KIPS Tr. Software and Data Eng.
Vol.3, No.1 pp.7~18 pISSN: 2287-5905

http://dx.doi.org/10.3745/KTSDE.2014.3.1.7

H-IMA : IMA based Hybrid Platform Architecture for Improving Portability of Flight Software

Yongjin Seo† · Sangpil Yun†† · Hyunwoo Joe††† · Cheolsoon Kwon†††† · Hyungshin Kim††††† · Hyeon Soo Kim†††††

ABSTRACT

Flight software operated on the on-board computers in the satellite has requirements such as real-time and 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 between them should be removed. And the work can be achieved by improving the portability of the flight software. In this paper, we propose a platform architecture based on the IMA architecture. The platform architecture is a hybrid one built by blending two kinds of realizations of the IMA architecture in order to maximize portability. In addition, we implement a prototype system and analyze the execution results of the system to justify the proposed architecture. The proposed architecture enables us to remove the dependency between fight software and operating environments.

Keywords : Flight Software, Software Architecture, Portability

H-IMA : 비행 소프트웨어의 이식성 향상을 위한 IMA 기반의 혼합형 플랫폼 아키텍처

서 용 진† · 윤 상 필†† · 조 현 우††† · 권 철 순†††† · 김 형 신††††† · 김 현 수†††††

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 종속성을 제거할 필요가 있으며, 이는 비행 소프트웨어의 이식성 향상을 통해 달성할 수 있다. 본 논문에서는 이를 위해 IMA 아키텍처 기반의 플랫폼 아키텍처를 제안한다. 이 아키텍처는 이식성을 극대화하기 위해 두 가지의 IMA 아키텍처 실현 방안을 기반으로 구축된 혼합형 아키텍처이다. 또한 혼합형 아키텍처의 검증을 위해 혼합형 아키텍처 기반의 시스템을 구현하고 동작 결과를 분석한다. 본 논문에서 제안한 아키텍처를 통해서 비행 소프트웨어와 동직 환경 사이의 종속성을 제거할 수 있다.G망을 활용하여 끊김없는 서비스를 지원할 수 있는 방안을 제안하였으며, 테스트베드 구현을 통하여 제안 방안의 우수성을 검증하였다.

키워드 : 비행 소프트웨어, 소프트웨어 아키텍처, 이식성

1. 서 론

비행 소프트웨어(Flight Software)는 인공위성의 탑재 컴퓨터에서 사용되는 소프트웨어로, 우주 공간이라는 특수한 환경에서 동작한다[1]. 비행 소프트웨어는 우주 공간이 갖는 접근성의 문제로 인해 유지보수가 거의 불가능하다. 따라서 탑재된 비행 소프트웨어는 내고장성(Fault Tolerance)과 고장 확산(Fault Propagation) 방지 기능, 고신뢰성이 요구된다. 뿐만 아니라 적재적소에 주어진 임무를 수행하여야 하기 때문에 실시간성 또한 요구되며, 탑재 컴퓨터라는 한정된 자원을 기반으로 동작하여야 한다. 이와 같은 요구사항을 만족하기 위해서 지금까지 비행 소프트웨어는 고신뢰성을 보장하는 실시간 운영체제에 종속적으로 개발되었다. 비행 소프트웨어가 특정 동작 환경(예: 실시간 운영체제, 프로

※ 본 연구는 한국연구재단을 통해 교육과학기술부의 우주기초원천기술개발 사업(NSL, National Space Lab)으로부터 지원받아 수행되었습니다(2011-0020905).
† 준 회 원 : 충남대학교 컴퓨터공학과 박사과정
†† 준 회 원 : 충남대학교 컴퓨터공학과 석사과정
††† 비 회 원 : 충남대학교 컴퓨터공학과 박사과정
†††† 비 회 원 : 충남대학교 컴퓨터공학과 석사과정
††††† 비 회 원 : 충남대학교 컴퓨터공학과 교수
†††††† 종신회원 : 충남대학교 컴퓨터공학과 교수
논문접수 : 2013년 10월 28일
수 정 일 : 1차 2013년 11월 29일
심사완료 : 2013년 11월 29일
* Corresponding Author : Hyeon Soo Kim(hskim401@cnu.ac.kr)

세서 등)에 종속성을 가짐으로 인해 비행 소프트웨어의 재사용성이 크게 떨어진다는 문제점이 발생하게 된다. 비행 소프트웨어를 개발하고 검증하기 위한 노력과 비용 등을 고려할 때 비행 소프트웨어의 재사용성은 중요한 이슈이며, 이를 위해 비행 소프트웨어는 동작 환경에 독립성을 갖도록 개발되어야 한다.

고신뢰성, 실시간성, 한정적 자원 기반의 동작과 같은 문제는 IMA 아키텍처를 통해 대부분 해결된다. 그러나 동작 환경에 대한 독립성은 IMA 아키텍처를 통해 완벽히 해소되지 않는다. IMA 아키텍처에서는 동작 환경에 대한 독립성을 해소하기 위해서 이식성을 제공한다. 이식성은 실현하는 방법에 따라서 그 특성이 바뀌어 상황에 따라 동작 환경에 대한 독립성을 제공하지 못한다. 이에 본 논문에서는 IMA 아키텍처의 이식성을 향상시킨 H-IMA 아키텍처를 제안한다. 다양한 상황에 대해서 모두 이식성을 제공할 수 있도록 확장함으로써, 비행 소프트웨어가 동작 환경으로부터 독립성을 갖도록 한다.

본 논문은 다음과 같이 구성된다. 2절에서는 H-IMA 아키텍처를 수립하는데 필요한 배경 지식인 IMA 아키텍처와 IMA 아키텍처를 실현한 P-IMA와 V-IMA가 제공하는 이식성의 차이를 살펴본다. 3절은 H-IMA의 수립 및 검증에 대한 내용을 담는다. H-IMA 아키텍처의 수립 과정과 H-IMA를 구성하는 다양한 파티션 유형에 대해서 소개한다. 4절은 본 논문에서 수립한 H-IMA 기반의 시스템 구현에 대한 내용을 담는다. 마지막으로 결론으로 마무리를 짓는다.

2. 관련 연구

2.1 IMA 아키텍처

IMA(Integrated Modular Avionics) 아키텍처는 항공 분야에서 제안된 아키텍처로, 모듈화 시스템(Modular System)의 집합으로 구성된다[2]. IMA 아키텍처에서는 모듈화 시스템을 파티션(Partition)이라 부르며, 비행 소프트웨어는 파티션 내부에 포함된다. 파티션들은 IMA 플랫폼을 통해 관리되며, IMA 플랫폼은 파티션에게 공유 자원을 제공한다[3]. 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.

IMA 아키텍처를 실현하는 방법에 따라서 가상화 기반의 IMA(Virtualization based IMA, 이후 V-IMA), 파티셔닝 기반의 IMA(Partitioning based IMA, 이후 P-IMA), 혼합형 IMA(Hybrid IMA, 이후 H-IMA)로 나눌 수 있다. 여기서 V-IMA와 P-IMA는 문헌 [4]에서 제시한 IMA 실현 방안으로부터 도출되며, H-IMA는 본 논문에서 제안하는 IMA 실현 방안이다. V-IMA는 가상화 기술을 바탕으로 개발된 IMA 시스템을 의미한다. IMA 아키텍처의 파티션을 가상화 기술의 가상머신(Virtual Machine)으로 구현한다. P-IMA는 ARINC 653 표준[5]을 따르는 IMA 시스템을 의미한다. 가상화 기술과 달리 추상 인터페이스를 통해서 이식성을 제공한다. H-IMA는 본 논문에서 제시하는 실현 방법으로 V-IMA와 P-IMA의 혼합형 IMA 시스템을 의미한다. V-IMA와 P-IMA는 모두 이식성을 제공하나, 특정 조건에 한하여 이식성을 제공한다. H-IMA는 V-IMA와 P-IMA가 갖는 이식성을 모두 제공한다.

