의 태스크 스케줄링, 태스크 동기화

등의 기능이 제공된다. 둘째, OSEK COM 은 태스크들이 메시지를 주고 받으며 서로 통신할 수 있도록 해 준다. OSEK COM 은 같은 ECU 에 있는 태스크들의 로컬 통신은 물론 네트워크를 통해 다른 ECU 에 있는 태스크로의 원격 통신도 지원한다. 셋째, OSEKtime OS 는 시간 기반(time triggered) 태스크 스케줄링을 지원한다. 시간 기반 태스크 스케줄링 에서는 태스크의 스케줄링 정보들이 미리 타임 테이블에 기록되어 정해진 시간에 정해진 순서대로 태스크들이 수행될 수 있도록 한다. OSEKtime OS 와 OSEK OS 는 함께 사용될 수도 있고, 또는 둘 중 하나만 사용될 수도 있다. 넷째, OSEK FTCom 은 OSEK COM 을 확장하여 태스크들이 주고받는 메시지에 고장 감내성을 부여한다. 다섯째, OIL 은 OSEK 의 여러 옵션들을 정의하고 있다. 사용자는 이 OIL 파일을 작성하여 OSEK 을 자신의 목적에 맞도록 최적화할 수 있다. Fig. 2는 이러한 다섯 가지 표준들의 상호 관계를 보여주고 있다. 여기에서는 이들 중 가장 중요한 모듈들인 OSEK OS, OSEK COM, 그리고 OIL 에 대해서만 자세히 설명한다.

2.1 OSEK OS

OSEK OS[4]는 OSEK 의 가장 핵심적인 모듈로 기본적인 실시간 운영체제를 정의한다. 구체적으로 OSEK OS 는 (1) 멀티 태스킹, (2) 이벤트/자원 관리, (3) 인터럽트 관리, (4) 알람(alarm), (5) 에러 처리의 다섯 가지 기능을 제공한다. 본 논문에서는 이들 중 가장 중요한 멀티 태스킹과 이벤트/자원 관리의 두 기능에 대해 설명한다.

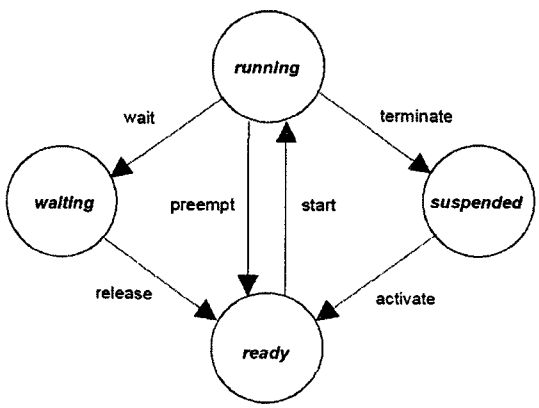


Fig. 3 Task state model of OSEK OS

2.1.1 멀티 태스킹

멀티 태스킹이란 한 CPU 위에서 여러 개의 태스크를 동시에 수행시키는 기능을 의미한다. OSEK OS 는 태스크의 수행 상태를 관리하거나 스케줄러를 통해 태스크의 수행 순서를 조정하는 역할을 한다.

OSEK OS 의 태스크는 Fig. 3 과 같이 running, ready, waiting, suspended 의 네 가지 상태를 가질 수 있다. 이들의 의미는 각각 다음과 같다.

- running: 태스크가 현재 수행 중이다.
- ready: 다른 태스크가 수행을 마치기를 기다리고 있는 중이다.
- waiting: 태스크가 어떤 이벤트가 발생하기를 기다리며 수행을 일시 중단한 상태이다. 이벤트가 발생하면 ready 상태가 된다.
- suspended: 태스크가 종료한 상태이다. 종료된 태스크는 다시 시작될 수 있다.

OSEK OS 는 선점형 실시간 운영체제이다. 이는 한 태스크가 수행 중이더라도, 언제든지 필요에 따라 다른 태스크가 현재 수행중인 태스크를 선점하고 즉시 수행될 수 있다는 뜻이다. 이를 위해 OSEK OS 는 각 태스크마다 별도의 메모리 영역을 할당하고 이 영역에 각 태스크의 상태(레지스터 값 등)를 저장하여 여러 태스크가 CPU 하나를 공유할 수 있도록 한다.

