AUTOSAR 마이그레이션(Migration)을 위한 레거시 ECU 소프트웨어 시스템 분석 방법

Li Jincheng*, 류기열**, 이정태***,
*아주대학교 대학원 컴퓨터공학과
**아주대학교 정보컴퓨터공학과
***아주대학교 소프트웨어융합학과
e-mail : jcdby@ajou.ac.kr

Legacy ECU software system analysis method for AUTOSAR migration

Li Jincheng*, Ki-yeol Ryu** , Jungtae Lee***
*Dept. of Computer Engineering, Graduate school, Ajou University
**Dept. of Information and Computer Engineering, Ajou University
***Dept. of Software Convergence Technology, Ajou University

요 약

AUOTSAR(Automotive Open system Architecture)는 자동차 ECU(Electronic Control Unit)에 내장되는 소프트웨어에 대한 표준 구조로서, ECU 소프트웨어의 품질 향상은 물론, 개발 및 관리 비용의 절감에 기여하는 등 많은 장점을 갖는다. AUTOSAR 의 이런 장점 때문에 많은 자동차 회사들이 ECU 소프트웨어에 AUTOSAR 적용을 추진하고 있다. 이에 따라 기존 레거시 ECU 소프트웨어 시스템을 AUTOSAR 표준에 맞는 ECU 소프트웨어 시스템으로 변환하는 방법에 대한 관심도 높아지고 있다. 그 이유는 이미 많은 ECU 레거시 소프트웨어 시스템들이 개발되어 사용되고 있으며, 이들에게는 이미 기능 및 안정성 검증을 위하여 많은 시간과 비용이 투자하여 되어 있다. 따라서 ECU 소프트웨어 시스템에 AUTOSAR 를 적용하는 경우 기존의 레거시 소프트웨어 시스템을 재사용할 수 있으면 생산성 및 품질 면에서 많은 장점을 갖는다. 본 연구에서는 C 언어로 작성된 기존의 ECU 소프트웨어 시스템을 AUTOSAR 플랫폼에서 재사용할 수 있도록 하기 위하여, 기존의 레거시 ECU 소프트웨어 시스템을 AUTOSAR 플랫폼으로 마이그레이션하는 방법에 대하여 연구하였다. 마이그레이션 과정은 크게 두 단계로 나누어 지는데, 이는 레거시 소프트웨어 시스템을 분석하여 마이그레이션이 가능하도록 기능별로 분해하는 것과, 분해된 구성 요소들을 AUTOSAR 플랫폼에 맞는 구조로 재구성하는 과정이다. 본 논문에서는 이중 첫 번째 과정인 레거시 소프트웨어 시스템의 분석 및 기능별 분해 방법을 제시하고자 한다.

1. 서론

AUTOSAR (Automotive Open system Architecture) [1]는 표준화 된 자동차 ECU 소프트웨어 구조이다. 이는 세 부분으로 나누어져 있다. 그림 1 에서 나오는 것 같이, 각각은 바로 SWC, BSW 와 RTE 이다. SWC 는 소프트웨어 컴포넌트(software component)이고 BSW 는 basic 소프트웨어 컴포넌트이고 RTE 는 runtime environment 이다. SWC 는 ECU 소프트웨어 시스템의 Business logic 을 담당하고, BSW 는 ECU 의 하드웨어 및 통신 기능을 사용하는데 필요한 기본 기능을 제공하고, RTE 는 SWC 와 BSW 사이에서 양측에 표준 인터페이스를 제공함으로써 SWC 와 BSW 계층을 분리 시켜주며, SWC 컴포넌트들 간 가상 통신 채널의 역할을 수행한다. 이런 표준 구조를 통하여 AUTOSAR 는 컴포넌트의 재사용성 및 하드웨어에 대한 이식성을 을 향상시켜줄 뿐만 아니라, 시스템의 복잡성을 줄임으로써 기능 안정성 등의 품질 관리 측면에 있어서도 많은 장점을 제공한다.

