논문 21-46-01-17 The Journal of Korean Institute of Communications and Information Sciences '21-01 Vol.46 No.01
https://doi.org/10.7840/kics.2021.46.1.141

객체-관계형 모델 기반의 ADN 메시지의 Designer/Dissector 구현

손 명 환•, 박 부 식°, 윤 종 호*

ADN Message Designer/Dissector Implementation Based on Object-Relational Model

Myeong-hwan Son•, Pu-sik Park°, Chong-ho Yoon*

요 약

네트워크 시스템에서 ICD (Interface Control Document)는 시스템 또는 장치 사이에 주고받는 메시지의 구조를 정의하는 중요한 정보이다. 이 정보는 다른 시스템과 공유하기 위해 문서 또는 여러 형식의 파일로 보관되고 시스템 개발자 및 메시지와 관련된 다양한 어플리케이션을 통해 활용된다. 본 논문은 ADN (Aircraft Data Network) 메시지 및 ICD의 데이터 구조를 체계적으로 정의하고 효율적으로 관리할 수 있는 객체-관계형 모델을 제안한다. 또한 메시지에 포함된 데이터를 해석하고 가공 및 재구성하여 사용자에게 제공하기 위해 필요한 ICD Designer와 Dissector를 어플리케이션의 형태로 구현한다. 그리고 구현된 어플리케이션을 사용하여 ADB (Aircraft Data Bus)의 메시지 해석에 필요한 ICD를 설계하고 수집된 메시지를 해석한다. 제안한 객체-관계형 모델 기반의 Designer 및 Dissector는 기존 유사 기술 대비 사용성이 개선되었다.

키워드 : ADB, ADN, ICD, 객체-관계형 모델, 네트워크 분석
Key Words : ADB, ADN, ICD, Object-Relational Model, Network Analysis

ABSTRACT

In a network system, an ICD (Interface Control Document) is important information that defines the structure of messages exchanged between systems or devices. This information is stored as documents or files in multiple formats to share with other systems and is utilized by system developers and various applications related to messages. We proposes an object-relational model that can systematically define and efficiently manage ADN (Aircraft Data Network) messages and ICD data structures. In addition, we implement the ICD Designer and Dissector in the form of applications needed to interpret, process and reconstruct the data contained in the message and provide it to the user. Then, using the implemented application, we design the ICD required to interpret the message of ADB (Aircraft Data Bus) and interpret the collected message. The proposed object-relational model based Designer and Dissector have improved usability compared to existing similar technologies.

※ 본 연구는 산업통상자원부가 지원하는 항공우주부품기술개발사업의 일환으로 수행되었습니다.(2003238)
• First Author : Korea Electronics Technology Insititute, mhson@keti.re.kr, 정회원
° Corresponding Author : Korea Electronics Technology Insititute, pusik.park@keti.re.kr, 종신회원
* Korea Aerospace University Aviation Electronics Information Engineering, yoonch@kau.ac.kr, 종신회원
논문번호 : 202007-168-D-RN, Received July 27, 2020; Revised October 14, 2020; Accepted October 15, 2020

Ⅰ. 서 론

항공 우주 및 무기 체계 분야에서 네트워크 시스템의 규모가 커지면서 네트워크 및 소프트웨어 관련 기술도 함께 발전하고 있다. 그와 동시에 항전 시스템 내에서는 여러 종류의 ADB (Aircraft Data Bus)를 통해 연결된 네트워크 시스템의 규모와 복잡성 또한 계속해서 증가하고 있다. 다수의 시스템을 네트워크 인터페이스를 사용하여 연결함에 있어 그 인터페이스의 입출력에 대한 정보는 필수적이고 중요한 정보이다. 이러한 인터페이스의 입출력 정보는 ADN (Aircraft Data Network) 메시지 및 ICD (Interface Control Document)로 정의된다. 기존의 ICD 분석 방법은 크게 두가지로 나뉜다. 첫 번째는 ICD를 XML, ASN.1, UML, JSON 같은 언어를 통해 추상화하고 이 데이터를 기반으로 분석 코드를 생성하여 ICD를 분석하는 방법이다[1-4]. 그리고 두 번째는 일부 메시지 분석 소프트웨어처럼 Lua 같은 스크립트 언어를 사용하여 직접 메시지를 해석하는 방법이다[5]. 전자와 같은 방식은 추상화 결과를 코드로 변환하고 응용 소프트웨어를 다시 빌드해야 하는 불편함이 있고, 후자의 경우에는 스크립트 언어를 배워야 하는 어려움이 있다. 또한, 정의된 ICD를 유지보수하거나 재사용하기가 불편하다.

한 예로는 한국항공우주산업에서 MS Access 파일을 편집하여 ADN (Aircraft Data Network) 메시지 구조가 정의된 XML 코드를 생성하고, 항전 장비를 점검하는 솔루션에서 그것을 활용한 사례가 있다. 이 경우에 항전 장비를 개발하는 과정에서 발생하는 빈번한 ICD의 수정에 대해 매번 이전에 작성한 XML 코드를 수정하여 재생성하고 어플리케이션에서 그것을 갱신해야하는 불편함이 존재하였다. 이러한 문제는 항전 장비를 포함하여 컴퓨터 네트워크를 사용하는 모든 시스템의 개발 과정에서 발생한다.

본 논문의 목표는 이 같은 문제점을 개선하기 위해 ICD를 손쉽게 설계하고 관리하는 도구를 개발하는 것이다. 이를 위해 사용자 정의 자료형을 지원하는 객체-관계형 모델 구조를 사용하여 ICD에 정의되는 복잡한 메시지 구조를 모델링 하고 ICD Designer와 Dissector를 직관적이고 편리한 어플리케이션 형태로 구현한다.

본 논문의 구성은 다음과 같다. 본 서론에 이어, 제 2장에서는 메시지 구조 및 ICD를 정의하기 위한 객체-관계형 데이터 모델을 제안하고 GPS PVT 정보를 전송하기 위한 AFDX 메시지를 예를 들어 정의한다. 제 3장에서는 객체-관계형 모델로 정의된 ICD를 사용하여 메시지를 식별하고 해석하는 알고리즘을 제시한다. 제 4장에서는 객체-관계형 모델을 기반으로 한 메시지 분석기의 구조를 제시하고 ICD Designer와 Dissector를 기존의 네트워크 진단 어플리케이션을 기반으로 구현한다. 마지막으로 제 5장에서는 결론을 제시한다.

Ⅱ. 객체-관계형 모델 기반의 메시지 구조 정의

2.1 객체-관계형 모델

XML과 같은 구조적 언어의 한계를 극복하기 위해 XML에 정의된 데이터들을 객체-관계형 데이터베이스를 통해 효율적으로 관리하는 방법론들이 등장하였다[6,7]. 객체-관계형 모델은 관계형 모델에 객체 지향적인 특성을 추가하여 기존의 관계형 모델로는 표현하기 어려운 복잡한 관계의 계층적인 자료구조를 표현할 수 있다. 본 논문에서는 복잡한 메시지 구조를 쉽게 정의하고 효율적으로 관리하기 위해서 객체-관계형 모델링을 통해 기존에 존재하는 자료형들의 집합으로 사용자 정의 자료형을 정의할 수 있도록 한다.

그림 1은 메시지 구조를 객체-관계형 모델로 구성하기 위한 주요 객체들과 그 상속 관계 및 연관 관계를 보여주는 E-R 다이어그램이다. 화살표는 객체들 간의 상속 관계 보여주며 Crow's foot은 객체들 간의 연관 관계를 보여준다. 객체들은 각각의 속성과 기능들을 가지고 있으며 모든 객체들은 PK (Primary Key)를 가진다. 또한 어떤 객체들은 FK (Foreign Key)를 사용하여 다른 객체들을 참조한다.

