ISSN 1598-0170 (Print)
ISSN 2287-1136 (Online)
http://www.jksii.or.kr

비행 소프트웨어의 이식성 향상을 위한 IMA 기반 플랫폼 아키텍처

IMA based Platform Architecture for Improving Portability of Flight Software

서 용 진[1] 김 현 수[1*]
Yongjin Seo Hyeon Soo Kim

요 약

비행 소프트웨어는 인공위성의 탑재 컴퓨터에서 사용되는 소프트웨어로, 실시간성과 고신뢰성이 요구된다. 이와 같은 요구사항으로 인해 비행 소프트웨어는 동작 환경에 대한 종속성을 갖게 된다. 이러한 문제는 새로운 시스템을 구축할 때마다 매번 다시 개발하여야 하는 상황을 초래한다. 따라서 비행 소프트웨어와 동작 환경 사이의 종속성 문제를 해결할 필요가 있으며, 이는 비행 소프트웨어의 이식성 향상을 통해 해결할 수 있다. 이식성을 향상시키기 위해서는 이식 비용의 최소화가 요구된다. 따라서 본 논문에서는 이식성 달성 방안으로부터 이식성을 분류한다. 또한 분류된 이식성의 종류를 이용하여 다양한 이식성을 제공하는 IMA 아키텍처 기반의 플랫폼 아키텍처를 제안한다. 본 논문에서 제안한 아키텍처를 통해서 비행 소프트웨어의 개발 문제를 해결할 수 있다.

☞ Keywords : 비행 소프트웨어, 소프트웨어 아키텍처, 이식성, IMA 아키텍처

ABSTRACT

Flight software operated on the on-board computers in the satellite has requirements such as real-time, high reliability. These requirements make dependency between the flight software and operating environments. Further, whenever a new system is built, such problem drives that all flight software are redeveloped. Thus, the dependency problem between them should be solved. And the problem can be resolved by improving the portability of the flight software. In order to improve the portability it is required to minimize the porting cost. In this paper, we classify the portability with the portability achieving methods. Using the classified portability, we propose a platform architecture that is based on the IMA concept and provides various portability capabilities. The proposed architecture enables us to solve the problem about the development of the flight software.

☞ Keywords : Flight Software, Software Architecture, Portability, IMA Architecture

1. 서 론

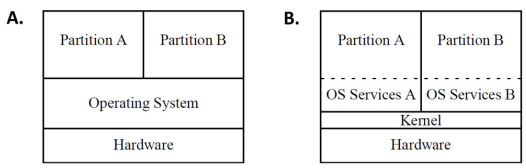
비행 소프트웨어(Flight Software)는 인공위성의 탑재 컴퓨터에서 사용되는 소프트웨어를 말한다. 비행 소프트웨어는 실시간성과 고신뢰성이 요구되며, 탑재 컴퓨터라는 제한된 자원을 기반으로 동작하여야 한다[1]. 이를 충족하기 위해서 비행 소프트웨어는 고신뢰성을 보장하는 실시간 운영체제를 기반으로 개발된다. 결국, 비행 소프트웨어는 특정 실시간 운영체제에 종속적으로 구현되는 것이다. 사실 한번 제작된 인공위성은 유지보수가 어렵기 때문에, 비행 소프트웨어가 특정 동작 환경에 종속성을 갖는 것은 큰 문제가 되지 않는다. 다만, 이와 같은 상황이 이후에 제작될 인공위성에 영향을 준다는 것이 문제가 된다. 새로 제작된 시스템의 동작 환경이 이전 시스템과 다르다면 이전에 개발된 비행 소프트웨어는 전혀 사용할 수 없다. 인공위성의 목적과 용도가 다를지라도 동일한 도메인의 시스템이기 때문에 이전 시스템에서 사용하였던 비행 소프트웨어를 다시 사용할 필요가 있다. 그러나 비행 소프트웨어가 갖는 종속성은 이 가능성을 약화시킨다. 또한 비행 소프트웨어는 고신뢰성이 요구되는 소프트웨어이기 때문에, 일반 소프트웨어와 달리 검증에

[1] Dept. of Computer Science and Engineering, Chungnam National University, Daejeon, 305-764, Korea
* Corresponding author (hskim401@cnu.ac.kr)
[Received 18 October 2013, Reviewed 29 October 2013, Accepted 19 November 2013]
☆ 본 연구는 한국연구재단을 통해 교육과학기술부의 우주기초원천기술개발 사업(NSL, National Space Lab)으로부터 지원받아 수행되었습니다 (2011-0020905).

소요되는 시간이 길다. 즉, 비행 소프트웨어를 새로 개발하는 것은 높은 개발 비용과 시간을 요구한다. 따라서 이와 같은 문제를 해결하기 위해서 비행 소프트웨어는 실시간성과 고신뢰성 외에도 동작 환경으로부터의 독립성 역시 요구되어야 한다.

IMA(Integrated Modular Avionics) 아키텍처는 본래 항공 분야에서 제안된 아키텍처로, 모듈화 시스템(Modular System)의 집합으로 구성된다[2,3]. IMA 아키텍처에서는 모듈화 시스템을 파티션(Partition)이라 부르며, 비행 소프트웨어는 파티션 내부에 포함된다. 파티션들은 IMA 플랫폼을 통해 관리되며, IMA 플랫폼은 파티션에게 공유 자원을 제공한다. 여기서 제공되는 공유 자원은 크게 시간적 자원(예: CPU 등)과 공간적 자원(예: 메모리, 통신채널 등)으로 나눌 수 있다. IMA 아키텍처를 통해 얻을 수 있는 이점은 다음과 같다. IMA 플랫폼을 통해서 파티션과 하드웨어를 분리할 수 있어 (1)비행 소프트웨어와 하드웨어와의 의존성을 낮출 수 있다. 또한 파티션은 시공간적으로 완벽하게 분리되어 구성되기 때문에 (2)파티션 혹은 비행 소프트웨어에서 발생된 오류가 시스템으로 전파되는 것을 막을 수 있다. 파티션 사이의 분리는 비행 소프트웨어의 병렬적인 개발을 가능하도록 하여 (3)비행 소프트웨어를 개발하는데 소요되는 시간을 줄여준다. 마지막으로 공간적 분리를 통해 (4)비행 소프트웨어의 실시간성을 보장할 수 있다. 따라서 IMA 아키텍처를 적용한다면 비행 소프트웨어가 갖는 요구사항을 모두 충족시킬 수 있다.

하지만 IMA 아키텍처는 개념에 불과하기 때문에, IMA 아키텍처의 이점을 활용하기 위해서는 이를 실현하여야 한다. IMA 아키텍처를 실현하는 방안으로는 두 가지가 존재한다[4]. Fig. 1은 실현 방안을 도식화한 것이다.



