개방형 시스템 아키텍처 기반의 소형 민간 무인항공기 시스템 개발 및 검증 방법

[1*]조현철, [2] 박근영

Development and Verification Methodology for Small Civil Unmanned Aerial Vehicle System based on Open System Architecture

[1*] Hyun-Chul Jo, [2] Keunyoung Park

요 약

무인 항공기 시스템은 주로 군사용 위주로 운용되어 왔으나, 민간 분야에서도 활발히 이용되고 있다. 민간 분야에서는 주로 저비용의 소형 무인 항공기 시스템을 이용하여 다양한 산업분야에서 이용되고 있다. 이러한 무인 항공기 시스템에서 운용되는 소프트웨어는 공통적인 기능들이 많이 존재한다. 하지만 이러한 특성에도 불구하고 하드웨어 장치가 변경되면 소프트웨어를 수정해야 하는 문제가 발생할 수 있다. 이러한 문제는 무인 항공기 시스템에서 상호 운용성, 모듈성, 이식성을 저하시킨다. 상기 문제들을 해결하기 위해 개방형 시스템 아키텍처가 제안되었으며, 본 논문에서는 다양한 개방형 시스템 아키텍처들 중 FACE(Future Airborne Capability Environment)를 기반으로 동작하는 무인 항공기 시스템 소프트웨어 구조를 제안한다. 본 논문에서 제안하는 소형 무인항공기 시스템은 소량 다품종으로 공급되는 민간 분야에서 다양한 플랫폼의 무인 항공기 시스템을 지원할 수 있으며, 소프트웨어에 대한 통합과 이식성이 뛰어난 장점을 가진다. 마지막으로 본 논문에서는 공개된 개발도구를 기반으로 하는 FACE 기반 무인 항공기 시스템 소프트웨어 개발 방법과 적합성 테스트에 대해 서술한다.

Abstract

The Unmanned Aerial Vehicle(UAV) system has been mainly used for military domains, but it also widely applied to used in the civilian domains. In civilian domains, low-cost and small-sized UAV systems are mainly applied in various industries. The software that operates UAV systems has a lot of common functions. However, even though there are many common functionalities of the software, changing the devices may cause a problem requiring software modification. These problems degrade interoperability, modularity and portability in UAV systems. In order to solve the problems, an Open System Architecture(OSA) has been proposed. In this paper, we propose a UAV system software architecture based on Future Airborne Capability Environment(FACE) standard. Our system can support UAV systems of various platforms in the civilian domains, which is supplied in small quantity batch production. And it has the advantages of software consolidation and portability. Finally, We describe the development and conformant methodology of the software based on the FACE standard using open development tools.

Keywords: Small Civil Unmanned Arial Vehicle, Ground Control System, Open System Architecture, Future Airborne Capability Environment, Conformance Test, Framework

[1]교신저자 한화시스템 항공연구센터 전문연구원 (hcjo47@hanwha.com)*

[2] 한화시스템 항공연구센터 전문연구원 (kpark12@hanwha.com)

Received: May. 12, 2020, Revised: June. 09, 2020, Accepted: June. 29, 2020

I. 서론

무인 항공기(UAV, Unmanned Aerial Vehicle) 시스템은 무인 항공기와 지상관제 시스템(GCS, Ground Control System)이 함께 운용된다[1]. 이러한 무인 항공기 시스템은 주로 감시 및 정찰 용도의 군사용 위주로 운용되어 왔으나, 최근 하드웨어와 소프트웨어 기술을 바탕으로 민간 분야로 영역을 넓혀가고 있다[2]. 민간 분야에서는 주로 저비용의 소형 무인 항공기 시스템을 이용하여 인프라 관리, 농업 방제, 엔터테인먼트 등 다양한 분야에서 이용되고 있다[3]. 민간 분야의 소형 무인 항공기 시스템은 군용 항공기에 비해 소프트웨어와 하드웨어를 포함한 상용제품(COTS, Commercial Off-the-Shelf)을 사용하고 있으며, 이 제품들은 소프트웨어 기능에 대한 유사성이 많다.

무인 항공기(UAV)에서 운용되는 소프트웨어[4]는 기체를 제어하기 위한 제어 소프트웨어와 정찰 임무를 수행하기 위한 스트리밍 소프트웨어로 구성되며 많은 무인 항공기에 탑재되어 운용되고 있다. 이러한 소프트웨어들은 유사한 기능을 하는 경우가 대부분이다. 지상관제 시스템(GCS)[5]에서는 디지털 지도 소프트웨어, 무인 항공기의 비행 경로(Waypoint)를 설정하여 비행을 설정하는 자동 비행(Autopilot) 소프트웨어와 항공기의 자세, 고도, 속도 등의 정보를 사용자에게 제공하는 PFD(Primary Flight Display) 등으로 구성되며, 이들 또한 소프트웨어 기능에 대한 공통성이 많다. 하지만 소프트웨어의 공통적인 기능이 많음에도 불구하고 하드웨어가 변경되면 장치를 지원하기 위한 디바이스 드라이버나 라이브러리 등을 같이 변경해야 하기 때문에 결론적으로 소프트웨어까지 변경해야 하는 문제점이 발생한다. 이는 개발 비용 및 개발 시간이 증가하는 문제점을 초래하게 된다. 이러한 문제점은 결국 무인 항공기 시스템에서 상호 운용성(Interoperability), 모듈성(Modularity) 및 이식성(Portability)을 저하시킨다.