2.2 IMA 아키텍처의 이식성

IMA 아키텍처의 이식성을 살펴보기 위해서는 P-IMA와 V-IMA가 제공하는 이식성을 살펴보아야 한다. IMA 아키텍처는 단순히 개념에 불과하여 구체적인 요소가 존재하지 않기 때문이다. P-IMA와 V-IMA가 제공하는 이식성은 Table 1과 같이 정리할 수 있다. Table 1에 명시된 내용은 1)과 2)절에서 자세히 설명한다.

Table 1. Portability of P-IMA and V-IMA

P-IMA	V-IMA
Portability from Operating System	Portability from Processor
Portability based on Source Code	Portability based on Binary Code
Portability for Flight Software	Portability for System

1) P-IMA의 이식성

P-IMA는 파티셔닝을 바탕으로 IMA 아키텍처를 달성한다. 파티셔닝은 ARINC 653 표준에 명시된 개념으로, 유사한 임계성(Criticality)을 갖는 비행 소프트웨어를 하나로 엮어 오류의 확산을 막는 방법이다. 이때, 비행 소프트웨어를 엮는 단위를 파티션이라 한다. 파티셔닝을 달성하기 위해서는 시간(예: CPU)과 공간(예: 메모리)에 대해 분리가 이루어져야 한다. 시간과 공간에 대한 분리를 위해 ARINC 653에서는 하드웨어와 파티션 사이에 자원 관리를 위한 계층을 제공하며, Fig. 1을 통해 확인할 수 있다.

ARINC 653 표준에서는 핵심 소프트웨어를 통해 하드웨어와 파티션이 분리된다. 핵심 소프트웨어는 ARINC 653 서비스를 제공하는데, 이는 일반적인 운영체제에서 제공하는 시스템 콜과 유사하다. 또한 APEX(APplication EXecutive)라는 별도의 인터페이스를 통해서 파티션 내부의 애플리케이션에게 이식성을 제공한다. 이처럼 P-IMA는 APEX와 같은 별도의 인터페이스를 통해 이식성을 제공한다. APEX를 일반적으로 추상 인터페이스라 부르며, 이는 이식성을 제공

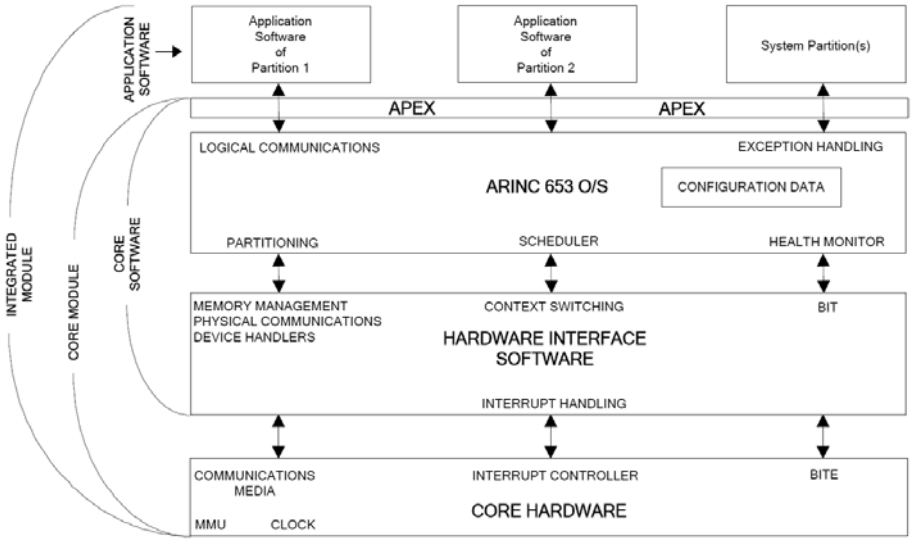

Fig. 1. System Structure of ARINC 653

하는 방법 중 하나이다[6].

추상 인터페이스를 통해 개발된 소프트웨어는 특정 운영체제의 시스템 콜에 종속되지 않아 운영체제에 대해 이식성을 갖는다. 또한 특정 운영체제의 시스템 콜을 가져다 쓰지 않기 때문에, 소프트웨어의 소스코드를 어떤 운영체제에서든 그대로 재사용할 수 있다. 이는 곧 소프트웨어의 수정이 용이함을 의미하며, 이와 같은 특징은 파티션의 구성을 자유롭게 한다. 어떤 파티션에도 종속되지 않고 시스템에 탑재될 수 있기 때문에 P-IMA 기반의 시스템에서는 소프트웨어 자체가 이식성을 갖는다. 따라서 P-IMA는 Table 1의 내용처럼 운영체제로부터의 이식성, 소스코드 기반의 이식성, 비행 소프트웨어 단위의 이식성을 제공한다.

2) V-IMA의 이식성

V-IMA는 가상화 기술을 바탕으로 IMA 아키텍처를 달성한다. 가상화는 특정 시스템이나 프로세스에게 동형이질성(Isomorphism)을 제공하는 것을 말한다[7]. 동형이질성이란, 본래 동작 환경과는 다른 구성의 시스템에서도 본래 동작 환경과 동일한 동작을 하는 것을 말한다. 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[8][9]. 가상화가 제공하는 다양한 기능은 VMM(Virtual Machine Monitor) 혹은 Hypervisor라고 부르는 요소를 통해서 제공된다. VMM은 게스트를 관리하고 게

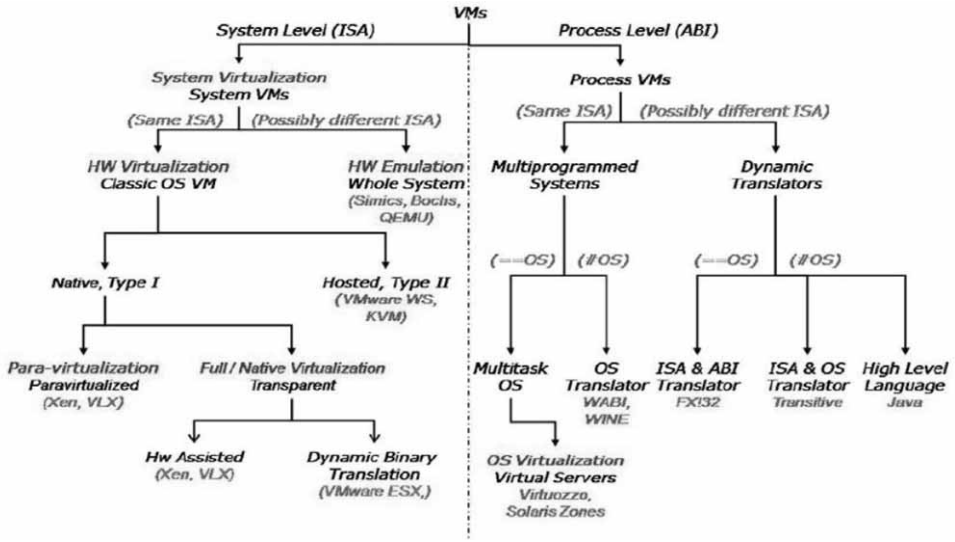

Fig. 2. Taxonomy of Virtualization

스트가 호스트의 하드웨어 자원을 사용할 수 있도록 도와준다. 가상화의 다양한 목적만큼 다양한 가상화 기술이 존재한다. Fig. 2은 다양한 가상화의 분류를 보여준다[7].