(Fig. 1) IMA Implementation Method

Fig. 1A는 마치 일반적인 운영체제와 같은 형태를 갖는다. 모든 파티션은 동일한 운영체제(Operating System, 이후 OS) 서비스를 제공받으며, 마치 프로세스처럼 동작한다. Fig. 1A 형태를 갖는 예로는 ARINC 653 표준[5]과 AIR[6]가 있다. ARINC 653 표준은 Fig. 1A 형태를 바탕으로 정의되었으며, AIR는 ARINC 653 표준을 준수한다. Fig. 1B는 Fig. 1A와 달리 각 파티션마다 독자적인 OS 서비스를 제공받는다. [4]에서는 이와 같은 특성을 통해 Fig. 1B는 가상화 시스템과 같다고 언급한다.

본 논문에서는 비행 소프트웨어가 동작 환경으로부터 독립성을 가질 수 있도록 하기 위해 IMA 아키텍처 기반의 우주용 플랫폼을 제안한다. 다시 말해, 본 논문에서는 IMA 아키텍처 기반의 플랫폼을 제공함으로써 비행 소프트웨어의 이식성을 달성하는 것을 목표로 한다. 문제는 어떠한 실현 방안을 선택할 것인지에 달려있다. Fig. 1의 두 가지 실현 방안은 모두 비행 소프트웨어가 이식성 품질을 만족할 수 있도록 도와준다. 다만, 두 실현 방안의 이식성 수준에서 차이가 있다. 결국 두 실현 방안의 이식성의 차이는 플랫폼에서 사용 가능한 소프트웨어의 유형을 결정한다. 특정 소프트웨어의 유형만을 지원하여서는 높은 수준의 이식성을 제공할 수 없기 때문에, 본 논문에서는 Fig. 1A와 Fig. 1B의 이식성을 모두 제공할 수 있는 혼합형 IMA 아키텍처 기반의 플랫폼을 제안한다. 따라서 본 논문에서 제안하는 플랫폼은 IMA 아키텍처의 이점을 모두 제공할 뿐만 아니라 다양한 유형의 비행 소프트웨어에 대해 이식성을 제공할 수 있다.

2. 이식성

IMA 아키텍처의 확장에 앞서 이식성의 정의와 달성 방안에 대해서 살펴본다. 달성 방안에 따라 이식성의 특성과 효과가 달라짐을 확인하기 위해서이다. 마지막으로 각 달성 방안의 특성을 바탕으로 이식성을 분류한다.

2.1 정의

이식성이란 소프트웨어가 자신이 목적으로 한 환경 외에도 다른 환경에서 동작할 수 있는 능력을 말하며, [7]에서는 이식성을 다음과 같이 정의한다.

"컴파일러 C로 컴파일된 소프트웨어 S가 운영체제 O와 하드웨어 H로 구성된 원본 컴퓨터(Source Computer)에서 동작한다. 소프트웨어 S와 기능상 동일하지만 컴파일러 C'으로 컴파일된 소프트웨어 S'이 운영체제 O'와 하드웨어 H'로 구성된 대상 컴퓨터(Target Computer)에서 동작한다. 이때, 소프트웨어 S에서 소프트웨어 S'으로 변환하는데 소요되는 비용이 소프트웨어 S'을 새로 개발하는데 소요되는 비용보다 작으면 소프트웨어 S

는 이식이 가능(Portable)하다고 한다."

[7]의 정의를 통해 두 가지 사실을 알 수 있다. 첫 번째는 이식성은 컴파일러, 운영체제, 하드웨어에 영향을 받는다는 것이다. 위 정의를 보면 한 소프트웨어는 컴파일러에 의해 구성되고 운영체제와 하드웨어로 구성된 환경에서 동작한다. 그러나 모든 경우에 이식이 가능한 것은 아니다. [8]에서는 호환의 문제로 이식이 불가능한 상황이 발생할 수 있음을 언급하며, 컴파일러-비호환성(Compiler Incompatibility), 운영체제-비호환성, 하드웨어-비호환성을 이야기한다. 따라서 이식성은 컴파일러, 운영체제, 하드웨어의 영향을 받는다. 두 번째는 이식성의 측정에 대한 것으로, 이식성은 이식의 비용이 재개발의 비용보다 적을 때 의미 있다는 것이다. [17]에서는 이를 아래와 같이 표현한다.

DP = 1 - (cost of porting / cost of redevelopment)

위 표현식에서 재개발 비용은 한 소프트웨어를 개발하는 비용과 같다. 따라서 DP(Degree of Portability)는 이식에 소요되는 비용이 적으면 적을수록 1에 가까운 값을 갖는다. 여기서 DP가 1의 값을 갖는다는 것은 완벽한 이식성을 의미한다. 그렇기 때문에 이식에 소요되는 비용을 최소화할 수 있는 방법을 찾는 것이 이식성을 향상시키는 방법이 된다.

2.2 달성 방안

이식성을 달성하는 방법은 크게 네 가지로 구분되며, 각각은 (1)구현-종속적인 부분의 분리, (2)추상 인터페이스, (3)MDA(Model Driven Architecture) 기법, (4)가상화 기술이다. 이번 절에서는 각 달성 방안을 통해 이식성이 어떻게 달성되는지 소개하고, 그로부터 이식성의 특징과 효과를 파악한다.

1) 구현-종속적인 부분의 분리

구현-종속적인 부분이란 동작 환경에 따라 변경될 수 있는 부분을 의미한다. 이식성에 비추어 보면 이식하기 위해 변환되어야 하는 부분, 즉 호환에 영향을 주는 부분을 의미한다. 이식을 위한 변환 과정이 필수적으로 발생하여야 하므로 이와 같은 상황에서 이식 비용을 최소화할 수 있는 방법은 추후에 있을 환경 변화에 대해 변경되어야 하는 부분과 그렇지 않은 부분을 분리하는 것이다. 만일 분리되지 않은 상태로 소프트웨어를 완성한다면, 변경되어야 하는 부분과 그렇지 않은 부분의 구분이 어렵기 때문에 본래 수정하여야 하는 범위보다 더 많은 범위를 수정해야 하는 상황이 발생한다. 따라서 구현-종속적인 부분을 분리하는 것은 이식성을 달성하는데 도움을 준다. 이 방법은 주로 응용 수준보다는 시스템 수준에서 많이 사용된다. 운영체제의 장치 구동기(Device Driver)에 대한 구현이 이 방법의 한 예로 볼 수 있다.

2) 추상 인터페이스

추상 인터페이스는 어떠한 동작 환경에서든 동일하게 사용되는 기능을 정의한 것을 의미하며, 공통 인터페이스라고도 부른다. 가장 간단한 예로는 POSIX(Potable Operating System Interface)가 있다. POSIX는 유닉스 계열의 운영체제를 위해 표준화된 추상 인터페이스이다. 만일 어떤 프로그램이 POSIX를 기반으로 구현된다면, 어떤 유닉스 기반의 운영체제에서도 동작할 수 있다. 즉, 해당 프로그램은 POSIX라고 하는 추상 인터페이스를 통해 이식성을 달성한다. 이처럼 추상 인터페이스는 특정 도메인에서 주로 사용되는 기능을 표준화된 API로 정의하기 때문에, 추상 인터페이스를 기반으로 구현된 소프트웨어는 해당 도메인 내에서 이식성을 달성한다.