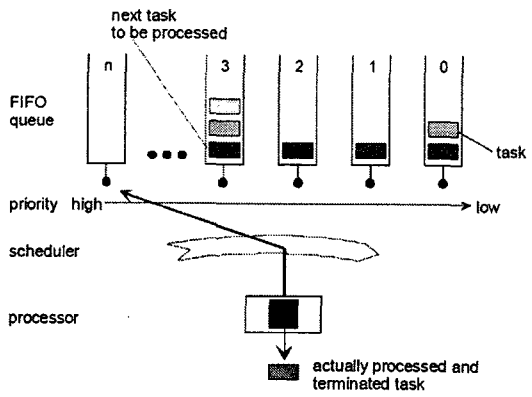


Fig. 4 Operation of OSEK OS scheduler

스케줄러는 ready 상태에 있는 태스크들 중에 다음에 수행할 한 태스크를 선택하는 역할을 한다. OSEK OS 는 16 가지의 우선순위를 지원하는 FIFO 스케줄러를 사용한다. 각 태스크는 0 부터 15 까지

의 우선순위를 가질 수 있으며, 우선순위가 높은 태스크가 먼저 선택된다. 같은 우선순위의 태스크가 여러 개 존재할 경우, 가장 오래 전에 ready 상태가 된 태스크가 선택된다. Fig. 4는 OSEK OS 의 스케줄러가 작동하는 방식을 보여주고 있다. 스케줄러는 16 개의 FIFO 큐를 가지고 있으며, 각 큐에는 해당하는 우선순위의 ready 태스크가 존재한다. 스케줄러는 이들 큐를 순차적으로 검사하면서 수행할 태스크를 선택한다.

OSEK OS 는 멀티 태스킹을 위해 다양한 API 를 제공한다. 이들의 이름과 그 의미는 다음 표와 같다.

Table 1 APIs for multi-tasking

API 이름	의미
DeclareTask()	태스크의 존재를 선언한다.
ActivateTask()	태스크를 시작시킨다.
TerminateTask()	태스크를 종료시킨다.
ChainTask()	현재 태스크를 종료시키고 다른 태스크를 시작시킨다.
Schedule()	스케줄러를 호출한다.
GetTaskID()	현재 수행중인 태스크를 반환한다.
GetTaskState()	태스크의 상태를 반환한다.

2.1.2 이벤트/자원 관리

이벤트 관리란 태스크가 특정 사건(예를 들어, 타이머 알람, 메시지 도착, 공유 자원의 상태 변화 등)을 기다릴 수 있도록 하는 것을 뜻한다. 자원 관리란 태스크들이 자원(메모리, CPU, 디바이스 등)을 안전하게 공유할 수 있도록 자원의 사용 순서를 조정해주는 것을 뜻한다.

OSEK OS 에서 이벤트는 시스템에서 발생할 수 있는 특정 사건을 나타내는 자료 구조이다. 이벤트에는 미리 정의된 것도 있고, 사용자의 필요에 따라 추가적으로 생성된 것도 있다. 태스크는 특정 이벤트가 발생하기를 기다릴 수 있는데, 이 경우 태스크의 상태는 waiting 으로 변하여 더 이상 수행되지 않는다. 한편 태스크는 특정 이벤트를 발생시킬 수 있다. 이 경우, 그 이벤트를 기다리고 있는 태스크들이 waiting 상태에서 ready 상태로 깨어나서 수행을 다시 시작할 수 있다. 이를 위해 이벤트 자료구조에는 wait queue 가 있어서 이 이벤트를 기다리고 있는 태스크들을 저장할 수 있다.

OSEK OS 는 자원 공유 메커니즘을 제공하여 태스크들이 자원을 사용하는 순서를 조정해 준다. 다양한 자원 공유 프로토콜이 연구되어 있는데 OSEK OS 는 우선순위 상한 프로토콜 (priority ceiling protocol)을 구현하여 일반적인 자원 공유 프로토콜에서 나타날 수 있는 데드락과 우선순위 역전 현상을 방지하고 있다.

