IEC 61850 통신 설명 기술 리포트

Description of IEC 61850 Communication Technical Report

Petr Matoušek, Faculty of Information Technology

한국어 번역본

초록

IEC 61850은 산업 통신 시스템(ICS), 특히 전력 시스템 통신을 위한 새로운 국제 표준이다. 이 표준은 추상 통신 서비스 인터페이스(ACSI)를 통해 접근되는 추상 객체(논리 노드, 데이터 객체)를 사용하여 시스템을 기술한다. 디바이스와 제어 스테이션 간의 통신은 MMS(Manufacturing Message Specification) 프로토콜을 이용한 클라이언트-서버 통신 또는 GOOSE(Generic Object-Oriented Substation Event) 프로토콜을 이용한 피어-투-피어 시스템으로 설계된다. 본 문서는 IEC 61850 표준에서 권장하는 시스템의 추상 모델과 GOOSE 및 MMS 통신 프로토콜 모두를 기술한다. 본 논문의 의도는 보안 모니터링에 초점을 맞추는 것이므로, 두 프로토콜 모두에 대한 상세한 설명이 제시된다.

1. 서론

Modbus, IEC 60870-5-101 또는 DNP3에서 구동되는 기존의 직렬 기반 SCADA 시스템은 현대적인 지능형 전자 장치(IED)의 차세대 기능을 지원하기에 충분하지 않다. IP 기반 프로토콜 변환 서비스를 사용하더라도 배포의 유연성이 부족하며, 궁극적으로는 RTU(원격 단말 장치)에서의 노후화된 직렬 통신에 의존하게 된다. 변전소 통신을 현대화하고 이더넷 및 IP의 이점을 활용할 수 있는 프로토콜을 도입하기 위해, IEC 기술 위원회 57(TC 57)은 IEC 61850 표준을 개발하였다.

IEC 61850은 전력 변전소 내의 지능형 전자 장치(IED)를 위한 통신 프로토콜을 정의하는 국제 표준이다. 이는 전력 시스템을 위한 국제 전기 기술 위원회(IEC) 기술 위원회 57 참조 아키텍처의 일부이다. IEC 61850에 정의된 추상 데이터 모델은 다수의 프로토콜에 매핑될 수 있다. 현재 표준에서의 매핑은 MMS(Manufacturing Message Specification), GOOSE(Generic Object Oriented Substation Event) 또는 SMV(Sampled Measured Values)로 이루어진다.

- MMS 프로토콜 (IEC 61850-8-1): IP를 통한 클라이언트/서버 통신을 지원하며 SCADA에 사용된다. 이는 모니터링 목적으로 사용된다.
- GOOSE (IEC 61850-8-1): IED 간 및 베이(Bay) 간 통신이 가능한 이더넷 기반 멀티캐스트(일대다) 통신을 사용한다. GOOSE는 전력 측정값을 전달하고 보호 계전기 간 통신뿐만 아니라 트립(tripping) 및 인터록(interlocking) 회로에도 자주 사용된다. 이는 상태 업데이트 및 제어 명령 요청 전송에 사용된다.
- 샘플링된 측정값 (SMV) (IEC 61850-9-2): 전력선의 전류 및 전압 값을 전달한다. SMV의 일반적인 용도는 모선(bus-bar) 보호 및 싱크로페이저(synchrophasor)[1]이다.

이러한 프로토콜들은 TCP/IP 네트워크 또는 고속 스위칭 이더넷을 사용하는 변전소 LAN 상에서 실행될 수 있으며, 이를 통해 보호 계전(protective relaying)에 필요한 4밀리초 미만의 응답 시간을 확보할 수 있다.

IEC 61850 프로토콜은 모든 보호, 제어, 측정 및 모니터링 기능을 하나의 공통 프로토콜로 통합할 수 있게 한다. 이는 고속 변전소 애플리케이션, 변전소 전체의 인터록 및 IED 간 상호 통신이 필요한 기타 기능들을 위한 수단을 제공한다. 잘 기술된 데이터 모델링과 변전소 내 최신 작업을 위해 명시된 통신 서비스는 이 표준을 현대 변전소 시스템의 핵심 요소로 만든다.

1) 싱크로페이저(Synchrophasors)는 위상과 전력을 모니터링하는 시간 동기화된 전기적 수치이다. 이들은 변전소 내의 위상 측정 장치(PMU)라고 불리는 장치에 의해 측정된다 [4].

2. IEC 61850 표준

IEC 61850 표준은 그림 1과 같이 10개의 주요 부분(Section)으로 변전소 통신 네트워크의 다양한 측면을 정의한다. 61850 표준의 아키텍처는 기반 프로토콜과 독립적인 데이터 항목/객체 및 서비스를 생성함으로써 데이터 항목과 서비스의 정의를 추상화한다. 이러한 추상적 정의는 데이터 및 서비스 요구사항을 충족할 수 있는 다른 모든 프로토콜로 데이터 객체와 서비스를 매핑할 수 있게 한다

표준의 파트 6은 IED와 관련된 변전소 내 통신을 위한 구성 기술 언어(CDL)를 기술한다. 파트 7은 기본 통신 구조를 기술한다. 이는 다음을 포함한다.

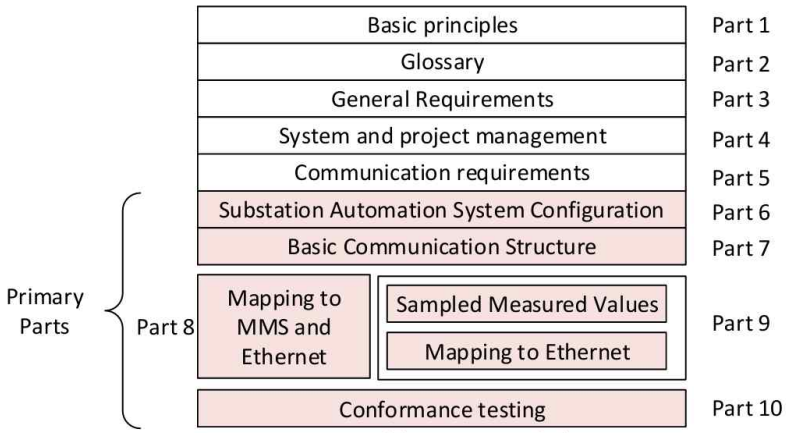


Figure 1: Structure of the IEC 61850 Standard

파트 7.2의 추상 서비스 정의와 파트 7.4의 논리 노드(Logical Node)라고 불리는 데이터 객체의 추상화를 포함한다. 공통 객체의 정의는 파트 7.3에 있다. 표준 파트 7의 구조는 다음과 같다.

- IEC 61850-7-1: 원칙 및 모델
- IEC 61850-7-2: 추상 통신 서비스 인터페이스 (ACSI)
- IEC 61850-7-3: 공통 데이터 클래스 (CDC)
- IEC 61850-7-4: 호환성 있는 논리 노드 클래스 및 데이터 클래스

데이터 및 서비스의 추상적 정의가 주어지면, 마지막 단계는 추상 서비스를 실제 프로토콜로 매핑하는 것이다. 파트 8은 다음을 포괄하는 특정 통신 서비스 매핑(SCSM)을 기술한다.

IEC 61850-8-1: MMS(ISO/IEC 9506 – 파트 1 및 2) 및 ISO/IEC 8802-3에 대한 매핑

파트 9 또한 다음에 대한 특정 통신 서비스 매핑을 정의한다.

- IEC 61850-9-1: 직렬 단방향 멀티드롭 점대점 링크 및 이더넷 데이터 프레임 상의 양방향 멀티포인트를 통한 샘플링된 값
- IEC 61850-9-2: ISO/IEC 8802-3 (프로세스 버스) 상의 샘플링된 값

변전소 자동화 시스템과 변전소(스위치야드) 간의 관계는 파트 6에 정의된 XML 기반의 변전소 구성 언어(SCL)로 공식적으로 기술된다.

2.1. IEC 61850 정보 모델

IEC 61850의 정보 모델은 물리 디바이스, 논리 디바이스, 논리 노드 및 데이터 객체로 구성된다(그림 2 참조).

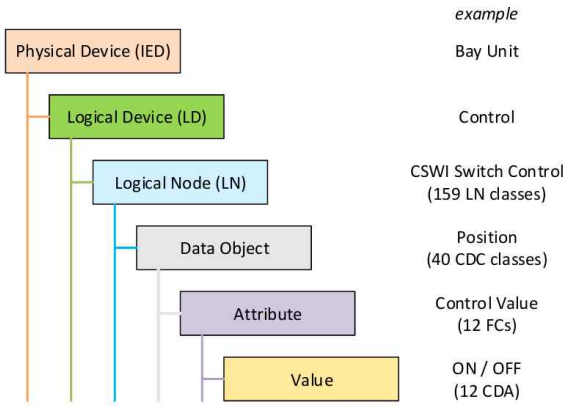


Figure 2: IEC 61850 information model

물리 디바이스는 논리 디바이스로 모델링되는 다양한 기능 모듈을 포함한다. 각 논리 디바이스는 논리 노드로 정의된 다양한 작업을 제공할 수 있다. IEC 61850-7-4 표준은 159개의 고유한 논리 노드 클래스를 정의한다. 논리 노드는 애플리케이션 기능을 나타내는 데이터 객체를 포함한다. 논리 노드의 변수는 공통 데이터 클래스(CDC)의 집합으로 표현된다. IEC 61850-7-3 표준은 40개의 서로 다른 CDC를 정의한다(부록 B 참조). 각 데이터 객체는 12가지 기능 제약(Functional Constraints)에 속하는 데이터 속성이라 불리는 요소들의 집합을 포함한다(부록 C3 참조). 속성은 공통 데이터 속성(CDA)에 의해 정의된 값을 포함한다.

2.1.1 물리 디바이스 (PD)

IEC 61850 디바이스 모델은 물리 디바이스(예: 계전기 또는 변전소)에서 시작된다. 따라서 물리 디바이스는 네트워크에 연결되는 장치이다. 물리 디바이스는 네트워크 주소에 의해 정의된다. 물리 디바이스는 때때로 IED(지능형 전자 장치)라고 불린다.

2.1.2 논리 디바이스 (LD)

각 물리 디바이스 내에는 하나 이상의 논리 디바이스가 존재할 수 있다. 논리 디바이스는 여러 디바이스의 데이터를 단일 물리 디바이스로 집계한다.
각 논리 디바이스는 다음 속성들을 포함한다.

- LDName: 네트워크상에서 논리 디바이스를 고유하게 정의한다.
- LogicalNode[1..n]: 논리 디바이스의 일부인 모든 논리 노드의 목록이다. 각 LD는 반드시 하나의 논리 노드 제로(LLN0)를 포함해야 한다. 이는 0개 이상의 논리 노드를 포함할 수 있다.
- GetLogicalDeviceDirectory: 클라이언트가 논리 노드에 접근할 수 있도록 모든 논리 노드의 RefObjects 목록을 반환하는 서비스이다.

각 논리 디바이스는 하나 이상의 논리 노드를 포함한다.

2.1.3 논리 노드 (LN)

IEC 61850은 변전소 장비(변압기, 차단기, 보호 기능 등) 내의 각 기능에 논리 노드를 할당한다. 논리 노드는 디바이스의 가상 표현이다. 이는 특정 변전소 기능과 관련된 데이터 및 서비스의 그룹이다. 따라서 변전소에서 생성된 모든 데이터는 특정 논리 노드에 할당될 수 있다. 표준에서 논리 노드는 데이터를 교환할 수 있는 가장 작은 개체로 명시된다.

논리 노드는 기능을 기반으로 그룹으로 결합된다. 자동 제어, 계량 및 관리, 감시 제어 등을 위한 논리 노드들이 존재한다. 표준은 159개의 고유 클래스를 가진 LN 클래스를 포함하는 19개의 서로 다른 LN 그룹을 정의한다(부록 A 참조).

특별한 그룹으로 시스템에 특화된 정보를 포함하는 시스템 논리 노드 그룹(L)이 있다. 여기에는 공통 논리 노드 정보(공통 LN 클래스, 예: LN 동작, 명판 정보, 동작 카운터)와 특정 하드웨어 관련 정보(물리 디바이스 정보 - LPHD 클래스)가 포함된다.

공통 LN 클래스는 다른 모든 LN 클래스에 대해 필수적이거나 조건부적인 데이터 객체를 제공한다. 또한 입력 참조, 통계 데이터 객체 등 다른 모든 LN 그룹에서 사용될 수 있는 데이터를 포함한다. 공통 LN 클래스의 구조는 그림 3에 묘사되어 있다.

Common LN Class			
Data Object Name	**Common data class**	**Explanation**	**Mandatory**
Data objects			
Mandatory and conditional LN information (shall be inhereted by ALL LN but LPHD)			
Description			
NamPlt	LPL	Name plate	C1
Status			
Beh	ENS	Behavior	M
Health	ENS	Physical status	C1
Blk	SPS	Dynamic function blocking	O
Measures			
SumSwARs	BCR	Sum of switched amperes, resetable	
Controls			
Mod	ENC	Mode	C1
CmdBlk	SPC	Control sequence blocking and activation of remote data objects	C2
Settings			
InRef1	ORG	Common input reference	O
BlkRef1	ORG	Blocking reference	O
Condition C1: Mod, Health and NamPlt shall LLN0 take from LD as mandatory, all other LNs as optional			
Condition C2: CmdBlk must be taken with Mod as optional data object by all LNs with control data objects			

Figure 3: Common LN class (without statistical LN information)

논리 디바이스 내에는 최소한 3개의 논리 노드가 존재해야 한다. 즉, 논리 디바이스의 공통적인 사안과 관련된 두 개의 LN(논리 노드 제로 LLN0 및 물리 디바이스 정보 LPHD)과, 어떤 기능을 수행하는 최소한 하나의 LN이 있어야 한다. 논리 노드의 전체 목록은 [2]에 정의되어 있다.

특수 L-그룹 클래스:

- 논리 노드 제로 (LLN0) - 자신이 속해 있는 가상 디바이스를 관리한다. 특히 통신 객체와 가상 디바이스의 로그를 정의한다.
- 물리 디바이스 논리 노드 (LPHD1) - 물리 디바이스를 나타내며, 특히 모든 논리 디바이스에 대해

동일한 통신 속성을 나타낸다.

모든 논리 노드는 일반 논리 노드 클래스(Generic Logical Node Class) 템플릿에 따라 구성된다(그림 4 참조).

GenLogicalNode class		
Attribute	**Attribute Type**	**Explanation**
LNName	ObjectName	Instance name unambiguously identifying the logical node within the scope of logical device, e.g., XCBR1
LNRef	ObjectReference	Unique path-name of the logical node: LDName/LNName, e.g., Q1B1W2/XCBR1
DataObject [1..n]	GenDataObjectClass	All data objects contained in the logical node.
DataSet [0..n]	DATA-SET	All DataSets contained in the logical node.
BufferedReportControlBlock [0..n]	BRCB	All buffered report control blocks contained in the logical node.
UnbufferedReportControlBlock [0..n]	URCB	All unbuffered report control blocks contained in the logical node.
LogControlBlocks [0..n]	LCB	All log control blocks contained in the logical node.
Only for LLN0		
SettingGroupsControlBlock [0..1]	SGCB	Setting group control block contained in the logical node.
Log [0..n]	LOG	All logs contained in the logical node.
GOOSEControlBlock [0..n]	GoCB	All GOOSE control blocks contained in the logical node.
MulticastSampledValues [0..n]	MSVCB	All multicast sampled value control blocks contained in the logical node.
UnicastSampledValues [0..n]	USVCB	All unicast sampled value control blocks contained in the logical node.
Services		
GetLogicalNodeDirectory		
GetAllDataValues		

Figure 4: Logical Node Model [3]

동일한 논리 이름을 가진 인스턴스가 두 개 존재하는 경우, 이들은 LN 뒤에 붙는 숫자로 구별된다. 예를 들어, 측정 클래스의 논리 이름이 MMXU라면 그 인스턴스들의 이름은 MMXU1과 MMXU2가 된다 (그림 5 참조). 또한 각 논리 노드는 해당 논리 노드의 목적을 추가적으로 식별하기 위해, 애플리케이션에 특화된 선택적 LN-접두사(LN-prefix)를 사용할 수도 있다.

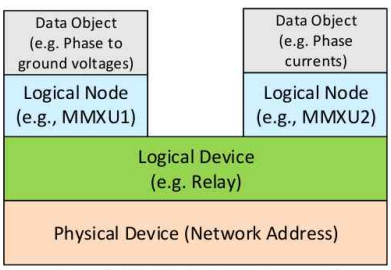


Figure 5: IEC 61850 Information Model

2.1.4 데이터 객체 (DO)

논리 노드는 애플리케이션(변전소) 객체를 나타내는 데이터 객체를 포함한다. 각 데이터 객체는 고유한 이름을 가진다. 이러한 데이터 이름은 표준에 의해 결정되며 기능적으로 전력 시스템의 목적과 관련이 있다.

주어진 논리 노드와 관련된 데이터 객체의 집합은 IEC 61850-7-3 표준에 의해 정의된다. 이 표준은 데이터 객체의 집합을 특정 클래스에 할당하는 40개의 공통 데이터 클래스(CDC, 부록 B 참조)를 기술한다. 예를 들어, 차단기는 XCBR 논리 노드로 모델링된다. 이는 동작이 원격인지 로컬인지 결정하기 위한 Loc(SPS 클래스, 단일 포인트 상태), 동작 횟수를 위한 OpCnt(INS 클래스, 정수 상태), 위치를 위

한 Pos(DPC 클래스, 제어 가능한 이중 포인트), 차단기 개방 명령 차단을 위한 BlkOpn(SPC 클래스, 제어 가능한 단일 포인트), 차단기 투입 명령 차단을 위한 BlkCls(SPC 클래스), 또는 차단기 동작 능력을 위한 CBOpCap(ENS 클래스, 제어 가능한 열거형 상태)을 포함한 다양한 데이터 객체를 포함한다(그림 6 참조).