추상 인터페이스를 이용한 이식성은 주로 표준화가 수반된다. 도메인 내에서 동일한 API를 사용해야 하기 때문에 표준화 작업이 존재하지 않는다면 적용되기 어렵다. 대신 표준화가 진행되고 도메인 내에 충분히 적용된다면 어떤 방법보다 높은 수준의 이식성을 제공할 수 있다.

3) MDA 기법

MDA(Model Driven Architecture)는 OMG(Object Management Group)에서 제시한 개발 패러다임으로, 정형화된 모델을 통해 구현물을 얻는 방법을 의미한다[9]. MDA에서 사용하는 모델에는 두 가지가 있다. 하나는 플랫폼 독립적 모델(Platform Independent Model, PIM)이며, 다른 하나는 플랫폼 종속적 모델(Platform Specific Model, PSM)이다. PIM은 도메인 내에서 사용되는 기술 플랫폼의 공통적 요소를 표현하기 위한 모델로, 이를 통해 시스템의 설계와 명세를 기술한다. PSM는 특정 기술 플랫폼의 요소를 표현하기 위한 모델로, 보통 하나 이상의 PSM

을 이용한다. PIM과 PSM 사이에는 변환 규칙이 존재하여 PIM을 통해 기술된 내용은 PSM으로 변환된다. 또한 PSM으로 표현된 내용 역시 실제 구현물로 변환되며, 이를 통해 구현물을 얻는다. 이 방법은 설계 단계에서 추상 계층을 제공함으로써, 단 한 번의 설계로 다양한 환경에 대한 구현물을 생성할 수 있도록 도와준다. 또한 새로운 동작 환경이 추가되더라도 기존 설계에 대한 수정 없이 이를 반영할 수 있다.

이 방법은 다른 방법과는 달리 소프트웨어의 실체가 존재하지 않는다. 다만, 정형화된 모델을 통해 소프트웨어의 설계와 명세만이 존재할 뿐이다. 또한 다른 동작 환경으로 이식되어야 할 때는 해당 동작 환경에 대한 PSM을 통해서 구현물이 생성되기 때문에 사실 상 이식을 위해 소요되는 비용이 거의 없다고 볼 수 있다. 문제는 정형화된 모델을 통해 소프트웨어를 명세하는 것이 어렵다는 것이다. 또한 명세와 설계로부터 생성된 구현물이 서로 동치인지 검증하는 문제도 역시 존재한다.

4) 가상화 기술

가상화는 특정 시스템이나 프로세스에게 동형이질성(Isomorphism)을 제공하는 것을 말한다[10]. 동형이질성이란 본래 동작 환경과는 다른 구성의 시스템이지만 실제 동작은 본래 동작 환경과 동일하도록 하는 것을 의미한다. 이때, 동형이질성을 제공받는 시스템을 게스트(Guest)라 하고, 동형이질성을 제공하는 시스템을 호스트(Host)라 한다. 본래 가상화는 이미 개발된 프로그램을 다른 환경에서 동작시키기 위해 등장하였으나, 현재는 그 외에도 시스템 간의 분리를 통해 효율적 서버 운영이나 보안 강화의 목적으로 사용되고 있다[11][12]. 가상화는 다양한 목적을 갖는 만큼 다양한 종류의 가상화 기술이 존재한다. Fig. 2는 가상화의 분류를 보여준다.

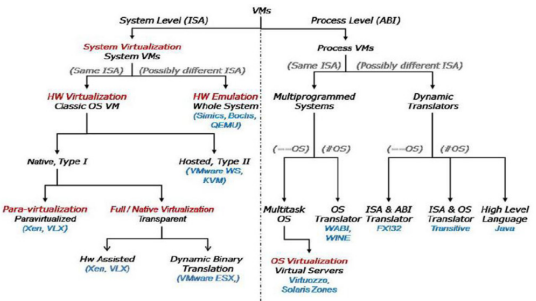


Fig. 2 Taxonomy of Virtualization〔10〕

기본적으로 가상화는 기존에 존재하는 게스트 시스템 혹은 게스트 프로세스를 호스트 시스템에서 동작시키는 것이 목적이다. 즉, 기존 소프트웨어(Legacy Software)를 재사용하는 것이 목적인 것이다. 기존 소프트웨어를 재사용하는 방법에는 두 가지가 있다. 하나는 변환(Conversion)하는 것이며, 나머지 하나는 래퍼(Wrapper)를 제공하는 것이다[8]. 가상화 기술은 기본적으로 래퍼를 제공하는 방식이며, 일부는 래퍼 내부에서 변환 작업을 수행한다. 래퍼를 제공하는 방식에는 시스템 가상화의 반가상화 기법이 존재하며, 변환 작업을 수반하는 방식에는 시스템 가상화의 전가상화 기법, 프로세스 가상화가 존재한다. 가상화에서 수행하는 변환 작업에는 게스트 환경에서 동작할 수 있는 이진코드를 호스트 환경에서 동작할 수 있는 이진코드로 변환하는 작업이 있다. 이와 같은 방법을 통해 소프트웨어 전체가 재사용될 수 있으며, 이는 곧 이식성의 달성을 의미한다. 중요한 것은 가상화 기술의 이식성을 이용하면 소프트웨어의 원본 코드가 존재하지 않더라도 이식이 가능하다는 것이다.

2.3 이식성의 분류

이번 절에서는 앞서 살펴본 이식성의 정의와 달성 방안을 바탕으로 이식성을 분류한다. 이식성을 분류하는 것은 본 논문에서 제안하는 이식성 향상 방안에 대한 타당성을 부여하기 위함이다. 본 논문에서는 이식성 향상을 위해 다양한 이식성을 제공하는 것을 목적으로 한다. 여기서 다양한 이식성이 존재함을 보이기 위해서는 이식성이 특정 기준을 통해 분류될 수 있음을 보여야 한다. 따라서 본 논문에서는 두 가지 기준을 통해 이식성을 분류한다. 기준은 이식 소프트웨어에 대한 (1)이식 단위(Porting Unit), (2)추상 단계(Abstraction Level)가 된다.

1) 이식 단위에 따른 분류

가상화 기술을 살펴보면, 시스템 가상화와 프로세스 가상화가 존재한다. 두 가상화의 차이는 간단히 말해서 운영체제를 포함하느냐 포함하지 않느냐로 구분할 수 있다. 프로세스 가상화는 단순히 한 소프트웨어만을 이식하는 방법이고, 시스템 가상화는 소프트웨어 묶음을 이식하는 방법이다. 다시 말해서 오직 하나의 소프트웨어만을 이식하는 경우뿐만 아니라 시스템 자체를 이식하여 사용하는 경우도 존재한다. 따라서 이식 단위에 따른 분류에는 소프트웨어 단위의 이식과 시스템 단위의 이식이 존재한다.