앞서 언급한 문제를 해결하기 위해 미 국방부(US Department of Defense)에서는 항공 시스템 개발의 비용 절감 및 효율성 향상을 목적으로 군 시스템에 개방형 시스템 아키텍처(OSA, Open System Architecture)를 적용하기 위한 전략들을 추진하였다[6]. 또한 특정 공급 업체의 의존성을 피하고 저비용 장비의 사용 역시 촉진하고 있다. 개방형 시스템 아키텍처에서는 인터페이스 요구사항을 정의하고 해당 인터페이스에 대한 응용소프트웨어와 기반 구조를 분리한다[7]. 이러한 흐름에 따라 군용 무인 항공기에 적용 가능한 개방형 시스템 아키텍처들이 제안되었다[8-10]. 다양한 개방형 시스템 아키텍처들 중 FACE(Future Airborne Capability Environment)[10]는 군사용 표준만이 아닌 POSIX 나 OpenGL 과 같은 민간 분야에서도 많이 사용되는 표준들을 기반으로 소프트웨어 공통화를 위한 가이드라인을 정의하였다. 또한 공개 개발 도구들이 다수 존재하여 민간분야에서도 쉽게 무인 항공 시스템을 개발할 수 있다. 따라서 FACE 를 UAV 와 GCS 를 포함한 민간 소형 무인 항공기 시스템에 활용하기에 충분한 가능성을 보인다. 본 논문에서는 개방형 시스템 아키텍처인 FACE 를 소형 무인 항공기 시스템에 적용한 소프트웨어 구조를 제안한다. 추가적으로 공개 개발 도구를 이용한 FACE 기반의 소프트웨어 개발 방법과 적합성 테스트 방법에 대해서 서술한다. 결론적으로 본 논문은 개방형 시스템 아키텍처인 FACE 기반 소프트웨어를 구성할 때 좋은 참조가 될 것이다.

II. 관련연구

항공 분야에서 개방형 시스템 아키텍처에 대한 연구가 군/민간 영역에서 다양하게 연구되고 있다. STANAG-4586[8]에서는 무인 항공기 및 여러 탑재 장치들과의 통신을 위한 메시지 인터페이스를 정의하여 상호운용성을 보장할 수 있도록 하였다. OMS[9]에서는 주요 인터페이스와 모듈에 대해 서비스 기반 구조의 디자인 패턴 및 원칙을 제시하여, 서비스 제공자와 서비스 이용자의 측면으로 항공 시스템의 기능들을 이용하도록 하였다. FACE[9]에서는 시스템의 기능적 분할이 가능한 지점들에 대해 각 기능들을 세그먼트로 분할한다. 이러한 FACE 표준을 준수할 경우 하드웨어와 소프트웨어의 분할된 세그먼트 형태를

유지할 수 있어 상호 운용성, 모듈성 및 이식성을 개선할 수 있다. 이 연구들은 주로 군에서 운용되는 항공 전자 시스템이나 군용 무인항공기를 대상으로 하고 있다.

민간 무인 항공기 분야에서는 주로 개방형 시스템 아키텍처를 서비스 측면에서 다루고 있다. E. Pastor 의 연구[11]에서는 무인 항공기에서 SOA(Service Oriented Architecture) 기반 통신을 적용한 USAL(UAV Service Abstraction Layer)을 제안하였다. 이 USAL 을 이용할 경우 무인 항공기가 제공하는 정보를 서비스화 할 수 있으며 이로 인해 정보제공 측면에서 상호운용성, 유연성, 확장성을 제공한다. K. Park 의 연구[12]에서는 REST 구조 기반의 무인 항공기 서비스 구조를 제안하며, 무인 항공기가 제공하는 위치정보와 영상정보를 수신하고 이를 웹서비스를 통해 제 3 자에게 서비스 할 수 있도록 하였다. W.Muller 의 연구[13]에서는 개방형 구조 기반의 센서 네트워크 시스템을 제안하여 UAV 의 대응행동을 위한 운용자의 의사결정을 지원할 수 있도록 하였다. 이들 연구는 UAV 와 GCS 사이에 주고받는 통신 관점에서의 개방형 시스템 아키텍처를 다루고 있다. 또한 민간 소형 무인 항공기 분야에서는 개방형 시스템 구조 중 하나인 IMA(Integrated Modular Architecture)를 적용하여 소프트웨어에 대한 이식성 및 모듈성을 증가시키는 방법들도 연구되었다. H. Jo[14]의 연구에서는 소형 민간 무인항공기에서 IMA 기반 OSAL(Operating System Abstraction Layer)을 적용함으로써 OS 상위에 동작하는 소프트웨어에 대해 이식성 및 모듈성을 향상시키는 방법이 제안 되었다. 이 연구에서는 UAV 에 탑재되는 운영체제 소프트웨어에 한정적으로 개방형 시스템 아키텍처를 적용하였으며, GCS 에 대해서는 적용되지 않았다.

본 연구에서는 군용 항공시스템에 주로 적용되었던 FACE 를 소형 무인 항공시스템에 적용할 수 있는 소프트웨어 구조를 제안한다. 본 연구에서 제안하는 FACE 기반 소형 무인항공기 시스템은 소량 다품종으로 공급되는 민간 분야에서 다양한 무인 항공 시스템을 지원하고 소프트웨어를 쉽게 이식할 수 있게 하는 장점을 가진다.

III. 개방형 시스템 아키텍처 기반 소형 무인항공기 시스템

본 논문에서는 상호 운용성, 모듈성 및 이식성을 제공하기 위해 개방형 시스템 아키텍처인 FACE 를 소형 무인항공기 시스템에 적용한 소프트웨어 구조를 제안한다. 먼저 3.1 장에서는 FACE 소프트웨어 구조를 설명하고, 3.2 장에서는 소형 무인 항공기 시스템인 UAV 와 GCS 에 FACE 를 적용한 소프트웨어 구조를 설명한다. 3.3 장에서는 FACE 기반 민간 소형 무인 항공기 시스템의 성능적 관점의 이슈에 대해 설명한다.

3.1 Future Airborne Capability Environment

FACE 는 항공 소프트웨어 분야의 개방형 시스템 아키텍처로써 그림 1 과 같은 FACE 참조 아키텍처(Reference Architecture)를 정의하여 소프트웨어의 이식성을 향상시키기 위한 설계 원칙을 준수하도록 권장하고 있다. 이를 통해 공통으로 사용되는 소프트웨어에 대하여 상호 운용성, 모듈성 및 이식성에 대한 문제를 해결하고 있다. FACE 참조 아키텍처는 표준화된 인터페이스와 데이터 모델을 정의하여 서로 다른 공급 업체(Vendor)에서 개발한 소프트웨어 구성 요소 사이에 이동 및 통합을 쉽게 할 수 있다.

FACE 참조 아키텍처는 OSS(Operating System Segment), IOSS(Input/Output Service Segment), PSSS(Platform-Specific Services Segment), TSS(Transport Services Segment) 및 PCS(Portable Components Segment)의 5 개 세그먼트로 구성되며 각 세그먼트들은 분산적으로 관리된다. OSS 는 FACE 세그먼트가 실행될 수 있는 컴퓨팅 환경을 제공하고, 다양한 운영체제 기능과 운영체제 레벨의 헬스모니터를 지원한다.