V-IMA는 이진코드 변환 기술을 통해 이식성을 제공한다. 이진코드 변환 기술은 게스트의 이진코드를 호스트에서 동작할 수 있는 이진코드로 변환하는 기술을 의미한다. 따라서 V-IMA는 어떤 프로세서 위에서 동작하든지 관계없이 비행 소프트웨어를 동작시킬 수 있다. 즉, 프로세서에 대한 이식성을 제공한다. 그러나 이진코드로부터 실행 가능한 이진코드를 얻어내기 때문에 수정이 용이하지 않다는 단점이 존재한다. 이러한 이유로 V-IMA에 탑재되는 시스템은 수정이 불가능하여 이전 시스템의 형태 그대로 탑재하여야 한다. 따라서 V-IMA는 Table 1의 내용처럼 프로세서로부터의 이식성, 이진코드 기반의 이식성, 시스템 단위의 이식성을 제공한다.

3. H-IMA(Hybrid IMA) 아키텍처

본 논문에서는 V-IMA와 P-IMA의 이식성을 모두 제공하기 위해 두 실현 방안의 혼합형 아키텍처인 H-IMA를 제안한다. V-IMA와 P-IMA는 동일한 목적을 달성하고 있으나, 엄연히 다른 구성을 갖는 시스템이기 때문에 하나의 아키텍처로 혼합하는 과정은 특정한 절차와 방법에 입각하여 수행하여야 한다. H-IMA를 수립하기 위해서 다음의 절차를 따른다. (1)P-IMA와 V-IMA 사이의 공통점과 차이점을 파악한다. 다음으로 파악한 (2)공통점을 바탕으로 H-IMA의 기본 구조를 정의한 뒤, (3)기본 구조에 P-IMA와 V-IMA의 차이점을 반영한다.

이와 같은 절차를 따르는 이유는 각 시스템의 장점을 최대한 유지하기 위함이다. 일반적으로 시스템의 장점은 시스템의 잘 갖추어진 구조로부터 나오기 때문에, 시스템의 장점을 살리기 위해서는 시스템의 구조를 최대한 준수하여야 한다. 이번 절에서는 주어진 절차에 따라 V-IMA와 P-IMA의 공통점과 차이점을 파악하고, 파악된 공통점과 차이점으로부터 H-IMA의 수립 방안을 제안한다.

3.1 V-IMA와 P-IMA의 공통점과 차이점

V-IMA와 P-IMA는 기본적으로 IMA 아키텍처의 실현 방안이기 때문에, IMA 아키텍처가 갖는 특징 자체가 두 실현 방안의 공통점이 된다. IMA 아키텍처의 특징은 (1)자원과 소프트웨어의 분리를 위한 계층을 제공한다는 것과 소프트웨어의 임계도(Criticality)를 바탕으로 분리하여 탑재한다는 점, 즉 (2)파티션을 제공하는 것이다. 자원과 소프트웨어를 분리하는 것은 일반적으로 운영체제에서 수행하는 것이다. 이러한 이유로 문헌 [4]에서는 IMA 실현 방안을 운영체제 서비스를 제공하는 방법에 따라 V-IMA와 P-IMA로 나눈다. 여기까지만 보면 일반적인 운영체제와 차이가 없어 보이지만, 파티션을 통해 소프트웨어가 탑재된다는 것이 일반적인 운영체제와의 차이점이다. 따라서 IMA 아키텍처에서는 일반적인 운영체제에서 제공하는 기능 외에도 파티션을 관리하기 위한 기능을 추가적으로 제공하여야 한다. 다시 말해 IMA 아키텍처는 자원과 파티션을 관리하는 계층이 요구되며, 이것이 곧 V-IMA와 P-IMA의 공통점이 된다.

V-IMA와 P-IMA의 차이점은 제공되는 이식성이다. V-IMA는 이진코드로 구성된 소프트웨어에 대해서 이식성을 제공하며, P-IMA는 추상 인터페이스를 기반으로 구현된 소프트웨어에 대해서 이식성을 제공한다. 다시 말해, V-IMA와 P-IMA는 이식성을 제공할 수 있는 소프트웨어의 유형이 다르다는 것이다. 이 차이로 인해 V-IMA와 P-IMA는 서로 다른 이식성의 달성 방법과 장점을 갖는다.

1) V-IMA의 장점

V-IMA는 이식에 대한 전제 조건이 없다. 여기서 전제조건이란 이식성을 제공받기 위해 소프트웨어가 충당하여야 하는 요소를 의미한다. P-IMA는 소프트웨어가 추상 인터페이스에서 제공하는 API를 기반으로 구현되지 않으면 소프트웨어에게 이식성을 제공할 수 없다. 즉, P-IMA는 추상 인터페이스를 사용하여야 한다는 것이 전제 조건이다. V-IMA는 자체에서 소프트웨어의 이식성을 위한 요소를 포함하고 있기 때문에 이식성을 위한 요소를 소프트웨어가 갖추고 있을 필요가 없다. 이와 같은 이유로 V-IMA는 실제 적용이 빠르다는 장점을 갖는다.

2) P-IMA의 장점

P-IMA는 소프트웨어의 활용도를 높일 수 있다. V-IMA는 이진코드로 구성된 소프트웨어를 사용하는 것을 기본으로 한다. 이러한 이유로 이진코드로 구성된 소프트웨어는 그 구조를 바꾸는 것이 거의 불가능하다. 예를 들어 V-IMA에서 사용되는 소프트웨어는 보통 운영체제와 함께 탑재된다. 이렇게 탑재된 소프트웨어는 함께 탑재된 운영체제를 다른 것으로 교체하는 것이 불가능하며, 더 나아가 소프트웨어 자체를 수정하는 것 역시 불가능하다. 하지만 P-IMA는 추상 인터페이스를 사용하기 때문에 어떠한 운영체제에도 탑재될 수 있다. 또한 추상 인터페이스를 기반으로 구현되기 때문에 소스코드 자체를 재사용할 수 있는 장점이 있다. 즉, 한 번 개발된 소프트웨어가 그 자체로만 사용되는 것이 아니라 새로운 소프트웨어를 개발할 때 또 다른 자원으로 활용될 수 있다. 따라서 P-IMA는 운영체제로부터의 독립성, 그리고 소스코드의 재사용성을 통해서 높은 활용도를 갖는 소프트웨어를 제공한다.

3.2 H-IMA 아키텍처 수립

앞서 파악된 공통점과 차이점을 통해 H-IMA 아키텍처를 수립한다. 본 논문에서는 공통점을 바탕으로 기본 구조를 정의하는 과정을 스케치(Sketch) 과정이라 하고, 기본 구조에 차이점을 반영하는 과정을 블랜드(Blend) 과정이라고 한다.

1) H-IMA 아키텍처 스케치

P-IMA와 V-IMA의 공통점을 통해 하드웨어(자원)와 파

티션 사이에서 파티션과 공유 자원을 관리하는 요소가 필수적임을 파악하였다. 따라서 H-IMA 아키텍처의 기본 구조를 Fig. 3과 같이 표현할 수 있다.

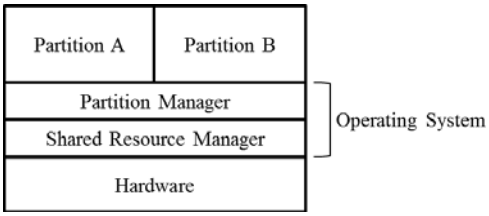

Fig. 3. Basic Structure of H-IMA

H-IMA의 기본 구조는 단순히 하드웨어와 파티션 사이에 운영체제가 존재하는 형태를 갖는다. 다만, 기존의 운영체제와 달리 파티션을 관리하는 계층, 즉 파티션 관리자(Partition Manager)가 존재하며, 파티션과 공유 자원을 관리하는 계층은 서로 분리된다. 운영체제를 두 개의 관리 계층으로 나눈 것은 관심의 분리(Separation of Concerns)를 위함이다. 파티션 관리자가 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 않는다면, 하드웨어의 변경에 따라 모든 것이 변경되어야 한다. 이와 같은 이유로 운영체제를 파티션 관리자와 공유 자원 관리자로 분리한다.