2) 이식 소프트웨어의 추상 단계에 따른 분류

이식 소프트웨어의 추상 단계는 소프트웨어가 어떤 형태로 구성되어 있는지를 통해 판단한다. 이식 소프트웨어의 추상 단계는 이진(Binary) 단계, 원본(Source) 단계, 설계(Design) 단계로 분류할 수 있다. 소프트웨어가 동작하기 위해서는 반드시 이진 단계에 도달하여야 한다. 그러나 이 분류는 다른 환경에 이식되기 직전의 추상 단계를 의미하는 것이 아니다. 개발자가 이식을 위해 변환 과정을 수행할 때, 그 시점의 소프트웨어가 갖는 추상 단계를 통해 결정된다. 예를 들어 전가상화를 통해 이식하고자 할 때의 소프트웨어는 이진 단계를 갖는다. 추상 인터페이스를 통해 이식하고자 할 때의 소프트웨어는 원본 단계를 갖는다. 이처럼 변환 과정에 놓인 소프트웨어의 추상 단계를 통해 이식성을 분류한다.

2.2절에서 언급한 이식성 달성 방안은 본 논문에서 제시한 이식성의 분류를 통해 Table 1과 같이 표현된다.

(Table 1) Categorization of Techniques for Achieving Portability

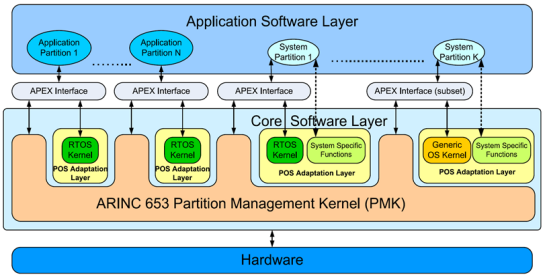
		Separation of Implement-dependent Part	Abstract Interface	MDA	Virtualization		
					Full-Virtualization	Para-Virtualization	Process Virtualization
Porting Unit	Software		✓	✓			✓
	System	✓			✓	✓	
Abstraction Level	Binary				✓		✓
	Source	✓	✓			✓	
	Design			✓			

3. IMA 기반 시스템의 이식성 분류

IMA 아키텍처를 적용한 솔루션으로는 AIR[6], XtratuM[13], PikeOS[14]가 존재한다. 각 솔루션은 IMA 아키텍처의 파티션을 고유의 방식으로 실현하며, 이는 곧 각기 다른 방식으로 이식성을 달성하였음을 의미한다. 본 논문에서는 각 솔루션의 이식성 달성 방안을 파악하고, 이로부터 각 솔루션을 통해 달성된 이식성의 분류를 확인한다.

3.1 AIR 프로젝트

AIR 프로젝트는 ARINC 653[5]의 실현 가능성을 살펴보기 위해 ESA(European Space Agency)의 지원을 받아 수행된 프로젝트이다. AIR는 실시간 운영체제 중 하나인 RTEMS를 대상으로 수행된 AIR와 불특정 다수의 실시간 운영체제를 대상으로 수행된 AIR-II가 존재한다. Fig. 3은 AIR의 아키텍처를 그림으로 표현한 것이다.



(Fig. 3) AIR Architecture [6]

AIR는 크게 파티션으로 구성되는 애플리케이션 소프트웨어 계층, APEX 인터페이스, PAL(POS Adaptation Layer)과 PMK(Partition Management Kernel)로 구성되는 핵심 소프트웨어 계층으로 구성된다. 각각은 다음과 같다.

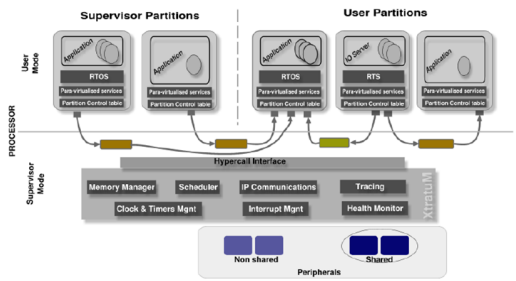
- 애플리케이션 소프트웨어 계층 : 하나 이상의 프로세스로 구성되는 애플리케이션 파티션과 하드웨어 자원을 다룰 수 있는 시스템 파티션으로 구성된다. 각 파티션에서 동작하는 프로세스는 APEX 인터페이스를 통해서 구현된다.
- APEX 인터페이스 : ARINC 653 서비스를 파티션 내부의 프로세스에게 제공하기 위한 요소이다.
- 핵심 소프트웨어 계층 : PAL과 PMK로 구성된다. PAL은 POS을 수정하지 않고 파티션에게 ARINC 653 서비

스를 제공할 수 있도록 도와주는 랩퍼 계층이다. PMK는 POS에서 제공하지 못하는 ARINC 653 서비스인 파티션 관리와 IPC 서비스를 제공하기 위한 요소이다. 공유 자원의 관리는 PMK에서 수행된다.

위 요소 중 APEX 인터페이스가 AIR에서 파티션과 비행 소프트웨어에게 이식성을 제공하는 요소이다. APEX 인터페이스는 핵심 소프트웨어 계층에서 제공하는 서비스를 파티션과 비행 소프트웨어에서 간접적으로 사용할 수 있도록 도와준다. 간접적으로 사용하도록 하는 것은 핵심 소프트웨어 계층으로부터의 독립성을 제공하기 위함이다. AIR의 핵심 소프트웨어 계층은 PAL과 PMK로 구성되는데, PAL은 내장되는 실시간 운영체제가 변경될 때마다 변경된다. 즉, 핵심 소프트웨어 계층은 상황에 따라 유동적이다. 그렇기 때문에 파티션과 비행 소프트웨어에게 항시 동일한 서비스를 제공하기 어렵다. 따라서 APEX 인터페이스를 통해 항상 동일한 서비스를 제공하도록 한다. 여기서 APEX 인터페이스가 곧 추상 인터페이스에 속하며, 이는 곧 AIR가 추상 인터페이스를 통해 이식성을 제공한다는 것을 말한다.

3.2 XtratuM

XtratuM은 실시간 임베디드 시스템을 대상으로 하는 시스템으로, 시간과 공간에 대한 파티션 환경을 보장한다. 초창기에서는 X86 아키텍처를 위한 시스템으로 설계되었으나, 후에 우주산업에서 널리 사용되는 SPARC LEON2 프로세서를 지원하기 위해 재설계 되었다.

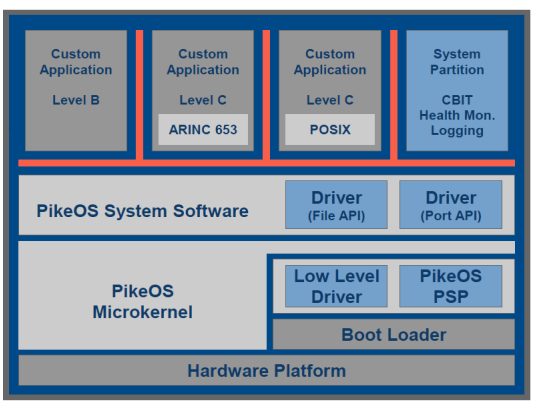


(Fig. 2) XtratuM Architecture [13]