IOSS 는 공급업체에서 제공하는 하드웨어에 대한 디바이스 드라이버가 존재하는 세그먼트로써 상위 세그먼트(PSSS)가 I/O 장치에 대해 독립적일 수 있도록 지원한다. 이를 위해 IOSS 는 비표준 API 에 대해 디바이스 드라이버에서 제공하는 데이터를 IO 메시지 모델로 변환하는 역할을 수행한다.

PSSS 는 플랫폼 의존적인 부분을 담당하며, PCS 에게 장치와 관련된 데이터를 제공하고, 표준화된 인터페이스를 통해 상위 세그먼트에게 플랫폼에 대한 독립성을 제공한다. 또한 PSSS 는 플랫폼 장치 서비스(Platform-Specific Device Service), 공통 서비스(Platform-Specific Common Service), 그래픽 서비스(Platform-Specific Graphic Service)를 담당한다. 플랫폼 장치 서비스는 플랫폼에 특화된 ICD(Interface Control Document)와 FACE 데이터 모델 사이의 변환을 담당한다. 플랫폼 공통 서비스는 Configuration 이나 Logging 과 같은 공통적으로 이용되는 서비스를 담당한다. 그리고 그래픽 서비스는 GPU 및 기타 그래픽 장치의 인터페이스를 추상화 하는 기능을 담당한다.

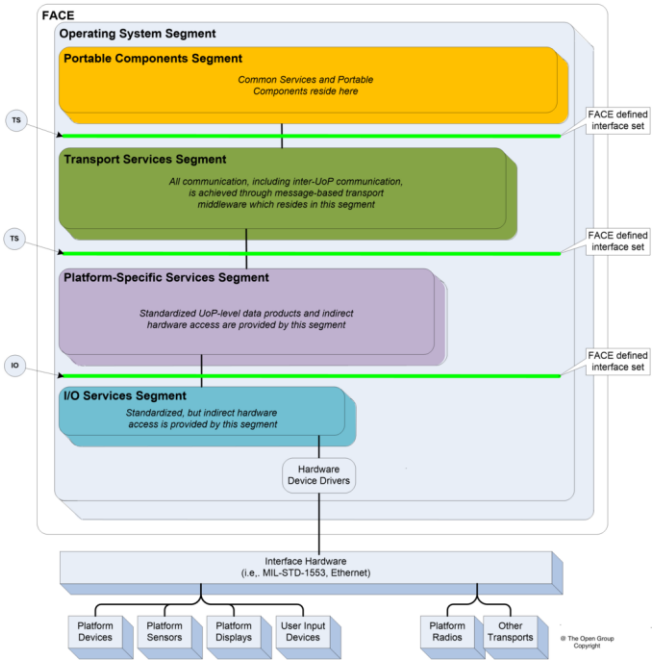

Figure 1. FACE Reference Architecture[10]

TSS 는 PCS 내 컴포넌트와 PSSS 컴포넌트 사이의 데이터 배포를 담당한다. TSS 에서는 데이터 전송 방식에 대한 의존성을 제거하여 소프트웨어 개발자가 데이터 변환 또는 데이터 전송에 관한 세부사항을 고려하지 않아도 되도록 지원한다.

PCS 는 FACE 에서 가장 플랫폼 독립적인 응용프로그램을 나타내는 세그먼트이다. FACE 에서는 PCS 를 장치 및 운영체제에 독립적인 소프트웨어를 배치함으로써 상호 운용성, 이식성 및 모듈성이 가장 두드러지는 세그먼트이다.

FACE 아키텍처 세그먼트 간 연결은 표준화된 인터페이스를 이용한다. FACE 의 인터페이스는 OSSI(OS Segment Interface), IOSI(I/O Service Interface), TSI(TS Interface)로 구성된다. OSSI 는 소프트웨어가 운영체제 내의 서비스 및 OSS 와 관련된 기타 기능을 활용할 수 있도록 POSIX 와 ARINC 653[15] 및 HMFM(Health Monitoring and Fault Management) API 가 포함한 표준화된 수단을 제공한다. IOSI 는 소프트웨어가 디바이스 드라이버와 통신할 수 있도록 표준화된 수단을 제공한다. 또한 IOSI 는 IOSS 내 소프트웨어에서 PSSS 의 소프트웨어로 제공되며 IO 메시지 모델을 기반으로 통신한다. TSI 는 소프트웨어가 통신 서비스를 활용하기 위한 표준화된 수단을 제공한다. TSI 에서는 FACE 데이터 모델을 사용하여 PSSS 와 PCS 사이의 통신을 담당한다.

FACE 는 UoC(Unit of Conformance)와 UoP(Unit of Portability)라는 컴포넌트 개념을 정의한다. UoC 는 단일 FACE 세그먼트의 적합성 요구사항을 충족시키기 위해 개발된 소프트웨어 구성요소로써 FACE 의 5 개 세그먼트에 포함된다. UoP 는 하나 이상의 기능을 실행하는데

필요한 서비스를 제공하는 소프트웨어 세트로 PCS, TSS, PSSS 에 포함되며, 이를 포함하는 세그먼트는 이식성 및 재사용성이 높은 세그먼트이다.

3.2 FACE 기반 민간 소형 항공기 시스템 구조

본 논문에서 대상으로 하고 있는 민간 소형 항공기 시스템은 오픈소스기반의 ArduPilot 커뮤니티[16]로부터 참조하였다. UAV 의 소프트웨어는 ArduPilot 을 참조하였으며, GCS 는 Mission Planner[17]를 참조하였다. UAV 와 GCS 은 MAVLink 을 통해 통신을 수행한다. 본 논문에서는 UAV 와 GCS 의 소프트웨어를 대상으로 하기 때문에 OSS 는 제외하였다.

3.2.1 FACE 기반 무인 항공기 구조

본 절에서는 비행제어, 네비게이션, 항공기 상태 알람, 데이터 로깅 임무를 수행하는 FACE 기반 무인 항공기 구조를 제안한다. 그림 2 는 FACE 기반 소형 무인 항공기 구조를 나타낸다.