그러므로 현재 전세계적으로 자동차 ECU 소프트웨어 시스템 연구 분야에서는 이미 생산된 자동차의 ECU 소프트웨어 시스템을 AUOTSAR 로 마이그레션하여 업그레이드 하거나, 새롭게 생산할 ECU 소프트웨어 시스템에 AUTOSAR 플랫폼을 적용하기 위한 많은 연구가 진행되고 있다.

본 연구에서는 이중 C 언어로 작성된 ECU 소프트웨어 시스템을 AUTOSAR 플랫폼으로 마이그레이션 하는 방법에 대하여 연구하였다.

기존의 레거시 시스템을 활용하지 않고 직접 AUTOSAR 플랫폼에서 새로이 소프트웨어를 개발할 경우, 이미 기존의 레거시 시스템이 만족하고 있는 요구 기능 및 안정성을

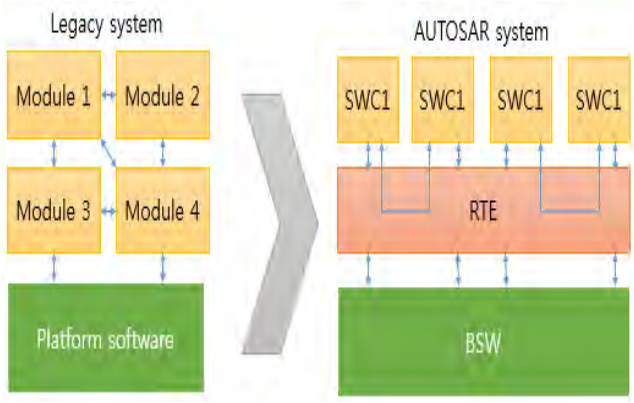
AUTOSAR 표준에 따라 다시 분석하고 설계해야 하며 이는 개발 비용 및 시간 측면에서 비효율적이다. 따라서 이미 개발된 ECU 소프트웨어 시스템을 적은 노력으로 AUTOSAR 플랫폼으로 변환할 수 있으면, 기존의 레거시 소프트웨어 시스템의 재사용이 가능하여 개발 비용 및 시간 측면에서 좀 더 효율적 일 수 있다.

최근에 기존의 레거시 소프트웨어 시스템으로부터 AUTOSAR 플랫폼으로 업그레이드 하는 방법에 대한 연구가 활발해지고 있다. 이중 기존의 레거시 소프트웨어 시스템을 AUTOSAR 플랫폼으로 마이그레이션 하는 방법을 개념적으로 제시한 연구가 있는데, 이는 (1)기존의 시스템을 기능별로(예, SWC 와 BSW) 분해하고, (2)그 분해된 것을 AUTOSAR 아키텍처에 맞게 다시 배치하고 (3)그 다음에 기존의 레거시 시스템에는 없으나, AUTOSAR 플랫폼으로 업그레이드된 시스템에서는 새롭게 필요한 요소들을 추가하는 과정으로 마이그레이션 과정을 제시한 연구가 있다[2]. 그러나 이 연구에서는 AUTOSAR 마이그레이션에 대한 구체적인 구현 방법을 제시하기 보다는, AUTOSAR 마이그레이션방법의 개념과 사례를 통한 구현 가능성만을 제시하였다.

본 연구에서는 [2] 에서 제시된 AUTOSAR 마이그레이션 개념을 토대로 이의 실제 구현 방법 및 이를 지원하는 도구의 개발에 대하여 연구하였으며, 본 논문에서는 이중 첫 번째 단계인 레거시 소프트웨어 시스템을 AUTOSAR 마이그레이션을 위하여 기능별로 분해하는 방법을 제시하고자 한다.