Basic Type, Enum Type, Stream Type, Array Type, 그리고 Complex Type은 Data Type을 상속받는다. Data Type은 Data Field 및 Array Type과 일대다 관계를 가지며 Complex Type은 Data Field 및 Message Type과 일대다 관계를 가진다. Enum Type은 Enum Member와 일대다 관계를 가진다. 이러한 관계는 관계형 데이터베이스에서 기본키 (PK)와 외래키 (FK) 속성을 통한 참조로 구현된다. Data Field 객체는 Type 속성을 통해 Data Type 객체 하나를 참조한다. 그리고 Message Type은 Type 속성을 통해 Complex Type 객체 하나를 참조하며 Complex Type 객체는 Data Field 객체의 Parent 속성을 통해 다수의 Data Field 객체를 참조한다. 다음은 각 객체의 역할과 속성에 대한 설명이다.

2.1.1 자료형 (Data Type)

메시지 구조를 정의하기 위해 자료형은 표 1과 같이 크게 5가지 유형으로 구분된다. 기본적인 원시 자

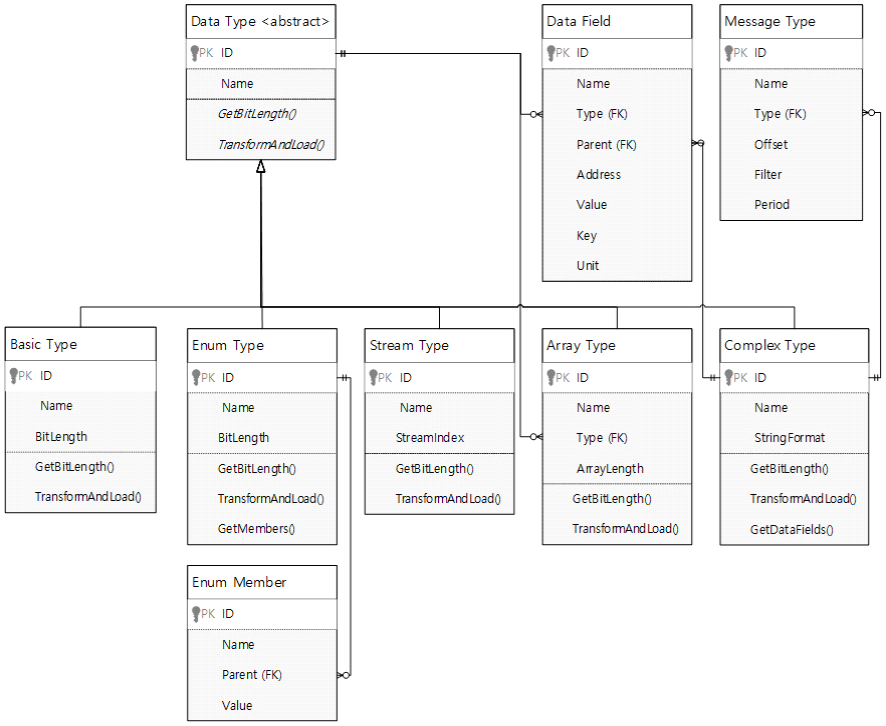


그림 1. 메시지 구조 정의를 위한 객체-관계형 모델의 E-R 다이어그램
Fig. 1. E-R diagram of object-relational model for defining message structure

료형인 Basic Type 외에도 Enum Type, Array Type, Stream Type, Complex Type의 자료형이 존재하며 특히 Complex Type은 복잡한 데이터 구조의 관계형 모델링을 위해 여러가지 유형의 자료형들로 구성할 수 있는 사용자 정의 자료형이다. Array Type과 Data Field 객체의 Type 속성에는 Basic Type 뿐아니라 Complex Type을 포함하여 모든 자료형이 사용될 수 있다. 단, 루프가 발생하는 경우를 방지하기 위해 하위 구조에 자기 자신이 포함되지 않게 구성해야 한다.

그림 1의 Data Type 객체는 자료형을 정의하는데 필요한 기본적인 속성과 기능을 선언한 추상 객체이다. Data Type 객체를 상속받는 5가지 자료형 (Basic Type, Enum Type, Array Type, Stream Type, 그리고 Complex Type) 객체들이 각각 개별적으로 데이터 해석 기능을 구현한다. 이 객체들은 각각의 자료 유형에 따라 추가적인 속성을 가질 수 있다. 표 2는 그림

표 1. 자료형의 유형
Table 1. Type of data type

유형	설명
Basic	크기와 해석 방법이 기본적으로 특정되어 있는 원시 자료형 (Primitive Type)
Enum	사용자가 명명하는 값으로 이루어진 열거 자료형
Stream	특정 포맷의 문자열, 이미지, 오디오, 비디오 스트림
Array	한 종류의 자료형의 배열로 정의된 자료형
Complex	여러가지 자료형들이 결합하여 복합적인 구조를 가지는 사용자 정의 자료형

표 2. 자료형의 속성과 기능
Table 2. Attributes of data type

자료형	이름	설명
속성		
Common	ID	자료형을 식별하기 위한 식별자
	Name	자료형의 이름
Basic,Enum	BitLength	자료형의 비트 단위의 크기
Stream	StreamIndex	스트림간 구분을 위한 식별자
Array	Type	배열을 구성하는 자료형
	ArrayLength	배열의 길이
Complex	StringFormat	문자열 표시 방법
기능		
Common	GetBitLength	자료형의 크기를 반환
	ProcessData	데이터를 변형하고 다른 시스템에 적재하는 기능
Enum	GetMembers	멤버들을 제공하는 기능
Compelx	GetDataFileds	하위 데이터 필드들을 제공

1에 나타난 자료형 객체들의 속성 및 기능에 대해 보여준다.

데이터 해석에 사용되는 모든 자료형 객체에는 목표가되는 데이터의 크기 및 해석 방법이 명확하게 정의 되어야한다. 여기서 해석이란 기존 데이터의 가공 및 가공된 데이터를 다른 시스템에 적재하는 것을 말한다. 이는 GetBitLength와 TransformAndLoad 기능을 통해 각각의 자료형 객체에서 정의된다. Basic Type은 Boolian 또는 8/16/32bit Integer와 같이 기존에 크기와 해석 방법이 정의되어 있는 원시 자료형이다. Enum Type은 명명된 값 (Enum Member)의 집합인 자료형으로 사용자가 멤버 집합과 자료형의 크기를 지정해야 한다. Enum Type은 Enum Member의 외래키인 Parent 속성을 통해 다수의 Enum Member들을 참조하는 GetMembers 기능을 제공해야 한다. Stream Type은 문자열, 이미지, 오디오, 비디오 데이터와 같은 스트림 데이터를 해석하고 표시하기 위한 자료형이다. 스트림 자료형은 스트림 메시지를 Stream Index 속성을 통해 스트림을 구분한다. Stream Type은 데이터 크기가 특정되어 있지 않은 경우, 메시지를 해석하는 과정에서 크기가 특정될 수도 있다. ArrayType은 하나의 자료형의 배열로 구성된 자료형으로 배열의 요소가 되는 자료형과 길이를 사용자가 지정함으로써 구조가 정의된다. 그리고 Complex Type의 경우 GetDataFields 기능을 통해 포함되는 하위 데이터 필드들을 얻고 그 데이터 필드들의 자료형의 정의에 따라 크기와 해석 방법이 특정된다. 그리고 String Format 속성을 사용하여 다양한 자료형의 하위 데이터들을 문자열로 반환하는 형식 문자열을 지정할 수 있다.