XtratuM의 구조를 보면 크게 XtratuM과 그 위에서 동작하는 파티션들로 구성된다. XtratuM은 각 파티션을 관리하며, 관리자 모드에서 동작한다. 여기서 XtratuM을 하이퍼바이저(Hypervisor)라 부르며, 이는 가상화 기술에서 가상 머신(Virtual Machine)을 관리하는 계층을 의미한다. 즉, XtratuM은 가상화 기술을 바탕으로 구성된 시스템이며, 파티션은 가상 머신 형태로 존재한다. 또한 파티션들은 하이퍼-콜이라 불리는 서비스를 통해 공유 자원에 접근한다. 따라서 XtratuM은 가상화 기술을 통해 이식성을 달성한다는 것을 알 수 있다. 특히, 가상화 기술 중에서 반가상화 기법을 사용한다. 이를 위해서 게스트 운영체제의 시스템-콜을 호출하면 내부적으로 하이퍼-콜을 호출하도록 수정하여야 한다. XtratuM은 가상화 기술 외에도 별도의 이식성 달성 방안을 사용한다. 위 그림에서 게스트 운영체제가 탑재되지 않은 파티션이 존재하는데, 이 파티션 내의 비행 소프트웨어는 XtratuM에서 제공하는 별도의 인터페이스 계층을 통해 동작한다. 단, 이 파티션에는 단일 소프트웨어만 탑재할 수 있다. 따라서 XtratuM에서는 반가상화 기법 외에도 추상 인터페이스를 통해 이식성을 달성한다는 것을 알 수 있다.

3.3 PikeOS

PikeOS는 SYSGO AG에서 개발한 상용 시스템으로, 항공우주, 국방, 자동차, 교통, 산업 자동화, 의료, 네트워크 인프라, 가전제품 등과 같이 기본적으로 안전-필수와 보안-필수라는 요구사항을 갖는 분야를 대상으로 한다. PikeOS는 Fig. 5와 같은 구조를 갖는다.



(Fig. 3) PikeOS Architecture [14]

PikeOS 아키텍처는 크게 가상화 플랫폼과 애플리케이션 계층으로 나뉜다. 가상화 플랫폼은 ARINC 653 서비스와 같은 기능과 가상화 기능도 함께 제공하는 요소이

며, PikeOS 마이크로 커널과 PikeOS 시스템 소프트웨어로 구성된다. 각 요소의 기능은 다음과 같다.

- PikeOS 마이크로 커널: 특권 모드에서 동작하며, 하드웨어 위에서 직접 동작하는 요소이다. 가상화 기술에서 VM(Virtual Machine)을 관리하는 VMM(Virtual Machine Manager)과 같은 역할을 수행한다. 즉, 가상화 기술의 실현을 위한 요소이다.
- PikeOS 시스템 소프트웨어: 사용자 모드에서 동작하며, 각 파티션과 애플리케이션에게 다양한 서비스를 제공한다. 제공되는 서비스로는 파티션의 생성 및 제어와 같은 파티션 관리, 파티션 설정, 헬스 모니터링, 애플리케이션의 적재 등이 있다. 또한 장치 드라이버를 다룰 수 있는 API를 제공한다. 즉, PikeOS 시스템 소프트웨어는 ARINC 653 표준에서 언급한 서비스를 제공하는 요소이다.

PikeOS는 가상화 기술을 통해 이식성을 달성하며, 여기서 사용되는 가상화 기술은 반가상화 기법이다.

4. IMA 아키텍처 이식성 향상 방안

이식성의 정의와 분류를 통해서 (1)이식성 달성 방안에 따라 이식성의 특성의 달라질 수 있으며, (2)이식의 비용을 최소화하는 것이 소프트웨어의 이식성을 향상시킬 수 있음을 파악하였다. 만일 어떤 소프트웨어를 이식하고자 할 때, 해당 소프트웨어가 특정 시스템에서 제공하는 이식 방법에 적합하다고 가정하자. 이와 같은 상황이라면 소프트웨어를 이식하는데 소요되는 비용은 아마 최소화될 것이다. 만일 그렇지 않다면, 그 소프트웨어는 좀 더 많은 비용을 통해 이식되거나 최악의 경우에는 이식되지 못하고 재개발될 것이다. 즉, 다양한 소프트웨어에 대해서 적합한 이식 방법을 모두 제공할 수 있다면 어떠한 소프트웨어든지 최소한의 비용을 통해 이식할 수 있으며, 이는 곧 이식성의 향상을 가져온다. 따라서 본 논문에서는 다양한 이식성 달성 방안의 적용, 즉 다양한 이식성을 제공함으로써 비행 소프트웨어의 이식성을 향상하고자 하는 것이다. 이를 위해서 IMA 아키텍처에서 제공하여야 하는 이식성의 범위를 설정한다. 본 논문에서 설정한 이식성의 범위는 다음과 같다.

$$R_P = \{(U_w, A_s), (U_s, A_s), (U_w, A_b), (U_s, A_b)\}$$

$$where: U = \{U_w, U_s\}$$
$$A = \{A_b, A_s, A_d\}$$

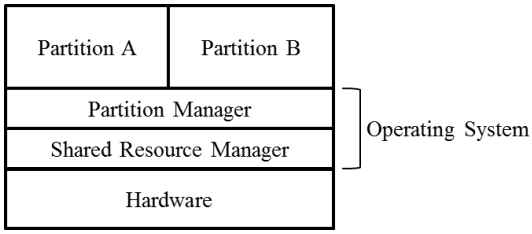
R_P는 이식성의 범위를 의미하며, 이식성의 범위는 이식 단위 U와 이식 소프트웨어의 추상 단계 A의 쌍으로 표현된다. 이식 단위 U는 소프트웨어 단위의 이식(U_w)과 시스템 단위의 이식(U_s)으로 구성되며 이식 소프트웨어의 추상 단계 A는 이진 단계의 이식(A_b), 원본 단계의 이식(A_s) 그리고 설계 단계의 이식(A_d)으로 구성된다.

설정된 이식성의 범위를 보면 설계 단계의 이식을 전혀 고려하지 않는다. 설정된 이식성의 범위를 충족하기 위해서는 각 이식성 범위에 맞는 달성 방안을 IMA 아키텍처에 적용하여야 한다. 그러나 설계 단계의 이식을 위한 시스템 요소는 존재하지 않을 뿐만 아니라 필요하지도 않다. 그 이유는 다음과 같다. 설계 단계의 소프트웨어는 추상화된 모델로 표현된 소프트웨어이며, 이것이 동작하기 위해서는 원본 단계의 소프트웨어로 변환되어야 한다. 만일 설계 단계의 소프트웨어로부터 원본 단계의 소프트웨어를 생성할 때 생성된 소프트웨어가 추상 인터페이스를 준수하도록 생성된다면, 여기서 생성된 원본 단계의 소프트웨어는 추상 인터페이스를 통해 이식성이 달성된다. 혹은 설계 단계의 소프트웨어로부터 이진 단계의 소프트웨어를 생성한다면, 설계 단계의 이식은 가상화 기술을 통해 달성된다. 따라서 설계 단계의 소프트웨어를 위한 별도의 시스템 요소는 필요하지 않다. 이와 같은 이유로 설계 단계의 이식을 고려하지 않는다. 만일 설계 단계의 이식이 고려되어야 한다면, IMA 아키텍처 내에서 고려하는 것이 아니라 별도의 방법론으로 이를 달성해야 한다.