본 논문의 구조는 다음과 같다. 2 장에서는 기존의 레거시 소프트웨어 시스템을 AUOTSAR 플랫폼으로 마이그레이션 하는 전체 과정을 소개한다. 3 장에서는 AUOTSAR 플랫폼으로 마이그레이션을 위하여 기존의 레거시 시스템을 기능별로 분해하는 방법에 대하여 설명하였으며, 4 장에서는 결론으로 끝을 맺는다.

2. AUOTSAR 마이그레이션

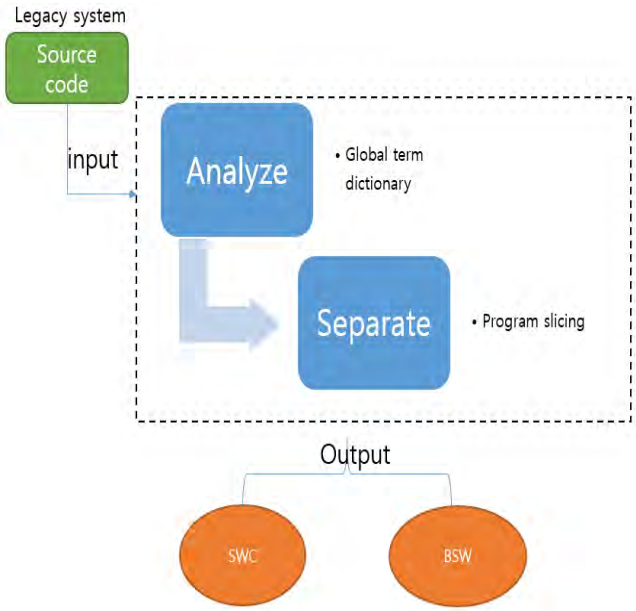


(그림 1) AUOTSAR 마이그레이션 개념

그림 1 은 기존의 레거시 소프트웨어 시스템이 AUOTSAR 플랫폼으로 마이그레이션하는 개념을 나타낸 것이다. 이 그림에서 레거시 소프트웨어 시스템은 모듈들과 플랫폼 소프트웨어로 구성되어 있다. 모듈은 함수들의 집합이고 플랫폼 소프트웨어는 real-time OS 와 하드웨어 접근 및 관리를 위한 공통 기능을 제공하는 library 등을 포함한 소프트웨어 실행 환경이다.

반면에 AUTOSAR 시스템은 전체 기능이 SWC 와 BSW 의 두 부분으로 분리되고, RTE 의 표준 인터페이스에 의하여 상호 연동되는 등 두 시스템은 구조에 큰 차이가 있다. 따라서 AUTOSAR 마이그레이션 과정은, 우선 기존의 레거시 소스 코드를 분석하여 기존 레가시 소스 코드 상의 모듈들을 SWC 에 해당하는 기능을 갖는 부분과 BSW 에 해당하는 기능을 갖는 부분으로 분리한 후, 이들을 다시 AUTOSAR 표준에 맞는 형태로 재구성하는 과정으로 진행되어야 한다.

3. 레거시 소프트웨어 시스템 분석 및 분해



(그림 2) 레거시 소프트웨어 시스템 분석 및 분해 과정

그림 2 은 기존의 레거시 소프트웨어 시스템을 분석 및 분해하는 과정을 나타낸 것이다.

AUTOSAR 시스템이 SWC, BSW, RTE 의 세 부분으로 구성되지만 RTE 는 SCW 와 BSW 를 분리 시키기 위한 표준 인터페이스 제공 및 가상 통신 기능 만을 제공한다. 따라서 레가시 소스 코드의 모듈은 단지 BSW 와 SWC 의 두 기능으로 분리되어야 하며, 레가시 소스 코드에서 BSW 기능과 관련된 부분을 찾아내면 나머지 부분은 모두 SWC 기능으로 간주 할 수 있다.