OSEK OS 는 이벤트/자원 관리를 위해 다양한 API 를 제공한다. 이들의 이름과 그 의미는 다음 표와 같다.

Table 2 APIs for event/resource management

API 이름	의미
DeclareEvent()	이벤트의 존재를 선언한다.
SetEvent()	이벤트를 발생시킨다.
ClearEvent()	이벤트의 발생을 취소한다.
GetEvent()	이벤트의 발생 여부를 반환한다.
WaitEvent()	이벤트의 발생을 기다린다.
DeclareResource()	자원의 존재를 선언한다.
GetResource()	자원의 사용을 시작한다.
ReleaseResource()	자원의 사용을 마친다.

2.2 OSEK COM

OSEK COM[5]은 태스크들이 서로 통신할 수 있는 방법을 제공해 주는 모듈이다. OSEK COM 의 가장 큰 두 특징은 (1) 메시지를 전송 단위로 하는 publish/subscribe 통신 모델을 사용하며, (2) 다양한 통신 옵션을 제공한다는 점이다. 이제 이들 각각에 대해 자세히 설명한다.

2.2.1 통신 모델

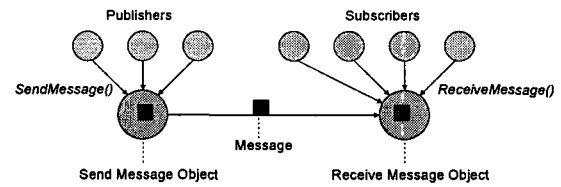


Fig. 5 Publish/subscribe communication model of OSEK COM

여기에서는 OSEK COM 이 제공하는 통신 모델에 대해 설명한다. 첫째, 이 모델은 데이터 전송

단위로 메시지를 사용한다. OSEK COM 에서 한 메시지는 메시지의 종류와 메시지의 내용으로 이루어진다. 사용자는 엔진 온도, 타이어 압력, 버튼의 눌림 여부 등 다양한 종류의 메시지를 정의할 수 있다. 메시지의 길이는 비트 단위로 세밀하게 명시할 수 있다.

둘째, 메시지는 publish/subscribe 방식으로 전송된다. Fig. 5에서 볼 수 있듯이, 이 방식을 사용하면 메시지를 보내고자 하는 태스크(publisher)는 통신 서브시스템에 메시지를 전달(SendMessage())하기만 하면 되고, 메시지를 받고자 하는 태스크(subscriber)는 통신 서브시스템으로부터 메시지를 구독(ReceiveMessage())하기만 된다. Publisher 가 특정 종류의 메시지를 OSEK COM 에게 전달하면, OSEK COM 은 그 종류의 메시지를 요청하는 모든 subscriber 들에게 메시지를 전송하는 역할을 한다.

이렇게 publisher 와 subscriber 가 서로 느슨하게 연결되어 있기 때문에, 태스크들은 서로 통신하기 위해 상대방의 존재를 알거나, 시스템의 전체적인 연결 상태를 파악할 필요가 없다는 장점이 있다. 또한 새로운 태스크를 추가하는 것이 매우 쉬우며, broadcast 를 전송 메커니즘으로 하는 자동차용 네트워크에 매우 효과적으로 구현이 가능하다.

2.2.2 통신 옵션

OSEK COM 은 기본적인 publish/subscribe 모델을 확장할 수 있도록 다양한 통신 옵션을 제공하고 있다. 그 중 몇 가지만 소개하면 다음과 같다. 첫째, 메시지를 주기적으로 전송하는 기능을 제공한다. 예를 들어, 별다른 프로그램 수정 없이 매 10ms 마다 타이어의 압력이 자동적으로 publish 되도록 할 수 있다. 둘째, 메시지 필터링을 제공한다. 이 기능은 메시지의 값이 특정한 조건을 만족할 때에만 실제로 subscriber 에게 전달되도록 하는 기능이다. 이 기능을 이용해서 엔진 온도가 특정 값 이상일 때에만 메시지가 전달되도록 할 수 있다. 셋째, 동기/비동기 메시지 전송을 모두 지원한다. 태스크는 메시지가 도착하기를 기다릴 수도 있고(동기), 메시지가 도착할 때 지정된 루틴이 수행되도록(비동기) 할 수도 있다.