XCBR class			
Data Object Name	**Common data class**	**Explanation**	**Mandatory**
LLName		The name shall be composed of the class name, the LN-Prefix and Ln-Instance-ID according to IEC 61850-7-2, Clause 22	
Data objects			
Description			
EEName	DPL	External equipment name plate	
Status			
EEHealth	INS	External equipment health	O
LocKey	SPS	Local or remote key (local means without substation automation communication, hardwired direct control)	O
Loc	SPS	Loc control mode	M
OpCnt	INS	Operation counter	M
CBOpCap	ENS	Circuit breaker operating capability	O
POWCap	ENS	Point on wave switching capability	O
MaxOpCap	INS	Circuit breaker operationg capability when fully charged	O
Dcs	SPS	Discrepancy	O
Measures			
SumSwARs	BCR	Sum of switched amperes, resetable	
Controls			
LocSta	SPC	Switching authrority at station level	O
Pos	DPC	Switch position	M
BlkOpn	SPC	Block opening	M
BlkCls	SPC	Block closing	M
ChaMotEna	SPC	Charger motor enable	O
Settings			
CBTmms	ING	Closing time of breaker	O

Figure 6: Example of Data Objects in a Logical Node XCBR [2]

앞서 살펴본 바와 같이, 특정 LN 클래스에 대해 정의된 데이터 객체는 다음 범주로 그룹화된다.

- 설명 (Description) – LN이 나타내는 전용 기능과는 독립적인 기본 정보 (예: 명판, 건전성 등)
- 상태 (Status) – 프로세스 상태 또는 LN 기능의 상태를 나타냄 (예: 스위치 유형, 스위치 위치)
- 측정 (Measures) – 프로세스에서 측정된 아날로그 데이터 (예: 선로 전류, 전압, 전력) 또는 LN 내부에서 계산된 데이터 (예: 총 유효 전력, 순 에너지 흐름)
- 제어 (Controls) – 명령에 의해 변경되는 데이터 (예: 개폐 장치 상태(ON-OFF), 탭 절환기 위치 또는 리셋 가능한 카운터)
- 설정 (Settings) – 논리 노드 기능을 위한 파라미터 (예: 1차, 2차 또는 3차 재폐로 시간, 투입 펄스 시간)

또한 표준 61850-7-2는 주어진 논리 노드에 대해 어떤 데이터 객체가 필수(M, mandatory), 선택(O, optional) 또는 조건부(C, conditional)인지를 정의한다.

2.1.5 공통 데이터 클래스 (CDC)

앞서 언급한 바와 같이, 논리 노드 내의 각 데이터 객체는 해당 데이터가 속한 공통 데이터 클래스의 사양을 따른다. 공통 데이터 클래스(CDC)는 데이터 객체를 기술하는 데 사용되는 공통 유형에 대한 구조를 정의한다. CDC 기술(description)은 논리 노드 내 데이터의 유형과 구조를 포함한다. 각 CDC는 정의된 이름과 속성 집합을 가지며, 이 속성들은 다시 정의된 이름, 정의된 유형 및 특정 목적을 가진

다. 또한, 데이터 속성 유형은 특정 기능 제약(FC)에 속한다.

데이터 속성은 기본형(예: BOOLEAN)이거나 복합형(구성형, 예: Quality)일 수 있다(부록 C 참조).

Single Point Setting (SPS) class					
Attribute	**Attribute Type**	**FC**	**TrgOp**	**Value/Range**	**M/O/C**
status					
stVal	BOOLEAN	ST	dchg	TRUE/FALSE	M
q	Quality	ST	qchg		M
t	TimeStamp	ST			M
substitution					
subEna	BOOLEAN	SV			PICS_SUBST
subVal	BOOLEAN	SV		TRUE/FALSE	PICS_SUBST
subQ	Quality	SV			PICS_SUBST
subID	VISIBLESTRING64	SV			PICS_SUBST
configuration, description and extension					
d	VISIBLESTRING255	DC	Text		O
dU	UNICODE STRING255	DC			O
cdcNs	VISIBLESTRING255	EX			AC_DLNDA_M
cdcName	VISIBLESTRING255	EX			AC_DLNDA_M
dataNs	VISIBLESTRING255	EX			AC_DLN_M

Table 1: Example of common data class (CDC)

표 1은 단일 포인트 상태(SPS) 클래스의 예를 보여준다. SPS 클래스는 3개의 상태 속성(ST), 4개의 대체 속성(SV), 2개의 설명 속성(DC), 그리고 3개의 확장 정의(ED) 속성으로 구성된다. SPS 상태 속성은 BOOLEAN 데이터 유형의 상태 값 stVal, Quality 데이터 유형의 품질 플래그 q, 그리고 TimeStamp 데이터 유형의 타임스탬프 t를 포함한다. 표준화된 데이터 유형의 목록은 부록 C와 D에 있다.

표 1의 처음 두 열은 SPS 클래스에 포함된 속성의 이름과 유형을 정의한다. 공통 데이터 클래스의 개별 속성들은 기능 제약(FC)에 따라 범주별로 그룹화된다. 트리거 옵션 열은 데이터의 리포팅이나 읽기 등이 언제 발생하는지를 정의한다. 네 번째 열은 속성의 사전 정의된 값이나 값 범위를 기술한다. 마지막 열은 데이터 속성이 필수(M), 선택(O) 또는 조건부(C)인지를 나타낸다. 예를 들어, 표 1의 첫 번째 데이터 속성은 stVal이다. 이는 BOOLEAN 데이터 유형을 가지며 상태 속성(ST)을 위한 기능 제약에 속한다. 트리거 옵션은 데이터 변경(dchg)이며, 이 속성은 필수 항목이다.

2.1.6 명명 체계 (Naming Scheme)

논리 노드(logical node)의 이름은 표준 논리 노드 인스턴스의 이름이며, 논리 디바이스(logical device) 내에서 고유하다(예: XCBR2).

객체 참조(Object reference)는 객체의 전체 경로이며, 기능 제약(functional constraint)으로 완성된다(IEC 61850-7-2, 22절 참조). 예를 들어, 객체 참조 EA1QA5/XCBR8.Pos.ctlVal ST(그림 7 참조)는 논리 디바이스 EA1QA5와 논리 노드 XCBR(차단기)의 인스턴스인 논리 노드 XCBR8을 지칭한다. 이 특정 논리 노드(이름 EA1QA5/XCBR8)는 공통 데이터 클래스 DPC(제어 가능한 이중 포인트)에서 파생된 데이터 객체 Pos(스위치 위치, 그림 6 참조)를 포함한다. DPC 클래스는 ctlVal, stVal, q, t 또는 ctlModel과 같은 다양한 속성을 포함한다. ctlVal 속성은 FALSE(스위치 끄기) 또는 TRUE(스위치 켜기) 값을 갖는 BOOLEAN 데이터 유형이다. 이 속성은 기능 제약 ST(상태 속성)를 포함한다.

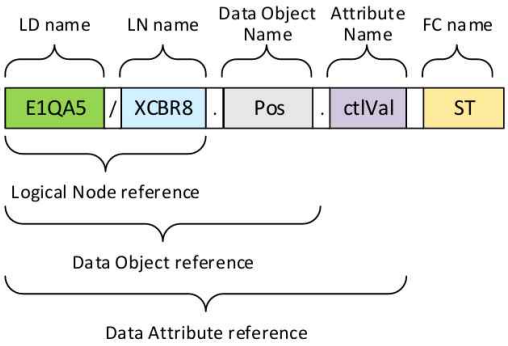


Figure 7: Example of an object reference

LN 참조, 데이터 객체 참조 또는 데이터 속성 참조는 특정 논리 노드, 해당 노드의 데이터 객체, 또는 주어진 데이터 객체의 속성을 지칭한다.

2.2 추상 통신 서비스 인터페이스 (ACSI)

IEC 61850의 추상 데이터 및 객체 모델은 모든 IED가 전력 시스템 기능과 직접적으로 관련된 동일한 구조를 사용하여 데이터를 표현할 수 있도록 하는 전력 시스템 디바이스 기술(describing)의 표준화된 방법을 정의한다. 추상 통신 서비스 인터페이스(ACSI) [3]는 다음을 위한 클라이언트와 원격 서버 간의 통신을 기술한다.

- 실시간 데이터 접근 및 검색
- 디바이스 제어
- 이벤트 리포팅 및 로깅
- 설정 그룹 제어
- 디바이스의 자기 기술(self-description) (디바이스 데이터 사전)
- 데이터 타이핑 및 데이터 유형 검색
- 파일 전송

또한 ACSI는 한 디바이스의 애플리케이션과 다른 디바이스들에 있는 다수의 원격 애플리케이션 간(게시자/구독자)의 빠르고 신뢰성 있는 시스템 전반의 이벤트 분배와, 샘플링된 측정값의 전송을 위한 추상 인터페이스를 제공한다.

ACSI 모델에는 두 그룹의 통신 서비스가 있다. 첫 번째 그룹은 IED로부터 데이터 값을 가져오는 것과 같은 클라이언트-서버 모델을 사용한다. 두 번째 그룹은 GOOSE 메시지를 사용하는 IED 간의 빠른 통신과 주기적인 샘플링된 측정값(SV) 전송에 사용되는 일반 변전소 이벤트(GSE) 서비스를 포함하는 피어 투 피어(peer-to-peer) 모델이다.

- 클라이언트-서버 통신은 클라이언트가 서버에 데이터를 요청하는 서비스이다. 서버는 논리 디바이스의 콘텐츠, 연관 모델, 시간 동기화 및 파일 전송을 포함한다. 이 클라이언트-서버 통신은 시간에 민감하지 않은(not time-critical) 대량의 데이터를 전송하는 데 사용된다.
- 샘플링된 측정값(SV)은 계측 및 측정과 관련된 메시지이다. 따라서 이들은 베이(bay) 레벨과 프로세스 레벨 사이에서 전송된다(그림 8 참조). SV 메시지는 시간에 민감하며(time-critical), 시간 순서대로 처리되어야 하고, 발생 가능한 손실이 감지되어야 한다. 이 메시지들은 하나의 수신자에게 유니

캐스트로 전송되거나 여러 수신자에게 멀티캐스트로 전송될 수 있다(표 2 참조).

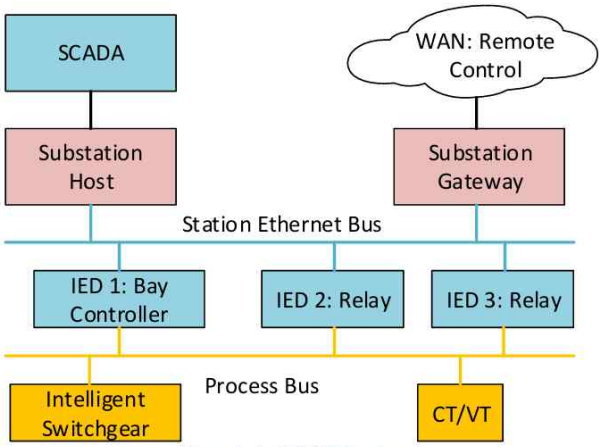


Figure 8: An IEC 61850 network

Recommended L2 multicast addresses		
Service	**Starting address**	**Ending address**
GOOSE	01:0c:cd:01:00:00	01:0c:cd:01:01:ff
GSE	01:0c:cd:02:00:00	01:0c:cd:02:01:ff
Sampled Values	01:0c:cd:04:00:00	01:0c:cd:04:01:ff

Table 2: Recommended multicast addresses [6]

- GOOSE 메시지는 IED 간의 빠른 수평 통신을 위해 정의되었다. 이들은 IED 간의 상태 및 제어 정보를 전송하는 데 사용된다. GOOSE 메시지는 LAN을 통해 멀티캐스트로 전송되며, 메시지를 수신하도록 구성된 모든 IED가 이를 구독할 수 있다.

ACSI는 모든 IED가 네트워크 동작 관점에서 동일한 방식으로 동작할 수 있도록 하는 서비스 집합과 그에 대한 응답을 정의한다. IEC 61850-8-1 [6]은 추상 객체와 서비스를 ISO 9506의 MMS(Manufacturing Message Specification) 프로토콜에 매핑한다.

IEC 61850 객체 및 서비스 모델의 MMS로의 매핑은 서비스 매핑을 기반으로 하며, 여기서 특정 MMS 서비스가 ACSI의 다양한 서비스를 구현하는 수단으로 선택된다(부록 I 참조). 그 후 IEC 61850의 다양한 객체 모델이 특정 MMS 객체에 매핑된다. 예를 들어, IEC 61850 논리 디바이스 객체는 MMS 도메인에 매핑된다.

2.3 객체 참조 및 데이터 속성 참조의 MMS 매핑

IEC 61850 객체는 표준 IEC 61850-8-1 [6]에 따라 MMS 객체에 매핑된다.

- 서버 클래스: IEC 61850-7-2 서버 클래스의 인스턴스는 MMS 가상 제조 장치(VMD) 객체에 일대일로 매핑된다(부록 E 참조).
- 논리 디바이스 (LD): 논리 디바이스 객체의 인스턴스는 MMS 도메인 객체에 매핑된다. MMS 도메인은 특정 이름과 연관된 정보의 집합을 나타낸다.

- 논리 노드 (LN): 논리 노드 클래스의 인스턴스는 단일 MMS NamedVariable(명명된 변수)에 매핑된다.

논리 노드는 하나 이상의 데이터 객체(DataObjects)로 구성된다. 데이터 객체의 이름은 MMS 명명된 변수 내에서 발견되는 데이터의 계층적으로 명명된 구성 요소를 기반으로 한다. 계층의 각 레벨은 데이터를 나타내는 MMS 이름 변수 내에서 "$"를 사용하여 결정된다:
'<LNVariableName>$<FC>$<LNDataObjectName1>', 예: XCBR1STPos (그림 7 참조).

데이터 객체의 데이터 속성 DataAttr은 데이터 객체와 유사한 방식으로 매핑된다:
'<LNVariableName>$<FC>$<LNDataName1>$<AttributeName1>', 예: XCBR1STPos$stVal.

2.4 통신 프로파일

OSI 모델을 사용하여 통신하기 위해, 통신 서비스는 서로 다른 통신 프로파일을 사용하여 실제 통신 프로토콜에 매핑되어야 한다. 프로파일은 IEC 61850-8-1에 정의되어 있다(그림 9 참조).

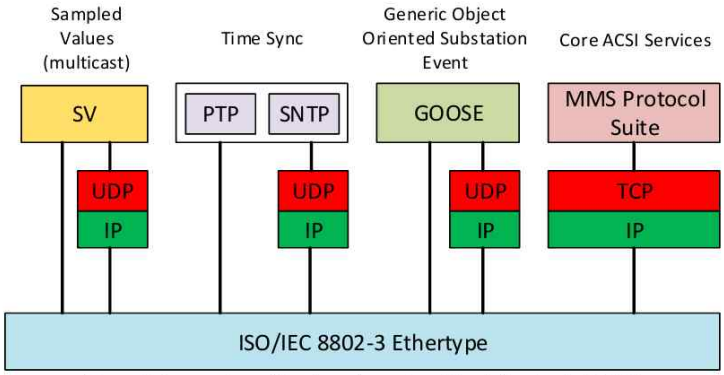


Figure 9: An overview of functionality and profiles defined in IEC 61850

클라이언트-서버 통신에는 MMS 프로토콜이 사용된다. 이 프로토콜은 본래 제조 분야를 위해 설계되었다. 복잡한 명명(naming)과 서비스를 지원하기 때문에 IEC 61850을 위해 선택되었다. MMS는 OSI 모델의 애플리케이션 계층을 담당하며, 전송 및 네트워크 계층은 TCP/IP 또는 ISO 프로토콜이 담당한다. MMS 프로토콜에 대한 자세한 내용은 4절에서 다룬다.

GOOSE 통신에는 비연결형(connection-less) OSI 및 비-MMS(non-MMS) 프로파일이 사용된다. 이는 전송 전에 IED 간의 연결을 확인하지 않는다는 것을 의미한다. GOOSE 메시지는 단순히 네트워크로 전송된다. 이는 GOOSE 통신의 시간에 민감한(time-critical) 요구사항을 충족하기 위해 필요하다. 그림 9에서 볼 수 있듯이, 중간 계층의 처리 시간을 제거하기 위해 GOOSE 메시지는 이더넷 데이터 프레임에 직접 매핑된다. GOOSE 프로토콜에 대한 자세한 내용은 3절에서 다룬다.

OSI 통신 스택에 대한 상세한 매핑은 그림 10에 묘사되어 있다. IEC 61850-8-1은 두 가지 프로파일을 정의한다. OSI 모델의 상위 3개 계층의 서비스와 프로토콜을 명시하는 애플리케이션 프로파일(A-Profile)과, 하위 4개 계층의 서비스와 프로토콜을 명시하는 전송 프로파일(T-Profile)이다.

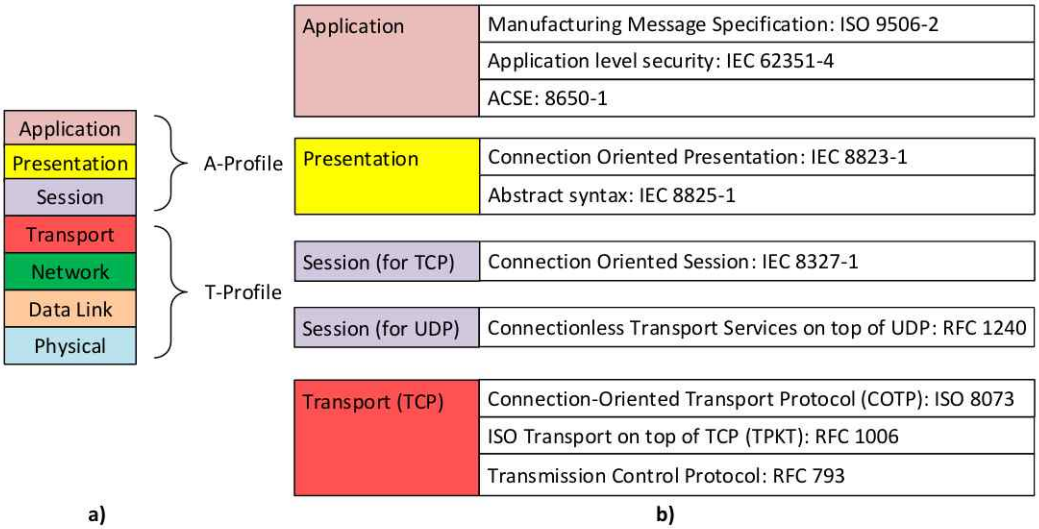


Figure 10: Communication stack: (a) A-profile, T-profile and (b) protocols for the client/server communication

특정 유형의 정보/서비스를 교환할 수 있도록 A-프로파일과 T-프로파일의 다양한 조합이 결합될 수 있다. IEC 61850-7-2에 명시된 서비스들은 A-프로파일과 T-프로파일의 네 가지 다른 조합으로 매핑되며, 다음 용도로 사용된다.