이 과정은 두 단계로 나누어 지는데 분석 단계와 분해 단계이다. 분석 단계는 Global term dictionary 를 구성하고 Global term dictionary 의 정보를 사용하여 레거시 소프트웨어 소스 코드를 검색하여 BSW 관련 기능을 수행하는 명령문들을 찾는 단계고, 분해 단계는 앞 단계에 찾아낸 BSW 관련 명령문으로부터 program slicing 을 이용해서 기존의 레거시 소프트웨어 소스 코드에서 이 BSW 기능 명령문과 연관된 명령문들을 찾아 이를 BSW 기능을 수행하는

부분으로 분리해 내는 단계이다.

3.1 분석 단계

이 단계에서는 Global term dictionary 를 통해서 기존의 레거시 시스템 소스 코드를 분석하고 BSW 관련 기능을 수행하는 명령문들을 찾는 방법을 제시하고자 한다.

```
/* Timer Registers */
#define TIMER_0_MATCH_REG          (*((uint32_t volatile *)0x40A00000))
#define TIMER_1_MATCH_REG          (*((uint32_t volatile *)0x40A00004))
#define TIMER_2_MATCH_REG          (*((uint32_t volatile *)0x40A00008))
#define TIMER_3_MATCH_REG          (*((uint32_t volatile *)0x40A0000C))
#define TIMER_COUNT_REG            (*((uint32_t volatile *)0x40A00010))
#define TIMER_STATUS_REG           (*((uint32_t volatile *)0x40A00014))
#define TIMER_INT_ENABLE_REG       (*((uint32_t volatile *)0x40A0001C))
```

(그림 3) 헤더 파일에서 매크로 정의

프로그램이 특정 하드웨어를 제어하려면 해당 하드웨어에 접근하기 위한 주소를 먼저 알아야 한다. 프로그램은 해당 주소에 하드웨어 제어를 위한 데이터를 읽거나, 씀으로써 해당 하드웨어의 동작을 제어하기 때문이다[7].

일반적으로 c 프로그램에서는 소스 코드에 해당 하드웨어에 접근하기 위한 주소를 직접 쓰지 않고 헤더 파일에 해당 주소 값을 의미하는 identifier 를 매크로로 정의하여 사용한다. 그림 3 은 이와 같은 매크로 정의 예이다. 따라서 이러한 매크로 identifier 는 레거시 소스 코드에서 BSW 와 관련 기능을 찾기 위한 키워드로서 사용될 수 있다. Global term dictionary 는 이런 키워드 정보를 제공한다. 이러한 Global term dictionary 는 레거시 소프트웨어 시스템 개발자에 의하여 만들어 지며 레거시 소스 코드 분석을 위한 기본 자료로 사용된다.

```
1  #include<mega883.h>
2  void main(void)
3  {
4      unsigned char injector = 0b11111111;//injector의 초기 설정값, 0xFF
5      DDRC = 0xFF;//*PORTC모두 출력으로 설정
6      DDRB = 0x00;//*PORTB모두 입력으로 설정
7
8      while(1)
9      {
10         injector = PINB.0 & 0b00000001;//스위치가 ON되면 PB0로 '0'입력
11         if(injector == 0b00000000)
12             PORTC = 0b11111110;//1번 인젝터 분사
13         else
14             PORTC = 0b11111111;//injector 분사 멈춤,0xFF
15     }
16 }
```

레지스터	기능
DDRB	포트 B의 데이터 방향을 결정(비트별 값이 '1':출력, 비트별 값이'0':입력)
DDRC	포트 C의 데이터 방향을 결정(비트별 값이 '1':출력, 비트별 값이'0':입력)
PORTC	DDRC에 의해서 출력으로 설정
PINB	DDRB에 의해서 입력으로 설정

(그림 4) I/O 포트와 관련된 레지스터

그림 4 는 ECU 소스 코드 및 Global term dictionary 의 예이다. 그림 4 의 Global term dictionary 는 ECU 소스 코드의 매크로 identifier 네 개를 가지고 있다. DDRC, DDRB, PORTC, PINB 로서 각 매크로 identifier 는 C port, B port 에 동작을 제어하기 위한 매크로 identifier 이다. 예를 들면 DDRC 는 C port 의 데이터 방향을 결정하는 레지스터의 주소 값 이며, DDRC = 0xFF 로 설정하면 PORTC 의 모든 핀은 출력으로 설정된다.