본 논문에서 제시하는 확장된 IMA 아키텍처는 다양한 이식성 달성 방안을 결합하여 수립되므로, 이 아키텍처를 혼합형 IMA(Hybrid IMA, 이후 H-IMA) 아키텍처라 한다. H-IMA 아키텍처를 수립하기 위해서 (1)H-IMA 아키텍처의 기본 구조를 구축하여야 한다. 다음으로 구축된 (2)기본 구조를 바탕으로 다양한 이식성 달성 방안을 적용한다. 본 논문에서는 기본 구조를 정의하는 과정을 스케치(Sketch) 과정이라 하고, 기본 구조에 다양한 이식성 달성 방안을 적용하는 과정을 블랜드(Blend) 과정이라 한다.

4.1 H-IMA 아키텍처 스케치

H-IMA 아키텍처를 수립할 수 있는 가장 큰 이유는 IMA 아키텍처가 모듈화 시스템의 집합으로 구성되며, 모듈화 시스템에게 공유 자원을 제공한다는 것이다. 따라서 하드웨어와 파티션 사이에서 파티션과 공유 자원을 관리하는 요소가 필수적으로 존재한다. [4]에서는 파티션과 하드웨어 사이에서 자원을 관리하는 요소를 운영체제로 표현한다. 그러므로 H-IMA 아키텍처의 기본 구조를 Fig. 6과 같이 표현할 수 있다.



(Fig. 6) Basic Structure of H-IMA

Fig. 6을 보면 H-IMA 아키텍처의 운영체제는 파티션과 공유 자원을 관리하는 요소로 분리된다. 이는 관심의 분리(Separation of Concerns)를 기반으로 한 것이다. 파티션 관리자(Partition Manager)가 수행하여야 하는 기능은 파티션과 밀접한 관계를 가지며, 공유 자원 관리자(Shared Resource Manager)는 하드웨어와 밀접한 관계를 갖는다. 만일 파티션 관리자와 공유 자원 관리자가 수행하여야 하는 작업이 명세되고, 이를 기반으로 플랫폼을 구축한다고 가정하자. 파티션 관리자는 어떠한 시스템에 탑재되더라도 변경되지 않는다. 파티션의 구성은 파티션 관리자에게 영향을 주기 어렵고, 하드웨어의 접근은 공유 자원 관리자를 통해 수행하기 때문이다. 그러나 공유 자원 관리자는 시스템의 변경, 특히 하드웨어의 변경에 민감할 수밖에 없다. 자신이 수행하는 기능은 변하지 않더라도 그 구현은 바뀔 수밖에 없다. 만일 두 요소가 분리되지 않았다면, 하드웨어의 변경에 따라 모든 것이 변경되어야 한다. 이와 같은 이유로 운영체제를 파티션 관리자와 공유 자원 관리자로 분리한다.

4.2 H-IMA 아키텍처 블랜드

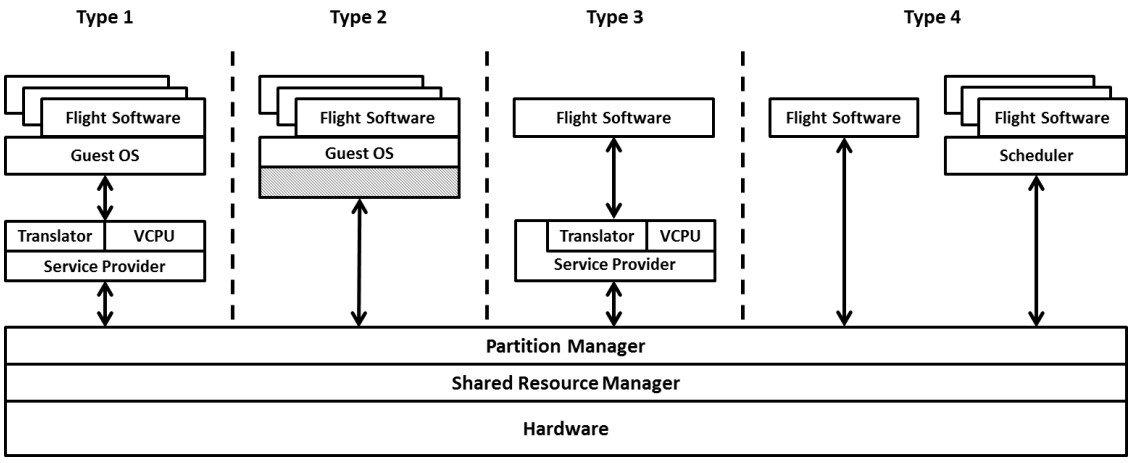
블랜드 과정에서는 설정된 이식성의 범위를 달성할 수 있는 방안을 H-IMA 아키텍처에 적용하는 작업을 수행한다. 각 이식성 달성 방안은 기본 구조에서 정의된 파티션 관리 계층과 자원 관리 계층에서 제공하는 서비스를 이용하여 이식성을 제공하여야 한다. 또한 IMA 아키텍처에서의 이식성은 파티션 단위로 이루어지기 때문에 파티션 단위로 각 달성 방안이 적용될 수 있는 형태가 되어야 한다. 따라서 블랜드 과정을 수행하기 위해서는 적용되는 이식성 달성 방안에 따른 파티션의 유형을 분류하여야 한다. 파티션을 분류하는 기준은 이식성의 범위에 따른다. 따라서 이식 소프트웨어의 추상 단계와 이식 단위를 기준으로 파티션을 분류하며, 분류된 결과는 Table 2와 같다.

(Table 2) Type of Partition

		Abstraction Level	
		Binary	Source
Porting Unit	System	**Type1** (Full-Virtualization)	**Type2** (Para-Virtualization)
	Software	**Type3** (Process Virtualization)	**Type4** (Abstract Interface)

제시된 기준을 통해 분류된 파티션은 총 네 가지이며, 각 파티션에 대한 설명과 각 파티션에 대한 이식성 달성 방안은 다음과 같다.

- (유형1) : 유형1 파티션은 이진 단계의 소프트웨어로 구성되는 파티션으로, 비행 소프트웨어와 게스트 운영체제가 함께 구성된다. 즉, 하나의 시스템에 대한 이미지 파일이 탑재되는 경우이다. 전가상화의 실현을 위해 이진코드 변환 요소가 요구된다. 이진코드 변환 요소는 변환기(Translator)와 가상 CPU로 구성된다. 변환기는 변환 규칙을 기반으로 작성되며, 추가적으로 게스트 명령어의 해석을 위한 디코더(Decoder)와 변환된 코드를 저장할 캐시(Cache) 등이 필요하다. 가상 CPU는 특정 프로세서에 특화된 요소를 해결하기 위해 사용되며, 주로 특권 명령어(Privileged Instruction)에 대한 처리를 위해 사용된다. 유형1 파티션에는 게스트 운영체제가 함께 탑재되기 때문에 비행 소프트웨어에 대한 스케줄링 요소를 추가적으로 제공하지 않아도 된다.
- (유형2) : 유형2 파티션은 원본 단계의 소프트웨어로 구성되는 파티션으로, 비행 소프트웨어와 게스트 운영체제가 함께 구성된다. 반가상화의 실현을 위해 게스