- 클라이언트-서버 서비스 (MMS)
- GOOSE/GSE 관리 서비스
- GSSE 서비스
- 시간 동기화 (PTP, SNTP)

MMS를 위한 A-프로파일에 정의된 프로토콜은 필수 사항이나, 애플리케이션 계층 보안(IEC 62351-4)은 조건부이다. GOOSE를 위한 A-프로파일은 UDP 상위에 RFC 1240 헤더를 요구한다. MMS를 위한 T-프로파일은 TCP가 사용될 때(MMS의 일반적인 경우) TCP 상위에 TPKT 프로토콜을 요구한다. MMS와 GOOSE 통신에 대한 더 자세한 내용은 다음 절에서 언급한다.

3. GOOSE 프로토콜

GOOSE(Generic Object-Oriented Substation Event) 프로토콜은 IEC 61850 장치 간의 전기 설비 보호와 같은 시간 임계적(time-critical) 이벤트의 전송을 구현한다. IEC 61850 표준은 클라이언트-서버 모델과 피어 투 피어(peer-to-peer) 모델이라는 두 가지 통신 서비스 그룹을 정의한다.

피어 투 피어 모델은 IED 간의 빠르고 신뢰성 있는 통신과 같은 시간 임계적 활동과 관련된 GSE(Generic Substation Event) 서비스에 활용된다. GSE 서비스와 관련된 메시지 중 하나는 LAN 전체에 브로드캐스트 또는 멀티캐스트 메시지 전송을 허용하는 GOOSE 메시지이다.

3.1 GOOSE 메시지 형식

GOOSE 메시지는 OSI 모델의 세 계층, 즉 물리 계층, 데이터 링크 계층 및 응용 계층과 연관된다. 데이터 링크 계층에서 GOOSE는 802.3 이더넷 프레임으로 캡슐화된다(그림 11 참조).

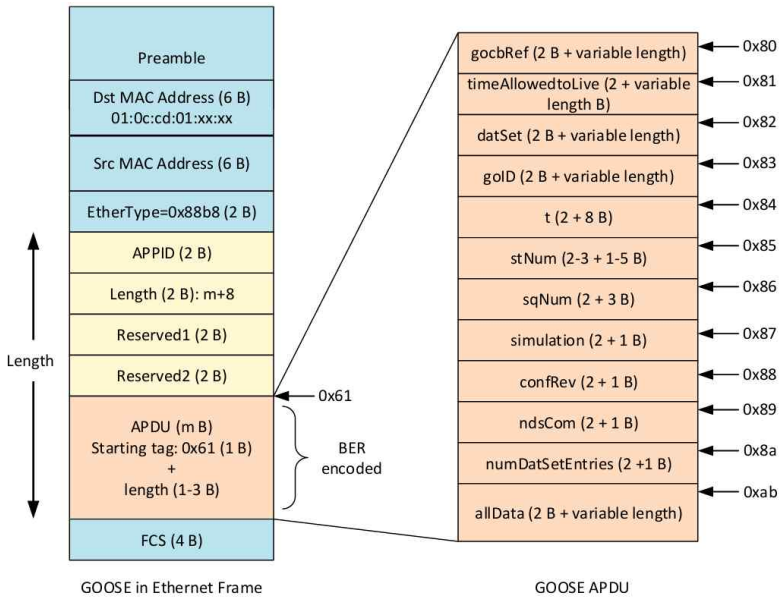


Figure 11: GOOSE message format

그림에서 볼 수 있듯이, GOOSE는 L2 계층에서 4개의 고정 길이 필드를 정의한다. APDU는 BER로 인코딩된 TLV(Type-Length-Value) 삼중항(triplet)의 시퀀스이며, 여기서 필드는 고정 길이 또는 가변 길이를 갖는다. TLV 구조는 1바이트 식별자(유형, 표 4 참조), 값의 길이 n을 정의하는 1바이트 이상의 바이트, 그리고 n바이트 값으로 구성된다. 위의 그림은 각 필드에 대한 ASN.1 식별자를 보여준다. 식별자는 필드의 시작 부분에 위치하며, 그 뒤에 길이와 값(TLV 삼중항)이 따른다. 일부 필드는 선택 사항일 수 있으며, 자세한 내용은 부록 F를 참조한다.

3.1.1 데이터 링크 계층에서의 GOOSE

GOOSE 메시지는 일반적으로 IEC 61850 기술 위원회 57(IEC-TC57)이 정의한 예약된 멀티캐스트 목적

지 MAC 주소 01:0c:cd:01:xx:xx(6바이트)를 사용하여 이더넷을 통해 전송된다(표 2 참조). 선택적으로, IEEE 802.1Q(VLAN) 형식이나 HSR(IEC 62439-3) 또는 PRP(IEC 62439-3) 링크 이중화를 사용한 확장 캡슐화를 추가할 수 있다([6, 부록 C] 참조).

GSE, GOOSE 및 샘플링된 값(Sampled Values)에 대한 Ethertype 값(2바이트)은 IEEE 기관에 등록되어 있다. 할당된 값은 표 3에 나열되어 있다([6, 부록 C.2] 참조).

Assigned Ethertype values			
Service	**Standard**	**Ethertype**	**APPID type**
GOOSE Type 1	IEC 61850-8-1	0x88b8	00
GSE Management	IEC 61850-8-1	0x88b9	00
Sampled Values	IEC 61850-9-2	0x88ba	01
GOOSE Type 1A	IEC 61850-8-1	0x88b8	10

Table 3: Assigned Ethertype values

Ethertype에 이어, 다음 4개의 특수 필드가 이더넷 프레임에 추가된다:

- 메시지로 전송되는 APPID(애플리케이션 식별자)는 수신 애플리케이션을 위한 핸들(handle)로 사용된다. 이는 GSE 관리 및 GOOSE 메시지를 포함하는 ISO/IEC-3 프레임을 선택하고 애플리케이션 연관(application association)을 구별하는 데 사용된다. APPID 값은 값의 최상위 비트로 정의되는 APPID 유형과 실제 ID의 조합이다. 가능한 값은 다음과 같다:
  - 기본값 (구성되지 않음): 0x0000
  - GOOSE 유형 1: 0x0000 – 0x3FFF
  - GOOSE 유형 1A (트립): 0x8000 – 0xBFFF

- Length는 APPID에서 시작하는 Ethertype PDU 헤더와 APDU(응용 프로토콜 데이터 단위)의 길이를 포함한 옥텟 수를 나타낸다. 따라서 길이 값은 8+m이어야 하며, 여기서 m은 APDU의 길이이고 m은 1492보다 작아야 한다. 길이 필드가 일치하지 않거나 유효하지 않은 프레임은 폐기되어야 한다.

- Reserved 1은 SendGOOSE 서비스의 서비스 매개변수 'Simulation'에서 매핑되는 S(시뮬레이션) 비트, 미래의 표준화된 애플리케이션을 위해 예약된 3개의 R(예약됨) 비트, 그리고 보안이 적용된 GOOSE가 전송될 때 사용되어야 하는 12개의 예약된 보안 비트를 포함하는 2바이트 구조이다. 보안이 적용되지 않는 경우 0으로 설정되어야 한다.

- Reserved 2는 보안 표준 IEC 62351-6에 의해 정의된 2바이트 필드이며, 보안이 적용된 GOOSE가 전송될 때 정의된 대로 사용되어야 하고, 그렇지 않을 경우 0으로 설정되어야 한다.

3.1.2 응용 계층에서의 GOOSE

응용 계층에서 GOOSE 메시지는 ASN.1 표기법을 사용하여 정의된다(부록 F 참조). PDU는 다음과 같은 구조를 갖는다:

```
IECGoosePdu ::= SEQUENCE {
        gocbRef                 [0]     IMPLICIT VISIBLE-STRING,
        timeAllowedtoLive [1]   IMPLICIT INTEGER,
```

```
        datSet                  [2]     IMPLICIT VISIBLE-STRING,
        goID                    [3]     IMPLICIT VISIBLE-STRING OPTIONAL,
        T                       [4]     IMPLICIT UtcTime,
        stNum                   [5]     IMPLICIT INTEGER,
        sqNum                   [6]     IMPLICIT INTEGER,
        simulation              [7]     IMPLICIT BOOLEAN DEFAULT FALSE,
        confRev                 [8]     IMPLICIT INTEGER,
        ndsCom                  [9]     IMPLICIT BOOLEAN DEFAULT FALSE,
        numDatSetEntries [10]   IMPLICIT INTEGER,
        allData                 [11]    IMPLICIT SEQUENCE OF Data,
}
```

GOOSE PDU의 구조는 IEC 61850 7-2 표준 [3]에 정의된 GOOSE 제어 블록(GOOSE Control Block) 객체에서 파생된다. 이는 다음 항목들로 구성된다:

- GoCBRef – GOOSE 제어 블록 참조(GOOSE control block reference)는 LLN0 내의 GOOSE 제어 블록 인스턴스에 대한 고유한 경로 이름이다. 형식은 LDName/LLN0.GoCBName이다.
  예를 들어 GEDeviceF650/LLN0GOgcb01의 경우, LD 이름은 GEDeviceF650, LN 클래스는 LLN0(논리 노드 제로), 기능 제약(functional constraint)은 GO(GOOSE 제어), 그리고 GoCB 인스턴스는 gcb01이다
- TimeAllowedtoLive – 속성 StNum이 증가된 시간이다. 이는 구독자(subscriber)에게 다음 메시지 반복을 얼마나 기다려야 하는지 알려준다.
- DatSet – 멤버들의 값이 전송되어야 하는 데이터 세트(data set)의 참조이다(예: GEDeviceF650/LLN0$GOOSE1). 데이터 세트의 멤버들은 1부터 시작하여 고유하게 번호가 매겨져야 한다. 이 번호는 해당 멤버의 멤버 오프셋(MemberOffset)이라고 한다. 데이터 세트의 각 멤버는 고유한 번호와 멤버 참조(MemberReference, 기능 제약 데이터 FCD 또는 데이터 속성 FCDA)를 갖는다(그림 12 참조).
- GoID – GOOSE ID는 사용자가 GOOSE 메시지에 식별자를 할당할 수 있게 하는 속성이다(예: F650_GOOSE1).
- T (타임스탬프) – 속성 StNum이 증가된 시간이다.
- StNum (상태 번호) – GOOSE 메시지가 전송되고 DatSet에 지정된 데이터 세트 내에서 값의 변경이 감지될 때마다 증가하는 카운터이다. 초기값은 1이어야 한다. 값 0은 예약되어 있다.
- SqNum – 보고서의 현재 시퀀스 번호이다. 이는 GOOSE 메시지가 전송될 때마다 증가해야 한다. StNum이 변경된 후, 카운터 SqNum은 값 0으로 설정되어야 한다. GoEna가 TRUE로 전송될 때 SqNum의 초기값은 1이다. 이 번호는 TCP의 시퀀스 번호와 유사해 보인다.
- Simulation (테스트 비트) – 만약 참(true)이면, 메시지와 그 값은 시뮬레이션 장치에 의해 발행되었으며 실제 값이 아니다. GOOSE 구독자는 수신 IED의 설정에 따라 실제 메시지 대신 시뮬레이션된 메시지의 값을 애플리케이션에 보고한다.
- ConfRev – 데이터 세트 멤버의 삭제, 멤버의 순서 변경, 또는 DatSet 참조 변경을 나타내는 구성 리비전(configuration revision)을 포함한다. 이 숫자는 DatSet 값에 의해 참조되는 데이터 세트의 구성이 변경된 횟수를 나타내야 한다.
- NdsCom – 메시지에서 일부 커미셔닝(commissioning)이 필요함을 나타낸다(need commission). 만약 참(TRUE)이면, GoCB는 추가 구성이 필요하다.
- NumDatSetEntries – 데이터 세트 항목의 수이다.
- allData – GOOSE 제어 블록에 지정된 MMS NamedVariableList의 사용자 정의 정보 목록이다.

3.2 통신

일반 변전소 이벤트(GSE) 모델은 입출력 값을 시스템 전역에 빠르고 신뢰성 있게 배포할 수 있는 기능을 제공한다. GSE 모델은 자율 분산(autonomous decentralization) 개념을 기반으로 하며, 멀티캐스트/브로드캐스트 서비스를 사용하여 동일한 일반 변전소 이벤트 정보를 하나 이상의 물리 디바이스에 동시에 전달할 수 있는 효율적인 방법을 제공한다.

IEC 61850-7-2 [3]에는 두 가지 제어 클래스와 두 가지 메시지 구조가 정의되어 있다:

- GOOSE(Generic Object Oriented Substation Event): 데이터 세트로 구성된 광범위한 공통 데이터의 교환을 지원한다.
- GSSE(Generic Substation State Event): 상태 교환 정보(비트 쌍)를 전달하는 기능을 제공한다.

피어 투 피어(Peer-to-peer) 통신은 일반 변전소 이벤트(GOOSE 및 GSSE; 멀티캐스트 기반)의 교환과 샘플링된 값(멀티캐스트 또는 유니캐스트 기반)의 교환을 위한 서비스를 제공한다. GOOSE 모델은 게시될 데이터 값들을 데이터 세트로 그룹화하여 사용한다. 아날로그, 바이너리 또는 정수 값과 같은 다양한 데이터와 데이터 속성을 사용하여 데이터 세트를 생성할 수 있다.

GOOSE 통신은 발행-구독(publish-subscribe) 메커니즘을 기반으로 한다. 발행자(publisher)는 송신 측의 로컬 버퍼에 값을 기록하고, 구독자(subscriber)는 수신 측의 로컬 버퍼에서 값을 읽는다. 구독자의 로컬 버퍼를 업데이트하는 것은 통신 시스템의 책임이다. 이 절차를 제어하기 위해 발행자에 있는 일반 변전소 이벤트 제어 클래스(GoCB)가 사용된다.

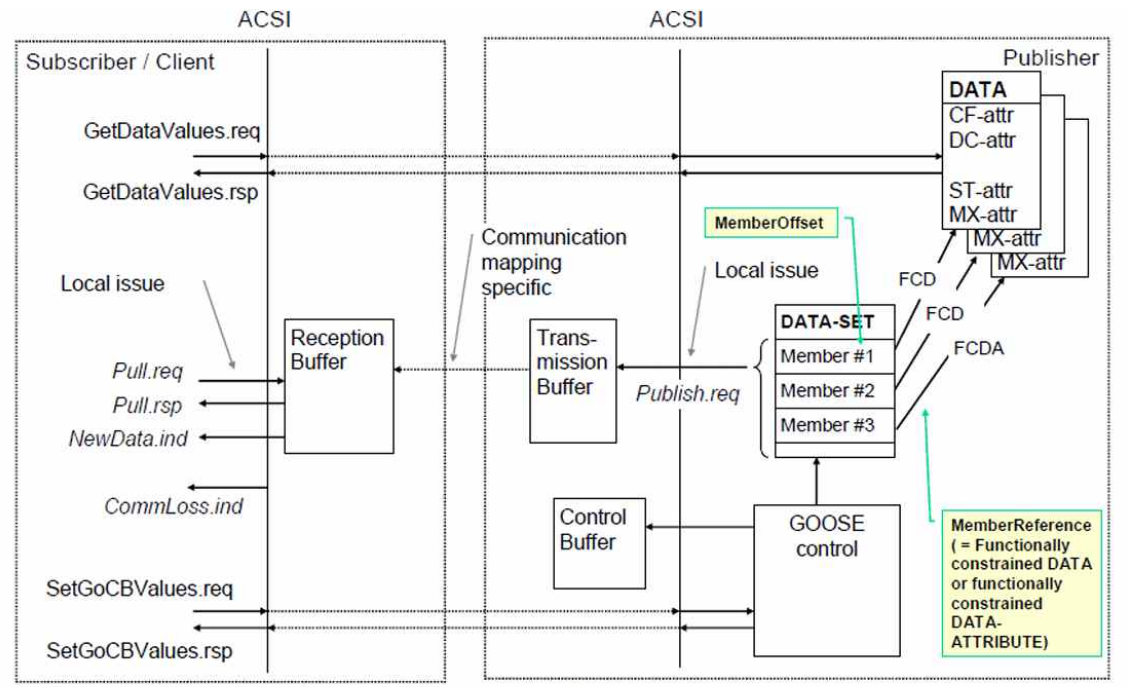


Figure 12: GoCB model [3]

그림 12는 GOOSE 모델의 클래스와 서비스에 대한 개요를 제공한다. 메시지 교환은 멀티캐스트 애플

리케이션 연관(multicast application association)을 기반으로 한다. 데이터 세트(DataSet) 내의 특정 기능 제약(예: ST)을 가진 하나 이상의 데이터 속성(DataAttributes) 값이 변경되면, 발행자(publisher)의 전송 버퍼가 로컬 서비스 publish로 업데이트되고, 모든 값이 GOOSE 메시지를 통해 전송된다. 데이터 세트는 여러 멤버를 가질 수 있다. 각 멤버는 특정 기능 제약(FC)을 가진 데이터 속성을 참조하는 멤버 참조(MemberReference)를 가져야 한다.

통신 네트워크의 매핑 특정 서비스(Mapping specific services)는 구독자(subscriber) 내의 버퍼 내용을 업데이트한다. 수신 버퍼에 수신된 새로운 값들은 애플리케이션에 신호된다.

GOOSE 메시지는 수신 장치가 상태가 변경되었음과 마지막 상태 변경 시간을 알 수 있도록 하는 정보를 포함한다. 마지막 상태 변경 시간은 수신 장치가 주어진 이벤트와 관련된 로컬 타이머를 설정할 수 있게 한다.

전원이 켜지거나 서비스로 복귀하여 새로 활성화된 장치는 데이터 객체(상태)의 현재 값 또는 값들을 초기 GOOSE 메시지로 전송해야 한다. 또한, GOOSE 메시지를 전송하는 모든 장치는 상태/값 변경이 발생하지 않았더라도 긴 주기 시간(long cycle time)으로 메시지를 계속 전송해야 한다. 이는 최근에 활성화된 장치가 피어 디바이스(peer devices)의 현재 상태 값을 알 수 있도록 보장한다.

GOOSE 메시지는 멀티캐스트되며, 이를 구독하도록 구성된 IED들에 의해 수신된다(그림 13 참조).

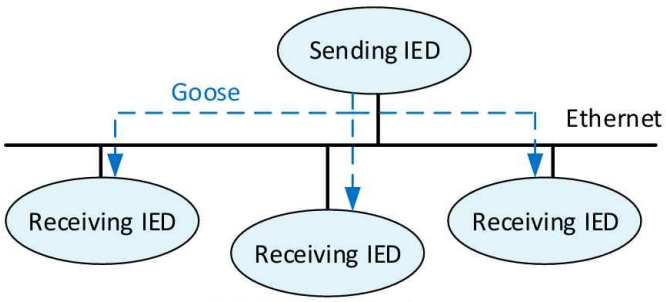


Figure 13: GOOSE publish-subscribe communication

우선, 다음 단계들을 포함하여 GOOSE 애플리케이션을 구성해야 한다:

1. GOOSE 데이터 세트를 준비한다.
2. 데이터 세트 전송 방법을 지정하기 위해 GOOSE 제어 블록 매개변수를 설정한다.
3. 구독을 통해 GOOSE 데이터 세트를 수신할 IED를 지정한다.

GOOSE 데이터 세트는 데이터 속성들의 집합이다. 사용자는 로컬에서 이 데이터 세트를 생성하고, 목록에 새로운 데이터 속성을 추가하거나 목록에서 이를 제거할 수 있다(그림 14 참조).