이 분석 단계에서는 Global term dictionary 에서 나오는 매크로 identifier 를 이용해서 소스 코드를 검색하여 BSW 관련 기능을 수행하는 명령문들을 찾을 수 있다.

그림 5 에서는 이와 같은 방법으로 찾아진 BSW 관련 기능을 수행하는 명령문들을 예를 보여 주고 있다.

하지만 이러한 과정만으로는 BSW 관련 기능을 수행하는 명령문들을 모두 찾을 수 없는 데 그 이유는 다음과 같다.

그림 5 의 line 10 에 있는 명령문 때문에, line 11 의 injector 가 line 10 에 있는 PINB 의 영향을 받는다. 그래서 line 11 의 injector 도 BSW 관련 기능을 수행하는 명령문으로 간주하게 된다.

이와 같은 BSW 과련 기능 명령문들을 찾기 위해서는 program slicing 를 이용해야 한다. 구체적인 방법은 다음 단계에서 설명하기로 한다.

```
1  #include<mega883.h>
2  void main(void)
3  {
4      unsigned char injector = 0b11111111;//injector의 초기 설정값, 0xFF
5      DDRC = 0xFF;//*PORTC모두 출력으로 설정
6      DDRB = 0x00;//*PORTB모두 입력으로 설정
7
8      while(1)
9      {
10         injector = PINB.0 & 0b00000001;//스위치가 ON되면 PB0로 '0'입력
11         if(injector == 0b00000000)
12             PORTC = 0b11111110;//1번 인젝터 분사
13         else
14             PORTC = 0b11111111;//injector 분사 멈춤,0xFF
15     }
16 }
```

(그림 5) BSW 정보 위치

3.2 추출 단계

Program slicing 는 Mark Weiser 가 1981 년에 최초로 소개한 개념[8]이다. 이는 특정 기준점(criteria)에 의해 프로그램에서 연관 명령문들을 추출하는 작업을 의미한다.

Program slicing 기술 중 하나인 forward slicing [4]은 함수 내에 특정한 identifier 의 값이 변하면 이 변화의 영향을 받는 명령문들을 찾는 것이다. 예를 들면, 그림 6 의 예제 프로그램에서 line 3 에 있는 identifier 인 'lines' 의 영향을 받는 명령문들은 그림 7 과 같이 나온다.

그림 7 은 예제 프로그램의 slice 를 나타낸 것이다. 이는 주어진 기준점인 slicing criterion<s, v>(이하 C<s, v>)

에 의해서 만들어진 것이다. C<s, v>에서 ‘s’ 는 프로그램의 명령문이고 ‘v’ 는 variable 이나 매크로이다. 따라서 레거시 ECU 프로그램에서 C<명령문, BSW 명령문 키워드>로 forward slicing 기술을 이용하면 이 BSW 명령문의 영향을 받는 부분을 찾아낼 수 있다. 현재 forward slicing 을 지원하는 많은 툴들이 이미 개발되어 있어 이들을 활용하면 위 방법은 쉽게 구현가능하다.[5][6]

위 과정들을 통해 기존의 레거시 소프트웨어 시스템 소스 코드에서 BSW 관련 기능을 수행하는 명령문들을 분리해 낼 수 있다. 소스 코드의 나머지 부분들은 앞에서 설명한 바와 같이 SWC 관련 기능을 수행하는 명령문으로 간주하면 된다.

```
1   read(text);
2   read(n);
3   lines = 1 ;
4   chars = 1;
5   subtext = "";
6   c = getChar(text);
7   while (c != '\eof')
8               if(c == 'n')
9               then lines = lines + 1;
10                      chars = chars + 1;
11              else chars = chars +  1;
12                      if (n != 0)
13                      then subtext = subtext ++ c;
14                              n = n - 1;
15              c = getChar(text);
16  write(lines);
17  write(chars);
18  write(subtext);
```