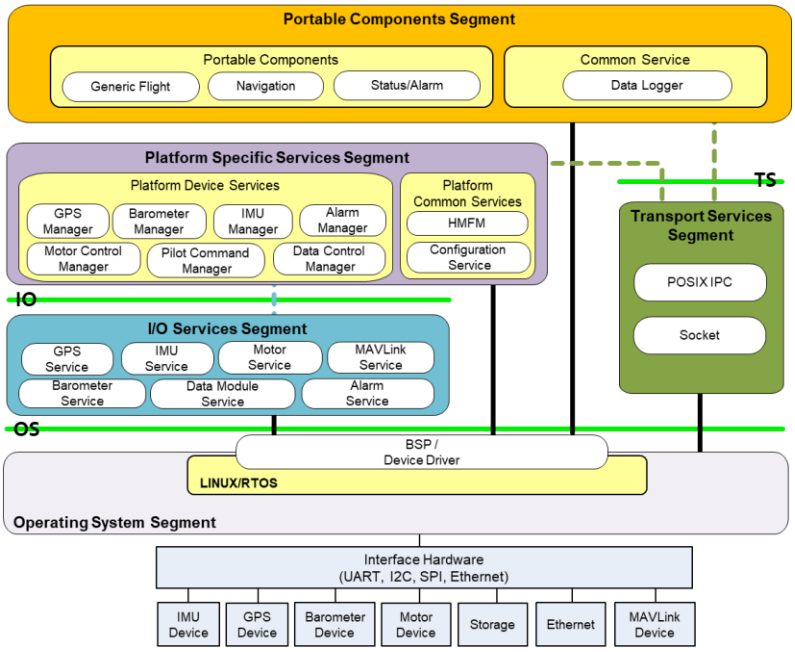

Figure 2. Small UAV Software Architecture

● **비행제어 소프트웨어**

비행제어 소프트웨어는 IMU, Barometer 센서와 MAVLink 장치를 통해 수신된 GCS 의 조종 명령을 이용해 기체를 제어한다. IMU, Barometer 와 MAVLink 디바이스 드라이버는 OSS 에 존재하며, IOSS 의 IMU, Barometer Service 는 디바이스 드라이버에서 장치 의존적인 데이터를 수신하여 IO 메시지 모델로 변환 하여 상위 세그먼트(IMU, Barometer Manger)로 전달된다. MAVLink Service 는 GCS 의 제어 명령 데이터를 수신하여 IO 메시지 모델로 변환하여 상위 세그먼트의 Pilot Command Manager 로 전달한다.

IO 메시지 모델로 변환된 데이터는 PSSS 의 IMU, Barometer Manager 그리고 Pilot Command Manager 에서 가공되어 FACE 데이터 모델로 변환된다. FACE 데이터 모델로 변환된 데이터는 TSS 를 통해 PCS 의 Generic Flight 로 전달되고, Generic Flight 에서 기체의 위치, 고도, 그리고 속도 제어를 처리한다. 여기서 기체 조종을 위한 일반적인 제어 명령이 처리된다.

Generic Flight 에서 처리된 제어 데이터는 TSS 를 통해 PSSS 에 존재하는 Motor Control Manager 와 Pilot Command Manager 로 전달된다. Motor Control Manager 는 제어 데이터를 받아 현재 UAV 기종에 맞는 제어 명령으로 변환하여 IOSS 의 Motor Service 로 전달하고,

OSS 에 존재하는 모터 제어 디바이스 드라이버를 통해 실제 Motor 를 제어할 수 있도록 한다. Pilot Command Manager 에서는 UAV 비행 상태 값을 MAVLink 포멧으로 변환하여 IOSS 의 MAVLink Service 로 전달하고, MAVLink 장치를 통해 GCS 로 비행 상태 값을 송신한다.

● 네비게이션 소프트웨어

네비게이션 소프트웨어는 GPS 데이터 및 MAVLink 를 통해 수신된 GCS 조종 명령을 이용하여 자율 비행(Autopilot) 을 담당한다. 이 GPS 데이터와 조종 명령은 OSS 내 GPS 장치와 MAVLink 장치로부터 수집되어 IOSS 로 입력되며, 각각 GPS Service 와 MAVLink Service 를 통해 PSSS 의 GPS Manager 와 Pilot Command Manager 로 각각 전달된다.

PSSS 의 GPS Manager 는 기체에 대한 현재 위치 정보를 추출하고, Pilot Command Manager 는 MAVLink Service 로부터 전달 받은 자율 비행 명령을 추출한다. 이 값들은 모두 TSS 를 통해 PCS 의 Navigation 으로 전달되며 현재 위치 정보와 자율 비행 정보를 이용하여 자율 비행 관련 기능이 수행된다. Navigation 에서 실제 기체 제어를 수행하지 않고 PCS 의 Generic Flight 에게 제어 데이터를 전달하여 기체 제어를 수행한다.

● 항공기 상태 알람 소프트웨어

항공기 상태 알람 소프트웨어는 현재 UAV 의 기체 상태를 표시하는 기능으로 PSSS 의 HMFM 으로부터 시스템 레벨의 기체와 소프트웨어 상태를 모니터링한 뒤 PCS 의 Status/Alarm 으로 상태 데이터를 전달한다.

Status/Alarm 에서는 전달받은 값을 통해 현재 기체 상태에 따른 알람의 종류를 판단하고, PSSS 의 Alarm Manager 로 알람 데이터를 보낸다.

Alarm Manager 에서는 해당 알람 데이터를 장치 종류에 맞는 데이터로 변환하고 IOSS 의 Alarm Service 로 데이터를 보낸다. 이후 Alarm Service 는 IO 메시지 모델 기반의 알람 데이터를 LED 와 같은 장치에 맞도록 변환하여 전송한다.

● 데이터 로깅 임무 소프트웨어

데이터 로깅 임무는 PCS 의 Generic Flight, Navigation, Status/Alarm 에게서 UAV 와 관련된 데이터를 받아 로그 데이터를 생성하고 UAV 내부 저장소에 로그 데이터를 저장하는 임무를 수행한다.

Data Logger 는 PCS 의 다른 UoP 들에게서 받은 UAV 데이터를 수집하여 TSS 를 통해 Data Control Manager 로 보낸다. Data Control Manager 는 IO 메시지 모델로 변환하여 Data Module Service 를 통해 Stroage 장치에 저장한다.