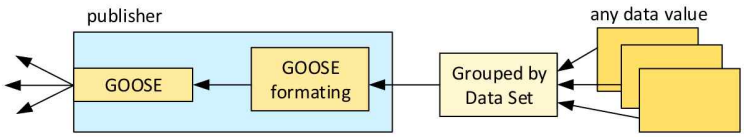


Figure 14: Peer-to-peer data value publishing model

3.3 메시지 파싱 예제

GOOSE 프로토콜은 이더넷 프레임에 직접 캡슐화되므로, GOOSE 프로토콜 파싱은 이더넷 레벨에서 시작된다(3.1절 참조).

3.3.1 데이터 링크 계층에서의 GOOSE

GOOSE를 나타내는 값 0x88b8을 가진 EtherType 필드 뒤에는, 2바이트 APPID 필드, 2바이트 Length 필드, 2바이트 Reserved1 필드, 그리고 2바이트 Reserved2 필드가 있다. 이 필드들은 모든 GOOSE 메시지에 고정 길이로 존재한다. 다음 예제는 GOOSE 프로토콜이 포함된 이더넷 프레임의 분석을 보여준다:

예제 1: 01 0c cd 01 00 01 00 09 8e fa b7 1c 88 b8 00 02 00 8e 00 00 00 00 61 81 83 80 1f 53 …

- 01 0c cd 01 00 01 (6바이트) - GOOSE용 멀티캐스트 목적지 MAC 주소 (접두사 0x 01 0c cd 01).
- 00 09 8e fa b7 1c (6바이트) - 송신 장치를 식별하는 유니캐스트 소스 MAC 주소.
- 88 b8 - GOOSE 프로토콜을 나타내는 EtherType (표 3 참조).
- 00 02 (2바이트) - 수신 애플리케이션을 식별하는 AppID (애플리케이션 ID).
- 00 8e (2바이트) - 이더넷 프레임의 일부를 포함한 GOOSE 메시지의 길이. 값 0x8e는 GOOSE 메시지의 142바이트를 나타낸다.
- 00 00 (2바이트) - Reserved1 필드.
- 00 00 (2바이트) - Reserved2 필드.

3.3.2 응용 계층에서의 GOOSE

응용 계층에서 GOOSE PDU는 ASN.1 표기법을 사용하여 인코딩된다. 즉, 전송된 데이터는 TLV(Type-Length-Value) 삼중항을 형성한다(부록 G 참조). GOOSE 헤더에 인코딩된 컨텍스트 특정(context-specific) 유형은 표 4에 제시되어 있다. 이 표는 IECGoosePdu 형식과 관련된 ASN.1 컨텍스트 특정 태그를 보여준다(부록 F 참조).

표준 61850-8-1 [6]은 GOOSE 메시지에 대한 고정 길이(Fixed-length) 속성을 정의한다. 이 속성은 발행자가 메시지의 각기 다른 필드에 대해 항상 고정된 오프셋을 사용한다는 것을 의미한다. 고정 길이 속성 구성은 각 GOOSE 제어 블록에 대해 수행된다. 표 4는 고정 길이 속성이 설정된 경우 주어진 필드에 대한 ASN.1 데이터 길이를 보여준다. 설정되지 않은 경우, 길이는 TLV 형식으로 지정된다.

다음 예제는 고정 길이 속성이 없는 GOOSE 메시지의 디코딩을 보여주며, 따라서 주어진 GOOSE 필드의 길이는 TLV 구조에 직접 제공된다.

Attribute Name	Data Type	ASN.1 Identifier	Tag number	ASN.1 Length (B)
goCBRef	Visible-string	0x80	0	variable
timeAllowedToLive	INT32U	0x81	1	5
datSet	Visible-string	0x82	2	variable
goID	Visible-string	0x83	3	variable
T	UtcTime	0x84	4	8
stNum	INT32U	0x85	5	5
sqNum	INT32U	0x86	6	5
simulation	Boolean	0x87	7	1
confRev	INT32U	0x88	8	5
ndsCom	Boolean	0x89	9	1
numDatSetEntries	INT32U	0x8a	10	5
allData	SEQUENCE of Data	0xab	11	variable

Table 4: ASN.1 Tags for context-specific data type in GOOSE message [6]

예제 2: 61 81 83 80 1f 53 49 50 43 54 52 4c 2f 4c 4c 4e 30 24 47 4f 24 43 6f 6e 74 72 6f 6c 5f 44 61 74 61 73 65 74 81 02 0b b8 82 14 53 49 50 43 54 52 4c 2f 4c 4c 4e 30 24 44 61 74 61 73 65 74 83 1d 53 49 50 2f 43 54 52 4c 2f 4c 4c 4e 30 2f 43 6f 6e 74 72 6f 6c 5f 44 61 74 61 73 65 74 84 08 59 31 8e 6a 25 e3 0a 89 85 01 05 86 03 0b 9b 2b 87 01 00 88 01 01 89 01 00 8a 01 02 ab 09 84 02 06 80 84 03 03 00 00

- Type 61 (이진수로 0110 0001)은 식별자 옥텟(identifier octet)으로, 데이터 유형 1을 가진 구성형(constructed, 1) 형태의 애플리케이션 클래스(01)를 설명한다. 애플리케이션 유형 01은 goosePDU를 의미한다(부록 F 참조).
- Length 81 83은 확장 길이 필드이다. 여기서 0x81(1000 0001)은 1옥텟으로 된 길이의 긴 정형(long definite form)을 설명하며, 0x83은 길이 값으로 131바이트를 의미한다.
  - ⊙ Type 80 (1000 0000)은 내장된 TLV 삼중항의 시작이다. 이는 컨텍스트 특정(context-specific) 클래스(10), 기본형(primitive, 0)이며 유형은 0으로 gocbRef이다(부록 F 참조).
    - Length 1f는 gocbRef 필드 내 VISIBLE STRING의 길이, 즉 31바이트이다.
    - Value 53 49 ... 74는 문자열 "SIPCTRL/LLN0GOControl_Dataset"을 나타내며, 이는 SIPCTRL 장치, 논리 이름 LLN0, 기능 제약 GO (GOOSE 제어) 및 제어 블록 이름 Control_Dataset을 참조한다.
  - TLV 81 02 0b b8은 기본형(0)의 컨텍스트 특정 데이터 유형(10)을 나타내며, 유형 1은 timeAllowedToLive 속성이다. 2바이트 값 0x 0b b8(10진수로 3000)을 가진다.
  - 다음 TLV 82 14 53 ... 74는 컨텍스트 특정 유형 2인 datSet을 가지며, 길이는 20바이트(16진수로 14)이고 ASCII 값은 "SIPCTRL/LLN0$Dataset"이다.
  - TLV 83 1d 53 ... 74는 길이 29바이트(16진수로 1d)인 컨텍스트 특정 유형 3 (GOOSE ID)을 가지며 ASCII 값은 "SIP/CTRL/LLN0/Control_Dataset"이다.
  - TLV 84 08 59 ... 89는 컨텍스트 특정 데이터 유형 T (UTC 시간)를 나타낸다. 8바이트 값은 "Jun 2, 2017 16:12:26.147995591 UTC"로 해석될 수 있다.
  - TLV 85 01 05는 값 5를 가진 stNum 필드를 인코딩한다.
  - TLV 86 03 0b 9b 2b는 값 760619를 가진 sqNum 필드를 인코딩한다.
  - TLV 87 01 00은 BOOLEAN 값 0(False)을 가진 simulation 필드를 인코딩한다.
  - TLV 88 01 01은 값 1을 가진 confRev 번호를 인코딩한다.
  - TLV 89 01 00은 BOOLEAN 값 0(False)을 가진 ndsCom 필드를 인코딩한다.
  - TLV 8a 01 02는 정수 값 2를 가진 numDatSetEntries 필드를 인코딩한다.
  - TLV ab 09 84 ... 00은 유형 0xab(10101011)를 가지며, 이는 컨텍스트 특정 유형(10), 구성형(1) 그리고 유형 11(이진수로 1011)인 데이터 시퀀스(SEQUENCE of Data)를 의미한다.
    - 길이는 9바이트이다.
    - 시퀀스의 첫 번째 항목은 유형 84(1000 0100)를 가지며, 이는 컨텍스트 특정 유형(10), 기본형(0)이다. 데이터 유형 식별자는 표 5의 CODED ENUM을 참조하며, 이는 비트 문자열(bit string)이다(부록 D 참조). 값의 길이는 2바이트이다. 값은 바이트 0x80(10 00 00 00)에 6개의 패딩 비트(0x60)를 포함하므로, 값은 2이다.
    - 시퀀스의 두 번째 항목 또한 유형 84(비트 문자열)를 가지며, 길이는 3바이트이고 값은 03 00 00이다. 값 필드는 3인 패딩 비트 길이와 0인 값으로 구성된다.

IEC 61850-7-2 data type	ASN.1 Identifier	ASN.1 Length (B)	Comments
Boolean	0x83	1	False (0), True (1)
INT8	0x85	2	signed 8 bit big endian
INT16	0x85	3	signed 16 bit big endian
INT32	0x85	5	signed 32 bit big endian
INT64	0x85	9	signed 64 bit big endian
INT8U	0x86	2	unsigned 8 bit big endian
INT16U	0x86	3	unsigned 16 bit big endian
INT24U	*not used*		
FLOAT32	0x86	5	32 bit IEEE 754 floating point
ENUMERATED	0x87	5	signed 8-bit big endian
CODED ENUM	0x84	2	bit-string: 1st byte=unused bytes, 2nd byte=value
OCTET STRING	0x89	20	20 bytes ASCII text, Null terminated
VISIBLE STRING	0x8a	35	35 bytes ASCII text, Null terminated
Timestamp	0x91	8	64 bit timestamp
Quality	0x84	3	bit-string

Table 5: ASN.1 tags for allData structure [6] and their length

3.4 GOOSE 데이터 세트

다음 섹션에서는 프로젝트를 위해 획득한 GOOSE 데이터 세트 내의 통신에 대해 간략하게 설명한다.

3.4.1 GOOSE.pcap 데이터 세트

GOOSE.pcap은 Wireshark가 게시한 샘플 파일이다[2]. 이 파일은 동일한 장치의 애플리케이션에서 전송된 8개의 GOOSE 패킷을 포함한다. APPID는 1, GoID는 F650_GOOSE1, 제어 블록은 GEDeviceF650/LLN0GOgcb01, 데이터 세트 이름은 GEDeviceF650/LLN0$GOOSE1이다. StNum 및 allData 값은 변경되지 않는다. 시간이 지남에 따라 timeAllowedtoLive 및 sqNum 값만 변경된다.

이 문맥에서 timeAllowedtoLive의 변화는 이상하며, 타임스탬프 t와 관련하여 볼 때 이러한 변화는 인위적인 것으로 보인다(표 6 참조).

goCBRef	goID	datSet	stNum	sqNum	timeAllowed toLive	t
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	10	40000	Jan 2, 2000 02:46:11.258165836 UTC
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	11	40000	Jan 2, 2000 02:46:11.258165836 UTC
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	12	40000	Jan 2, 2000 02:46:11.258165836 UTC
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	1	1000	Jan 2, 2000 02:47:29.927595853 UTC
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	2	1000	Jan 2, 2000 02:47:29.927595853 UTC
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	3	1000	Jan 2, 2000 02:47:29.927595853 UTC
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	4	2000	Jan 2, 2000 02:47:29.927595853 UTC
GEDeviceF650/LLN0GOgcb01	F650_GOOSE1	GEDeviceF650/LLN0$GOOSE1	1	5	40000	Jan 2, 2000 02:47:29.927595853 UTC

Table 6: Analyzing GOOSE data

3.4.2 goose1.pcapng 데이터 세트

이 파일은 전력 시스템 연구실에서 생성되었다. 데이터 세트는 3개의 서로 다른 장치에서 5개의 서로 다른 멀티캐스트 그룹(01:0c:cd:01:00:00, 01, 02, 03, 04)으로 전송된 1,093개의 GOOSE 패킷을 포함한

2) https://wiki.wireshark.org/SampleCaptures?action=AttachFile&do=view&target=GOOSE.pcap.gz 참고 [2018.6]

다.

하나의 IED에서 하나의 목적지 멀티캐스트 주소로 전송되는 GOOSE 패킷은 2초 또는 10초 간격으로 전송된다. 주어진 목적지 MAC 주소에 대해, APPID(애플리케이션 ID), goCBRef(제어 블록 주소), goID(GOOSE ID) 및 DatSet(데이터 세트 참조)은 일정하며, 오직 sqNum(시퀀스 번호) 값만 증가한다(표 7 참조). allData 블록에는 변경 사항이 없다.

Src MAC	Dst MAC	APPID	goCBRef - goID - datSet	stNum	timeAllowed toLive	sqNum range	Packets
00:09:8e:fa:b7:1a	01:0c:cd:01:00:00	0x00000001	SIP1CTRL/LLN0GOControl_Dataset SIP1/CTRL/LLN0/Control_Dataset SIP1CTRL/LLN0$Dataset	3	3000	760626-760835	210
00:09:8e:fa:b7:1a	01:0c:cd:01:00:03	0x00000004	SIP1PROT/LLN0GOControl_Dataset_1 SIP1/PROT/LLN0/Control_Dataset_1 SIP1PROT/LLN0$Dataset_1	186	3000	760548-760757	210
00:09:8e:fa:b7:1c	01:0c:cd:01:00:01	0x00000002	SIPCTRL/LLN0GOControl_Dataset SIP/CTRL/LLN0/Control_Dataset SIPCTRL/LLN0$Dataset	5	3000	760617-760826	210
00:09:8e:fa:b7:1c	01:0c:cd:01:00:02	0x00000003	SIPPROT/LLN0GOControl_Dataset_1 SIP/PROT/LLN0/Control_Dataset_1 SIPPROT/LLN0$Dataset_1	81	3000	760617-760826	210
00:09:8e:fa:b7:1c	01:0c:cd:01:00:04	0x00000005	SIPCTRL/LLN0GOControl_Dataset_1_1 SIP/CTRL/LLN0/Control_Dataset_1_1 SIPCTRL/LLN0$Dataset_1_1	75	3000	760617-760826	210
00:21:c1:25:08:a2	01:0c:cd:01:00:00	0x00000001	AA1J1Q01A1LD0/LLN0GOLEDs_info AA1J1Q01A1LD0/LLN0.LEDs_info AA1J1Q01A1LD0/LLN0$LEDs_ON_OFF	23	11000	112568-112610	43
							1093

Table 7: GOOSE packets values in the dataset

예를 들어 00:09:8e:fa:b7:1c와 같은 하나의 아웃스테이션(GOOSE 발행자)을 분석할 때, 우리는 이 장치가 GOOSE 구독자가 데이터를 읽는 세 개의 서로 다른 멀티캐스트 그룹으로 GOOSE 메시지를 반복적으로 전송하는 것을 볼 수 있다. 표준에 명시된 바와 같이, 발행된 데이터는 데이터 세트로 그룹화되며 각 데이터 세트는 해당 애플리케이션(인스턴스)에 의해 제어된다. 데이터 속성 값의 모든 변경 사항은 allData 필드에서 감지된다. 상태가 변경되면, stNum이 증가한다.

아웃스테이션에서 다음 애플리케이션들이 활성화되어 있다:

- APPID=2인 애플리케이션은 제어 블록 주소 'SIPCTRL/LLN0GOControl_Dataset', 데이터 세트 'SIPCTRL/LLN0$Dataset', 그리고 GOOSE ID 'SIP/CTRL/LLN0/Control_Dataset'을 사용하여 그룹 '01:0c:cd:01:00:01'로 PDU를 전송한다.
- APPID=2인 애플리케이션은 제어 블록 주소 'SIPCTRL/LLN0GOControl_Dataset_1_1', 데이터 세트 'SIPCTRL/LLN0$Dataset_1_1', 그리고 GOOSE ID 'SIP/CTRL/LLN0/Control_Dataset_1_1'을 사용하여 그룹 '01:0c:cd:01:00:04'로 PDU를 전송한다.
- APPID=3인 애플리케이션은 제어 블록 주소 'SIPPROT/LLN0GOControl_Dataset_1', 데이터 세트 'SIPPROT/LLN0$Dataset_1', 그리고 GOOSE ID 'SIP/PROT/LLN0/Control_Dataset_1'을 사용하여 그룹 '01:0c:cd:01:00:02'로 PDU를 전송한다.

PDU는 모든 멀티캐스트 그룹에 대해 2초마다 전송된다. 송신 측에 변경 사항이 없으므로, stNum(상태 번호)은 동일하게 유지된다.

시퀀스 번호 값 sqNum은 새 메시지가 전송될 때마다 증가한다. 이 변수는 각 목적지에 대해 동일한

시작 값으로 초기화되었으므로, 전송 중 그 값은 모든 목적지에 대해 동일하다.

2분간의 GOOSE 통신 후, 아웃스테이션(서버)과 MAC 주소 00:0a:f7:4d:93:fc를 가진 새로운 장치(클라이언트) 사이에 MMS 통신이 열린다. 클라이언트는 연결을 수립하고 데이터를 요청한다. MMS 통신은 다음 단계들을 포함한다(자세한 내용은 4.4.1절 참조):

1. 연결 초기화 (Connection initialization): 연결 매개변수가 협상되고 사용 가능한 서비스가 공지될 때, 'MMS Initiate-Request' 및 'MMS Initiate-Response' PDU를 사용하여 수행된다.
2. 데이터 세트 초기화 (Dataset initialization): 클라이언트가 'getNameList', 'getVariableListAttributes' 서비스를 사용하여 사용 가능한 논리 노드, 데이터 세트, 변수 및 속성 이름을 발견할 때 수행된다.
   - 이 사례에서는 다음 논리 노드(MMS 도메인)들이 발견된다: 'SIPCTRL', 'SIPDR', 'SIPMEAS', 'SIPPROT'.
   - 이 LN들에 대해, 다음 데이터 세트들이 발견된다: 'SIPCTRL'의 경우 – 'LLN0$Dataset' 및 'LLN0$Dataset_1_1', 'SIPDR'의 경우 – 데이터 세트 없음, 'SIPMEAS'의 경우 – 데이터 세트 없음, 그리고 'SIPPROT'의 경우 – 'LLN0$Dataset_1'.
   - 데이터 세트 발견 후, 사용 가능한 속성 이름이 요청된다:
     - 'SIPCTRL/LLN0$Dataset'에 대해: 'XSWI1$STPosstVal', 'XSWI1STPos$q'
     - 'SIPCTRL/LLN0$Dataset_1_1'에 대해: 'XCBR1$ST$TripOpnCmd$stVal', 'XCBR1STTripOpnCmd$q'
     - 'SIPPROT/LLN0$Dataset_1'에 대해: 'ID_PTOC1STStr$general', 'ID_PTOC1$STStrq'
3. 데이터 접근 (Data access): 다음 단계는 읽기(read) 서비스를 사용하여 데이터에 접근하는 것이다.
   - 각 LN은 변경 사항 확인을 위해 'LLN0DCNamPlt$configRev' 속성을 요청받는다. 그 값은 클라이언트에 의해 각 데이터 세트마다 5초 간격으로 정기적으로 읽힌다.