2.1.2 데이터 필드 (Data Field)

데이터 필드는 자료형을 속성으로 가지는 객체이며 Complex type은 이 데이터 필드들의 집합으로 새로운 자료형을 정의한다. 데이터 필드 객체는 자료형 외에도 식별을 위한 ID, 메시지 해석과 표시에 필요한 주소 (Address), 단위 (Unit), 기본 값 (Value) 등과 같은 속성을 포함하며 하나의 자료형을 가지고 여러개의 데이터 필드 객체를 생성할 수 있다.

표 3은 데이터 필드의 속성 (Attribute)들을 보여준다. ID는 데이터베이스 전체에서 데이터 필드를 식별하기 위한 식별자이다. Name은 사용자가 명명하는 데이터 필드의 이름이다. Type은 데이터 필드의 자료형이다. 일반적으로 데이터 필드의 크기와 해석 방법은 자료형에 의해서 결정된다. 단, Stream type의 경우 크기가 고정되지 않아 다른 데이터 필드의 값을 사용하는 방식이 필요할 수 있다. Address는 Complex Type 내에서 이 데이터 필드의 시작 위치이다. 다시 말하면 Complex Type 내에서 해당 데이터 필드 앞에 위치한 다른 데이터 필드들의 크기의 합이다. Value는 메시지를 생성하거나 유효성을 검사할 때 사용되는 값 또는 식이다. Key는 이 데이터 필드가 메시지를 식별하기 위해 사용되는지 결정한다. Key의 값이 true이면 실제 메시지 필드의 값이 Value의 값과 일치하는 경우에 이 데이터 필드를 사용한 해석방법이 유효한 것으로 판단한다.

표 3. 데이터 필드의 속성
Table 3. Attributes of data field

유형	설명
ID	데이터 필드의 식별자
Name	데이터 필드의 이름
Type	데이터 필드의 자료형
Parent	이 데이터 필드를 소유하는 Complex Type 자료형
Address	상위 Complex Type 에서 해당 데이터 필드의 시작 위치
Value	메시지의 생성에 사용되는 값 또는 식
Key	Value를 메시지 식별에 사용할 것인 지 결정하는 true/false 값
Unit	표시할 단위

2.1.3 메시지 타입 (Message Type)

사용자는 최종적으로 어떤 메시지의 해석 방법을 정의하기 위해 Complex Type을 자료형으로 가지는 메시지 타입을 정의한다. 메시지 타입은 자료형 외에도 메시지 해석을 위해 전체 메시지 데이터에서 해당 자료형이 시작하는 위치와 메시지의 식별에 필요한 조건식 등을 속성으로 포함한다.

표 4. 메시지 타입의 속성
Table 4. Attributes of message type

유형	설명
Name	메시지를 식별하는 이름
Type	메시지를 해석하기 위한 Complex Type의 자료형
Offset	전체 메시지 데이터에서 해석할 데이터의 시작 위치
Filter	메시지의 식별에 사용되는 식
Period	메시지의 송신 주기

표 4는 메시지 타입이 가지는 속성들을 보여준다. ID는 데이터베이스 전체에서 메시지 타입을 식별하기 위한 식별자이다. Name은 사용자가 명명하는 메시지 타입의 이름이다. Type은 메시지를 해석하는데 필요한 Complex Type의 자료형이다. Offset은 전체 메시지 데이터에서 해석할 데이터 (Payload 영역)이 시작되는 주소 또는 주소를 특정할 수 있는 값이다. Filter는 어떤 메시지에 대해 이 메시지 타입을 적용해야 할지 식별하기 위한 조건식이다. 메시지를 해석하는 툴에서는 메시지가 수신되면 이 조건식을 만족시키는 메시지 타입을 찾아 Type 속성에 할당된 Complex Type 자료형을 사용하여 메시지를 해석할 수 있다. Period는 메시지가 반복되는 주기를 나타낸다. 네트워크 고장 진단 도구는 이 속성을 통해 주기적으로 전송되는 제어 메시지가 적절한 주기로 수신되고 있는지 진단할 수 있다.

2.2 메시지 구조 정의를 위한 관계형 데이터베이스 및 정규화

제안하는 객체-관계형 모델에서는 메시지의 구조를 정의하는 각각의 개체들은 그 계층 위치와 관계없이 객체 종류에 따라 구분된 데이터베이스를 통해 관리된다. 메시지의 구조는 데이터베이스의 구조와 독립되어 오로지 데이터베이스에 포함된 내용에 의하여 결정되며 데이터베이스의 구조는 데이터베이스의 내용을 통해 메시지의 계층 구조를 재현하는 도구 역할을 한다. 따라서 아무리 복잡한 계층 구조의 메시지라도 그것을 관리하기 위한 데이터베이스의 구조는 변하지 않는다.

관계형 모델로 구성된 자료형의 정의와 편집에 있어서 참조 무결성을 보장하기 위해서 어떠한 자료형이나 데이터 필드의 속성이 변경되었을 경우에 그 변경 내용은 데이터베이스 전체에 적용시키는 정규화를 수행해야한다. 이러한 정규화를 위해 데이터베이스에서는 변경된 객체와 연관 관계를 가지는 모든 객체들에게 참조 관계를 고려하여 변경된 내용을 적용해야 한다.

예를 들어 Data Type 객체하나가 제거되었을 때 해당 Data Type 객체와 연관 관계에 있으면서 해당 Data Type을 참조하는 Data Field 객체들을 제거해야 한다. 하지만 Data Field 객체가 제거되었을 때는 Data Type 이 Data Field를 참조하지 않기 때문에 Data Type를 제거할 필요가 없다. 마찬가지로 Complex Type 객체 하나가 제거되었다면 해당 Complex Type을 참조하는 Message Type과 Data Field 객체들을 제거해야 한다. 하지만 Message Type이 제거되었을 때 Message Type이 참조하는 Complex Type 객체를 제거할 필요는 없다.

2.3 ADN 메시지 구조 정의 및 컴파일

2장에서 제안한 모델은 복잡한 구조의 ADN 메시지 및 ICD를 설계하는데 사용할 수 있다. 그림 2는 100M이더넷 기반의 ADB인 AFDX 네트워크에서 사용되는 메시지의 구조를 보여준다. AFDX 메시지는 UDP을 기반으로 FDS (Functional Data Set)의 필드 구조들이 UDP Payload 영역에 탑재된다. FDS는 FSS (Functional Status Set)와 최대 4개의 DS (Data Set)으로 구성된다. DS는 기본 데이터의 집합으로 구성된 데이터 필드이며 FSS는 4byte로 구성되어 각각의 바이트가 DS1, DS2, DS3, DS4의 유무 또는 상태를 나타내는 Functional Status (FS)에 해당한다. Functional Status는 4개의 상태를 가진다. 예를 들어 FDS에서 DS가 하나만 존재한다면 FS2, FS4, FS4는 NO_DATA (0)이고 FS1은 NO_DATA 이외의 값이 된다.

그림 2와 같은 구조의 AFDX 메시지의 예시 중 하나로써 GPS PVT 메시지는 한 개의 FDS에 한 개의 DS를 포함하고 있다[8]. 그리고 DS는 모두 Basic Type의 자료형의 집합으로 구성할 수 있는 구조이다. 이 메시지의 데이터 구조를 정의하기 위해 다음과 같이 새로운 객체들을 생성하는 작업이 필요하다.

Enumerated Type Table에 Fuctional Status라는 이름의 Enumerated type 자료형을 생성하고 멤버들을 정의하여 Member Table에 추가한다.

Complex Type Table에 GPS PVT DS1이라는 이름의 Complex Type 자료형을 생성하고 Data Type Table의 자료형들을 이용하여 데이터 필드들을 정의하고 Data Field Table에 추가한다.