2.3 OIL (OSEK Implementation Language)

OIL[6]은 OSEK OS, OSEK COM 등 다양한 OSEK 모듈들과 응용 프로그램의 속성을 설정하는 규약이다. 예를 들어 사용자는 OIL 을 작성하여, ECU 에 어떠한 OSEK 모듈들이 포함될 지, 어떤 태스크들이 수행되며, 어떤 이벤트, 자원, 메시지들이 사용될 지 등을 정할 수 있다. 또한 각 태스크의 우선순위나 메시지의 크기, 사용될 통신 옵션의 종류 등을 정할 수도 있다.

Fig. 6은 OSEK 응용 프로그램의 개발 과정을 보여준다. 이 그림에서 system generator 는 OIL 을 이용하여 특화된 OSEK OS 와 OSEK COM 을 생성하고 있다. 이렇게 생성된 OSEK 모듈들은 응용 프로그램과 함께 링킹되어 최종적으로 하나의 수행 파일을 구성한다.

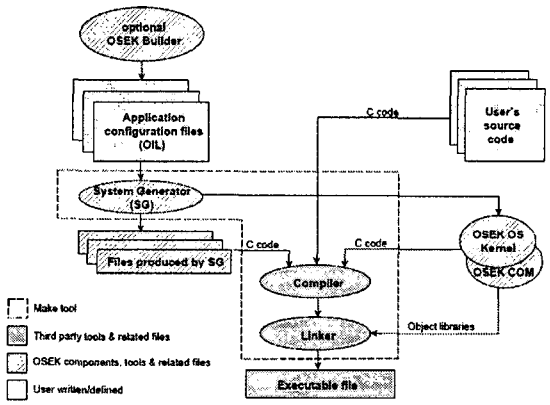


Fig. 6 Development process of OSEK applications

3. 차량용 미들웨어 – AUTOSAR

AUTOSAR(Automotive Open System Architecture)는 자동차 업체가 공통으로 사용할 수 있도록 설계된 차량용 소프트웨어 규격과 실행 환경이다. AUTOSAR 의 궁극적인 목적은 소프트웨어 모듈의 *plug & play* 이다. 즉, 하나의 차량용 소프트웨어 모듈(예를 들어 자동 주차 모듈)이 AUTOSAR 표준을 만족하는 모든 자동차에 변경 없이 실행되도록 하는 것이다. AUTOSAR 는 BMW, Bosch, Continental, DaimerChrysler 와 같은 유럽계 자동차 회사들을 주축으로 개발되었다.[7] 대표적인 AUTOSAR 제품으로는 3Soft 사의 tresosECU, LiveDevices 사의 RTA, Vector 사의 DaVinci 등이 있다.

3.1 개념

AUTOSAR 에서 자동차용 소프트웨어는 컴포넌

트를 단위로 모듈화된다. 이들 컴포넌트들은 설계 단계에서 VFB(Virtual Functional Bus)라는 가상의 네트워크를 통해 서로 통신하도록 구성된다. 그 후 매핑 단계에서는 각 컴포넌트들이 어떤 ECU 위에서 수행될 지 결정된다. 이렇게 ECU 에 할당된 컴포넌트들은 런타임에 각 ECU 마다 존재하는 RTE(Run-Time Environment)를 통해 서로 정보를 교환하며 필요한 작업을 수행한다.[8] 이 과정을 거치는 동안 각 소프트웨어 컴포넌트들과 자동차의 각 ECU 들의 속성과 제약조건들이 기록된 설정 파일들이 사용된다. Fig. 7이 이러한 과정을 보여주고 있다.