2) H-IMA 아키텍처 블랜드

P-IMA와 V-IMA의 차이점은 제공되는 이식성에 있다. IMA 아키텍처에서 이식성의 주체는 파티션이므로, 결국 파티션 자체가 두 시스템 사이의 차이점을 만들어 낸다고 볼 수 있다. 따라서 P-IMA와 V-IMA의 차이점을 구체화하기 위해서는 H-IMA 아키텍처에서 사용되는 파티션의 유형을 파악하는 것이 중요하다. H-IMA 아키텍처에서 사용되는 파티션은 간단하게 P-IMA 기반의 파티션과 V-IMA 기반의 파티션으로 나뉜다. 추가적으로 파티션에 탑재되는 소프트웨어의 개수를 통해 파티션을 분류할 수 있다. 이 기준은 단순하게 생각하면 비효율적일 수 있지만, 소프트웨어의 개수는 관리 측면에서 차이가 있다. 만일 파티션 내부에 소프트웨어가 하나만 존재한다면, 내부의 소프트웨어 자체가 하나의 파티션과 동일하기 때문에 태스크 관리에 대해 고려하지 않아도 된다. 반면, 파티션 내에 소프트웨어가 두 개 이상이면 태스크 관리는 반드시 필요하다. 즉, 태스크 관리 측면에서 단일 소프트웨어로 구성된 파티션과 다수의 소프트웨어로 구성된 파티션으로 분류할 수 있다. 이와 같은 기준으로 Table 2와 같이 파티션을 분류할 수 있다.

Table 2. Type of Partition

		Software Type	
		Binary	Source
# of FSW	Multiple	Type1	Type2
	Single	Type3	Type4

제시된 기준을 통해 분류된 파티션은 총 네 가지이며, 각 파티션에 대한 설명은 다음과 같다.

- (유형1) : 유형1 파티션은 V-IMA 기반의 파티션으로, 두 개 이상의 이진 소프트웨어로 구성되는 파티션이다. 비행 소프트웨어의 집합만으로 구성되기 보다는 게스트 운영체제와 함께 구성된다. 즉, 하나의 시스템에 대한 이미지 파일이 탑재되는 경우이다. 유형1 파티션에는 게스트 운영체제가 함께 탑재되기 때문에 비행 소프트웨어에 대한 스케줄링 요소를 추가적으로 제공하지 않아도 된다.
- (유형2) : 유형2 파티션은 P-IMA 기반의 파티션으로, 두 개 이상의 원본 소프트웨어로 구성되는 파티션이다. 유형1과 달리 게스트 운영체제가 포함되지 않으며, 오로지 비행 소프트웨어의 집합으로만 구성된다. 유형 2 파티션은 추상 인터페이스와 함께 다수의 비행 소프트웨어를 스케줄링 할 수 있는 기능이 별도로 요구된다.
- (유형3) : 유형3 파티션은 V-IMA 기반의 파티션으로, 하나의 이진 소프트웨어로 구성되는 파티션이다. 게스트 운영체제를 포함하지 않으며, 비행 소프트웨어 단독으로 탑재된다. 유형3 파티션은 단일 소프트웨어로 구성되어 비행 소프트웨어에 대한 스케줄링 요소를 고려하지 않아도 된다.
- (유형4) : 유형4 파티션은 P-IMA 기반의 파티션으로, 하나의 원본 소프트웨어로 구성되는 파티션이다. 게스트 운영체제를 포함하지 않으며, 비행 소프트웨어 단독으로 탑재된다. 단일 소프트웨어로 구성되어 비행 소프트웨어에 대한 스케줄링 요소를 고려하지 않아도 된다.

분류된 파티션을 H-IMA 아키텍처의 기본 구조에 반영함으로써, H-IMA 아키텍처를 완성할 수 있다. 그러나 분류된 파티션이 동작할 수 있는 시스템의 구성은 각 파티션마다 다르다. 예를 들면, 유형1 파티션은 전가상화 기법을 기반으로 구성된 시스템에서 동작하며 유형2 파티션은 추상 인터페이스를 제공하는 시스템에서 동작할 수 있다. 즉, 분류된 파티션을 반영하기 위해서는 별도의 요소가 제공되어야 한다. 그러나 이러한 요소는 특정 파티션에 종속적으로 구성될 수밖에 없기 때문에 Fig. 3의 운영체제에서 직접 이를 제공하면 H-IMA 아키텍처 스케치 과정에서 의도하였던 관심의 분리를 위배할 뿐만 아니라 이식성을 해친다. 따라서 파티션 지원 계층(Partition Supporting Layer)을 통해 파티

션의 동작 환경을 제공한다. 즉 V-IMA 기반의 파티션인 유형1과 유형3을 위해서는 이진코드 변환 요소가 반드시 지원되어야 하고, P-IMA 기반의 파티션인 유형2와 유형4를 위해서는 추상 인터페이스가 제공되어야 한다. 이런 내용을 H-IMA 아키텍처의 기본 구조에 반영하면 Fig. 4와 같은 H-IMA 아키텍처를 완성할 수 있다.

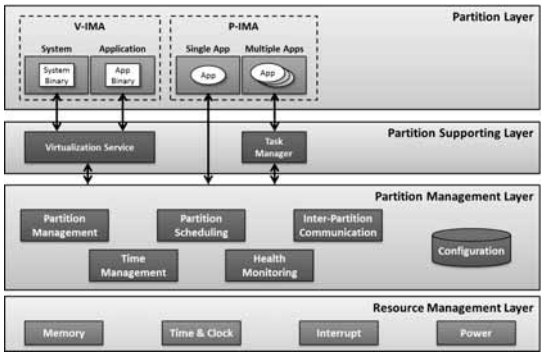

Fig. 4. H-IMA Architecture

3.3 H-IMA의 타당성 검증

H-IMA 아키텍처의 목적은 P-IMA와 V-IMA의 이식성을 모두 제공하는 것에 있다. 따라서 H-IMA 아키텍처의 타당성을 제시하기 위해서는 H-IMA 아키텍처가 제공하는 이식성에 대한 타당성에 대해서 증명하여야 한다. H-IMA 아키텍처가 P-IMA와 V-IMA의 이식성을 모두 제공할 수 있다는 것은 P-IMA와 V-IMA보다 더 많은 상황에 대해서 이식성을 제공한다는 것이다. 이식성의 이점이란 개발 비용과 시간의 절감을 의미한다. 따라서 H-IMA 아키텍처의 이식성에 대해서 이야기하기 위해서는 H-IMA 아키텍처 기반의 시스템에 비행 소프트웨어를 탑재할 때 소요되는 개발 비용과 시간을 확인하여야 한다. 더 나아가 같은 상황에서 P-IMA와 V-IMA보다 더 적은 비용과 시간을 소요하는지 확인함으로써 H-IMA 아키텍처를 평가할 수 있다.

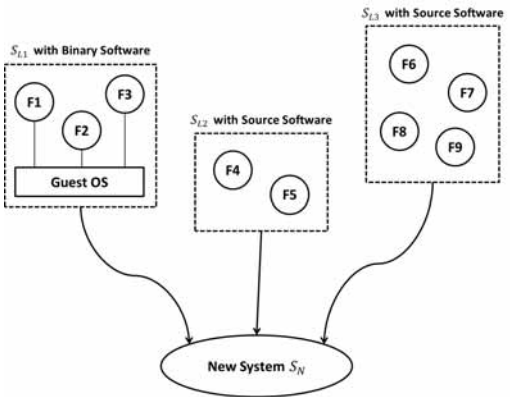

Fig. 5. Structure of Legacy Systems

이와 같은 방식으로 H-IMA 아키텍처의 이식성을 평가하기 위해서 시나리오를 정의한다. 본 논문에서는 기존 시스템(Legacy System, S_L로 표기)가 갖는 비행 소프트웨어를 바탕으로 새로운 시스템(New System, S_N로 표기) S_N을 구축하는 시나리오를 사용하고자 한다. 이때, 세 가지 상황을 고려한다. (1)S_L의 비행 소프트웨어를 그대로 S_N에 탑재하는 경우, (2)S_L의 비행 소프트웨어 외에 새로 추가되는 비행 소프트웨어가 존재하는 경우, (3) S_L의 비행 소프트웨어 중 일부를 제외하고 탑재하는 경우. S_N을 구축하는데 사용되는 S_L은 Fig. 5와 같은 구성을 가진다. Fig. 5에서는 세 가지의 S_L이 사용되며 구분을 위해 S_{L1}, S_{L2}, S_{L3}와 같이 표기하도록 한다. 또한 각 S_L 내부에 존재하는 F1, F2, ... , F9는 S_L에 탑재되어 있는 비행 소프트웨어를 의미한다. S_N은 다음과 같은 가정 하에 구축된다.