제안된 FACE 기반 무인 비행기 소프트웨어 구조의 PCS 와 PSSS 의 소프트웨어는 이식성과 상호운용성이 높다. 다만 IOSS 나 OSS 같은 경우 장치 의존적인 부분이 존재하기 때문에 다른 UAV 기종에서 IOSS 와 OSS 를 제공한다면 소프트웨어 수정 없이 포팅하여 운용할 수 있다.

3.2.2 FACE 기반 지상 관제 시스템 구조

본 절에서는 전자 지도, PFD, Way Point, UAV 조종 기능을 수행하는 FACE 기반 지상 관제 시스템을 제안한다. 그림 3 은 FACE 기반 지상 관제 시스템 전체 구조를 나타낸다.

● 전자지도 소프트웨어

전자 지도 소프트웨어는 소형 UAV 에서 MAVLink 를 통해 전송된 위치 정보를 사용자에게 보여주는 기능을 수행한다. 이를 위해 MAVLink 장치에서는 UAV 정보를 수신하여 IOSS 의 MAVLink Service 로 전달한다. 전달된 값은 IO 메시지 모델로 변환되고, PSSS 의 Flight Information Manager 로 전달된다.

전달된 값은 UAV 의 위치, 속도, 고도 등의 비행 정보를 포함하는데, 이 값은 TSS 를 통해 PCS 의 GPS Aiding 으로 전달된다. 이후 GPS Aiding 에서는 GPS 데이터와 각종 비행 정보들을 처리하여 다른 PCS 컴포넌트들에게 비행 정보를 제공하는 역할을 수행한다.

PCS 의 Digital Map 은 비행 정보를 받아 현재 기체의 위치를 식별하고, 해당 지도 데이터를 데이터 스트림 형태로 PSSS 의 GLX Service 로 전달한다. 이후 GLX Service 에서는 해당 지도 데이터를 디스플레이 드라이버로 보내고 사용자에게 출력되도록 한다.

GLX Service 는 GLX 서버 역할을 하며 그래픽 렌더링을 수행한다. GLX 서버를 사용함으로서 PCS 의 구성요소와 그래픽 엔진을 분리할 수 있다. OpenGL 은 표준 API 이므로 IOSS 를 거치지 않고 디스플레이 드라이버로 데이터를 전달한다.

● PFD 소프트웨어

PFD 는 비행 정보를 그래픽으로 보여주는 역할을 하며, 전자 지도와 유사한 구조로 수행된다. UAV 에서 전송한 비행 데이터는 MAVLink 를 통해 GCS 로 전달된다. 또한 GPS Aiding 에서 기체의 고도, 자세, 속도, 방향 등의 비행 정보를 PFD 에서 수신 받는다. PFD 에서 비행 정보를 입력으로 받아 PFD 화면에 대한 정보를 업데이트하고, GLX Service 로 GCS 의 화면 정보를 스트림으로 전송하여 GLX Service 에서 렌더링을 수행한다.

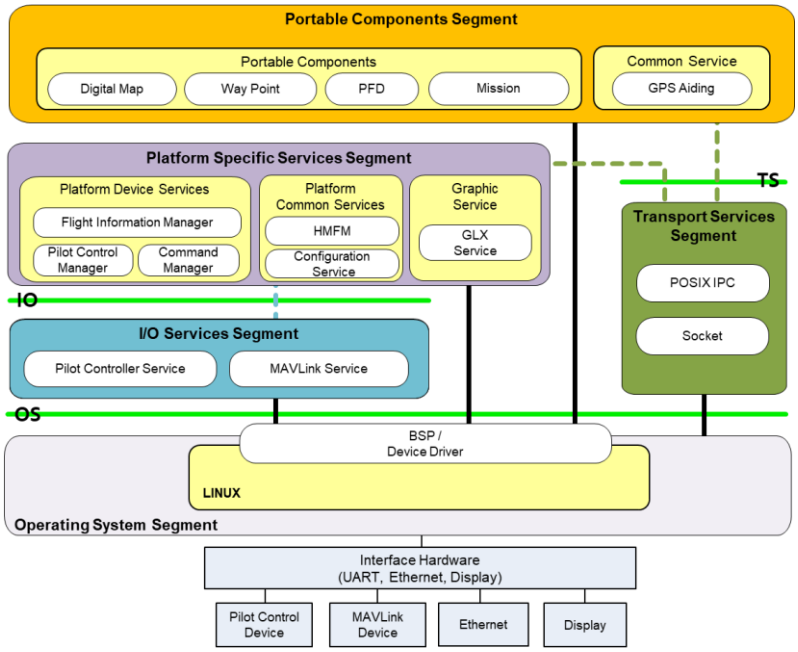

Figure 3. GCS Software Architecture

● Way Point 소프트웨어

Way Point 는 GCS 의 사용자가 전자지도를 기반으로 비행 경로를 설정하고, 설정된 비행 경로대로 자동 비행을 설정하는 역할을 한다. 사용자가 GCS 의 화면에서 비행 경로를 입력하면, IOSS 의 Pilot Controller Service 에서 사용자 입력 데이터를 디바이스 드라이버에서 수신하여 IO 메시지 모델로 변환 후 PSSS 의 Pilot Control Manager 로 보낸다.

Pilot Control Manager 는 사용자 입력 데이터 값을 FACE 데이터 모델로 변환 후 PCS 의 Way Point 로 보낸다. Way Point 는 지도와 사용자 입력 데이터를 이용하여 좌표 값과 비행 경로를 계산하여 PCS 의 Mission 으로 전달한다. Mission 에서 비행 경로 데이터를 전달받아 자동 비행 설정과 관련된 명령을 처리하고 비행 명령을 PSSS 의 Command Manager 로 전달한다. Command Manager 는 비행 명령을 입력으로 받아 IO 메시지 모델로 변환하여 IOSS 의 MAVLink Service 로 데이터를 전달 후 MAVLink 장치를 통해 UAV 로 비행 명령이 전달된다.

● **UAV 조종 소프트웨어**

UAV 조종 소프트웨어은 GCS 에서 사용자가 기체를 수동 조종하는 기능이며, GCS 의 Pilot Control 장치에서 비행 조종 입력이 들어오면 IOSS 의 Pilot Controller Service 에서 비행 조종 입력 데이터를 수신 받는다. 이후 Pilot Controller Service 는 비행 조종 입력 데이터를 IO 메시지 모델로 변환 후 PSSS 의 Pilot Control Manager 로 보낸다.