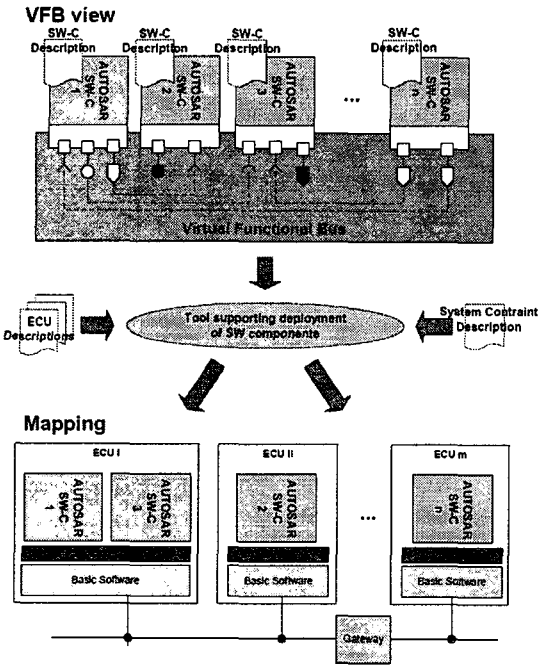


Fig. 7 Concept of AUTOSAR

3.2 아키텍처

Fig. 8에서 볼 수 있듯이 AUTOSAR 의 런타임 아키텍처는 크게 (1) RTE, (2) OS, (3) Communication, (4) ECU Abstraction, (5) Microcontroller Abstraction, (6) Services, (7) Complex Device Drivers 의 일곱 가지 모듈들로 나눌 수 있다. 여기에서는 이 중 핵심적인 RTE, OS/Communication, ECU/Microcontroller Abstraction 에 대해 설명한다.

3.2.1 RTE

RTE 는 AUTOSAR 의 필수적인 모듈로서, AUTOSAR 컴포넌트들이 서로 통신할 수 있도록 인터페이스를 제공한다. RTE 는 대상 컴포넌트가 어떤 ECU 에 있는지, ECU 사이의 통신이 어떤 프로토콜(CAN, LIN, FlexRay, 또는 TTP 등)에 의해 이루어지는지 알 필요가 없게 해 준다.

RTE 는 client/server 와 sender/receiver 의 두 가지 통신 모델을 지원한다. Client/server 모델은 client 컴포넌트들이 server 컴포넌트에게 미리 정의된 서비스를 요청하고 그 결과를 되돌려 받는 통신 모델을 의미한다. Sender/receiver 모델은 앞서 언급한 publish/subscribe 모델과 동일하다. 이 모델에서 sender 컴포넌트는 데이터를 배포하며, receiver 컴포넌트들은 이 데이터를 능동적으로 읽는다. 이 두 통신 모델은 각각 서비스 중심적인 통신과 데이터 중심적인 통신을 효과적으로 지원한다.

RTE 는 포트라는 객체를 통해 컴포넌트들 사이의 연결 상태를 관리한다. 각 컴포넌트는 두 가지 종류의 포트들을 가진다. 한 가지는 서비스를 요청받거나 데이터를 받기 위한 포트들이며, 다른 한 가지는 서비스를 요청하거나 데이터를 보내기 위한 포트들이다. 한 컴포넌트가 포트를 통해 서비스를 요청하거나 데이터를 쓰면, RTE 는 그 포트에 연결된 상대편 포트에게 이러한 사실을 전송한다. 그러면 상대편 포트는 자신이 속한 컴포넌트에게 서비스가 요청되었음을 알리거나 전송된 데이터를 제공한다. 이렇듯 통신할 컴포넌트에 대한 정보가 포트에 숨겨져 있기 때문에, 각 컴포넌트는 상대편 컴포넌트의 위치나 존재 여부 조차 알 필요가 없다는 장점이 있다.