위에서 언급된 MMS 통신은 아웃스테이션이 변경 사항을 알릴 때 모든 멀티캐스트 그룹으로 전송하는 GOOSE 메시지와 교차하여(interleaved) 나타난다. 토폴로지의 예는 그림 15에 있다.

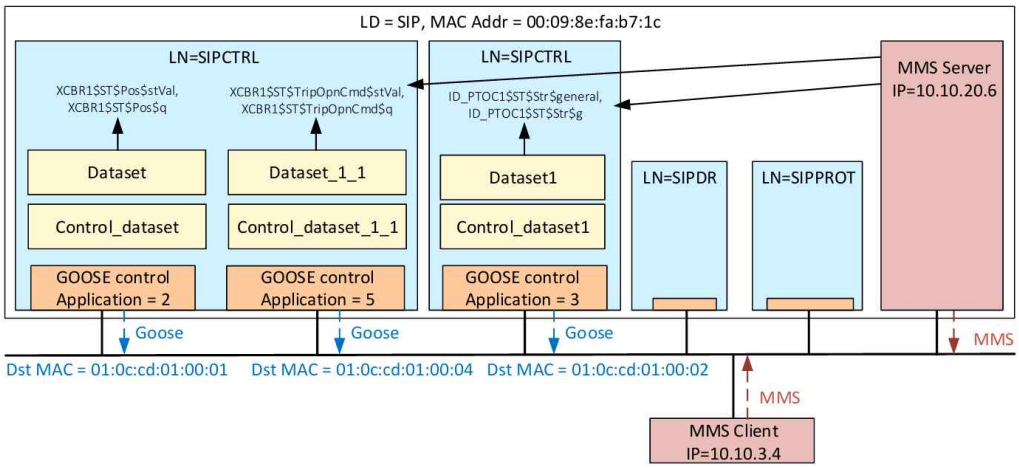


Figure 15: Example of communication topology

3.5 요약

GOOSE 프로토콜에 대한 설명과 가용한 데이터 세트의 분석을 바탕으로, 다음과 같은 관찰을 할 수 있다:

- GOOSE 프로토콜은 송신자(발행자)가 GOOSE 메시지가 담긴 멀티캐스트 이더넷 프레임을 수신자(구독자)에게 전송하는 피어 투 피어(peer-to-peer) 모드를 사용하여 통신한다. 하나의 송신자는 여러 멀티캐스트 그룹으로 데이터를 전송할 수 있다.
- 발행자는 GOOSE 제어 블록을 사용하여 발행될 변수들의 집합을 정의한다. 발행된 데이터는 GOOSE PDU 내의 DatSet 변수를 사용하여 참조된다.
- GOOSE PDU는 이더넷 프레임에 캡슐화될 때, 01:0c:cd:01로 시작하는 예약된 목적지 L2 주소와 0x88b8인 EtherType에 의해 쉽게 식별될 수 있다.[3)]
- GOOSE 메시지는 keep-alive 메커니즘으로서 정기적으로 전송된다. 전송 시간은 로컬에서 구성된다. 발행자 측에 변경 사항이 없으면, 전송된 메시지는 시퀀스 번호 sqNum만 증가할 뿐 거의 동일하다.
- 표준 IEC 61850-8-1 [6]은 여러 GOOSE 서비스(GetGoReference, GetGOOSEElementNumber, GetGoCBValues, SetGoCBValues, SendGOOSEMessage) [3]를 정의하지만, 표준에는 오직 두 가지 유형의 GOOSE PDU, 즉 MngtPdu와 IECGoosePdu만 기술되어 있다(부록 F 참조).
- 가용한 데이터 세트를 분석할 때, IECGoosePdu 프레임만 발견되었다. 모든 GOOSE 메시지는 물리 디바이스(Physical Device)에 의해 피어 투 피어 모드로 멀티캐스트 주소로 전송되었다.
- PDU 내에 데이터 속성(Data Attribute) 참조가 제공되지 않기 때문에 allData 필드에서 전송된 데이터는 쉽게 해석될 수 없다. 발행자 측에서는 원본 데이터를 참조하기 위해 DatSet 식별자가 사용된다(그림 14 참조).
- 데이터 전송의 변경 사항은 상태 번호 StNum을 볼 때 식별할 수 있다. 이 번호는 데이터 세트(DataSet) 내에서 값 변경이 감지될 때마다 증가한다.

3) http://standards-oui.ieee.org/ethertype/eth.txt 참고 [2018.8]

4. MMS 프로토콜

MMS(Manufacturing Message Specification)는 실제 장치 및 기능을 모델링하고, 실제 장치에 대한 정보, (실시간 조건 하에서의) 공정 데이터, 그리고 감시 제어 정보를 네트워크로 연결된 장치 및/또는 컴퓨터 애플리케이션 간에 교환하기 위한 메시징 시스템이다.

MMS는 표준 ISO/IEC 9506-1(서비스) 및 ISO/IEC 9506-2(프로토콜)에 의해 정의된다.

- 서비스 사양(Service specification)은 가상 제조 장치(VMD, Virtual Manufacturing Device)의 정의, 네트워크의 노드 간에 교환되는 서비스(및 메시지), 그리고 VMD 및 서비스와 관련된 속성과 매개변수를 포함한다.
- 프로토콜 사양(Protocol specification)은 통신 규칙, 즉 네트워크 전반에 걸친 메시지의 시퀀싱(sequencing), 메시지의 형식 및 인코딩, 그리고 통신 네트워크의 다른 계층과 MMS 계층 간의 상호작용을 정의한다.

MMS는 클라이언트-서버 모델을 사용하여 통신한다. 클라이언트는 서버에 데이터나 작업을 요청하는 네트워크 애플리케이션 또는 장치(예: 모니터링 시스템, 제어 센터)이다. 서버는 MMS 클라이언트가 접근할 수 있는 가상 제조 장치(VMD)와 그 객체(예: 변수)를 포함하는 장치 또는 애플리케이션이다. VMD 객체는 다른 모든 객체가 위치하는 컨테이너를 나타낸다(그림 16 참조). 클라이언트는 MMS 서비스 요청을 발행하고 서버는 이러한 요청에 응답한다.

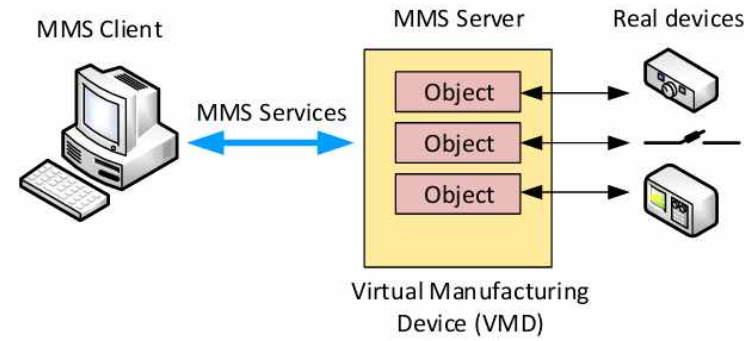


Figure 16: MMS client-server model

MMS는 객체 클래스(이름 있는 변수, 도메인, 프로그램 호출), 인스턴스, 그리고 메서드(읽기, 쓰기, 저장, 시작, 중지 등)를 사용하는 객체 지향적 접근 방식을 취한다.

4.1 VMD 모델 및 MMS 객체

VMD 모델은 다음을 정의한다:

- 서버에 포함된 객체(예: 변수) 및 속성(예: 이름, 값, 유형)
- 객체에 접근하고 관리하기 위한 서비스(예: 읽기, 쓰기)
- 서비스를 처리할 때 장치가 나타내야 하는 동작

VMD 모델은 통신의 네트워크 가시적 측면(network visible aspects)만을 명시한다. 실제 장치가 VMD 모델을 어떻게 구현하는지에 대한 내부 세부 사항은 MMS에서 정의하지 않는다.

MMS는 일반적인 장치에서 찾아볼 수 있는 다양한 객체들을 정의한다. 각 객체에 대해, 표준 ISO 9506-2 [9]는 대응하는 서비스를 정의한다. 표 12는 MMS 객체와 IEC 61850-7-2 [3]에 정의된 IEC 61850 객체 간의 매핑을 보여준다. 표 11의 객체 외에도, ISO 9506-2는 다음 객체들을 정의한다: 프로그램 호출(Program Invocation), 유형(Type), 세마포어(Semaphore), 오퍼레이터 스테이션(Operator Station), 이벤트 조건(Event Condition), 이벤트 동작(Event Action), 이벤트 등록(Event Enrollment), 그리고 트랜잭션(Transaction).

MMS Object	IEC 61850 Object	MMS Services
Application Process VMD	Server	Initiate
		Conclude
		Abort
		Reject
		Cancel
		Identify
Named Variable Objects	Logical Nodes and Data	Read
		Write
		InformationReport
		GetVariableAccessAttribute
		GetNameList
Named Variable List Objects	Data Sets	GetNamedVariableListAttributes
		GetNameList
		DefineNamedVariableList
		DeleteNamedVariableList
		Read
		Write
		InformationReport
Journal Objects	Logs	ReadJournal
		InitializeJournal
		GetNameList
Domain Objects	Logical Devices	GetNameList
		GetDomainAttributes
		StoreDomainContents
Files	Files	FileOpen
		FileRead
		ObtainFile
		FileClose
		FileDirectory
		FileDelete

Table 12: MMS Objects and Services

모든 객체(이름 없는 변수 제외)는 객체 이름으로 식별되며, 이는 다음과 같이 구분된다:

- VMD 특정 (VMD-specific): 영구적이고 사전 로드(pre-loaded)되어 있으며, 모든 클라이언트가 동일한 객체를 본다.
- 도메인 특정 (Domain-specific): 해당 도메인이 존재하는 동안에만 존재한다.
- 애플리케이션 연관 특정 (Application-Association specific): 클라이언트가 연결되어 있는 동안에만 존재한다. 이는 클라이언트가 생성한 데이터 세트와 같은 비영구적 객체에 적용된다.

모든 객체에 대한 접근은 어떤 클라이언트가 객체를 삭제하거나 수정할 수 있는지를 지정하는 특수 객체인 접근 제어 목록(Access Control List)에 의해 제어될 수 있다.

MMS 서비스(메서드)는 MMS 객체를 대상으로 작동한다. 서비스는 객체 생성 또는 삭제(creation, deletion), 객체 값 읽기(get, report), 객체 값 수정(write, alter), 업로드 또는 다운로드(도메인, 파일), 그리고 명령 수행(start, stop 등)을 할 수 있다.

또한 MMS는 VMD에 대한 정보를 얻기 위해 Status, Unsolicited Status, Identify와 같은 서비스를 제공한다. GetNameList 서비스는 VMD 내의 모든 이름 있는 객체의 이름과 유형을 검색하여, VMD에 정의된 객체에 대한 정보를 관리하고 획득하는 기능을 제공한다(부록 E 참조).

MMS는 이름 있는 변수와 이름 없는 변수를 사용한다.

- 이름 없는 변수 (unnamed variables, vadr)는 VMD 내의 고정 물리 주소로 식별된다. 예: numericAddress (0xAF043BC0), symbolicAddress (MW%1004), 또는 unconstrainedAddress (0x76AA).
- 이름 있는 변수 (named variables, vnam)는 객체 이름으로 식별된다.

데이터에 접근할 때, MMS는 전체로서 전송될 변수들의 그룹인 **데이터 세트(Data Set)**를 구성하는 서비스를 제공한다. 이는 일반적으로 각 클라이언트에 대해 개별적으로 수행된다(애플리케이션 연관 특정 유형). 클라이언트는 목록을 정의하고 변수 이름과 전송 모드를 채워 넣는다.

MMS 도메인은 실제 장치 내의 특정 자원을 나타내는 이름 있는 MMS 객체이다. 많은 일반적인 애플리케이션에서, 도메인은 장치 내의 메모리 영역을 나타내는 데 사용된다. 객체(변수, 이벤트, 프로그램 호출 등)는 도메인에 종속될 수 있다. 각 도메인은 MMS의 상태 머신에 의해 제어된다.

4.2 MMS 캡슐화

MMS는 클라이언트와 서버의 주소를 지정하는 방법을 명시하지 않으며, 하위 프로토콜의 주소 지정 체계에 의존한다. 실제로 클라이언트와 서버는 IP 주소로 식별되며, MMS는 TCP 포트 102를 통해 캡슐화된다. 그러나 포트 102는 TCP 상에서 ISO 모델 프로토콜의 일반적인 캡슐화 방식인 ISO TSAP Class 0를 위해 할당된 것이다. 상위 계층은 ISO 식별자(예: TSAP(전송 서비스 액세스 포인트), COTP 소스 및 목적지 참조, OSI 호출(calling) 및 피호출(called) 세션 선택자 등)를 사용한다.

캡슐화는 ISO 스택의 일부인 여러 ISO 프로토콜을 포함한다(그림 17 참조). 모든 MMS 메시지에 모든 프로토콜이 포함되어야 하는 것은 아니다. 다음 본문에서는 각 캡슐화 프로토콜에 대해 설명한다.

<table>
<tr><th>Layer</th><th>PDU</th><th>Protocols</th></tr>
<tr><td rowspan="2">Application (L7)</td><td rowspan="2">APDU</td><td>Manufacturing Message Specification (MMS): ISO 9506</td></tr>
<tr><td>Association Control Service Element (ACSE): ISO/IEC 8650/X.227</td></tr>
<tr><td>Presentation (L6)</td><td>PPDU</td><td>OSI Connection Oriented Presentation ISO 8823/X.226</td></tr>
<tr><td>Session (L5)</td><td>SPDU</td><td>OSI Connection Oriented Session: ISO 8327/X.225</td></tr>
<tr><td rowspan="3">Transport (L4)</td><td rowspan="3">TPDU</td><td>Connection-Oriented Transport Protocol: ISO/IEC 8073/X.224</td></tr>
<tr><td>ISO Transport over TCP (TPKT): RFC 1006</td></tr>
<tr><td>Transmission Control Protocol (TCP): RFC 793</td></tr>
<tr><td>Network (L3)</td><td>NPDU</td><td>Internet Protocol (IP): RFC 791</td></tr>
<tr><td>Data Link (L2)</td><td>Data Frame</td><td rowspan="2">Ethernet: ISO/IEC 8802-3</td></tr>
<tr><td>Physical (L1)</td><td>Bits</td></tr>
</table>

Figure 17: MMS OSI model over TCP/IP

4.2.1 전송 계층 (L4): TPKT 및 COTP

전송 계층에서 MMS 패킷은 TPKT 및 COTP 프로토콜로 캡슐화된다. TPKT(ISO Transport over TCP)는

RFC 1006 [10]에 정의된 프로토콜이다. 이는 TCP/IP 상에서 ISO TP0 프로토콜(전송 프로토콜 클래스 0)을 구현한다. TCP와 TP0의 근본적인 차이점은 TCP가 명시적인 경계 없이 연속적인 옥텟 스트림을 관리한다는 점이다. 반면 TP0는 정보가 이산적인 객체인 NSDU(네트워크 서비스 데이터 단위)로 전송되고 전달되기를 기대한다. 클래스 0의 경우, NSDU는 TPDU(전송 프로토콜 데이터 단위)와 동일하다. 즉, 하나의 TPDU가 단일 NSDU 내에서 전송된다.

TPKT 프로토콜은 TPDU를 구분하기 위해 간단한 패킷화 체계를 사용한다. 각 패킷은 가변 길이의 정수 개 옥텟으로 구성된 객체로 간주된다. TPKT 헤더의 형식은 그림 18에 나와 있다. 헤더는 버전 번호 3, 1바이트 예약 필드, 그리고 2바이트 길이 필드를 포함한다. 길이는 헤더를 포함한 TPKT PDU의 전체 길이이다. TPKT PDU는 목적지 포트 102(ISO-TSAP 클래스 0)를 사용하여 TCP를 통해 전송된다.

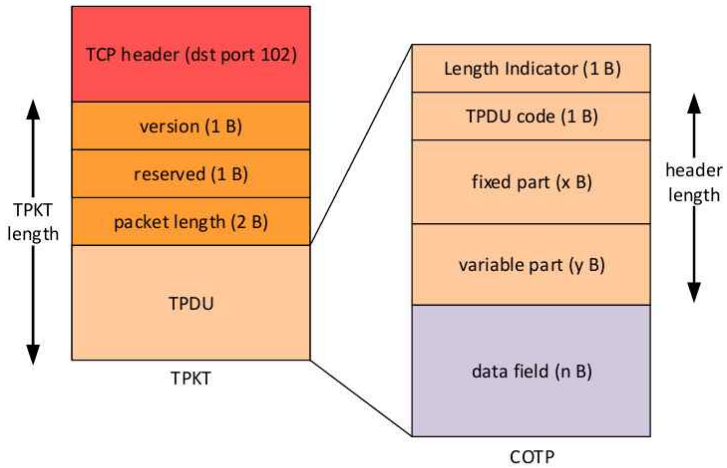


Figure 18: TPKT and COTP encapsulation

연결 지향 전송 프로토콜(COTP)은 ISO 8073/X.224 표준 [11]과 RFC 905 [12]에 정의되어 있다. 이 표준은 여러 가지 COTP 메시지를 정의한다(표 13 참조).