Pilot Control Manager 는 FACE 데이터 모델로 변환 후 PCS 의 Mission 으로 전송하고, Mission 에서는 비행 조종 입력 값과 현재 기체의 비행 정보 값을 이용하여 비행 조종 데이터를 연산한다. 이후 PSSS 의 Command Manager 로 비행 조종 명령을 보내고, IOSS 의 MAVLink Service 로 데이터 전달 후 MAVLink 장치를 통해 UAV 로 비행 조종 명령이 전달된다.

제안된 FACE 기반 지상 관제 시스템 구조에서는 앞서 제안한 무인 비행기 소프트웨어 구조와 마찬가지로 PCS 와 PSSS 의 컴포넌트를 이기종 GCS 에 수정 없이 포팅되어 운용할 수 있을 것이다.

3.3 FACE 기반 민간 소형 항공기 시스템 고찰

본 논문에서 제안한 구조는 FACE 5 개의 세그먼트 중 OSS 를 제외한 PCS, PSSS, TSS, IOSS 내의 컴포넌트로 구성된다. 제안한 구조는 민간 소형 무인 항공기 시스템 분야에서 소프트웨어에 대한 재사용성 및 이식성을 향상할 수 있는 구조이다. 하지만, 본 논문에서 제안한 FACE 기반 무인항공기 시스템은 기존 무인 항공기 시스템에 비해 성능적인 측면에서 문제를 야기할 수 있다. FACE 기반의 소프트웨어는 런타임 동안 I/O 장치의 데이터를 IOSS, PSSS 및 TSS 계층을 지나 PCS 로 전달한다. 이는 각 계층에 존재하는 스레드들의 데이터 복사와 세마포어와 같은 동기화로 인해 오버헤드를 발생시킨다. 또한 IOSS 와 PSSS 계층에서 데이터 변환이 발생하는데, 소프트웨어에서 요구하는 메시지가 많아질수록 데이터 변환 또한 많이 발생하여 오버헤드를 발생시킨다. 결국 기존 소프트웨어보다 데이터에 대한 Latency 가 증가한다. 또한 FACE 소프트웨어 설계 시 IOSS, PSSS 및 PCS 의 크기를 작은 단위(Fine Grained)로 구성하면 소프트웨어 성능이 저하될 수 있다. 소프트웨어 기능을 작은 단위로 구성하게 되면, 기능을 담당하는 스레드 수 또한 증가하게 되므로 태스크에 대한 문맥 교환(Context Switching) 비용이 증가 한다. 그리고 PCS 의 구성요소들이 증가하게 되므로 시스템이 복잡해지고 이를 유지하기 위한 비용이 증가하게 된다.

다품종 소량생산으로 공급되는 민간 소형 무인 항공기 시스템에서는 소프트웨어에 대한 이식성이 중요한 요소 중 하나이다. FACE 기반 소형 무인항공기 시스템 소프트웨어는 이식성에 대한 문제를 해결하지만, 앞서 언급한 성능과 이식성에 대한 절충(Trade-off)을 고려해야 한다. 다만 최근 무인 항공기에 탑재되는 컴퓨팅 노드들의 소형화와 멀티코어와 같은 기술들이 적용되어 성능이 향상되고 있다. 예를 들어 Raspberry Pi 와 같은 저비용 고성능의 멀티코어 기반 소형 임베디드 보드가 장착된 소형 드론[18]이 민간 분야에서 널리 적용되고 있다. 이는 기존 마이크로프로세서 기반의 소형 무인 항공기에 비해 성능적 문제를 완화할 수 있을 것이라 판단된다. 또한, 무인 항공기 탑재 보드는 임무 수행과 관련된 소프트웨어들이 사전에 정의되어 있고, 소프트웨어 기능들이 동적으로 추가되지 않는 환경이다. 따라서 저비용 고성능의 멀티코어 기반 컴퓨팅 보드를 이용하고, 사전에 최악 실행시간 분석과 소프트웨어 성능 분석을 철저히 수행하여 FACE 기반 무인 항공기 시스템 소프트웨어를 구성한다면 재사용성과 이식성이 뛰어난 민간 소형 무인 항공기 시스템이 될 것이다.

IV. 개방형 시스템 아키텍처 기반 소프트웨어 개발 및 검증 방법

본 장에서는 FACE 기반 소프트웨어 개발 방법과 FACE 규격을 준수 했는지에 대한 적합성 테스트를 수행하는 방법에 대해 설명한다. 본 논문에서는 민간 분야에서 쉽게 접근할 수 있는 공개된 개발 도구 및 적합성 테스트 도구를 기반으로 서술한다.

4.1 FACE 개발 프로세스

그림 4 는 FACE 기반의 소프트웨어를 개발하기 위한 프로세스를 나타낸다. FACE 는 상호 운용성 요구사항을 지원하면서 기존 UoP 를 효율적으로 이식하거나 포팅하기 위한 FACE 데이터 모델에 관하여 정의한다. FACE 기반 소프트웨어를 개발하기 위해서는 FACE 가 교환하는 메시지 구조를 XMI(XML Metadata Interchange)형식의 파일로 제공해야 하며, 확장자는 ".face"를 갖는다. 이후 FACE 데이터 모델링을 수행할 때 FACE 컨소시엄에서 제공하는 SDM(Shared Data Model)[18]을 기반으로 FACE 데이터 모델을 작성한다. 작성된 데이터 모델을 USM(UoP Supplied Data Model)이라 한다. 생성된 USM 을 이용하여, Code Generator 를 통해 PCS 와 PSSS 사이의 통신을 위한 TS 인터페이스와 데이터 타입이 포함된 소스코드를 생성할 수 있다. FACE 표준에서는 현재 C, C++, JAVA, Ada 언어를 지원하고 있다. 개발자는 생성된 소스코드를 이용하여 세부적인 기능들을 추가한다. 개발 후 개발된 코드와 데이터 모델 파일을 기반으로 FACE 적합성 테스트(Conformance Test)를 수행한다. FACE 적합성 테스트는 데이터 모델과 FACE 5 개의 세그먼트에 대해 각각 수행된다.

Figure 4. FACE Software Development Process

4.2 FACE 개발 도구

FACE 개발을 위한 상용 및 공개 개발도구들이 존재한다. 본 논문에서는 다양한 FACE 개발 도구들 중 민간 분야에서 접근 가능한 공개된 개발 도구에 대해서 설명한다.