Complex Type Table에 GPS PVT Data라는 이름

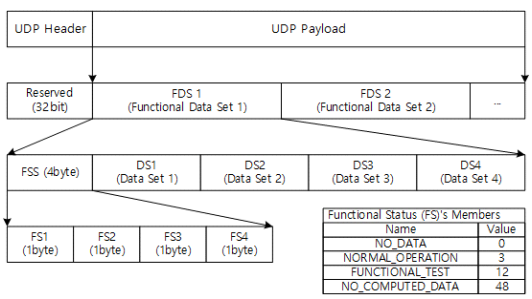


그림 2. AFDX 메시지의 구조[8]
Fig. 2. The structure of the AFDX Message[8]

의 Complex type 자료형을 생성하고 앞서 정의한 Functional Status와 GPS PVT DS1을 사용하여 그림 1과 같이 데이터 필드들을 생성하고 Data Field Table에 추가한다.

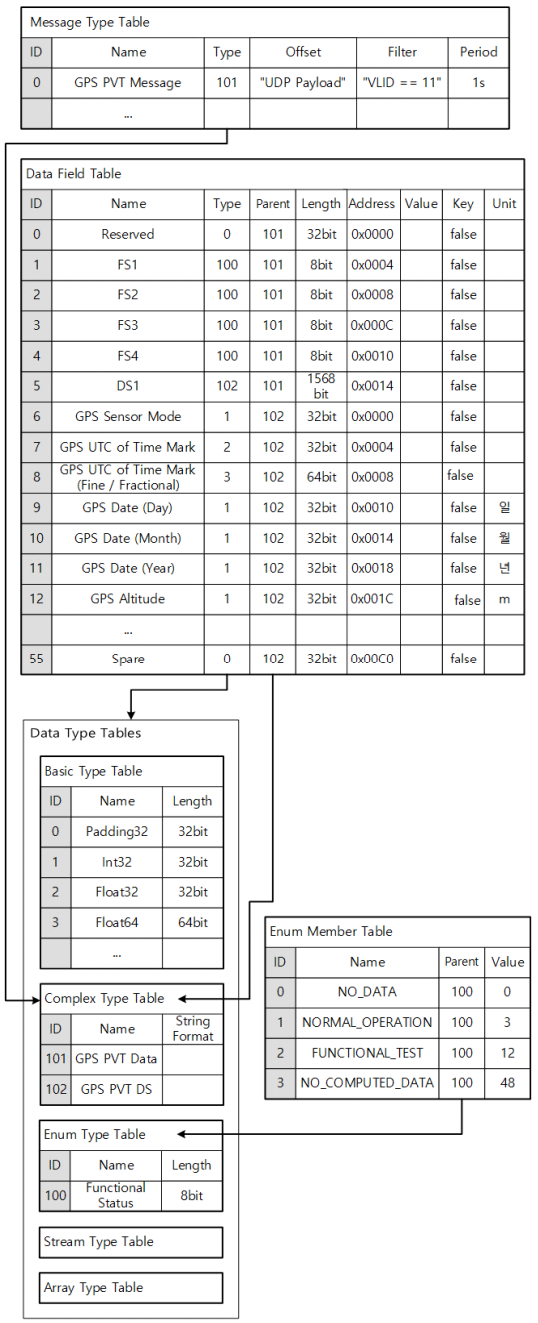


그림 3. GPS PVT 메시지의 구조가 정의된 관계형 데이터베이스의 구조
Fig. 3. Relational database structure with the structure of GPS PVT messages

마지막으로 Message Type Table에 GPS PVT Message라는 이름으로 GPS PVT Data 자료형의 메시지 타입을 생성하고 Offset, Filter 등의 나머지 속성들에 값을 추가한다.

그림 3은 이러한 작업으로 정의된 메시지가 컴파일되어 객체-관계형 데이터베이스에 저장된 것을 보여준다. 객체 종류 별로 테이블이 존재하며 참조 관계를 화살표를 통해 보여준다. Message Type의 Type 속성은 Complex Type 테이블을 참조한다. Data Field의 Type 속성은 Data Type 테이블을 참조하고 Parent 속성은 Complex Type 테이블을 참조한다. Data Type 테이블 집합은 모든 Data Type을 포함하며 Data Field 테이블에서 특정 레코드의 Type을 참조하기 위해 Data Type 테이블 집합에 속하는 모든 테이블의 ID를 검색한다. 특정 Complex Type의 하위 데이터 필드들을 참조하기 위해서는 Data Field 테이블에서 Parent 속성이 해당 Complex Type의 ID인 것을 검색한다. 마찬가지로 Enum Type의 멤버들을 참조하기 위해서는 Enum Member 테이블에서 Parent 속성이 해당 Enum Type의 ID인 것을 검색한다.

그림 3에 정의된 구조뿐만 아니라 메시지는 다양한 구조로 정의될 수 있다. 예를 들어 FDS를 하나의 Complex type으로 정의하여 사용하거나 DS1의 하위 데이터 필드 목록 중에서 관계있는 여러 가지 자료형을 하나의 Complex type으로 정의하여 DS1의 데이터 필드로 사용할 수 있을 것이다.

Ⅲ. 메시지의 데이터 구조 해석

3.1 메시지 타입 및 데이터 영역 검색

메시지를 해석하기 위해서는 먼저 해석에 필요한 Complex Type의 자료형을 데이터베이스에서 검색해야 한다. 이것은 메시지 타입의 조건식과 데이터 필드의 Key/Value 속성을 이용한다. 그리고 메시지에서 해석하려 하는 데이터 영역을 메시지 타입의 Offset 속성을 사용하여 검색한다. 이후에 메시지 타입의 Complex Type 자료형을 사용하여 해당 영역을 해석한다. 그림 4는 메시지를 수신했을 때 메시지 타입을 검색한 뒤 DataETL 함수를 호출하여 처리하는 알고리즘을 보여준다.

메시지 (Message)가 수신되면 데이터베이스의 메시지 타입 목록 (MessageTypeList)에서 조건식을 만족시키는 메시지 타입 (mType)을 찾는다. 그리고 메시지 타입의 오프셋 (Offset)을 이용하여 메시지를 해석해야할 데이터 영역 (paylaod)를 얻는다. 그리고

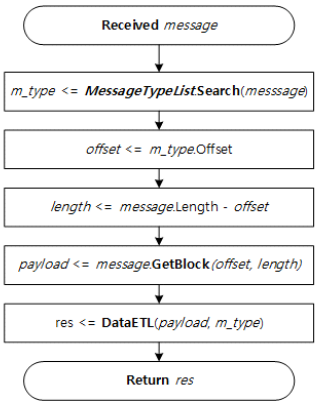


그림 4. 수신한 메시지를 처리하는 알고리즘의 순서도
Fig. 4. Flowchart of algorithm for processing received message

DataETL 함수에 그 데이터와 메시지 타입을 전달한다. 함수 이름의 DataETL은 데이터의 추출(Extraction), 변형 (Transform), 적재 (Load)를 의미하며 데이터를 분해하고 분해된 데이터들을 변형한 후 사용자에게 전달하기 위한 시스템에 적재하는 역할을 한다.

그림 3의 경우, Message Type의 Filter 속성에서 이더넷 헤더에 존재하는 VLID (Virtual Link ID) 값이 11이면 ID가 101인 Complex Type을 사용하여 UDP의 페이로드 영역을 해석하도록 GPS PVT Message라는 이름의 메시지 타입이 정의되었다.