- S_L은 각각 하나의 파티션으로 탑재된다.
- 파티션 내부의 비행 소프트웨어를 새로운 시스템에 탑재하는 방법으로는 P-IMA 기반의 이식(P)과 V-IMA 기반의 이식(V), 그리고 재개발(R)로 나눌 수 있다.
- S_L의 원본 소프트웨어는 P-IMA 기반의 시스템에서 제공하는 서비스로 구현되어 있다고 가정한다. 이는 추가적인 비용 없이 P-IMA 기반의 이식이 가능함을 의미한다.
- P-IMA 기반의 이식을 통한 비행 소프트웨어 개발 비용 $Cost_P$, V-IMA 기반의 이식을 통한 비행 소프트웨어 개발 비용 $Cost_V$, 재개발을 통한 비행 소프트웨어 개발 비용 $Cost_R$에 대해서 다음과 같은 관계를 정의할 수 있다: $Cost_P \leq Cost_V < Cost_R$.

먼저, S_N은 IMA 아키텍처 기반으로 구축되므로 비행 소프트웨어에 대한 이식성을 제공한다. 따라서 이식이 가능한 상황이라면 항상 이식을 통한 개발 비용($Cost_P$, $Cost_V$)이 재개발을 통한 개발 비용($Cost_R$)보다 작다. 다음으로 $Cost_P$가 $Cost_V$보다 더 작은 이유는 V-IMA 기반의 이식을 달성하기 위해서는 이진코드 변환 요소를 제공하여야 하기 때문이다. 이진코드 변환 요소는 "변환 규칙"을 바탕으로 동작하며, 이는 환경에 따라 변경될 수 있다. 따라서 $Cost_P$가 $Cost_V$보다 더 작을 수밖에 없다. 이와 같은 가정을 바탕으로 H-IMA 아키텍처의 이식성을 평가한다. 이때, H-IMA 아키텍처 단독으로 평가가 불가하므로 비교 대상이 될 수 있는 시스템을 선정한다. 비교 대상 시스템은 Table 3과 같다.

Table 3. Target Systems for Portability Evaluation

ID	System	Partition Type
S1	H-IMA based System	Type1, 2, 3, 4
S2	AIR	Type2
S3	XtratuM	Type2, 4
S4	PikeOS	Type2
S5	V-IMA based System	Type1

Table 3의 S1, S2, S3, S4, S5는 이식성 평가를 위한 대상 시스템을 표현한다. S3과 S4에 해당하는 XtratuM[10]과 PikeOS[11]는 반가상화 기반의 솔루션이지만, 본 논문에서는 P-IMA의 예제로써 포함된다.

1) S_L의 비행 소프트웨어를 그대로 사용하는 경우

S_L의 비행 소프트웨어를 S_N에 탑재하기 위해서는 Table 4와 같은 탑재 방법을 사용하여야 한다.

Table 4. Methods for Porting (Scenario 1)

	F1	F2	F3	F4	F5	F6	F7	F8	F9
S1	V	V	V	P	P	P	P	P	P
S2	R	R	R	P	P	P	P	P	P
S3	R	R	R	P	P	P	P	P	P
S4	R	R	R	P	P	P	P	P	P
S5	V	V	V	V	V	V	V	V	V

※ P : P-IMA based Porting
※ V : V-IMA based Porting
※ R : Re-development

Table 4를 보면 S2, S3, S4의 경우에는 이진 소프트웨어에 대한 고려가 전혀 없기 때문에 이진 소프트웨어에 해당하는 F1, F2, F3에 대해서 재개발(R)을 해야 함을 알 수 있다. 특이한 것은 S5의 경우로, 어떤 유형의 소프트웨어든지 V-IMA 기반의 이식(V)을 통해 해결함을 알 수 있다. S1은 모든 유형의 파티션을 지원하므로 소프트웨어의 유형에 알맞은 방법으로 이식을 달성한다. Table 4를 통해 두 가지 사실을 알 수 있다. 첫 번째는 S1 기반의 시스템을 구축하는데 가장 적은 비용이 소요된다는 것이다. 두 번째는 S5가 S2, S3, S4에 비해 쉽게 이식성의 효과를 볼 수 있다는 점이다. 다시 말해서 비행 소프트웨어에 대해서 V-IMA가 P-IMA보다 많은 이식성의 기회를 제공해 준다는 것이다. 왜냐하면 V-IMA는 소프트웨어의 유형과 관계없이 이식성을 제공하지만, P-IMA는 이진 소프트웨어에 대해서 이식성을 제공할 수 없기 때문이다. 따라서 각 대상 시스템에서 소요되는 개발 비용을 기반으로 아래와 같이 이식성의 효과를 평가할 수 있다.

$$S1 > S5 > S2 = S3 = S4$$

2) S_L의 비행 소프트웨어 외에 추가되는 비행 소프트웨어가 존재하는 경우

이런 상황을 만들기 위해 F10, F11을 추가하며, 두 소프트웨어는 원본 소프트웨어에 해당한다. F10과 F11은 새로 만들어지는 소프트웨어이므로 어떤 시스템이든지 개발하는 방법을 선택하여야 한다. 따라서 F10과 F11은 이식성을 평가하는데 전혀 영향을 끼치지 못한다. 또한 기존 시스템의 비행 소프트웨어 역시 1)절의 상황과 동일할 수밖에 없다. 그러므로 이 절의 상황은 1)절의 상황과 동일한 결과를 내놓는다.

3) S_L의 비행 소프트웨어의 일부를 탑재하는 경우

이런 상황을 만들기 위해 F2, F5를 제거하여 탑재한다. F2는 이진 소프트웨어로, 이것이 제거되기 위해서는 하나의 이미지 파일이 쪼개져야 하는 상황이 발생한다. F5는 원본 소프트웨어로, 이것을 제외하는 것은 간단한 일이 된다. 이와 같은 상황에서 비행 소프트웨어를 S_N에 탑재하기 위해서는 Table 5와 같은 탑재 방법을 사용하여야 한다.