Message	Code	Message	Code
Connection Request (CR)	1110 xxxx	Expedited Data (ED)	0001 0000
Connection Confirm (CC)	1101 xxxx	Data Acknowledgement (AK)	0110 zzzz
Disconnect Request (DR)	1000 0000	Expedited Data ACK (EA)	0010 0000
Disconnect Confirm (DC)	1100 0000	Reject (RJ)	0101 zzzz
Data (DT)	1111 0000	TPDU Error (ER)	0111 0000

Table 13: COTP messages

MMS 통신은 주로 CR (TPDU 코드 0xd0), CC (TPDU 코드 0xe0) 및 DT (TPDU 코드 0xf0) 메시지를 사용한다. 이들의 구조는 그림 19에 나타나 있다. CR 및 CC 메시지는 연결 설정 시 사용되며, DT 패킷은 정상 운영 단계에서 사용자 데이터를 전송한다.

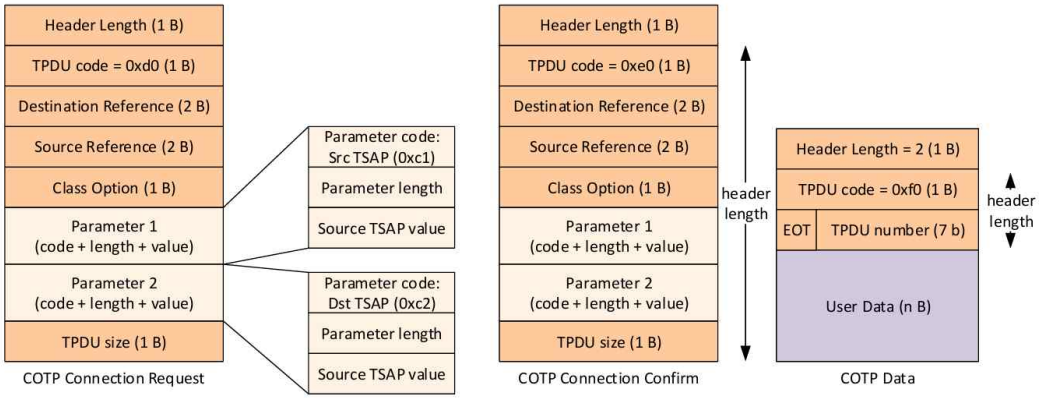


Figure 19: Format of COTP CR, CC and DT messages

TPDU의 첫 번째 옥텟은 길이 지시자 필드(LI)이다. 이는 LI 필드와 사용자 데이터를 제외한 헤더의 길이를 옥텟 단위로 나타낸다. 메시지의 구조는 다음 2바이트의 TPDU 코드에 의해 결정된다. 유효한 TPDU 코드는 표 12에 나와 있으며, 여기서 xxxx 또는 zzzz는 흐름 제어에 사용되는 CDT(크레딧) 필드를 의미한다. 이는 피어 엔티티(peer entity)가 할당한 윈도우 상한값의 초기값을 나타낸다. 전송 클래스 TP0 및 TP1의 경우 이 값은 0으로 설정된다.

이러한 메시지의 구조는 가변적이며 TPDU 코드에 따라 결정된다. CR 및 CC 메시지는 TCP 상에서 전송 세션을 수립한다. 이 메시지들은 전송 연결 식별을 위한 소스 및 목적지 참조(reference), 소스 및 목적지 TSAP 값(TCP/UDP 포트 번호와 유사), 그리고 최대 TPDU 크기를 포함한다. CR 및 CC 메시지는 다른 파라미터들도 전송할 수 있다. 클래스 TP0의 CR 및 CC 메시지에서는 사용자 데이터가 허용되지 않는다. COTP 데이터 패킷 내 사용자 데이터의 길이는 명시되지 않으며, 하위 계층(즉, TPKT 길이)에서 얻어야 한다.

COTP를 사용하여 TCP 세션을 수립할 때, 그림 20과 같이 CR 및 CC 메시지를 사용하여 통신 파트너 간에 소스 및 목적지 참조 번호가 교환된다. 처음에 송신자는 DstRef가 초기화되지 않은 상태에서 자신의 SrcRef를 전송한다. CC 메시지에 의해 확인되면, SrcRef와 DstRef가 모두 설정된다.

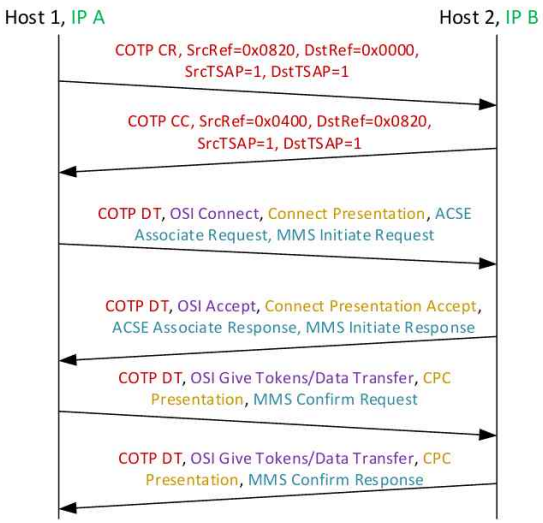


Figure 20: Opening MMS data connection

COTP 세션이 수립된 후, L5에서의 OSI 연결 지향 세션이 생성되어야 한다. 이는 ISO 8327/X.225 프로토콜 [13]과 연계 제어 서비스 요소(ACSE) 서비스 [14]에 의해 제공되며, 아래 내용을 참조한다. ACSE 상위에는 MMS Initiate Request(개시 요청) 및 Response(응답)가 있다.

L4 및 L5 세션이 수립되는 즉시, COTP는 TCP 상에서 TP0 TPDU를 전송하기 위해 데이터 메시지(DT)를 사용한다. COTP DT는 고정된 길이 크기(2바이트)를 가지며 TPDU 코드는 0xf0이다. TP0의 경우 TPDU 번호는 항상 0으로 설정된다.

4.2.1.1 COTP 분할 및 재조립 (Segmenting and Reassembling)

다음 COTP DT 바이트의 첫 번째 비트(EOT, End of TSDU Mark)는 특별한 의미를 갖는다. 이는 시퀀스 내에 후속 TPDU가 있는지 여부를 나타낸다. 데이터 전송의 목적은 전송 연결에 의한 TSDU(Transport Service Data Unit)의 양방향 전송을 허용하는 것이다. 큰 TSDU는 송신 전송 엔티티에서 여러 TPDU로 분할되고 수신 전송 엔티티에서 원래 형식으로 재조립될 수 있다.

값 1은 현재 DT TPDU가 전체 DT TPDU 시퀀스의 마지막 단위임을 나타낸다. 0으로 설정된 경우, 이는 시퀀스의 마지막 DT TPDU가 아님을 의미한다(분할). 이 경우, 최종 수신자는 상위 계층 PDU(예: MMS)를 추출할 수 있도록 전송 계층에서 DT TPDU를 재조립해야 한다. 이때 TPKT 길이는 전체 TSDU가 아닌 현재 DT TPDU 세그먼트의 길이만을 나타낸다. 이러한 상황은 예를 들어 MMS가 getNamedVariableListAttributes를 요청할 때 발생할 수 있다.

4.2.2 OSI 연결 지향 세션 (L5)

OSI 참조 모델은 L5의 세션 프로토콜에 의해 제공되는 연결 지향 세션 서비스를 정의한다. 세션 서비스는 ISO/IEC 8326/X.215에 명시되어 있으며, 연결 지향 및 비연결성 세션 서비스 프리미티브, 연결 수립 및 데이터 전송을 기술한다. 표준 ISO-IEC 8327/X.225 [13]은 MMS에서 사용하는 연결 지향 세션 프로토콜을 정의한다.

이 표준은 30개 이상의 서로 다른 SPDU(Session Protocol Data Unit)를 정의하지만, MMS는 세 가지 유형만 사용한다. 연결 수립을 위한 Connect (SPDU ID = 13)(그림 21 참조)와 Accept (SPDU ID = 14), 그리고 통신을 위한 Give Tokens/Data Transfer (SPDU ID = 1)이다.

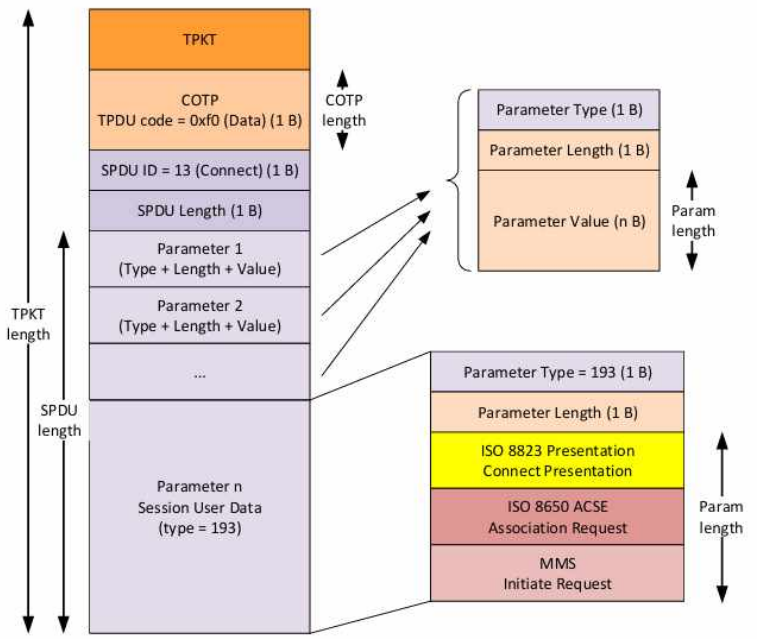


Figure 21: OSI Connect SPDU transmitted over TCP/IP

SPDU의 구조는 고정되어 있다. SPDU 식별자(1바이트)로 시작하며, 길이 지시자(LI, 1바이트)와 파라미터 필드 목록이 뒤따른다. 각 파라미터 필드는 유형(1바이트), 길이(1바이트) 및 값으로 구성된 TLV 구조를 갖는다.

4.2.2.1 Connect SPDU

Connect SPDU는 연결 지향 세션 연결을 개시한다. Connect SPDU의 유형은 13(10진수)이며, SPDU 길이와 파라미터 필드 목록을 포함한다(그림 21 참조). 가능한 파라미터로는 Connect Accept Item(파라미터 유형=5), Session Requirement(유형=20), Calling Session Selector(유형=51), Called Session Selector(유형=52) 또는 Session User Data(유형=193)가 있으며, [13, 부속서 C.2]를 참조한다. 마지막 파라미터(Session User Data)는 상위 계층 PDU인 ACSE 및 MMS를 캡슐화한다.

Connect SPDU는 표준 X.227 [14]에 기술된 ACSE Association Request(AARQ) APDU를 캡슐화한다. AARQ PDU는 TLV 구조를 갖는 BER 인코딩 패킷이다(부록 H 참조). AARQ APDU는 OID iso(1).standard(0).iso9506(9506).part(2).mms-annex version1(3)을 사용하여 MMS 컨텍스트를 정의한다. 사용자 정보(user-information) 필드에서 ACSE AARQ는 MMS Initiate Request를 전송한다. MMS Initiate Request는 MMS 버전(proposedVersionNumber), 적합성 파라미터(proposedParameterCBB, 적합성 빌딩 블록), 지원되는 MMS 서비스 목록(serviceSupportedCalling) 및 기타 파라미터를 포함한다(4.2.4절 참조).

4.2.2.2 Accept SPDU

Connect SPDU에 대한 응답은 유사한 형식을 갖는 Accept SPDU이다(그림 22 참조). 애플리케이션 계층에서 이는 ACSE Association Response(ACSE-AARE) 및 MMS Initiate Response를 캡슐화한다. MMS는 협상된 프로토콜 버전(negotiatedVersionNumber), 적합성 파라미터(negotiatedParameterCBB), 지원되는 서비스 목록(servicesSupportedCalled) 등을 전송한다.

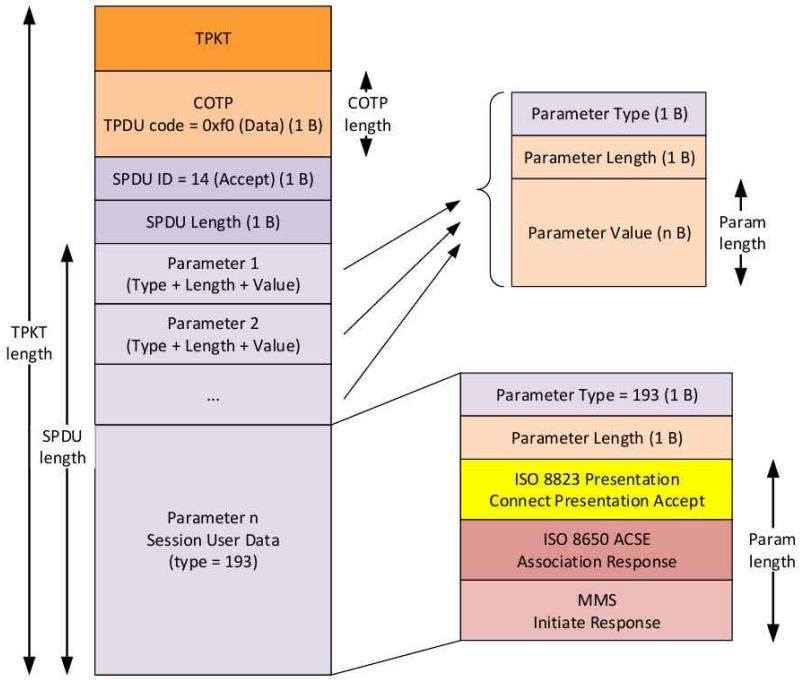


Figure 22: OSI Accept SPDU transmitted over TCP/IP

4.2.2.3 Give Tokens/Data Transfer SPDU

OSI L5 Connect 및 Accept 메시지가 성공적으로 교환된 후, 이어지는 MMS 메시지들은 두 개의 L5 PDU인 Give Tokens 및 Data Transfer, 프레젠테이션 프로토콜 ISO 8823, 그리고 MMS 프로토콜로 구성된 시퀀스 내에만 캡슐화된다. 더 이상 ACSE 캡슐화는 사용되지 않는다. PDU의 구조는 그림 23에 나타나 있다.

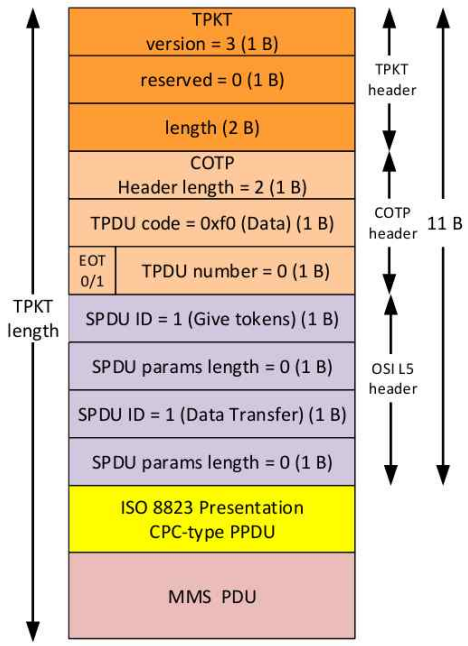


Figure 23: OSI Data Transfer SPDU transmitted over TCP/IP

그림 23에서 보는 바와 같이, 표준 X.225는 SPDU 연결(concatenation)을 허용한다 [13, parts 6.3.7, 7.16]. 이 경우 기본 연결이 적용된다. 즉, Give Tokens SPDU가 Data Transfer SPDU와 연결된다. 두 SPDU 모두 파라미터가 없다. 표준에서는 Data Transfer SPDU를 단독으로 전송하는 것을 허용하지 않으므로, Give Tokens SPDU는 연결된 SPDU 시퀀스를 시작하는 데 사용된다. Data Transfer SPDU는 연결이 수립된 후 사용자 데이터를 전송하는 데 사용된다.

PDU는 고정된 형식을 갖는다: TPKT용 03 00 xx xx (4바이트), COTP용 02 0f 80/00 (3바이트), 그리고 L5 헤더용 01 00 01 00 (4바이트)이며, 즉 11바이트의 OSI 헤더로 구성된다. 그 뒤에 프레젠테이션 계층이 이어진다.

4.2.3 OSI 연결 지향 프레젠테이션 (L6)

OSI 프레젠테이션 계층(L6)은 프레젠테이션 연결 설정 중 협상 및 적합성 요구사항 명세를 제공한다. 또한 데이터 전송 중 데이터 인코딩을 제공한다. 표준 ISO 8822/X.216은 프레젠테이션 서비스를 정의하며, 표준 ISO 8823/X.226 [15]은 프레젠테이션 프로토콜 데이터 단위(PPDU)를 기술한다.

PPDU의 구조는 ASN.1 [15, part 8]에 의해 정의된다. MMS 통신은 단 세 가지 유형의 PPDU만 사용한다: Connect Presentation PPDU (CP-유형), Connect Presentation Accept PPDU (CPA-유형), 그리고 사용자 데이터를 포함하는 CPC-유형 PPDU이다. 이들 PPDU의 형식 기술은 ASN.1로 기술되어 있으며, 부록 I를 참조한다. 기술된 바와 같이, 그 구조는 가변적이며 BER 인코딩된다. PPDU의 길이는 PPDU 내에 지정된 옵션에 따라 달라진다. 다음 부분에서는 이 세 가지 PPDU의 형식에 대해 논의한다.

4.2.3.1 Connect Presentation PPDU

Connect Presentation (CP) PPDU는 연결 시작 시에만 L5 Connect SPDU 내에서 전송되며, L7의 AARQ 및 MMS Initiate Request를 캡슐화한다(그림 21 참조). L6 CP PPDU는 L5 Connect SPDU에 캡슐화된 Session user data 파라미터(유형=193)의 일부이다. CP PPDU는 SPDU Session user data의 파라미터 유형 및 길이 뒤에 위치한다.