3.2 재귀적 데이터 추출

제안하는 모델에서 모든 자료형은 자체적으로 데이터를 변형하고 다른 시스템에 적재하는 기능을 가지고 있어야한다. 이 기능은 DataType 객체의 TransformAndLoad 기능에 정의된다. Complex Type과 같이 하위 데이터 필드가 존재하는 자료형은 하위 데이터 필드의 자료형의 데이터를 추출하여 그 자료형에 대해 TransformAndLoad 기능을 수행함으로써 이 기능을 제공한다. 그리고 메시지 타입은 하나의 Complex Type의 자료형을 가져야 한다. 따라서 데이터와 그 데이터를 해석할 특정 자료형을 입력으로 가지는 DataETL 함수를 하나 선언한 후에 그것을 재귀적으로 사용하면 메시지 전체의 모든 구간에 대해 데이터를 추출한 뒤 변형과 적재를 수행할 수 있다.

그림 5는 DataETL의 알고리즘의 순서도를 보여준다. DataETL 함수는 해석할 실제 데이터 (data)와 해석에 필요한 자료형 (type)을 입력 받는다. 만약 자료형이 Complex Type이 아니라면 자료형의

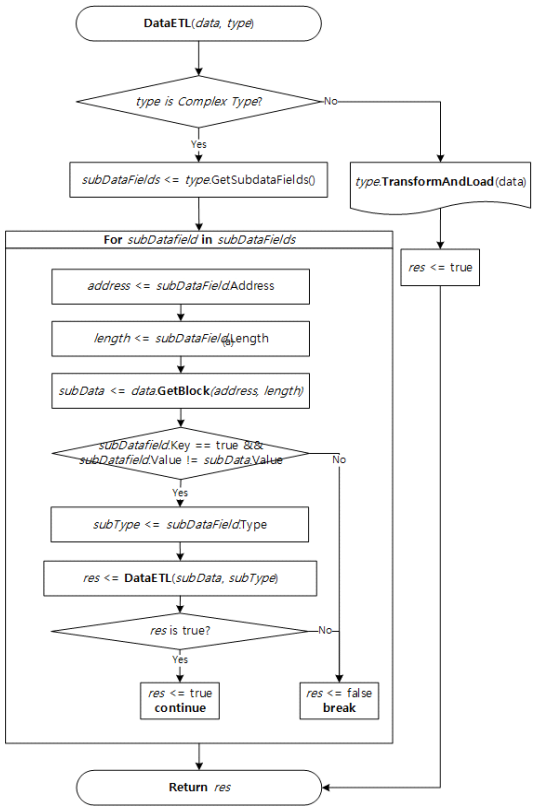


그림 5. DataETL 함수의 알고리즘의 순서도
Fig. 5. Flowchart of DataETL function algorithm

TransformAndLoad 기능에 정의된 방법대로 데이터를 해석한다. 데이터 필드의 자료형이 Complex Type이라면 하위 데이터 필드와 그에 해당하는 데이터 블록에 대해 재귀적으로 DataETL 함수를 수행하여 메시지의 모든 데이터 필드에 대해서 데이터의 해석을 진행한다. 이때 데이터 필드의 Key값이 True이고 데이터 필드의 value 값이 data의 값과 다르다면 해석을 그만두고 false를 반환하여 메시지의 해석을 중단한다.

3.3 해석된 데이터의 가공 및 표시

자료형을 통해 해석한 데이터는 그 구조와 값을 보관하기에 효율적으로 가공하고 사용자에게 정보를 전달하기에 유리한 형태로 표시되어야 한다. 그림 5의 알고리즘을 구현하기 위해 모든 자료형 객체는 이러한 역할을 수행할 TransformAndLoad 기능을 일관적인 입출력 구조를 가지도록 구현해야 한다. TransformAndLoad 기능의 입력으로는 data 외에도 구조를 구성하는데 필요한 상위 데이터 필드들 또는 전체 메시지에서 해당 data의 위치와 같은 정보들이 추가적으로 필요할 수 있으며 이러한 정보들은 TransformAndLoad 기능의

구현 범위에 따라 달라질 수 있다.

본 논문에서는 객체-관계형 모델을 사용하여 메시지 구조를 정의하였지만 메시지 구조는 사용자 친화적인 어플리케이션을 통해서 기존의 계층적인 트리 형태나 메시지 구조를 나타내는 그림으로 표현하는 것이 정보 제공에 유리하다. 트리 형태로 정보를 제공하기 위해서 모든 자료형 객체들의 TransformAndLoad 기능은 메시지의 구조를 이루는 데이터 필드들을 기능이 구현되는 어플리케이션의 자료 체계로 가공하고 상위 데이터 필드들과의 관계를 통해 계층적인 트리 형태의 구조로 보관하며 해당 트리의 노드들은 가공된 데이터를 정보 제공에 유리한 방식으로 표시하여 메시지의 전체적인 구조를 시각화해야 한다.

Ⅳ. 구 현

4.1 기존의 메시지 분석기

그림 6을 기존의 언어를 사용한 방식과 객체-관계형 모델을 사용한 방식을 사용하여 구현한 메시지 분석기의 구조를 보여준다. 6-(a)는 사용자가 XML, ASN.1, UML과 같은 언어로 메시지 구조를 정의한다. 그리고 그것에서 어플리케이션을 위한 코드를 생성하여 메시지 분석 어플리케이션을 빌드한다.

빌드 된 어플리케이션에는 사용자가 정의한 메시지를 분석하기 위한 ICD Dissector와 Presenter가 포함되어 있다. 6-(b)는 사용자가 Lua 같은 언어를 사용하여 메시지 해석 방법을 보여준다. Lua로 메시지를 해석하는 Dissector 함수를 정의하고 ICD Dissector에 입력하면 그것을 사용하여 Lua 인터프리터에서 메시지를 해석한 뒤 Presenter를 통해 출력한다.

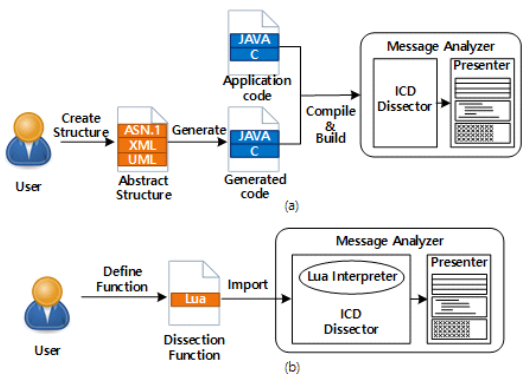


그림 6. 기존의 메시지 분석기의 구조
Fig. 6. Structure of existing message analyzer

4.2 객체-관계형 모델 기반의 메시지 분석기

그림 7은 제안하는 객체-관계형 모델을 사용한 메시지 분석기의 구조를 보여준다. 객체-관계형 모델을 관리할 데이터베이스가 필요하며 ICD Designer/Dissector가 데이터베이스에 정의된 메시지 구조를 공유한다. ICD Designer가 제공하는 직관적이고 편리한 UI를 제공해야 하며 사용자는 이것을 통해 ICD를 정의할 수 있다. 정의된 ICD는 데이터베이스에 저장된다. 메시지 분석기가 메시지를 수신했을 때 메시지 Dissector는 해당 메시지의 해석에 필요한 객체를 검색하여 메시지를 해석하고 그 결과를 Presenter를 통해 출력한다.

그림 8은 기존의 메시지 분석기에서 필요한 메시지의 추상적 구조 또는 해석에 필요한 함수를 생성하기 위해서 객체-관계형 모델을 활용하는 활용 방안을 제시한다. 사용자는 ICD Designer가 제공하는 UI를 통해 메시지를 구성하고 객체-관계형 모델로 관리하며 컴파일러는 그것을 추상적 메시지 구조 또는 Dissector 함수로 변환한다.