(그림 6) Example program

```
3)   Lines = 1;
9)       then lines = lines + 1;
16)  Write(lines);
```

(그림 7) <3, {lines}>에 의한 그림 4 의 forward static slice

위와 같은 방법에 의하여 SWC 기능과 BSW 기능으로 분석, 분리된 정보는 마이그레이션을 위한 다음 단계에서 사용 가능한 데이터 구조로 표현되어야 한다. 데이터 구조에서는 각 모듈의 명령문을 BSW set 와 SWC set 로 나누어 저장하는데, 이때 BSW set 는 이 set 를 만들기 위하여 사용된 키워드 정보가 함께 표시된다. 그리고 해당 set 에는 각 set 에 포함되는 명령문들의 식별할 수 있는 명령문의 번호들이 기억된다. <표 1> 은 이러한 데이터 구조를 테이블 형식으로 요약하여 표현한 것 이다.

<표 1> 분석정보 데이터 구조

모듈 이름	함수 이름	BSW set		SWC set
		키워드	명령문 라인 번호	
그림4 예제프로그램	main	DDRC	5	나머지 라인 번호
		DDRB	6	
		PINB	10,11	
		PORTC	12,14	
...	...	...	...	...

표 1 을 그림 4 의 프로그램을 예로 설명하면 다음과 같다. 우선 모듈과 함수의 이름 칸에는 이들에 해당하는 이름을 기록한다. BSW set 는 키워드와 명령문 라인 번호의 두 부분으로 나누어 지는데, 키워드 부분에는 DDRC, DDRB, PINB, PORTC 를 기록하고, 명령문 라인 번호 부분에는 각 키워드에 해당하는 라인 번호를 기록한다. SWC set 에는 나머지 명령문 라인 번호를 기록한다.

4. 결론

본 연구에서는 기존의 레거시 ECU 소프트웨어 시스템을 AUTOSAR 플랫폼으로 마이그레이션 하기 위해서 필요한 레거시 소프트웨어 시스템을 분석하고, 분석 결과를 토대로 추후 AUTOSAR 마이그레이션 시 SWC 와 BSW 로 할당되어야 할 기능별로 분해하는 방법에 대해서 제시하였다. 본 논문에서는(1)Global term dictionary 를 통해서 기존의 레거시 소스 코드에서 BSW 관련 키워드를 찾아내고, (2)그 찾아낸 BSW 관련 키워드로부터 program slicing 을 이용해서 기존의 레거시 소스 코드에서 그 BSW 관련 키워드와 관련된 부분을 찾아내는 방법을 제시하였다. 또한 이렇게 찾아진 정보들을 마이그레이션 다음 단계에서 활용하기 위한 데이터 구조로 표현하는 방법도 제시하였다.

참고문헌

[1] Automotive Open system Architecture: www.autosar.org

[2] Daehyun Kum, Gwang-Min Park, SeongHun Lee and Wooyoung Jung,”AUTOSAR migration from Existing Automotive software”, International Conference on Control, 2012

[3] Joche Quante, Mohammed Tarabain, Janet Siegmund, “Towards recovering and Exploiting Domain Knowledge from C code: A Case study on Automotive software”, IEEE, 2014

[4] JOSEP SILVA, “A Vocabulary of Program Slicing-Based Techniques”, ACM Computing Surveys, 2012

[5] http://research.cs.wisc.edu/wpis/slicing_tool/

[6] http://frama-c.com/index.html

[7]Michael Barr, Anthony Massa, “programming embedded systems, second edition with C and GNU development tools”, O`Reilly Media, Oct, 2006

[8] Mark Weiser. "Program slicing". Proceedings of the 5th International Conference on Software Engineering, pages 439–449, IEEE Computer Society Press, March 1981.