(Fig. 7) H-IMA Architecture

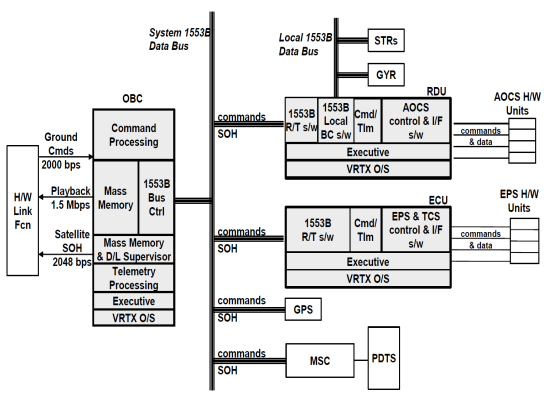
트 운영체제의 수정이 요구된다. 게스트 운영체제가 제공하는 서비스가 호출되면, 내부적으로 파티션 관리자와 공유 자원 관리자가 제공하는 서비스를 이용할 수 있도록 수정되어야 한다. 유형1 파티션과 마찬가지로 게스트 운영체제가 함께 탑재되므로 비행 소프트웨어에 대한 스케줄링 요소를 추가적으로 제공하지 않아도 된다.

- (유형3) : 유형3 파티션은 이진 단계의 단일 소프트웨어로 구성되는 파티션이다. 게스트 운영체제를 포함하지 않으며, 비행 소프트웨어 단독으로 탑재된다. 유형1 파티션과 마찬가지로 이진코드 변환 요소를 통해 이식성을 달성할 수 있다. 유형3 파티션을 위한 이진코드 변환 요소에는 변환기와 가상 CPU 외에도 API 변환기가 추가로 요구된다. API 변환기는 유형3 파티션 내부의 소프트웨어서 호출하는 시스템 콜과 API를 H-IMA 아키텍처의 서비스로 연결하는 역할을 수행한다. 유형1 파티션은 게스트 운영체제가 함께 탑재되어 시스템 콜 혹은 API에 대한 부분을 고려하지 않아도 되지만, 유형3 파티션은 비행 소프트웨어 단독으로 탑재되기 때문에 이와 같은 요소가 필수적이다. 유형3 파티션은 단일 소프트웨어로 구성되어 비행 소프트웨어에 대한 스케줄링 요소를 고려하지 않아도 된다.
- (유형4) : 유형4 파티션은 원본 단계의 소프트웨어로 구성되는 파티션이다. 본래 소프트웨어 단위의 이식을 목적으로 하지만, 소프트웨어의 집합을 시스템으로 가정한다면 시스템 단위의 이식도 가능하다. 다만 게스트 운영체제를 포함하지 않기 때문에, 시스템 단위의 이식을 수행하기 위해서는 별도의 스케줄러를 구현하여야 한다.

위 내용을 H-IMA 아키텍처의 기본 구조에 반영하면 Fig. 7과 같은 H-IMA 아키텍처를 완성할 수 있다.

5. 적용 사례

이번 절에서는 H-IMA 아키텍처가 실제 시스템에 어떤 방식으로 적용될 수 있는지 제시한다. 이를 위해 본 논문에서는 다목적실용위성 2호(KOMPSAT-2)[15]의 비행 소프트웨어를 예시로 사용한다. 다목적실용위성 2호는 Fig. 8과 같은 시스템 구조를 갖는다.



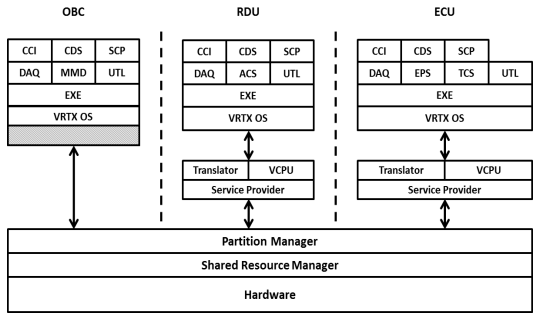
(Fig. 8) System Architecture of KOMPSAT-2 [16]

다목적실용위성 2호는 탑재 컴퓨터(OBC, On-Board Computer), 전력계 제어 장치(ECU, Electrical power subsystem Control Unit), 원격 구동 장치(RDU, Remote Drive Unit)와 같은 세 가지 하드웨어 모듈로 구성된다 [16]. 탑재 컴퓨터는 지상으로부터 명령을 받고 명령을 알맞은 컴퓨터로 분배한다. 또한 센서 데이터와 위성체 상태 데이터를 수집하여 저장하고 이를 지상으로 송신한다. 전력계 제어 장치는 전력 발생 및 분배, 그리고 열 제어를 관리하며, 원격 구동 장치는 자세 제어 센서들로부터 정보를 받아 위성체 자세 및 궤도를 제어한다. 세 개의 하드웨어 모듈은 1533B 데이터 버스에 의해 연결되어 상호 통신을 수행한다. 각 하드웨어 모듈에 탑재되는 비행 소프트웨어는 Table 3과 같다.

(Table 3) Flight Software of KOMPSAT-2

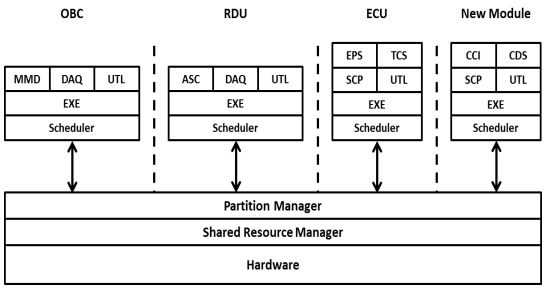
CSC	Description
CCI	Command & Communication Interface
CDS	Command Dispatch Supervisor
SCP	Stored Command Processing
DAQ	Data Acquisition
EXE	Executive
MMD (Only OBC)	Mass Memory & Downlink Management
UTL	Utilities
ACS (Only RDU)	Attitude Control Subsystem
EPS (Only ECU)	Electrical Power Subsystem
TCS (Only ECU)	Thermal Control Subsystem

본 적용 사례에서는 소프트웨어 단위의 이식보다는 시스템 단위의 이식을 목표로 한다. 또한 이미 개발되어 사용되는 비행 소프트웨어이기 때문에 추상 인터페이스를 이용하여 이식성을 달성할 수 없다. 따라서 여기서 사용할 수 있는 이식성 달성 방안은 전가상화와 반가상화 기법이다. 본 논문에서는 탑재 컴퓨터의 비행 소프트웨어는 원본 단계로 존재하며, 전력계 제어 장치와 원격 구동 장치의 비행 소프트웨어는 이진 단계로 존재한다고 가정한다. 그러므로 탑재 컴퓨터의 비행 소프트웨어는 반가상화 기법으로 통해 이식성이 달성되며, 전력계 제어 장치와 원격 구동 장치의 비행 소프트웨어는 전가상화 기법을 통해 이식성이 달성된다. 또한 하드웨어 모듈은 각각 파티션으로 변환되어 하나의 하드웨어 모듈에 탑재된다. 따라서 H-IMA 아키텍처가 적용된 시스템의 모습은 Fig. 9와 같이 표현된다.