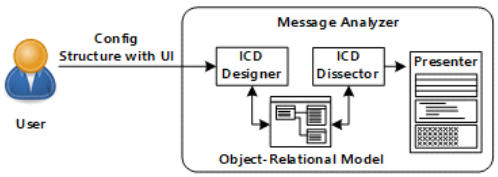


그림 7. 제안하는 객체-관계형 모델 기반의 메시지 분석기의 구조
Fig. 7. The structure of the proposed object-relational model based message analyzer

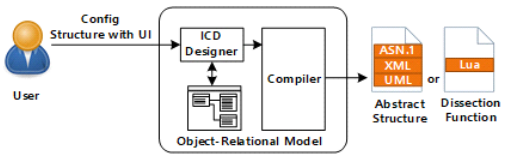


그림 8. 메시지의 추상적 구조 또는 해석에 필요한 함수를 생성하기 위한 객체-관계형 모델의 활용 방안
Fig. 8. How to use the object-relational model to generate the necessary functions for the abstract structure or interpretation of the message

4.3 구현된 어플리케이션

본 논문에서 제안하는 객체-관계형 모델링 기반의 ICD Designer/Dissector는 한국전자기술연구원에서 개발한 AFDX 네트워크 진단 어플리케이션을 기반으로 구현되었다. 구현 과정에서 하위 데이터 필드에 Complex Type을 포함하지 않는 Complex Type을

Pure Complex Type라는 이름으로 기존의 Complex Type과 구분하여 다른 테이블 공간에서 관리하도록 하였다.

4.3.1 Complex Type 정의 컨트롤러

그림 9는 Complex Type을 정의하기 위한 컨트롤러 UI를 보여준다. 왼쪽의 컨트롤은 전체 데이터베이스의 Complex Type의 목록, 중앙의 컨트롤은 선택한 Complex Type가 가지는 데이터 필드 목록, 오른쪽의 컨트롤은 새로운 데이터 필드를 추가하기 위한 자료형의 목록이다. 그림에서는 왼쪽의 Complex Type 목록에서 GPS PVT DS1이 선택되었고 중앙의 테이블에서는 GPS PVT DS1을 Parent 속성으로 가지는 데이터 필드 목록을 보여준다. 오른쪽 자료형 목록으로부터 새로운 데이터 필드를 선택된 Complex Type에 추가할 수 있다.

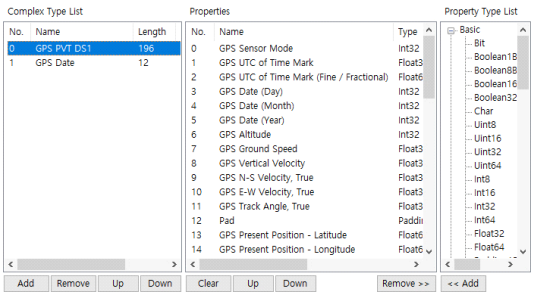


그림 9. Complex Type 자료형을 정의하기 위한 컨트롤러 UI
Fig. 9. Controller UI for defining Complex data type

4.3.2 Enum Type 정의 컨트롤러

그림 10은 Enum Type을 정의하기 위한 컨트롤러의 UI를 보여준다. 좌측의 컨트롤에서 Enum Type의 목록, 우측의 컨트롤에서 선택한 Enum Type의 멤버 목록을 보여준다. 그림에서 선택된 Enum Type은 그림 2의 AFDX 메시지 구조에 사용되는 Functional Status Enum Type이 정의된 것이다.

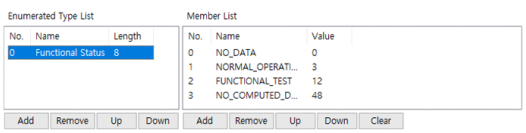


그림 10. Enum Type 자료형을 정의하기 위한 컨트롤러 UI
Fig. 10. Controller UI for defining Enum data type

4.3.3 메시지 타입 정의 컨트롤러

그림 11은 메시지 타입을 정의하기 위한 컨트롤러의 UI를 보여준다. 오른쪽의 Complex Type 목록에서 메시지의 해석 방법이 정의된 Complex Type을 선택하여 추가한 뒤 왼쪽 테이블에서 직접 다른 속성들을 수정한다. 그림에는 GPS PVT 1HZ라는 이름으로 GPS PVT Data를 자료형으로 사용한 메시지 타입이 정의되었다. 그리고 이 메시지 타입은 VLID가 1001인 AFDX 메시지의 UDP Payload를 해석하는데 사용된다.

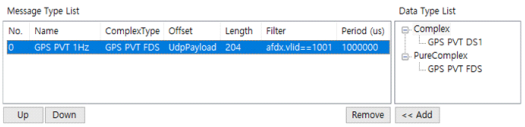


그림 11. 메시지 타입을 정의하기 위한 컨트롤러의 UI
Fig. 11. Controller's UI for defining Message types

4.3.4 메시지의 해석 결과를 표시하는 컨트롤러

메시지의 해석과 표시는 3장의 기능과 알고리즘을 네트워크 진단 어플리케이션 내에 구현하였고 이 어플리케이션을 통해 수집된 패킷을 해석하여 그림 12와 같이 트리 구조의 계층 형태로 표시하도록 하였다. 그림 12에서는 VLID가 1001인 AFDX 메시지가 UDP Payload영역부터 GPS PVT 자료형을 사용하여 해석된 것을 보여준다. 트리의 각 노드에는 데이터 필드의 이름과 자료형 객체를 통해 처리한 내용이 단위와 함께 표시된다.

그림 13은 그림 9의 GPS PVT DS1을 또 다른 구조로 정의한 것을 보여준다. 그림 9의 데이터 필드 목록에서는 GPS Date (Day), GPS Date (Month), GPS Data (Year)라는 이름의 데이터 필드가 각각 32bit의 Integer 자료형으로 구성되어 있다. 그림 13-(a)에서는 이 데이터 필드들을 GPS Date라는 Complex Type의

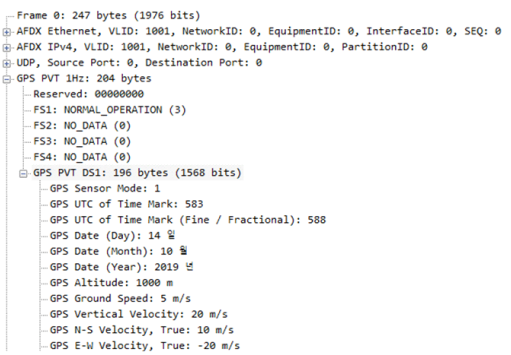


그림 12. 진단 어플리케이션 내에 표시된 GPS PVT 메시지
Fig. 12. GPS PVT messages displayed within diagnostic applications

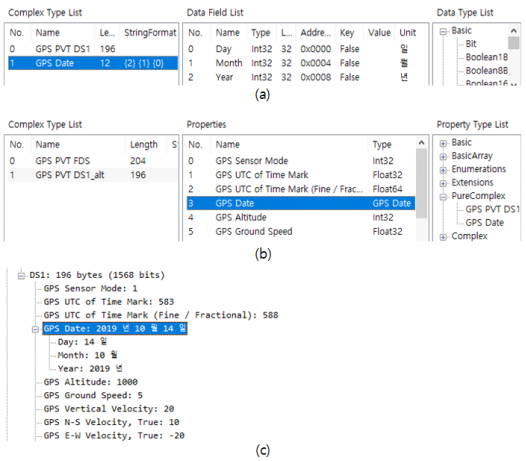


그림 13. GPS Date Complex Type 자료형을 사용하여 구성한 GPS PVT DS1과 그것을 해석한 모습
Fig. 13. GPS PVT DS1 constructed using GPS Date Complex data type and its interpretation

자료형을 새롭게 정의하였다. 이 Complex Type의 자료형은 12Byte의 길이를 가지게 된다. String Format은 각각의 데이터 필드 문자열이 역순으로 나열되도록 설정되었다. 그림 13-(b)에서는 GPS Date 자료형이 GPS PVT DS1의 데이터 필드에 있던 GPS Date와 관련된 데이터 필드들을 대체하였다. 그리고 그림 13-(c)는 GPS PVT 메시지가 새롭게 정의된 ICD 구조에 의해 해석된 결과를 보여준다. GPS Date (Day), GPS Date (Month), GPS Data (Year) 노드들이 GPS Date 노드의 하위 노드로 나타난다.