CP PPDU는 BER 인코딩된다. 즉, TLV 세 쌍(triplets)으로 구성된다. PPDU 형식은 다음과 같다:

```
CP-type ::= SET {
        mode-selector                   [0]         IMPLICIT   Mode-selector,
        normal-mode-parameters    [2]         IMPLICIT   SEQUENCE {
                        protocol-version                        [0]         IMPLICIT Protocol-version DEFAULT {version-1},
                        calling-presentation-selector           [1] IMPLICIT Calling-presentation-selector OPTIONAL,
                        called-presentation-selector            [2] IMPLICIT Called-presentation-selector OPTIONAL,
                        presentation-context-definition-list    [4] IMPLICIT Presentation-context-definition-list
OPTIONAL,
                        default-context-name                    [6] IMPLICIT Default-context-name OPTIONAL,
                        presentation-requirements               [8] IMPLICIT Presentation-requirements OPTIONAL,
                        user-session-requirements               [9] IMPLICIT User-session-requirements OPTIONAL
                        user-data                               User-data OPTIONAL
            } OPTIONAL
}
Mode-selector ::= SET {
            mode-value              [0]         IMPLICIT    INTEGER {
                                    x410-1984-mode          (0),
                                    normal-mode             (1)         }
}
```

```
Calling-presentation-selector ::= Presentation-selector
Called-presentation-selector ::= Presentation-selector
Presentation-selector ::= OCTET STRING
Presentation-context-definition-list ::= Context-list
Context-list ::= SEQUENCE OF SEQUENCE {
        presentation-context-identifier     Presentation-context-identifier,
        abstract-syntax-name                Abstract-syntax-name,
        transfer-syntax-name-list           SEQUENCE OF         Transfer-syntax-name
}
Presentation-context-identifier ::= INTEGER
Abstract-syntax-name ::= OBJECT IDENTIFIER
Transfer-syntax-name ::= OBJECT IDENTIFIER
Presentation-requirements ::= BIT STRING {
        context-management  (0),
        restoration         (1)
}
User-data ::= CHOICE {
        simply-encoded-data  [APPLICATION 0]  IMPLICIT  Simply-encoded-data,
        fully-encoded-data   [APPLICATION 1]  IMPLICIT  Fully-encoded-data
}
Fully-encoded-data ::= SEQUENCE OF PDV-list
PDV-list ::= SEQUENCE {
        transfer-syntax-name                Transfer-syntax-name                OPTIONAL,
        presentation-context-identifier     Presentation-context-identifier,
        presentation-data-values            CHOICE {
                single-ASN1-type            [0]         ABSTRACT-SYNTAX.&Type (CONSTRAINED BY{
                                            -- Type corresponding to presentation context identifier --}
},
                octet-aligned               [1]         IMPLICIT  OCTET STRING,
                arbitrary                   [2]         IMPLICIT  BIT STRING
        }
}
```

CP PPDU의 예시는 다음과 같다: 31 81 9e a0 03 80 01 01 a2 81 96 81 02 00 01 82 04 00 00 00 01 a4 23 30 0f 02 01 01 06 04 52 01 00 01 30 04 06 02 51 01 30 10 02 01 03 06 05 28 ca 22 02 01 30 04 06 02 51 01 88 02 06 00 61 61 30 5f 02 01 01 a0 5a 60 58 80 02 07 80 a1 07 06 05 28 ca 22 02 03 a2 06 06 04 2b ce 0f 17 a3 03 02 01 17 a6 06 06 04 2b ce 0f 17 a7 03 02 01 17 be 2f 28 2d 02 01 03 a0 28 a8 26 80 03 …

이 PPDU는 TLV 구조의 유형(Type)을 나타내는 0x31 바이트로 시작한다. 0x31(이진수로 0011 0001)은 태그 번호 17(10001)인 Set(집합)을 갖는 구성적 형태(constructive form, 1)의 유니버설 데이터 유형(universal data type, 00)을 나타낸다. 그다음 바이트들은 길이(Length) 필드를 형성한다. CP PPDU 데이터가 127바이트보다 길기 때문에, ASN.1 길이 필드는 'Long definite length' 형식을 사용한다. 이 형식에서 길이 필드는 여러 바이트를 포함하며, 사용 가능한 데이터세트에서 볼 수 있듯이 일반적으로 2바이트이다. 길이 필드 바이트의 수는 첫 번째 길이 필드 바이트의 하위 7비트에 의해 결정된다(부록 G 참조).

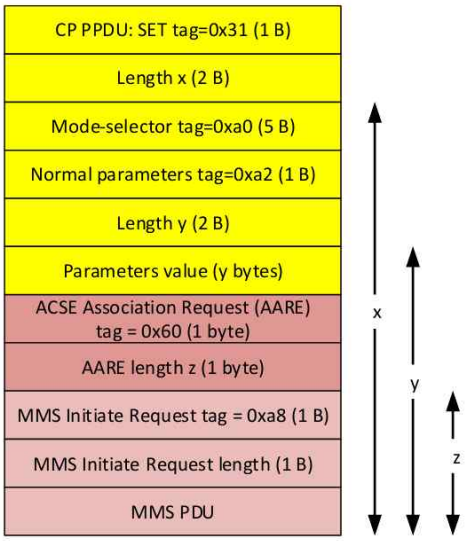


Figure 24: Format of CP PPDU for opening L6 association with embedded AARE and MMS messages

길이(Length) 필드 뒤에는 식별자 0xa0 (1010 0000)을 가진 TLV 세 쌍이 온다. 이는 구성적 형태(constructive form)의 컨텍스트 특정 데이터 유형과 태그 0, 즉 mode-selector(모드 선택자)를 나타낸다. 이것이 구성적 형태이므로, 식별자 0x80을 가진 내장된 TLV를 캡슐화한다. 이 내장된 TLV는 태그 0(mode-value)과 값 1(normal-mode)을 가진 컨텍스트 특정 데이터 유형을 참조한다.

다음 TLV는 0xa2 (1010 0010) 바이트로 시작하며, 이는 normal-mode-parameters SEQUENCE(시퀀스)를 참조하고 값과 함께 선택적 파라미터 목록을 포함한다. 자주 사용되는 파라미터는 다음과 같다.

- calling-presentation-selector 및 called-presentation-selector 파라미터는 각각 L6 목적지 및 L6 소스 주소를 형성한다. 이 주소들은 PPDU 어소시에이션(association)이 수립된 후 대화 내에서 더 이상 사용되지 않는다.
- 시작 바이트 0xa4 (1010 0100)를 가진 TLV는 구성적 형태의 컨텍스트 특정 데이터 유형을 나타낸다. 태그 4는 presentation-context-definition-list를 의미한다. 이 리스트에는 두 개의 항목이 있다.
  - ⊙ 리스트의 각 항목은 SEQUENCE를 나타내는 0x30 (0011 0000) 바이트로 시작한다.
  - ⊙ 첫 번째 항목은 값 1을 갖는 presentation-context-identifier(태그 2, INTEGER), 값 2.2.1.0.1 (ACSE 추상 구문 버전 1)을 갖는 abstract-syntax-name(태그 06, OID), 그리고 값 2.1.2 (ASN.1 BER)를 갖는 transfer-syntax-name을 포함한다.
  - ⊙ 두 번째 항목은 값 3을 갖는 presentation-context-identifier, OID 1.0.9506.2.1 (mms-abstract-syntax-version1)을 갖는 abstract-syntax-name, 그리고 값 2.1.2 (ASN.1 Basic Encoding Rules)를 갖는 transfer-syntax-name을 포함한다.

  이 정보는 presentation-context-identifier가 사용자 데이터(user-data)에 나타나 콘텐츠의 유형(즉, ACSE 패킷인지 MMS 패킷인지)을 나타내기 때문에 중요하다. 컨텍스트 리스트는 presentation-context-identifier(정수)를 abstract-syntax-name(OID)에 매핑한다. OID는 ACSE 구문 또는 MMS 구문을 참조한다. 따라서 PDV 리스트(PDV-list) 내의 presentation-context-identifier는 콘텐츠가 ACSE인지 MMS인지를 알려준다. 이 정보는 L5 SPDU 유형으로부터 유추할 수도 있는데, 보통 ACSE 패킷은 Connect 또는 Accept SPDU에 캡슐화되는 반면, MMS 패킷은 Data Transfer SPDU에 위치하기 때문이다.
- 컨텍스트 리스트(context-list) 뒤에는 식별자 0x88 (1000 1000)을 가진 SEQUENCE가 오는데, 이는

presentation-requirements 유형(태그 8)을 참조한다. 이 2바이트 필드는 BIT STRING으로, 첫 번째 바이트 0x06은 사용되지 않는 비트 수(6비트)를 나타내며, 나머지는 값을 설명하는데 두 비트 모두 0이다. 그러나 이 필드는 테스트된 모든 CP PPDU에 존재하는 것은 아니다.

- 마지막 유용한 필드는 식별자 0x61 (0110 0001)로 시작하며, 이는 애플리케이션 태그 1을 가진 구성적 형태의 애플리케이션 특정 클래스, 즉 fully-encoded-data를 설명한다. 식별자 0x30 (SEQUENCE)을 가진 내장된 TLV는 presentation-context-identifier가 1 (ACSE 데이터)로 설정된 PDV-list와 single-ASN1-type (식별자 0xa0)으로 인코딩된 애플리케이션 데이터를 포함한다. 식별자 뒤에는 1바이트의 콘텐츠 길이와 식별자 0x60으로 시작하는 AARQ, 그리고 태그 0xa8을 가진 MMS Initiate Request가 따른다(4.2.4절 참조).

4.2.3.2 Connect Presentation Accept PPDU

Connect Presentation Accept (CPA) PPDU는 L6 연결의 초기 단계에서 CP PPDU에 대한 응답이다. CPA PPDU는 L7의 AARE 및 MMS Initiate Response와 함께 L5 Accept SPDU 내에서 전송된다(그림 22 참조). 패킷의 형식은 ASN.1에 의해 다음과 같이 기술된다.

```
CPA-PPDU ::= SET {
        mode-selector                          [0]   IMPLICIT  Mode-selector,
        normal-mode-parameters                 [2]   IMPLICIT  SEQUENCE {
            protocol-version                           [0] IMPLICIT Protocol-version DEFAULT {version 1},
            responding-presentation-selector           [3] IMPLICIT Responding-presentation-selector OPTIONAL,
            presentation-context-definition-result-list [5] IMPLICIT Presentation-context-definition-result-list OPTIONAL,
            presentation-requirements                  [8] IMPLICIT Presentation-requirements OPTIONAL,
            user-session-requirements                  [9] IMPLICIT User-session-requirements OPTIONAL,
            user-data                                  User-data      OPTIONAL
        } OPTIONAL
}
Responding-presentation-selector ::= Presentation-selector
```

CPA PPDU의 형식은 4.2.3.1절에 기술된 CP PPDU와 유사하다(그림 25 참조).

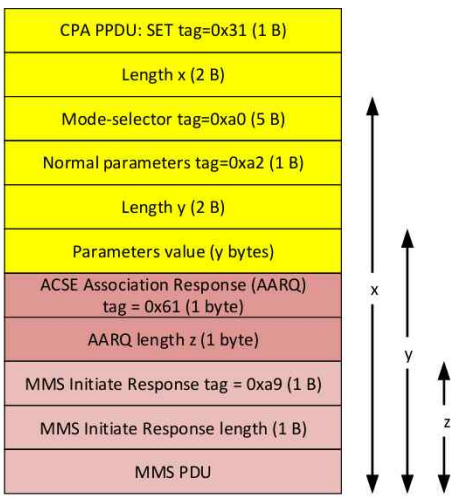


Figure 25: Format of CPA PPDU with embedded AARQ and MMS messages

이 메시지는 태그 0x31로 시작한다. Normal-mode-parameters 섹션 내의 주요 파라미터는 다음과 같다.

- Responding-presentation-selector (식별자 0x83)는 CP PPDU의 called-presentation-selector 값을 유

지한다.

- presentation-context-definition-list (식별자 0xa5)는 CP PPDU에서 제안된 transfer-syntax-names와 함께, 제안된 컨텍스트에 대한 수락 또는 거부 결과를 전송한다.
- 또한 presentation-requirements (식별자 0x88) 및 user-data (식별자 0x61) 필드도 존재한다.
- user-data는 CP PPDU와 유사하게 presentation-context-identifier (식별자 0x02, 값 0x01) 및 presentation-data-value (식별자 0xa0)를 갖는 PDV-list (식별자 0x30, SEQUENCE)를 포함한다. CPA PPDU의 경우, presentation-context-identifier는 ACSE 추상 구문 버전 1 (OID 2.2.1.0.1)을 나타내는 컨텍스트 1을 가리킨다.
- 캡슐화된 ACSE PDU는 Application Response (AARQ) 유형이며 presentation-data-value 필드 뒤에 온다. 그 식별자는 0x61이다(부록 H 참조). AARQ 내의 user-information 필드는 태그 0xa9로 시작하는 MMS Initiate-Response PDU를 포함한다(4.2.4절 참조).

CPA PPDU의 예시는 다음과 같다: 31 81 86 a0 03 80 01 01 a2 7f 83 04 00 00 00 01 a5 12 30 07 80 01 00 81 02 51 01 30 07 80 01 00 81 02 51 01 88 02 06 00 61 5f 30 5d 02 01 01 a0 58 61 56 80 02 07 80 a1 07 06 05 28 ca 22 02 03 a2 03 02 01 00 a3 05 a1 03 02 01 00 a4 06 06 04 2b ce 0f 17 a5 03 02 01 17 be 2e 28 2c 02 01 03 a0 27 a9 25 80 02 …

4.2.3.3 CPC-유형 PPDU

CPC-유형 PPDU는 데이터 전송 단계에서 전송되는 MMS 데이터를 캡슐화한다(그림 26 참조). 이들 PPDU는 전송 과정에서 가장 빈번하게 나타나는 PDU이다. 이 PPDU의 형식은 ASN.1 표기법을 사용하여 명시되어 있으며, 다음 정의를 참조한다.

```
CPC-type ::= User-data
User-data ::= CHOICE {
        simply-encoded-data         [APPLICATION 0]     IMPLICIT   Simply-encoded-data,
        fully-encoded-data          [APPLICATION 1]     IMPLICIT   Fully-encoded-data
}

Fully-encoded-data ::= SEQUENCE OF PDV-list

PDV-list ::= SEQUENCE {
        transfer-syntax-name            Transfer-syntax-name            OPTIONAL,
        presentation-context-identifier Presentation-context-identifier,
        presentation-data-values        CHOICE {
                single-ASN1-type        [0]         ABSTRACT-SYNTAX.&Type (CONSTRAINED BY{
                                        -- Type corresponding to presentation context identifier --
}),
                octet-aligned           [1]         IMPLICIT   OCTET STRING,
                arbitrary               [2]         IMPLICIT   BIT STRING
        }
}
```

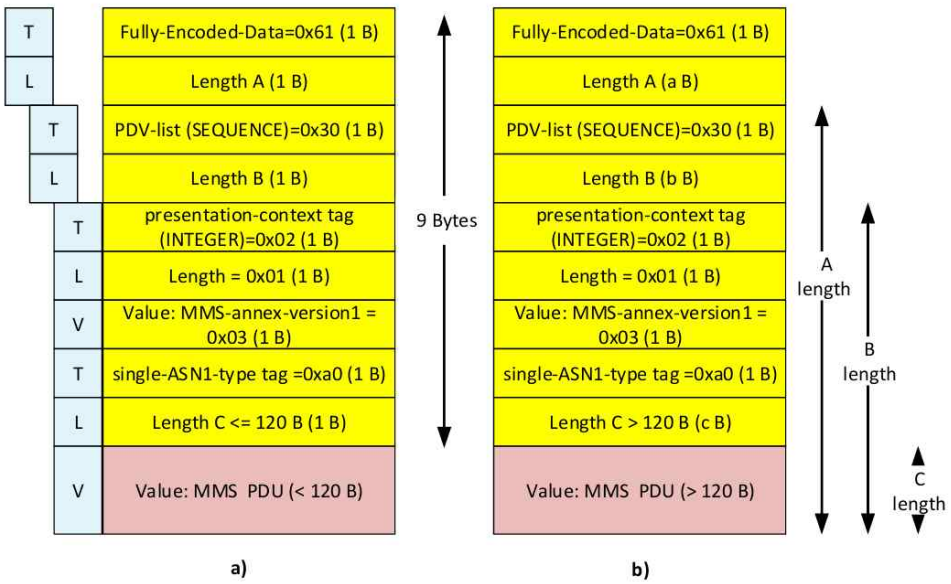


Figure 26: Format of CPC-type PPDU for MMS data transfer with MMS PDU lesser than 120 B (a) and greater (b).

CPC-유형 PPDU의 예:
61 18 30 16 02 01 03 a0 11 a0 0f 02 02 06 27 a1 09 a0 03 80 01 09 a1 02 80 00

이 PPDU는 바이트 0x61 (0110 0001)로 시작하는데, 이는 데이터 유형 Fully-encoded-data를 나타내는 태그 1을 가진 구성적 형태(constructive form)의 애플리케이션 특정 유형을 의미한다. 이 바이트 뒤에는 데이터 길이와 식별자 0x30 (SEQUENCE, 태그 16)을 가진 새로운 내장된 TLV 구조가 이어진다. 이 태그는 PDV-list (presentation-data-values)를 참조한다. 선택적 필드인 transfer-syntax-name은 프레젠테이션 컨텍스트에 대해 둘 이상의 전송 구문 이름이 제안된 경우에만 존재하는데, 여기서는 해당하지 않는다.

ISO 9506 [9]은 OID 1.0.9506.2.3 (mms-annex-version1)을 갖는 하나의 프레젠테이션 컨텍스트만을 정의한다. 다음 TLV는 태그 0x02와 값 0x03으로 presentation-context-identifier를 인코딩했다. 이 값은 앞서 설명한 CP 및 CPA PPDU에서 협상된 presentation-context-list를 참조한다.

마지막 TLV는 0xa0 (1010 0000)으로 시작하며, 이는 태그 0 (single-ASN1-type)을 가진 구성적 형태의 컨텍스트 특정 데이터 유형을 의미한다. 이는 PDV-list 값이 정확히 하나의 프레젠테이션 데이터 값을 포함함을 의미하며, 이는 BER [15]에 따라 인코딩된 단일 ASN.1 유형이다. 이어지는 바이트는 내장된 PDU의 길이를 나타내며, 그 뒤에 태그 0xa0으로 시작하는 MMS 메시지가 따른다.

PPDU 헤더의 형식은 대부분의 MMS 메시지에서 변경되지 않는다. PPDU가 127바이트보다 짧은 메시지, 즉 MMS PDU 길이가 120바이트 미만인 경우, PPDU 내부 TLV의 길이 필드는 1바이트의 고정 길이를 갖는다. 이는 PPDU 헤더의 전체 길이가 9바이트임을 의미한다(그림 24 a 참조). 이는 TPKT 길이가 145바이트 미만인 메시지에 유효하다.

MMS PDU가 120바이트보다 긴 메시지의 경우, PPDU의 TLV 길이 필드는 1바이트보다 길 수 있다(부록 G, 'long definite length' 참조). 이 경우 길이 필드는 0x80 이상의 값으로 시작한다. 즉, 최상위 비트가 1이며, 나머지 값은 길이 필드의 바이트 수를 나타낸다. 이 경우, 헤더 내의 길이 필드에 따라 PPDU 헤더는 9바이트보다 길어진다(그림 24 b 참조).