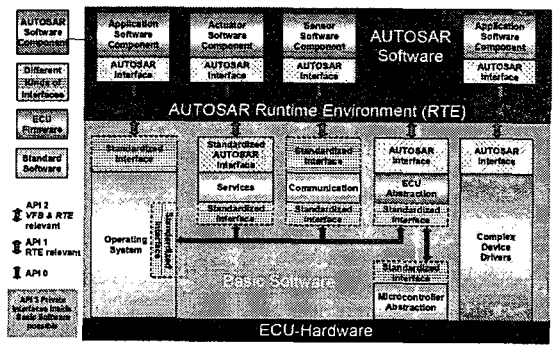


Fig. 8 Architecture of AUTOSAR

3.2.2 OS/Communication

AUTOSAR 는 OS 모듈과 Communication 모듈로 각각 OSEK OS 와 OSEK COM 을 사용한다. 이들 두 모듈은 응용 컴포넌트나 RTE 를 비롯한 다른 모듈들이 멀티 태스킹이나 태스크간 통신을 필요로 할 때에 사용될 수 있다. 예를 들어 RTE 는 내부적으로 OSEK OS 와 OSEK COM 을 이용하여 그 기능을 구현한다. 이 두 모듈은 그 기능이 필요 없는 일부 간단한 ECU 들에서는 포함되지 않아도 되는 선택적인 모듈이다.

3.2.3 ECU/Microcontroller Abstraction

AUTOSAR 는 한 자동차 안에 여러 개의 ECU 가 네트워크로 연결되어 있으며, 한 ECU 안에는 하나 혹은 여러 개의 Microcontroller 와 다수의 장치들(RAM, Flash, 네트워크, I/O)이 작동한다고 가정하고 있다. ECU abstraction 모듈은 해당 ECU 안에 장치들이 어떻게 구성되어 있는지(메모리 맵, I/O 맵, SPI 등)에 상관없이 그 장치에 접근할 수 있도록 해 준다. 반면 microcontroller abstraction 모듈은 실제로 특정 장치를 작동시키기 위한 장치 드라이버들을 구현하고 있는 모듈이다. 이 두 모듈을 이용하면, ECU 위치에 무관하게 응용 프로그램이 구현될 수 있다.

3.3 개발 과정

Fig. 9 는 AUTOSAR 를 이용하여 자동차용 소프트웨어를 개발하는 과정을 설명한다. 이는 크게 (1) 시스템 전체적인 명세로부터 각 ECU 별 명세를 매핑하는 과정과 (2) 이를 바탕으로 각 ECU 별로 소프트웨어 모듈들을 구현하거나 생성하여 하나의 실행 파일을 만드는 두 과정으로 나누어 볼 수 있다. 이제 이 두 과정 각각에 대해 설명한다.

3.3.1 매핑 과정

개발 과정의 가장 첫 단계는 소프트웨어 컴포넌트들의 인터페이스와 연결 상태 등을 기술하는 SW-Component Description, 각 ECU 별 하드웨어 특성을 기술하는 ECU Resource Description, 그리고 시스템의 전반적인 제약 사항을 기술하는 System Constraint Description 을 작성하는 단계이다. 이러한 사항들이 기술되고 나면 어떤 컴포넌트가 어떤 ECU 에서 수행되는 것이 가장 효율적이고 제약 조건을 잘 만족시켜 줄 수 있을지를 결정하는 매핑 단계를 거친다. 이 작업은 도구에 의해 자동화된 방식으로 이루어질 수도 있고, 개발자에 의해 수동으로 진행될 수도 있다. 이 단계를 거치면 각 ECU 별로 ECU Configuration Description 이 생성되는데 여기에는 RTE, OSEK, 응용 컴포넌트들이 어떻게 설정되는지 기술되어 있다.