Table 5. Methods for Porting (Scenario 3)

	F1	F3	F4	F6	F7	F8	F9
S1	V	V	P	P	P	P	P
S2	R	R	P	P	P	P	P
S3	R	R	P	P	P	P	P
S4	R	R	P	P	P	P	P
S5	RV	RV	V	V	V	V	V

※ P : P-IMA based Porting
※ V : V-IMA based Porting
※ R : Re-development
※ RV : Re-development and V-IMA based Porting

Table 5를 보면 S2, S3, S4는 1)절의 상황과 크게 다르지 않음을 알 수 있다. 다만, F5가 제거되면서 F4가 단독으로 파티션에 탑재되는 상황이 발생한다. 이때, S2와 S4의 경우에는 유형2 파티션으로 탑재되지만, S3은 유형4 파티션으로 탑재된다. 사실 이는 개발 비용과 시간에는 영향을 거의 주지 않지만, 동작 과정에서의 효율성 측면에서 S3가 S2와 S4보다 더 나은 방법임을 알 수 있다. 반면, S5는 1)절의 상황과는 판이하게 다르다. 이진 소프트웨어임에도 불구하고 유형3 파티션을 지원하지 않기 때문에 재개발(R)되어야 하는 상황이 발생한 것이다. 게다가 재개발하더라도 이를 탑재할 수 있는 방법은 V-IMA 기반의 이식(V) 밖에 없기 때문에 새로 개발했음에도 불구하고 V-IMA 기반의 이식을 수행하여야 하는 단점이 존재한다. 이로 인해 S5는 $(Cost_R + Cost_V)$의 비용이 소요된다. S1의 경우에는 유형3 파티션을 통해서 쪼개어진 이진 소프트웨어를 탑재할 수 있다. 물론 기존 시스템과 다른 배포 형태를 갖게 되지만, 개발 비용과 시간 측면에서 바라보았을 때 그 목적을 달성할 수 있다. 따라서 이 절의 상황에서는 아래와 같은 순서로 이식성의 효과를 평가할 수 있다.

$$S1 > S3 \geq S2 = S4 > S5$$

4) H-IMA 아키텍처의 이식성 평가 결과

우리는 이식성 평가를 통해서 몇 가지 사실을 알 수 있다. 첫째, V-IMA는 어떠한 유형의 소프트웨어든지 쉽게 이식할 수 있는 방법을 제공한다. 이진 소프트웨어는 V-IMA를 통해서만 이식성을 제공할 수 있으며, 원본 소프트웨어는 컴파일을 통해서 이진 소프트웨어로 변환할 수 있으므로

이에 대해서 이식성을 제공할 수 있다. 이와 같은 사실은 V-IMA가 P-IMA에 비해서 쉽게 이식성의 효과를 얻어낼 수 있다는 것을 보여준다. 따라서 비행 소프트웨어의 이식성을 단기간 내에 달성하기 위해서는 V-IMA를 선택하는 편이 탁월함을 알 수 있다. 둘째, P-IMA는 V-IMA에 비해 비행 소프트웨어의 활용도를 높이는 방법을 제공한다. P-IMA는 원본 소프트웨어 기반의 이식성을 제공함으로써 비행 소프트웨어의 조합을 재구성할 수 있다는 장점을 갖는다. 물론 V-IMA 역시 원본 소프트웨어에 대해 재구성을 수행할 수 있지만, P-IMA 기반의 이식보다 더 높은 비용을 소요하는 V-IMA 기반의 이식을 따른다는 것이 문제가 될 수 있다. 따라서 장기적으로 P-IMA는 V-IMA가 가지지 못하는 재사용성을 제공할 수 있다는 장점을 가진다. 셋째, 본 논문에서 제안한 네 가지의 파티션 유형은 소프트웨어의 유형에 알맞은 이식성을 제공할 수 있도록 도와준다. 유형1과 유형2 파티션은 기존 솔루션에서도 제공하는 파티션 유형이지만, 유형3과 유형4 파티션은 흔히 볼 수 없는 형태의 파티션이다. 이러한 이유로 유형3과 유형4 파티션은 크게 효용성이 없는 파티션으로 보일 수 있다. 그러나 3) 절을 통해서 오히려 유형3과 유형4 파티션이 제공됨으로써 이식성의 효과뿐만 아니라 시스템의 효율성 측면에서도 유용함을 확인할 수 있다. 뿐만 아니라 세 가지 상황에 대해서 가장 낮은 개발 비용이 소요되는 시스템은 H-IMA 기반의 시스템임을 확인할 수 있다.

4. H-IMA 기반의 시스템 구현

본 논문에서는 H-IMA에 대한 검증을 위해 H-IMA를 기반으로 시스템을 구현한다. 이때, 구현하는 시스템은 (1)자원 관리 계층과 파티션 관리 계층이 분리되어 구현되어야 하며, (2)단일 시스템 내에 다양한 소프트웨어 유형을 탑재할 수 있어야 한다.

H-IMA 기반의 시스템을 구축하기 위해서는 자원 관리 계층, 파티션 관리 계층, 네 가지 유형의 파티션을 지원하기 위한 요소를 구현하여야 한다. 이 중에서 파티션 지원 요소는 파티션의 유형에 따라 요구되는 기능이 상이하여 한 번에 개발하는 것은 무리가 있다. 또한 네 가지 유형의 파티션 중 하나만을 지원하더라도 시스템이 동작하는데 아무런 문제가 없다. 따라서 본 논문에서는 파티션 유형에 대해서 점진적 개발을 수행한다.

4.1 H-IMA 기반의 시스템 구축 과정

H-IMA를 기반으로 시스템을 구축하는 과정에서 P-IMA 측면에서의 시스템 구축은 AIR[12]를 통해 그 가능성을 입증하였다. 또한, XtratuM과 PikeOS는 반가상화 기법을 바탕으로 한 솔루션이지만 그 형태는 P-IMA와 유사성을 보인다. 따라서 P-IMA에 해당하는 유형2와 유형4 파티션을 H-IMA에 적용하는 것이 가능함을 보여준다. 그러나 V-IMA의 경우는 실제 그 솔루션이 존재하지 않는다. 전가상화 기법을 바탕으로 한 솔루션의 예로는 VMware가 존재하지만, 이것은 임베디드 시스템을 대상으로 하지 않기 때문에 적합한 예가 될 수 없다. 즉, V-IMA에 해당하는 유형1과 유형3 파티션의 적용 가능성을 증명할 수 있는 방법이 존재하지 않는다. 따라서 유형1과 유형3 파티션을 적용하는 것이 다양한 유형의 소프트웨어를 탑재할 수 있음을 증명하는데 반드시 필요하다. 따라서 이번 절에서는 자원 및 파티션 관리 계층과 유형1 파티션 지원 요소에 대한 설계 및 구현에 대해서 기술한다. 본 논문에서 개발한 유형1 파티션을 지원하는 시스템의 아키텍처[13]는 Fig. 6과 같다.

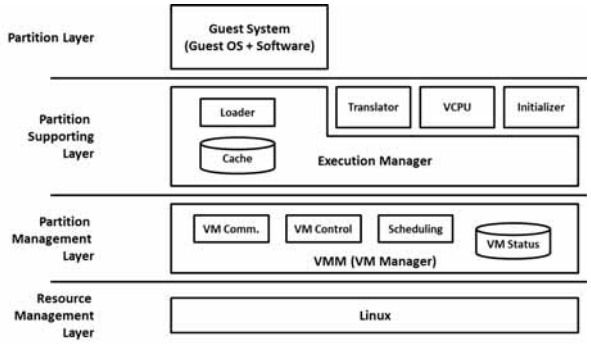

Fig. 6. H-IMA based System for Partition type1

1) 자원 관리 계층

자원 관리 계층(Resource Management Layer)은 시간 자원(예: CPU 등)과 공간 자원(예: 메모리, 통신 채널 등)에 대해 접근할 수 있는 API를 제공한다. 흔히 알고 있는 운영체제의 기능 중 일부를 담당하는 계층이다. H-IMA의 자원 관리 계층은 실제로 구현하지 않고, COTS(Commercial off-the-shelf)로 대체될 수 있다. 물론 인공위성에 탑재될 때에는 마이크로 커널(Micro-kernel) 형태로 재구성되어야 한다.

2) 파티션 관리 계층

파티션 관리 계층(Partition Management Layer)은 파티션을 관리하는데 필요한 기능을 제공한다. 주로 파티션의 생성 및 제어를 담당하며, 추가적으로 파티션에 대한 설정값을 관리하는 역할도 수행한다. 자원 관리 계층과 달리 파티션 관리 계층은 대체할 수 있는 COTS가 없기 때문에 해당 부분은 구현한다. Fig. 6의 VMM(Virtual Machine Manager)이 파티션 관리 계층에 해당한다. VMM의 기능은 (1)VM 자료구조 관리, (2)VM 제어, (3)VM과의 통신이다. 이와 같은 기능을 수행하기 위해서 Table 6과 같은 API를 정의한다.