V. 구현 결과 및 분석

표 5는 기존의 방법들로 구현된 메시지 분석기들과 제안하는 방법으로 구현된 메시지 분석기의 차이점을 비교하여 보여준다. 방법 A는 그림 6-(a)와 같이 XML, ASN.1 등을 사용하여 메시지 구조를 정의한 뒤 코드를 생성하고 메시지 분석기와 함께 빌드하는 방법을 가리키고 방법 B는 그림 6-(b)과 같이 Lua를 사용하여 Dissector Function을 작성하여 메시지 분석기에 입력하는 방법을 가리킨다. 그리고 제안한 방법은 제안하는 객체-관계형 모델을 사용하여 4장에서 구현된 어플리케이션을 사용하는 방법을 가리킨다.

먼저 방법 A는 메시지 구조를 추상화하기 위해 XML을 사용하고 방법 B는 바이트 배열로부터 메시지를 분석하는 Dissector Function을 작성하기 위해 Lua를 사용한다. 반면에 제안한 방법은 사용자가 어떠한 컴퓨터 언어도 구사할 필요가 없으므로 사용자의 편의성과 접근성이 높다.

표 5. 구현 방법들의 비교
Table 5. Comparison of implementation methods

	방법 A	방법 B	제안한 방법
요구되는 언어	1개 (XML)	1개 (Lua)	없음
코드 수정	어플리케이션 수정	플러그인 수정	필요 없음
Dissector 알고리즘	분석기에 포함	구현 필요	분석기에 포함
Designer 도구	별도의 편집 도구 활용 (XML, MS Access)	별도의 편집 도구 활용 (Lua, Code Editor)	전용 GUI Designer 제공
참조 무결성	보장하지 않음	보장하지 않음	보장
절차 단계 수	4	3	2

방법 A으로 구현된 기존의 한 시스템에서는 사용자가 XML 파일을 직접 작성하는 대신 추가적인 도구를 사용하는 방법이 사용되기도 하였는데 그것은 MS Access를 통해 데이터 구조를 정의하고 DB파일로 저장한 뒤 그것을 다시 별도의 도구를 사용하여 XML파일을 생성하는 것이다. 이 방법에서는 사용자가 XML를 구사해야할 필요는 없어졌지만 MS Access라는 도구와 DB파일을 XML파일로 변환하는 도구를 추가적으로 사용해야한다.

또한 방법 A는 생성한 추상화 파일을 분석기 어플리케이션에 적용하기 위해서 어플리케이션을 다시 빌드하여 수정해야 한다. 그리고 방법 B는 작성한 Lua를 어플리케이션에 플러그인으로 적용시키기 위한 코드 작업과 플러그인 삽입 작업이 필요하다.

한편 각 방법에 따라 Dissector 알고리즘의 구현 방법에도 차이가 있다. 방법 B는 사용자가 Lua를 사용하여 바이트 배열을 분해하고 Presenter에 표시하는 Dissector 알고리즘을 Dissector 함수에 정의한다. 반면에 방법 A와 제안한 방법은 메시지의 구조만을 메시지 분석기가 요구하는 규칙에 따라 정의하며 Dissector 알고리즘은 메시지 분석기에 포함되어있다. 특히 제안한 방법은 3장의 해석 알고리즘을 통해 Dissector 알고리즘이 구현된다. 따라서 방법 A와 C는 사용자가 Dissector 알고리즘을 구현할 필요가 없으나 Dissector 구현할 수 있는 범위가 방법 B에 비하여 제한적일 수 있다.

그리고 방법 A와 B는 메시지 구조 또는 Dissector Function을 정의하기 위해 별도의 편집기나 개발 도

구를 사용해야한다. 그러나 제안한 방법은 그림 9~11과 같이 메시지 구조를 정의하기 위한 전용 GUI Designer를 구현된 어플리케이션에서 제공한다.

또한 각 방법에 따라 참조 무결성의 보장 여부의 차이가 있다. 방법 A와 B 그리고 제안한 방법 모두에서 메시지 구조 또는 Dissector 함수는 파일로 저장되어 보관된다. Lua의 경우에는 그림 14와 같은 Dissector 함수 하나를 정의하기 위해 하나의 파일을 생성해야하며 정의한 Dissector 함수는 다른 Dissector 함수 내에서 참조되어 사용될 수 있다. A, B의 경우 메시지의 종류가 많거나 메시지의 구조가 복잡하여 데이터 필드가 다양할 경우에 파일은 여러 개로 분할될 수 있거나 분할해야하며 이때 서로 다른 파일 간에 데이터를 참조하는 경우가 발생한다.

여러 개의 파일이 서로의 데이터를 복잡하게 참조하는 경우에 사용자는 많은 양의 데이터를 관리하기가 어려워진다. 특히 메시지 구조의 특정 데이터 필드의 이름이나 크기가 수정되었을 때 해당 데이터를 참조하는 또 다른 데이터 필드나 메시지들이 모두 수정되어야 하는 경우가 발생할 수 있다. 이때 수정 작업이 제대로 이루어지지 않으면 데이터의 참조 무결성이 위반되어 메시지 Dissector 기능이 올바르게 수행되지 않는다.

```
-- Create a protocol
p_gpspvt = Proto ("gpspvt", "GPS PVT")

-- Define Fields
local f = p_gpspvt.fields
f.rsvd0 = ProtoField.uint32("gpspvt.rsvd0", "RESERVED", base.HEX, nil, 0x00)
f.fs1 = ProtoField.uint8("gpspvt.fs1", "FS1", base.HEX)
f.fs2 = ProtoField.uint8("gpspvt.fs2", "FS2", base.HEX)
f.fs3 = ProtoField.uint8("gpspvt.fs3", "FS3", base.HEX)
f.fs4 = ProtoField.uint8("gpspvt.fs4", "FS4", base.HEX)
f.gps_sen_mod = ProtoField.int32("gpspvt.gps_sen_mod", "GPS SENSOR MODE", base.DEC)

...

(중략)

...


-- Define the dissector function
function p_gpspvt .dissector (buffer, pinfo, tree)
  if buffer:len() == 0 then return end

  subtree = tree:add(p_gpspvt , buffer(0))
  subtree:add(f.rsvd0, buffer(0, 4))
  subtree:add(f.fs1, buffer(5, 1))
  subtree:add(f.fs2, buffer(6, 1))
  subtree:add(f.fs3, buffer(7, 1))
  subtree:add(f.fs4, buffer(8, 1))
  subtree:add(f.gps_sensor_mode, buffer(9, 4))

...

(중략)

...

  pinfo.cols.protocol = p_tpcstream.name
  pinfo.cols.info = "GPS PVT MESSAGE"

end
```

그림 14. Lua로 Wireshark를 위한 GPS PVT 메시지의 Dissector Function을 작성한 모습의 예
Fig. 14. An example of creating a Dissector Function of GPS PVT message for Wireshark with Lua