FACE 소프트웨어 개발 시 데이터 모델링은 필수이다. 따라서, FACE 데이터 모델링을 하기 위해서는 FACE 기술 표준을 지원하는 모델링 툴이 필요하다. Vanderbilit 대학의 MTF(Modeling Tools for FACE Software Development)[20]는 FACE 소프트웨어 개발자와 시스템 통합자에게 FACE 데이터 모델링 도구와 코드 생성기(Code Generator)를 제공한다. MTF 모델링 도구는 시각적으로 FACE 데이터 모델링을 수행할 수 있으며, FACE 데이터 모델링을 통해 생성된 XMI 파일을 기반으로 PCS 와 PSSS 내 UoP 간 데이터 전송을 위한 소스코드를 생성할 수 있다. 하지만 MTF 는 현재 FACE 표준 2.1 만 지원하고 있다는 단점이 있다. Vanderbilit 대학에서는 추가적으로 Rational Rhapsody, Sparx Enterprise Architect 도구를 기반으로 FACE 데이터 모델링을 위한 플러그인을 무료로 지원하고 있다[20].

OSATE(Open Source AADL Tool Environment)[21]는 실시간 시스템 소프트웨어 및 하드웨어를 모델링하는 아키텍처 언어인 AADL(Architecture Analysis & Design Language)[22]을 개발하는 오픈소스 도구이다. OSATE 에서는 플러그인 형태로 FACE 3.0 데이터 모델을 지원하고, 텍스트 형태로 FACE 데이터 모델을 생성 할 수 있다. 그리고 FACE 데이터 모델을 AADL 모델로 변환하여 시스템의 스케줄링 가능성(Schedulability), 흐름 제어(Flow Control)과 같은 분석을 수행할 수 있다.

4.3 FACE 적합성 프로세스

FACE 표준을 기반으로 소프트웨어를 개발 했다면, FACE 표준 규격을 준수했는지 확인하기 위해 FACE 적합성 테스트를 수행해야 한다. FACE 컨소시엄에서는 FACE CTS(Conformance Test Suite)[23]를 제공한다. FACE CTS 는 컴파일러(Compiler), 링커(Linker), 아카이버(Archiver)를 포함한 툴 체인과 개발 언어의 규칙(예를 들어 C99 등), 데이터 모델, 개발자 코드를 테스트한다.

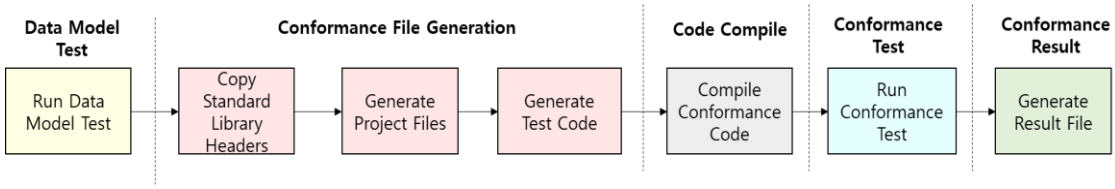

Figure 5. FACE Conformance Test Flow

그림 5 는 FACE 적합성 테스트의 실행 흐름을 보여준다. FACE CTS 는 FACE 데이터 모델에 대한 무결성 검사를 수행한다. 이때 FACE 컨소시엄에서 제공하는 SDM 과 사용자가 생성한 USM 두개의 데이터 모델 모두 FACE 데이터 모델 규칙을 준수하는지에 대한 적합성 테스트를 수행한다.

다음으로 CTS 는 적합성 테스트를 위한 파일을 생성하는데, 먼저 CTS 는 FACE 에서 지원하고 있는 표준 라이브러리에 대한 헤더 파일을 테스트 프로젝트 폴더에 복사한다. 다음으로 프로젝트 테스트 파일을 생성하는데, 이는 개발자 코드의 함수 구조, 데이터 타입 및 상수를 포함하는 테스트 코드를 생성한다. 그리고 CTS 에서 개발자의 코드를 테스트하기 위한 적합성 테스트 코드를 생성한다. 적합성 테스트 코드는 CTS 의 메인 실행 지점이 된다.

적합성 관련 코드들이 생성이 되었으면, 코드를 컴파일 한다. 이때 FACE 런타임 환경에 대해 적합성 테스트를 수행한다. 만약 컴파일 및 링크가 실패하면 현재 환경이 FACE 환경에 적합하지 않다는 것을 의미하며, 성공하면 런타임 환경에 적합하다는 것을 의미한다.

테스트 코드에 대한 컴파일 수행 후 CTS 는 적합성 테스트를 실행한다. 적합성 테스트는 CTS 의 메인 지점인 적합성 테스트 코드에서 사용자 코드로 FACE 데이터를 주입(Inject)하여 FACE 규격을 준수하는지를 테스트 한다. 즉, CTS 는 FACE 소프트웨어의 기능에 대한 테스트보다 인터페이스가 FACE 표준 규격에 맞게 올바르게 정의 되었는지 확인한다. 마지막으로 적합성 결과를 파일 형식으로 보고가 된다.

FACE 컨소시엄은 FACE 적합성 프로그램(FACE Conformance Program)을 통해 FACE 기술 표준에 따라 개발하는데 필요한 적합성 기준 및 프로세스를 제공한다. 그림 6 은 FACE 적합성 프로세스를 나타내는 그림이다. FACE 적합성 프로그램은 Verification, Certification 및 Registration 으로 구성된다. Verification 은 FACE 표준 요구사항에 대한 구현의 적합성을 결정하는 프로세스로 소프트웨어 개발 산출물을 포함한 FACE CTS 결과에 대한 검증이 수행된다. Verification 이 종료되면, FACE 인증 기관(Certification Authority)에게 인증을 신청한다. 마지막으로 FACE Registration 은 FACE Library 에 등록하는 프로세스를 수행한 뒤 FACE 적합성 인증서와 FACE 상표를 사용할 권리를 부여한다.