VM 자료구조 관리 기능은 VM의 생성을 담당하는 기능이다. 일반적인 프로세스와 같이 VM를 관리하기 위해서는 VM에 대한 정보를 담는 자료구조가 필요하다. 따라서 VM을 생성하는 과정에서 VM 자료구조를 생성하고 초기화하는 기능이 필요하다. 이때 사용되는 API는 *create_vm* 함수이다. VM 제어 기능은 말 그대로 VM를 제어하는 기능이다. VMM은 VM을 시작/정지/재시작/종료할 수 있으며, 이와 같은 기능은 *ctrl_vm* 함수를 통해 이루어진다. 마지막으로 VM과의 통신 기능은 VM 자료구조 관리 기능과 VM 제어 기능을 수행하기 위한 기능이다.

Table 6. API for VMM

Return	API Name	Parameters
int	init_vmmanager	void
int	exit_vmmanager	void
BOOL	create_vm	BYTE4 vid
		BYTE4 pid
		char* path
BOOL	ctrl_vm	BYTE4 vid
		BYTE4 cmd
int	vmmanager_ioctl	inode* inode
		file* flip
		BYTE4 cmd
		long data

3) 이진코드 변환 요소

이진코드 변환 요소는 파티션 지원 계층(Partition Supporting Layer)에 속하는 요소로, 크게 변환기(Translator), 가상 CPU(Virtual CPU), 서비스 제공자(Service Provider)로 구성된다. 변환기는 변환 규칙을 기반으로 작성되고, 가상 CPU는 특정 프로세서에 특화된 요소를 해결하기 위해 사용되며, 주로 특권 명령어(Privileged Instruction)에 대한 처리를 위해 사용된다. 이와 같은 요소를 고려하여 실제 구현된 결과는 Fig. 7과 같다.

서비스 제공자는 실행 관리자(Execution Manager)를 통해 구현된다. 서비스 제공자는 단순히 파티션과 자원 관리 계층 사이의 인터페이스 역할을 수행하는 요소였으나, 구현 과정에서 이진코드 변환을 명령하는 요소가 필요하여 실행 관리자라는 요소를 구현하였다. 파티션에서 파티션 관리 계층이나 자원 관리 계층에 접근하는 시점을 파악하기 위해서는 우선적으로 이진코드의 변환이 발생하여야 하며, 이진코드 변환의 제어를 파티션 관리 계층에서 담당하는 것은 H-IMA의 목적에 어긋나기 때문에 서비스 제공자를 확장하여 구성하였다. Fig. 7을 보면 변환기, 가상 CPU와 같은 요소가 실행 관리자와 분리되어 있다. 변환기와 가상 CPU 요소는 탑재되는 소프트웨어에 따라 변경되어야 하지만, 실행 관리자는 탑재 소프트웨어와 무관하게 항상 동일한 기능을 수행한다. 따라서 이진코드 변환 요소의 유동성을 위해 Fig. 7과 같이 구성한다.

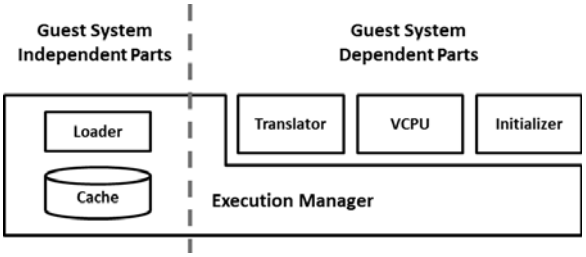

Fig. 7. Binary Code Translation Component

a) 실행 관리자(Execution Manager)

게스트 시스템, 즉 게스트 운영체제 및 소프트웨어를 실행시키는 모듈이다. 게스트 시스템의 적재와 초기화, 그리고 변환을 수행한다. 여기서 초기화와 변환에 대한 기능은 게스트 시스템의 구성에 따라 변경되는 부분이기 때문에 초기화 관리자와 변환기의 API를 호출하여 기능을 수행한다. 그 외에도 가상 CPU와 연동하여 예외 처리와 하드웨어 처리를 수행한다. 실행 관리자가 제공하는 API는 Table 7과 같다.

Table 7. API for Execution Manager

Return	API Name	Parameters
void	tb_exec	VCPU *vcpu
		BYTE4 addr
void	save_condition_codes	int *n
		int *z
		int *v
		int *c
void	restore_condition_codes	int n
		int z
		int v
		int c
BYTE4	sparc7_sensitive_handler	VCPU *vcpu
		int instr
void	write_sysreg	BYTE4 addr
		BYTE4 value
BYTE4	read_sysreg	BYTE4 addr
BYTE4	load_file_srec	BYTE1 *filepath
		VM *vm
void	cpu_init	VCPU *vcpu

b) 초기화 관리자(Initializer)

게스트 시스템과 관련된 자료구조를 초기화하는 모듈이다. 초기화 관리자를 통해 초기화되는 자료구조는 VM, 가상 CPU이다. 여기서 초기화란, 게스트 시스템이 동작할 수 있도록 초기 상태로 설정하는 것을 의미한다. 초기화 관리자가 제공하는 API는 Table 8과 같다. 본 논문에서 사용한 게스트 시스템이 SPARC v7의 ERC32 프로세서를 기반으로 하기 때문에 Table 8과 같은 API를 구현한다.

Table 8. API for Initializer

Return	API Name	Parameters
void	sparc7_init	VCPU *vcpu
void	erc32_init	VM *vm

c) 변환기(Translator)

VM에 탑재된 게스트 이미지의 이진코드를 실제 시스템에서 동작할 수 있는 이진코드로 변환하는 모듈이다. 명령어의 해석 이후에 명령어의 변환이 수행된다. 변환기가 제공하는 API는 Table 9와 같다.

Table 9. API for Translator

Return	API Name	Parameters
void	translate	VM *vm
		BYTE4 addr
void	decode	VM *vm
		BYTE4 addr
BYTE4	unused_reg	BYTE4 rds
		BYTE4 unused_regs

d) 가상 CPU (Virtual CPU)

게스트 시스템이 갖는 CPU에 대한 자료구조이다. 게스트 시스템에 대한 하드웨어 처리를 위해 정의된 요소이며, 하드웨어 처리를 가상 동작하도록 만든다. Table 10은 해당 시스템에서 사용하는 가상 CPU 자료구조로, ERC32 프로세서가 갖는 레지스터를 표현한다.

Table 10. Data Structure of VCPU

Field	Type	Description
gregs	BYTE4[]	Global Register
iregs	BYTE4[]	General purpose registers
fregs	BYTE4[]	Floating point registers
psr	BYTE4	processor state register
win	BYTE4	Window invalid mask register
tbr	BYTE4	Trap base register
y	BYTE4	Y register
pc	BYTE4	Program counter
npc	BYTE4	next program counter
regwptr	BYTE4 *	Point to current window
psret	BYTE4	Enable trap bit of PSR
psrs	BYTE4	Supervisor bit of PSR
psrps	BYTE4	Previous supervisor bit of PSR
psrpil	BYTE4	Processor interrupt level bit of PSR
instr	BYTE4	Instruction
delayed	BYTE4	Delayed Instruction
is_broken	BOOL	Is branch broken?
psref	BYTE4	Enable floating point unit bit of PSR
psrn	BOOL	Negative bit of PSR
psrz	BOOL	Zero bit of PSR
psrv	BOOL	Overflow bit of PSR
psrc	BOOL	Carry bit of PSR
trap	BYTE4	Is trap generated?

4.2 H-IMA 기반의 시스템 동작 결과

구현된 시스템에 세 가지의 비행 소프트웨어를 탑재하여 동작시켰다. 탑재된 비행 소프트웨어는 KOMSAT(아리랑 위성)에서 사용된 것으로, Table 11과 같이 구성된다.