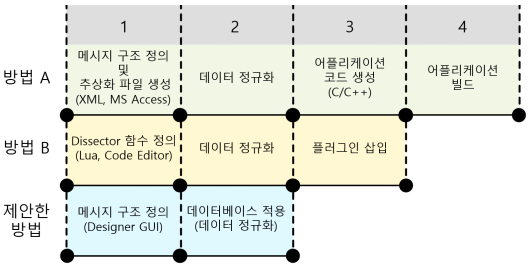


그림 15. 각 방법 별 메시지 구조 또는 Dissector 함수 작성 후 적용 절차
Fig. 15. An example of creating a Dissector Function of GPS PVT message for Wireshark with Lua

방법 A와 B가 이러한 데이터의 참조 무결성에 대한 데이터 및 파일의 관리를 사용자에게 전부 부담시키는 반면에 제안한 객체-관계형 모델을 사용하는 방법은 하나의 파일 또는 데이터베이스에 많은 메시지 구조를 정의할 수 있으며 서로의 참조 관계 또한 명확하게 정의하고 저장한다. 그리고 구현된 어플리케이션은 하나의 객체에 수정이 발생하는 경우 데이터베이스 내에서 참조 관계를 모두 검색하여 해당되는 모든 객체에 변경사항을 적용하는 데이터 정규화를 진행한다. 따라서 제안한 방법에서 구현된 어플리케이션은 데이터의 참조 무결성을 보장한다.

그림 15는 각 방법에서 메시지 구조를 정의하거나 Dissector 함수를 작성한 후에 메시지 분석기에 적용되기까지의 절차를 도식화 한 것이다. 메시지 구조 정의를 위한 Designer와 해석을 위한 Dissector를 어플리케이션에서 직접 제공하고 데이터 정규화를 어플리케이션이 자동으로 진행하면서 방법 A와 B보다 절차의 단계수가 감소하였다.

Ⅵ. 결 론

본 논문에서는 복잡한 메시지 구조를 정의하기 위한 객체-관계형 모델을 제안하였고 제안한 모델을 사용하여 ICD Designer/Dissector를 전자부품연구원에서 개발한 ADB 네트워크 진단 어플리케이션을 기반으로 구현하였다. 그리고 AFDX 네트워크에서 사용하는 ADN 메시지를 정의하고 해석함으로써 제안한 객체-관계형 모델과 ICD Designer/Dissector의 기능을 검증하였다.

제안한 객체-관계형 모델을 통한 ADN 메시지 및 ICD 정의는 기존의 자료형을 가지고 사용자 정의 자료형을 구성하기에 적합하며 복잡한 계층 구조의 메시지 구조를 쉽게 정의하고 효율적으로 관리할 수 있

다. 그리고 해석하는 알고리즘이 일관적이고 간단한 구조를 가지며 Dissector가 제안하는 알고리즘을 통해 해석 기능을 제공한다면 사용자는 ICD를 정의할 때마다 해석을 위해 추가적인 작업을 수행 하지 않아도 된다.

그리고 본 논문에서는 객체-관계형 모델을 사용하여 기존의 메시지 분석기의 불편함을 해소한 메시지 분석기의 구조를 제안하였다. 제안하는 구조의 메시지 분석기는 Designer/Dissector가 객체-관계형 모델을 관리하는 기능과 함께 하나의 어플리케이션 내에서 구현되어 사용자에게 메시지 구조 정의를 위한 직관적이고 편리한 인터페이스를 제공하며 제안한 해석 알고리즘을 통해 기존의 트리 형태의 계층구조로 해석한 메시지를 시각화한다. 또한 데이터 정규화를 통해 객체-관계형 모델 통해 정의된 메시지 구조와 관련된 데이터들의 참조 무결성을 보장할 수 있다. 이 방식으로 구현한 메시지 분석기는 사용자가 시스템 개발 중 잦은 ICD 수정에 대해 효율적으로 대처할 수 있도록 한다.

References

[1] H. R. Kim and C. T. Lim, "The design and implementation of an enhanced ASN. 1 compiler for open system application" *The IEEK* - A, vol. 33, no. 3, pp. 398-407, Mar. 1996.

[2] Y. Kim, S. Kim, J. Choi, C.-W. Yoo, and D. Lee, "Design and implementation of ASN.1 development environment for source code generation of network management," *J. KISS(C): Computing Practices*, vol. 4, no. 6, pp. 826-835, Dec. 1998.

[3] H.-I. Seo and E.-G. Kim, "Design and implementation of IEEE Std 1609.2 message encoder/decoder for vehicular communication security," *J. KIICE*, vol. 21, no. 3, pp. 568-577, Mar. 2017.

[4] A. Sasikumar, N. Prem, and J. V. Satyanarayana, "Avionics ICD representation using JSON for configurable data analyzer," *2019 ICORT*, pp. 1-5, Balasore, India, Feb. 2019.

[5] Ulf Lamping, *Wireshark Developer's Guide. Version 3.3.0, Chapter 11. Wireshark's Lua API Reference Manual*, https://www.wireshark.org/download/docs/developer-guide.pdf.

[6] T. Shimura, M. Yoshikawa, and S. Uemura, "Storage and retrieval of XML documents using object-relational databases," *Int. Conf. Database and Expert Syst. Appl.*. pp. 206-217, Springer, Berlin, Heidelberg, Aug. 1999.

[7] J. Song and W. Kim, "Extensible indexing technique for efficient storage and extraction of XML documents using relational databases," *IEEK - IE*, vol. 41, no. 4, pp. 93-102, Dec. 2004.

[8] Aircraft Data Network, "*Part 7, Avionics Full-Duplex Switched Ethernet Network,*" ARINC, 2009.

손 명 환 (Myeong-hwan Son)

2017년 2월 : 한국항공대학교 항공전자 및 전자공학과 학사 졸업

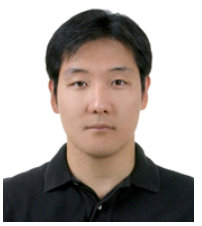
2019년 2월 : 한국항공대학교 항공전자 및 전자공학과 석사 졸업

2019년 2월~현재 : 전자부품연구원 모빌리티플랫폼연구센터 연구원

<관심분야> TSN (Time Sensitive Network), 항공 및 차량 네트워크

[ORCID:0000-0001-7298-5927]

박 부 식 (Pu-sik Park)



1999년 2월 : 한국항공대학교 항공통신정보공학 학사 졸업

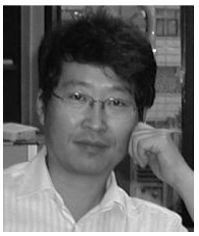
2001년 8월 : 한국항공대학교 정보통신공학 석사 졸업

2002년 1월~현재 : 전자부품연구원 모빌리티플랫폼연구센터 책임연구원

<관심분야> TSN (Time Sensitive Network), AFDX (Avionics Full-Duplex Switched Ethernet), AeroRing, TCN (Train Communication Network), Seamless Redundancy, 항공IT융합, 철도IT융합 등

[ORCID:0000-0002-4308-5708]

윤 종 호 (Chong-ho Yoon)



1984년 : 한양대학교 전자공학과 학사 졸업
1986년 : 한국과학기술원 전기 및 전자공학과 석사 졸업
1990년 : 한국과학기술원 전기 및 전자공학과 박사 졸업
1991년 9월~현재 : 한국 항공대학교 항공전자 및 정보통신공학부 교수
2000년 5월~현재 : 한국이더넷포럼 부의장
<관심분야> 차량용 이더넷, TSN(Time Sensitive Network), 시간동기