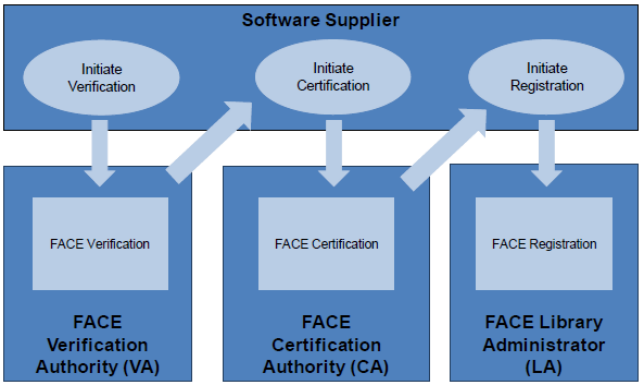

Figure 6. FACE Conformance Process Flow[10]

V. 결론

본 논문에서는 군용 항공시스템에서 주로 다루었던 개방형 시스템 아키텍처인 FACE 를 민간 소형 무인 항공기 시스템에 적용한 소프트웨어 구조를 제안하였다. 본 논문에서는 FACE 참조 아키텍처를 기반으로 UAV 의 기체 제어와 임무 소프트웨어, GCS 에서는 기체 운용 및 모니터링 소프트웨어를 제안하였다. 제안된 소프트웨어는 공통된 기능을 하는 소프트웨어와 하드웨어 의존적인 소프트웨어를 FACE 세그먼트로 나누어 구분하였다. 결론적으로 본 논문에서 제안한 소형 무인항공기 시스템 소프트웨어 구조는 다양한 무인 항공기 시스템 간의 이동 및 통합이 쉽게 이루어 질 것이다. 또한 공개 개발도구들을 기반으로 FACE 기반의 소프트웨어를 개발하는 방법론과 FACE 적합성 테스트 및 FACE 인증 방법에 대해 서술하였다. 이는 민간 분야에서 쉽게 FACE 기반의 소프트웨어를 구성할 때 좋은 참조가 될 수 있을 것이다.

VI. 참고문헌

[1] R. Austin, "Unmanned Aircraft Systems: UAVS Design Development and Deployment," John Weily and Sons, vol. 54, 2011.
[2] N. H. Motlagh et al., "Low-Altitude Unmanned Aerial Vehicles-Based Internet of Things Services: Comprehensive Survey and Future Perspectives," in IEEE Internet of Things Journal, vol. 3, no. 6, pp. 899-922, December 2016.
[3] H. Shakhatreh et al., "Unmanned Aerial Vehicles(UAVs): A Survey on Civil Applications and Key Research Challenges," in IEEE Access, vol. 7, pp. 48572-48634, 2019.
[4] E. Pastor et al., "UAV Payload and Mission Control Hardware/Software Architecture," in IEEE Aerospace and Electronic Systems Magazine, vol. 22, no. 6, pp. 3-8, June 2007.
[5] D. Perez et al., "A Ground Control Station for a Multi-UAV Surveillance System," Journal of Intelligent and Robotic Systems, vol. 69, no. 1-4, pp. 119-130, 2013.
[6] S. Welby, "Modular Open Systems Architecture in DoD Acquisition," 17th Annual NDIA Systems Engineering Conference, October 29 2014.
[7] J. L. Tokar, "A Comparison of Avionics Open System Architectures," ACM SIGAda Ada Letters, vol. 36, no. 2, pp. 22-26, May 2017.
[8] NATO STANAG 4586, Standard Interfaces of UAV Control System(UCS) for NATO UAV Interoperability, Edition 4, April 2017.
[9] Open Mission Systems, https://www.vdl.afrl.af.mil/programs/uci/oms.php, Accessed 13 April 2020.
[10] Future Airborne Capability Environment, http://www.opengroup.org/face, Accessed 13 April 2020.
[11] E. Pastor et al., "An Open Architecture for the Integration of UAV Civil Applications," Aerial Vehicles, InTech, New York, Chap. 24, pp. 511–536, January 2009.
[12] K. Park et al., "A Representational State Transfer Based Architecture for Web Services Using Unmanned Aerial Vehicles, " Advanced Science Letters, vol. 20, pp. 2060-2063, October 2014.
[13] W. Müller, et al., "Open Architecture of a Counter UAV System," in Proc. of the SPIE 10651, Open Architecture/Open Business Model Net-Centric Systems and Defense Transformation, Florida, 2018.
[14] H. Jo, et al., "Portable and Configurable Implementation of ARINC-653 Temporal Partitioning for Small Civilian UAVs," in IEEE Access, vol. 7, pp. 142478-142487, 2019.
[15] Avionics Application Software Standard Interface(Part 1): Required Services, Document ARINC Specification 653P1-4, 2015.
[16] ArduPilot, https://ardupilot.org/, Accessed 13 April 2020.
[17] Mission Planner, https://ardupilot.org/planner/docs/mission-planner-ground-control-station.html, Accessed 13 April 2020.
[18] Erle-Copter, https://robots.ros.org/erlecopter/, Accessed 13 April 2020.
[19] FACE Shared Data Model, https://www.opengroup.org/face/docsandtools, accessed 13 April 2020.
[20] Modeling Tools for FACE Software Development, https://www.isis.vanderbilt.edu/FACE, Accessed 13 April 2020.
[21] Open Source AADL Tool Environment, https://osate.org/, Accessed 13 April 2020.
[22] SAE International, Architecture Analysis & Design Language, AS5506C, January 2017.

[23] FACE Conformance Test Suites, https://www.opengroup.org/face/conformance-testsuites, Accessed 13 April 2020.

저자 소개

조현철*(Hyun-Chul Jo)*

2012 년 2 월 건국대학교 컴퓨터공학 학사
2014 년 2 월 건국대학교 컴퓨터공학 석사
2019 년 8 월 건국대학교 컴퓨터공학 박사
2019 년 10 월 ~ 현재 한화시스템 항공연구센터 전문연구원

관심분야 : 실시간 시스템, 임베디드 시스템, 운영체제

박근영*(Keunyoung Park)*

2013 년 2 월 안양대학교 디지털미디어공학 학사
2015 년 2 월 건국대학교 인터넷미디어공학 석사
2019 년 8 월 건국대학교 인터넷미디어공학 박사
2019 년 9 월 ~ 2020 년 3 월 국립농업과학원 박사후연구원
2020 년 3 월 ~ 현재 한화시스템 항공연구센터 전문연구원

관심분야 : 임베디드 시스템, 인공지능, 컴퓨터비전