(Fig. 9) Use-case 1 of H-IMA

이미 구축된 시스템을 바탕으로 H-IMA 아키텍처를 적용하는 예제에 대해서는 추상 인터페이스를 적용할 수 없다. 추상 인터페이스를 사용하기 위해서는 비행 소프트웨어가 추상 인터페이스를 바탕으로 구현되어야 하기 때문이다. 추상 인터페이스는 당장에 직면한 이식성 문제를 해결하기 위한 달성 방안이 아니다. 추상 인터페이스를 통해 이식성이 달성되기 위해서는 추상 인터페이스 기반의 비행 소프트웨어가 충분히 존재하여야 하며, 이를 달성하기까지는 어느 정도의 시간이 요구된다. 그러나 대부분의 비행 소프트웨어가 추상 인터페이스를 기반으로 구현된 경우라면 좀 더 효율적인 이식이 가능하다. 만일 다목적실용위성 2호의 비행 소프트웨어가 모두 추상 인터페이스를 기반으로 구현되었다면 Fig. 10과 같은 구성도 가능하다.



(Fig. 10) Use-case 2 of H-IMA

가상화 기술만을 사용하였을 때는 하드웨어 모듈을 그대로 파티션으로 사용하였다. 그러나 추상 인터페이스

를 사용한 경우에는 필요에 의해 파티션을 생성할 수 있다. 위의 적용 예제에서는 세 개의 하드웨어 모듈이 공통적으로 가지고 있던 비행 소프트웨어를 모아 새로운 모듈을 생성한 모습이다. 추상 인터페이스를 기반으로 구현된 경우에는 파티션의 구성을 자유롭게 할 수 있다.

5. 결 론

본 논문에서는 비행 소프트웨어가 갖는 이슈 중 하나인 동작 환경과의 종속성에 주목하였다. 동작 환경과의 종속성으로 인해 비행 소프트웨어의 재사용성은 매우 낮을 수밖에 없어 새로운 시스템을 구성할 때마다 재개발되는 문제가 발생한다. 이는 결국 인공위성을 구축하는데 소요되는 시간과 비용을 증대시키는 결과를 초래한다. 이와 같은 문제는 IMA 아키텍처가 제공하는 이식성을 통해서 해결될 수 있다. 이식성을 통해서 비행 소프트웨어의 재사용성을 향상시킬 수 있으며, 이는 곧 인공위성을 구축하는데 소요되는 시간과 비용 역시 감소시킨다. 그러나 현재 IMA 아키텍처를 적용한 솔루션들은 한정적인 이식성을 제공함으로써 특정 조건에서만 비행 소프트웨어의 이식성을 달성한다. 이에 본 논문에서는 다양한 이식성 달성 방안을 결합함으로써 IMA 아키텍처가 다양한 이식성을 제공하도록 확장하였다. 이식성은 이식의 비용을 줄임으로써 향상되기 때문에 이와 같은 방법을 통해 IMA 아키텍처의 이식성을 향상시켰으며, 더 나아가 다양한 상황에 대해서 이식성을 달성할 수 있는 방법을 제공하였다. 추후에는 제시된 H-IMA 아키텍처를 바탕으로 시스템을 구축할 수 있도록 구축 절차를 정의하고 정의된 절차를 바탕으로 실제 시스템을 구현할 예정이다.

참 고 문 헌(Reference)

[1] J. Lee, S. Cha, “Development trends of satellite flight software”, Journal of Computing Science and Engineering, vol. 25, no. 2, pp. 43-48, 2007.

[2] C. B. Watkins, “Integrated modular avionics: managing the allocation of shared intersystem resources”, 25th Digital Avionics Systems Conference (DASC), Portland, Oregon, 2006.

[3] R. L. C. Eveleens, “Integrated Modular Avionics Development Guidance and Certification Considerations”, National Aerospace Laboratory NLR, 2006.

[4] DOT/FAA/AR-99/58, “Partitioning in Avionics Architectures : Requirements, Mechanisms, and Assurance”, 2000.

[5] ARINC 653-Part 1, “avionics application software standard interface Part 1 - Required Services”, Airlines electronic engineering committee(AEEC), 2006.

[6] J. Rufino, J. Craveiro, “Robust Partitioning and Composability in ARINC 653 Conformant Real-Time Operating Systems”, 1st INTERAC Research Network, 2008.

[7] S. R. Schach, “Object-Oriented and Classical Software Engineering”, 8th ed., ,2002.

[8] H. M. Sneed, “Measuring Reusability of Legacy Software Systems”, Software Process: Improvement and Practice, vol. 4, pp 43-48, 1998.

[9] OMG, “Developing in OMG’s Model Driven Architecture”, ftp://ftp.omg.org/pub/docs/omg/01-12-01.pdf

[10] J. E. Smith, R. Nair, “Virtual machines: versatile platforms for systems and processes”, Morgan Kaufmann Publishers, San Francisco, CA, USA, 2005.

[11] A. Aquiar, F. Hessel, “Current techniques trends in embedded system’s virtualization”, Softw., Pract. Exper. vol. 42, no. 7, pp. 917-944, 2012.

[12] Li, Y., et al,. “A Survey of Virtual Machine System: Current Technology and Future Trends.”, Third International Symposium on Electronic Commerce and Security, 2010.

[13] M. Masmano, I. Ripoll, A. Crespo, “XtratuM Hypervisor for LEON3-user manual”, 2011.

[14] R. Kaiser, “Combining Partitioning and Virtualization for Safety-Critical Systems”, SYSGO White Paper, 2007.

[15] J. Choi, J. Lee, J. Lee, “The Design of Executive Flight Software CSC for KOMPSAT-2”, Proc. of the 30th KIISE Fall Conference, vol.30, no.2(II), pp.262-264, 2003.

[16] S. Kang, J. Lee, J. Choi, J. Lee, “Flight Software Development for KOMPSAT-2 On-Board Computers”, Proc. of the 30th KIISE Fall Conference, vol.30,

no.2(II), pp.2346-348, 2003.
[17] J. D. Mooney, “Bringing Portability to the Software Process”, Technical Report TR-97-1, Dept. of Statistic and Computer Science, West Virginia University, 1997.

◐ 저 자 소 개 ◑

서 용 진
2011년 충남대학교 컴퓨터공학과 졸업(학사)
2011년~현재 충남대학교 컴퓨터공학과 (박사 과정, 석박사통합)
관심분야 : 소프트웨어 테스팅, 스마트폰, UX/UI
E-mail : yjseo082@cnu.ac.kr

김 현 수
1988년 서울대학교 계산통계학과 졸업(학사)
1991년 한국과학기술원 전산학과 졸업(석사)
1995년 한국과학기술원 전산학과 졸업(박사)
1995년~1995년 한국전자통신연구원 Post Doc.
1996년~2001년 금오공과대학교 조교수
2001년~현재 충남대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어 공학, 소프트웨어 테스팅, 소프트웨어 아키텍처
E-mail : hskim401@cnu.ac.kr