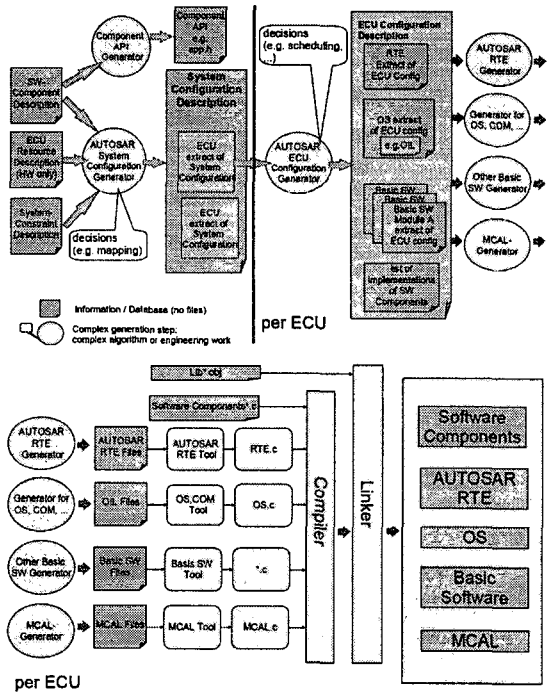


Fig. 9 Development process of AUTOSAR

3.3.2 구현 및 생성 과정

앞의 매핑 과정에서 생성된 각 ECU 별 설정 사항은 자동화된 생성 도구의 입력으로 전달되어 RTE, OSEK 등의 AUTOSAR 모듈들이 최적으로 생성될 수 있도록 한다. 이와는 별도로 응용 컴포넌트 개발자는 그 ECU 위에서 수행되어야 할 응용 컴포넌트를 구현하게 된다. 이렇게 구현된 응용 컴포넌트의 소스 코드들과 자동으로 생성된 AUTOSAR 모듈들이 함께 컴파일되고 링킹되어 하나의 실행 파일이 만들어진다. 마지막으로 이 실행 파일을 ECU 에 적재하여 모든 개발 과정이 완료된다.

4. 결 론

본 논문은 최근 자동차 산업계가 겪고 있는 급격한 변화를 극복하고 살아남기 위해서는 무엇보다도 소프트웨어의 복잡성을 해결하는 것이 중요하며, 이는 위해서는 운영체제나 미들웨어와 같은 체계화된 런타임 소프트웨어 플랫폼을 사용하여야만 하는 이유를 설명하였다. 이어서 그 대표적인 소프트웨어 플랫폼들로 OSEK 과 AUTOSAR 를 소개하였다. OSEK 은 실시간 멀티 태스킹과 태스크 간 통신 등을 지원하는 표준 운영 환경을 제공한다. AUTOSAR 는 OSEK 을 확장하여, 컴포넌트 단위로 소프트웨어를 설계하고 배포할 수 있도록 다양한 추상화 모듈을 제공하는 동시에 컴포넌트 간 표준화된 인터페이스를 정의하고 있다.

이러한 런타임 소프트웨어 플랫폼을 완벽히 활용하기 위해서는 아직 해결되어야 할 문제들이 많다. 그 중 중요한 것들을 언급하면 다음과 같다. 자동차용 소프트웨어의 전반적인 아키텍처를 설계하는데 사용할 방법론과 이를 지원하는 도구가 고안되어야 한다. 또한 프로그래머들이 소프트웨어 컴포넌트들을 손쉽게 개발할 수 있도록 해 주는 그래픽 기반의 개발 환경이 필요하다. 개발 비용을 줄이기 위해 실제 하드웨어가 없는 상황에서도 전체 시스템을 시뮬레이션 할 수 있는 테스트 환경이 제공되어야 한다. 자동차 업계 및 학계는 현재 이러한 문제들을 해결하기 위해 노력하고 있으며 전망은 밝다.

후 기

본 연구는 산업자원부에서 시행하는 자동차부품기반기술개발사업(사업명: 네트워크 기반 분산형 실시간 제어 시스템 설계 기반 기술 개발)으로 지원받아 수행한 연구결과이다. 연구비를 지원해주신 관계자 여러분께 감사를 표한다.

참고문헌

1. Hardung, Bernd, Kolzow, Thorsten and Kruger, Adnreas, "Reuse of Software in Distributed Embedded Automotive Systems," Proceedings of the 4th ACM International Conference on Embedded Software, ACM Press, pp. 203-210, 2004.
2. Klaus, Grimm, "Software Technology in an Automotive Company – Major Challenges," Proceedings of the 25th International Conference on Software Engineering, IEEE, pp. 498-503, 2003.
3. OSEK VDX Portal, http://www.osek-vdx.org.
4. OSEK, "OSEK/VDX Operating System Version 2.2.3," 2005.
5. OSEK, "OSEK/VDX Communication Version 3.0.3," 2004.
6. OSEK, "OSEK/VDX OIL Version 2.5," 2004.
7. AUTOSAR, http://www.autosar.org.
8. AUTOSAR, "AUTOSAR Technical Overview," 2006.