Table 11. FSW on KOMSAT

FSW	Period	Description
tONE	120 ms	• Flight SW Management • Request and Obtainment of Information about Devices
tTWO	120 ms	• Send Commands to Devices • Telemetry
tBACK	x	• Long-term Task

Table 11의 비행 소프트웨어의 우선순위는 tONE = tTWO > tBACK이며, 높은 우선순위를 가질수록 스케줄링에서 우선권을 갖는다. 시스템에서는 60ms 단위로 타이머 인터럽트가 발생되며, tONE과 tTWO는 타이머 인터럽트를 통해 자신이 동작하여야 하는 주기가 돌아오면 실행된다. tBACK은 tONE과 tTWO가 모두 대기 상태가 되면 실행된다. 즉, tBACK은 동작하다가 타이머 인터럽트가 발생되면 tONE이나 tTWO에게 CPU를 넘겨준다. Table 11의 비행 소프트웨어는 위와 같은 방식으로 동작하며, 실제 구현된 시스템에서도 동일한 동작을 수행하는지 확인하였다. 그 실행 결과는 Fig. 8과 같다.

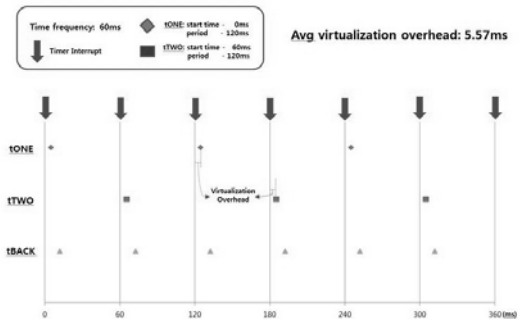

Fig. 8. FSW Execution Result

Fig. 8은 각 FSW의 시작 시간을 측정한 것을 도식화한 것이다. 이를 보면 설정되어 있는 tONE의 시작 시간은 0ms이고, tTWO의 시작 시간은 60ms이지만, 실제 시작 시간은 설정된 값보다 지체된 것을 볼 수 있다. 이는 가상화로 인해 발생하는 오버헤드(Overhead)로 인한 것이다. 시작 시간은 조금 지체되었으나, FSW가 갖는 본래 스케줄 흐름은 동일하다. tONE과 tTWO는 120ms의 주기마다 실행되며, tBACK은 tONE과 tTWO과 작업을 완료한 이후, 즉 자신 외의 비행 소프트웨어가 모두 대기 상태가 되었을 때 실행된다.

본 논문에서 구현한 H-IMA 기반의 시스템의 가능성, 즉 유형1 파티션의 가능성은 기존에 존재하는 비행 소프트웨어를 그대로 탑재하였을 때, 탑재된 비행 소프트웨어가 기존

에 동작하던 방식대로 동작하는지를 통해 확인된다. Fig. 8의 결과를 보면 기존의 스케줄 흐름을 그대로 따르고 있으므로, 본 논문에서 구현한 H-IMA 기반의 시스템의 가능성을 확인할 수 있다.

5. 결 론

IMA 아키텍처는 비행 소프트웨어가 갖는 고신뢰성, 실시간성, 그리고 환경-종속성에 대한 요구사항을 충족하는 아키텍처이다. 특히, IMA 아키텍처는 이식성을 제공함으로써 환경-종속성에 대한 문제를 해결한다. 기존의 IMA 아키텍처 기반의 솔루션들은 가상화 기술이나 파티셔닝 기법을 이용하여 IMA 아키텍처를 실현하였다. 본 논문에서는 IMA 아키텍처의 이식성이 실현되는 기술에 따라 서로 다른 특성을 띄는 것에 주목하고, 이를 모두 제공할 수 있는 H-IMA를 제시하였다. H-IMA에서는 각 솔루션에서 제공하는 이식성을 분석하고 이를 네 가지 유형의 파티션으로 분류하여 제공한다. 이와 같이 다양한 파티션을 제공함으로써, 다양한 상황에 따라 다양한 유형의 소프트웨어를 탑재할 수 있을 뿐만 아니라 시스템을 구축하는데 소요되는 시간을 줄일 수 있다. 본 논문에서는 제안한 H-IMA에 대한 검증을 위해 두 가지 방법을 사용하였다. 첫 번째 방법은 기존의 P-IMA와 V-IMA 기반의 시스템과 비교하였을 때, 더 적은 개발 비용을 소요하는지 확인하는 방법이며, 두 번째 방법은 유형1 파티션을 대상으로 하는 H-IMA 기반의 시스템을 구현하는 방법이다. 첫 번째 방법을 통해서 P-IMA와 V-IMA의 이식성을 모두 제공하는 것만으로도 개발 비용의 절감이 존재함을 확인하였으며, 두 번째 방법을 통해서 기존의 비행 소프트웨어를 구현된 시스템에 그대로 탑재하여 동작시켜 이식이 가능함을 보였다. 이를 통해서 H-IMA 기반의 시스템에 대한 가능성을 확인할 수 있다. 추후에는 유형1 파티션 외의 나머지 파티션에 대해서 지원할 수 있도록 파티션 지원 계층에 대한 구현을 완료하고, 다양한 파티션을 기반으로 한 시스템 구축 절차에 대해서도 수립할 예정이다.

참 고 문 헌

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

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

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

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

[10] M. Masmano, I. Ripoll, A. Crespo, “XtratuM Hypervisor for LEON3-usermanual”, 2011.

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

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

[13] H. Joe, H. Jeong, Y. Yoon, H. Kim, S. Han, H. W. Jin, “Full virtualizing micro hypervisor for spacecraft flight computer”, 31th Digital Avionics Systems Conference (DASC), 2012.

서 용 진

e-mail : yjseo082@cnu.ac.kr

2011년 충남대학교 컴퓨터공학과(학사)

2011년～현 재 충남대학교 컴퓨터공학과 박사과정, 석박사통합

관심분야 : 소프트웨어 테스팅, 스마트폰, UX/UI

윤 상 필

e-mail : sambitious@cnu.ac.kr

2012년 충남대학교 컴퓨터공학과(학사)

2012년～현 재 충남대학교 컴퓨터공학과 석사과정

관심분야 : 소프트웨어 테스팅, 분산 컴퓨팅 개발 및 검증

조 현 우

e-mail : jhwzero@cnu.ac.kr

2005년 충남대학교 컴퓨터공학과(학사)

2007년 충남대학교 컴퓨터공학과(석사)

2008년～현 재 충남대학교 컴퓨터공학과 박사과정

관심분야 : 임베디드 시스템 가상화, 항공우주 시스템

권 철 순

e-mail : cskwon@cnu.ac.kr
2013년 충남대학교 컴퓨터공학과(학사)
2013년~현 재 충남대학교 컴퓨터공학과 석사과정
관심분야 : 임베디드 시스템 가상화, 항공우주 시스템

김 형 신

e-mail : hyungshin@cnu.ac.kr
1990년 한국과학기술원 전산학과 (학사)
1991년 Univ. of Surrey 위성통신학과 (석사)
1992년~2001년 한국과학기술원 인공위성 연구센터 선임연구원
2003 한국과학기술원 전산학과(박사)
2003년~2004년 Carnegie Mellon University Post Doc.
2004년~현 재 충남대학교 컴퓨터공학과 교수
관심분야 : 항공우주 시스템, 저전력 컴퓨팅, 임베디드 시스템 소프트웨어

김 현 수

e-mail : hskim401@cnu.ac.kr
1988년 서울대학교 계산통계학과(학사)
1991년 한국과학기술원 전산학과(공학석사)
1995년 한국과학기술원 전산학과(공학박사)
1995년~1995년 한국전자통신연구원 Post Doc.
1996년~2001년 금오공과대학교 조교수
1999년~2000년 Colorado State University 방문교수
2007년~2008년 Purdue University 방문연구교수
2001년~현 재 충남대학교 컴퓨터공학과 교수
관심분야 : 소프트웨어 공학, 소프트웨어 테스팅, SOA