4.3 MMS 프로토콜

ISO 9506 [9]에 정의된 MMS 프로토콜은 두 가지 유형의 통신을 구현한다.

- 확인(Confirmed) MMS 서비스
  - 확인 MMS 서비스는 요청 서비스를 포함하는 확인 요청(Confirmed-Request) PDU를 통해 요청된다(부록 J 참조). 표준 ISO 9506은 getNameList, read, write, getVariableAccessAttributes 등 87개의 서로 다른 서비스를 정의한다.
  - 각 확인 요청 PDU는 확인 응답(Confirmed-Response) PDU 또는 확인 오류(Confirmed-Error) PDU에 의해 확인된다.
  - 또한, 확인 요청 PDU는 취소 요청(Cancel-Request) PDU에 의해 중단될 수 있으며, 이는 취소 응답(Cancel-Response) PDU 또는 취소 오류(Cancel-Error) PDU를 사용하여 확인된다.
  - 요청 PDU의 각 인스턴스는 32비트 부호 없는 정수인 InvokeID를 사용하여 해당 응답 PDU와 상호 연관된다.

- 비확인(Unconfirmed) MMS 서비스
  - 비확인 MMS 서비스의 경우, 응답 PDU나 오류 PDU가 수신되지 않는다. 또한, 비확인 MMS 서비스는 취소할 수 없다.
  - 이 표준은 세 가지 다른 비확인 서비스를 정의한다: informationReport, unsolicitedStatus, eventNotification.

이 표준은 14가지 유형의 MMS PDU를 정의하며, 이들은 표준에서 부여한 태그 번호를 가진 TLV 식별자를 통해 쉽게 식별할 수 있다(부록 J 및 표 14 참조). 추가 데이터 세트를 포함하는 서비스는 Type 필드의 P 비트를 1 (1010 = 0xa)로 설정하는 반면, 단순한 내용을 가진 서비스는 Type 필드를 0 (1000 – 0x8)으로 설정한다는 점에 유의한다.

MMS PDU	TLV identifier	Tag number
confirmed-RequestPDU	0xa0	0
confirmed-ResponsePDU	0xa1	1
confirmed-ErrorPDU	0xa2	2
unconfirmed-PDU	0xa3	3
rejectPDU	0xa4	4
cancel-RequestPDU	0x85	5
cancel-ResponsePDU	0x86	6
cancel-ErrorPDU	0xa7	7
initiate-RequestPDU	0xa8	8
initiate-ResponsePDU	0xa9	9
initiate-ErrorPDU	0xaa	10
conclude-RequestPDU	0x8b	11
conclude-ResponsePDU	0x8c	12
conclude-ErrorPDU	0xad	13

Table 14: Types and identifiers of all MMS PDUs

4.3.1 통신 개시

연결은 InitiateRequest PDU와 InitiateResponse PDU를 사용하여 개설된다(부록 J 참조). 이들은 연결의 세부 사항과 지원되는 서비스를 협상하는 데 사용된다. 지원되는 서비스는 비트열(bit string)로 인코딩된다. 다음 예제는 InitiateRequest 및 InitiateResponse PDU를 보여준다.

예제 1: a8 26 80 03 00 fa 00 81 01 0a 82 01 0a 83 01 05 a4 16 80 01 01 81 03 05 e1 00 82 0c 03 a0 00 00 00 00 02 00 00 00 ed 10

- 컨텍스트 특정(context-specific) 태그 8을 가진 0xa8 (1010 1000)은 길이 38바이트의 InitiateRequest PDU를 지정한다.
- 태그 0을 가진 0x80 (1000 0000)은 localDetailCalling을 설명하며, 이는 값 64,000을 가진 3바이트 정수이다.
- 태그 1을 가진 0x81 (1000 0001)은 값 10을 가진 ProposedMaxServOutstandingCalling 매개변수를 설명한다.
- 마찬가지로, 0x82는 값 10을 가진 ProposedMaxServOutstandingCalled 매개변수를 설명한다.
- 태그 0x83은 값 5를 가진 proposedDataStructureNestingLevel 매개변수를 참조한다.
- 0xa4 (1010 0100)는 길이 22바이트(16진수로 16)인 initRequestDetail의 시퀀스(sequence)이다. 이는 다음 데이터를 포함한다:
  - 0x80은 정수 값 1을 가진 proposedVersionNumber를 참조한다.
  - 0x81은 proposedParameterCBB (적합성 빌딩 블록, conformance building block)를 참조한다. 이는 값 0xe1 00 (1110 0001 0000 0000)을 가진 11비트 문자열이며, 옵션 str1 (배열 지원), str2 (구조체 지원), vnam (명명된 변수 지원) 및 vlis (명명된 변수 목록)에 해당한다.
  - 0x82는 사용 가능한 서비스를 인코딩하는 11바이트 비트열을 나타낸다. 값은 1010 0000 0000 0000 0000 0000 0000 0000 0000 0000 0010 0000 0000 0000 0000 0000 0000 1110 1101 0010 0 이며, 이는 사용 가능한 서비스가 status, identify, obtainFile, fileOpen, fileRead, fileClose, fileDelete, fileDirectory, informationReport 및 conclude임을 의미한다

예제 2: a9 25 80 02 7d 00 81 01 0a 82 01 08 83 01 05 a4 16 80 01 01 81 03 05 e1 00 82 0c 03 ee 08 00 00 04 00 00 00 01 ed 18

- 선행 바이트 0xa9는 길이 37바이트(16진수로 25)인 InitiateResponse PDU를 나타낸다.
- 0x80은 값 32,000 (0x7d00)을 가진 localDetailCalled 매개변수를 설명한다.
- 태그 0x81과 0x82를 가진 다음 두 매개변수는 각각 값 10과 8을 가진 negotiatedMaxServOutstandingCalling 및 negotiatedMaxServOutstandingCalled를 참조한다.
- 0x83은 값 5를 가진 negotiatedDataStructureNestingLevel을 나타낸다.
- 0xa4는 협상된 매개변수들의 새로운 시퀀스(sequence)를 시작한다:
  - 0x80은 1인 negotiatedVersionNumber를 참조한다.
  - 0x81은 negotiatedParameterCBB를 참조하며, 이는 값 0xe10을 가진 비트열이다(위 참조).
  - 0x82는 servicesSupportedCalled를 참조하며, 이는 값 ee:08:00:00:04:00:00:00:01:ed:18 (1110 1110 0000 1000 0000 0010 0000 0000 0000 0000 0000 0000 0000 0001 1110 1101 0001 1)을 가진 11바이트 문자열이다. 이 비트열은 다음 서비스를 인코딩한다:
    status, getNameList, identify, read, write, getVariableAccessAttributes, getNameVariableListAttributes, getDomainAttributes, getCapabilityList, fileOpen, fileRead, fileClose, fileDelete, fileDirectory, informationReport, conclude, cancel.

initRequest 및 initResponse를 사용한 협상된 매개변수의 예가 그림 29에 나타나 있다.

parameters	initRequest	initResponse
localDetailCalling	64000	32000
MaxServOutstandingCalling	10	10
MaxServOutsendingCalled	10	8
DataStructureNesting	5	5
Version	1	1
ConformanceBlock options	str1	str1
	str2	str2
	vnam	vnam
	vlis	vlis
Supported Services	status	status
		getNameList
	identify	identify
		read
		write
		getVariableAccessAttributes
		getNameVariableAttributes
		getDomainAttributes
		getCapabilityList
	obtainFile	
	fileOpen	fileOpen
	fileRead	fileRead
	fileClose	fileClose
	fileDelete	fileDelete
	fileDIrectory	fileDirectory
	informationReport	informationReport
	conclude	conclude
		cancel

Figure 29: Example of MMS parameter negotiation

4.3.2 데이터 전송

MMS 통신의 대다수는 확인 요청(Confirmed-Request) 및 확인 응답(Confirmed-Response) 메시지에 의해 이루어진다. 확인 요청 PDU는 요청의 명확한 식별자인 InvokeID와 ConfirmedServiceRequest의 유형(예: status, getNameList, read, write 등)을 포함한다. 이러한 각 서비스는 요청 유형에 따라 서로 다른 데이터를 전송한다.

데이터 전송은 두 가지 단계로 구성된다.

- 데이터 세트 초기화 - 이 단계에서 클라이언트는 getNameList, getVariableListAttributes, getNamedVariableListAttributes 등을 사용하여 사용 가능한 논리 노드(logical nodes), 데이터 세트, 변수 및 속성의 이름을 요청한다.
- 데이터 액세스 - 초기화 후, 읽기(read), 쓰기(write) 및 기타 작업을 위해 데이터 객체(data objects)에 액세스한다.

다음 부분에서는 가장 빈번하게 사용되는 MMS PDU의 형식을 보여주며, 이러한 패킷을 식별하고 파싱하는 방법에 대한 권장 사항을 제시한다. 실제 통신 예제는 4.4절에 기술되어 있다.

4.3.2.1 페이로드 크기가 120B보다 작은 MMS PDU

확인 응답(Confirmed-Response) PDU는 동일한 InvokeID를 가지며, 요청 결과(값, 상태 등)가 포함된 confirmedServiceResponse를 포함한다. 120바이트보다 작은 MMS PDU의 형식은 그림 30에 나타나 있다.

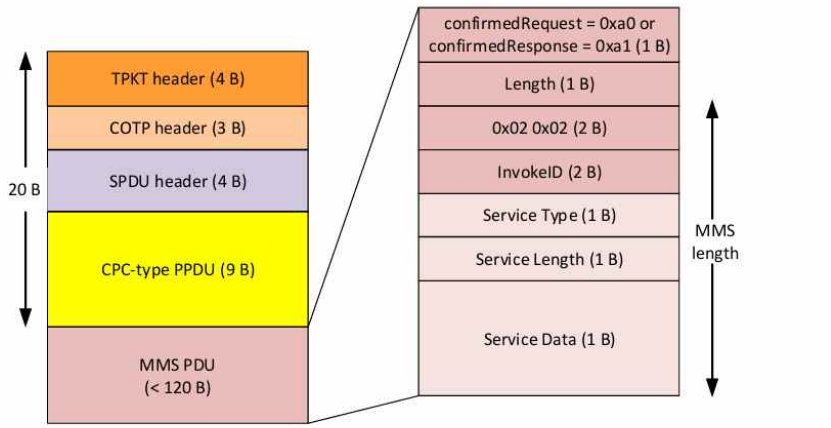


Figure 30: Format of MMS Confirmed-Request/Response packet with MMS PDU length less than 120 B

다음 예제들은 확인 서비스(Confirmed Services)에 의해 교환되는 MMS PDU를 보여준다. 예제 1과 2는 getNameList 서비스를 보여주며, 예제 3과 4는 read 서비스를 보여준다.

예제 1: a0 0f 02 02 06 28 a1 09 a0 03 80 01 09 a1 02 80 00

- 0xa0은 15바이트(0x0f) 페이로드를 가진 MMS 확인 요청(confirmedRequest) PDU를 나타낸다.
- InvokeID 매개변수는 유형 0x02 (INTEGER)와 길이 2바이트로 시작한다. 값은 1576이다.
- 0xa1은 서비스의 유형을 참조한다(부록 J 참조). 태그 1은 getNameList 서비스를 나타낸다.
  - GetNameListRequest는 objectClass, objectScope, ContinueAfter 지시자의 시퀀스(sequence)이다. 서비스 데이터의 길이는 9바이트이다.
  - 0xa0은 길이가 3인 구성형 데이터 타입(constructive data type)인 ObjectClass (태그 0)를 참조한다. 이는 태그 0 (basicObjectClass), 길이 1, 값 9 (domain)를 가진 내장된 TLV 구조를 포함한다.
  - 0xa1은 길이가 2인 구성형 데이터 타입인 ObjectScope (태그 1)를 참조하며, 여기에는 길이 0인 값(NULL)을 가진 태그 0 (vmdSpecific)인 내장된 TLV가 있다.

예제 2: a1 27 02 02 06 28 a1 21 a0 1c 1a 0b 4b 4f 43 31 30 34 43 31 4c 44 30 1a 0d 4b 4f 43 31 30 34 43 31 53 45 53 5f 31 81 01 00

- 0xa1은 39바이트(0x27) 페이로드를 가진 MMS 확인 응답(confirmedResponse) PDU를 나타낸다.
- InvokeID는 유형 0x02와 길이 2바이트로 시작한다. 그 값은 1576 (0x0628)이다.
- 0xa1은 서비스의 유형을 참조한다(부록 J 참조). 태그 1은 getNameListResponse를 나타낸다.
  - GetNameListResponse는 listOfIdentifiers와 moreFollows 지시자의 시퀀스이다. 서비스 데이터의 길이는 33바이트이다.
  - 0xa0은 길이가 28인 구성형 데이터 타입인 listOfIdentifier (태그 0)를 참조한다. 이는 태그 0x1a (0001 1010)를 가진 내장된 TLV들의 시퀀스를 포함하는데, 이 태그는 유니버설 클래스 데이터 타입인 VisibleString (태그 26)을 참조한다. 문자열의 길이는 11바이트(0x0b)이며, 그 ASCII 값은 KOC104C1LD0이다. 시퀀스의 다음

항목 또한 길이 13바이트(0x0d)와 값 KOC104C1SES_1을 가진 visible string (0x1a)이다.

- 0x81은 길이 1바이트와 값 0 (FALSE)을 가진 moreFollows 매개변수(태그 1)를 참조한다.

예제 3: a0 30 02 02 06 2a a4 2a a1 28 a0 26 30 24 a0 22 a1 20 1a 0b 4b 4f 43 31 30 34 43 31 4c 44 30 1a 11 4c 4c 4e 30 24 42 52 24 52 65 70 43 6f 6e 46 30 31

- 0xa0은 길이 48바이트(0x30)인 확인 요청(confirmedRequest) PDU를 참조한다.
- 0x02는 값 1578 (0x06 2a)을 가진 InvokeID를 나타낸다.
- 0xa4는 읽기 서비스(read service, 태그 4)를 참조한다. 서비스 데이터의 길이는 42바이트이다. Read-Request는 시퀀스(sequence)이다.
  - 0xa1은 길이 40바이트(0x28)인 VariableAccessSpecification 유형을 의미한다. 이는 길이 38바이트(0x26)인 유형 0xa0 (listOfVariables)의 또 다른 TLV 구조를 포함한다.
  - 0x30은 길이 36바이트인 VariableSpecification의 유니버설 데이터 타입 SEQUENCE (태그 16)를 나타낸다.
  - 0xa0은 ObjectName이라는 또 다른 TLV인 choice name (태그 0)을 참조한다.
  - 0xa1은 두 항목(domainID와 itemID)의 구조체(SEQUENCE)인 도메인 특정 유형(domain-specific type)을 나타낸다.
    - 태그 0x1a를 가진 첫 번째 내장된 TLV는 길이 11바이트(0xb)와 값 KOC104C1LD0를 가진 Visible String이다.
    - 다음 내장된 TLV는 길이 17바이트(0x11)와 값 LLN0BRRepConF01을 가진 태그 0x1a (Visible String)로 시작한다.

예제 4: a1 65 02 02 06 2a a4 5f a1 5d a2 5b 8a 0c 4d 41 4d 45 5f 52 65 70 43 6f 6e 46 83 01 00 8a 1a 4b 4f 43 31 30 34 43 31 4c 44 30 2f 4c 4c 4e 30 24 53 74 61 74 4e 72 6d 6c 44 86 01 01 84 03 06 00 03 86 02 01 f4 86 01 00 84 02 02 64 86 01 00 83 01 00 83 01 00 89 08 00 00 00 00 00 00 00 00 8c 06 00 00 00 00 00 00

- 0xa1은 길이 101바이트인 확인 응답(confirmedResponse) PDU를 참조한다.
- 0x02는 값 1578 (0x06 2a)을 가진 InvokeID를 나타낸다.
- 0xa4는 읽기 서비스(read service, 태그 4)를 참조한다. ReadResponse는 variableAccessSpecification (선택 사항)과 listOfAcccessResult의 시퀀스이다.
  - 0xa1은 listOfAccessResults를 나타낸다. 이는 AccessResults의 시퀀스이다.
    - 0xa2는 성공적인 결과와 데이터 타입 구조체(태그 2)를 참조한다. 이는 길이 91바이트인 Data의 시퀀스이다.
      - 0x8a는 VISIBLE string (컨텍스트 특정 태그 10)이다. 길이는 12이며 값은 MAME_RepConF이다.
      - 0x83은 길이 1바이트와 값 0 (FALSE)을 가진 BOOLEAN이다.
      - 0x8a는 길이 26바이트와 값 KOC104C1LD0/LLN0$StatNrmlD를 가진 또 다른 VISIBLE string이다.
      - 0x86은 길이 1바이트와 값 1을 가진 정수(integer)이다.
      - 0x84는 길이 3바이트와 값 0을 가진 BIT STRING이다.
      - 0x86은 길이 2와 값 500을 가진 또 다른 정수를 시작한다.
      - 0x86은 길이 1바이트와 값 0을 가진 또 다른 정수를 시작한다.
      - 0x84는 길이 2바이트와 값 25를 가진 또 다른 BIT STRING을 시작한다.
      - 0x86은 길이 1바이트와 값 0을 가진 또 다른 정수를 시작한다.
      - 0x83은 길이 1바이트와 값 0 (FALSE)을 가진 또 다른 BOOLEAN이다.
      - 0x83은 길이 1바이트와 값 0 (FALSE)을 가진 또 다른 BOOLEAN이다.
      - 0x89는 길이 8바이트와 값 0x00 00 00 00 00 00 00 00인 OCTET STRING을 시작한다.
      - 0x8c는 길이 6바이트와 값 0x00 00 00인 TimeOfDay이다.

4.3.2.1 페이로드 크기가 120B보다 큰 MMS PDU

앞서 기술된 메시지들은 120바이트 미만의 MMS 패킷을 전송한다. 이 경우, PPDU 내의 TLV 구조는 길이가 1바이트이며(그림 26 참조), L5-L7 헤더의 총 길이는 20바이트이다. 따라서, MMS 메시지 파싱을 위한 시작 지점을 찾는 것은 어렵지 않다.

- 목적지 포트가 102 (ISO TSAP Class 0)
  - 목적지 포트만으로는 MMS 패킷임을 증명하기에 충분하지 않다. 추가적인 확인이 수행되어야 한다:
    - COTP의 TPDU 코드는 COTP Data에 대해 0xf0이어야 한다 (TCP 페이로드의 시작에서 오프셋 5바이트).
    - SPDU 유형은 L5에서 0x01 (Give Tokens)이어야 한다 (TCP 페이로드의 시작에서 오프셋 7바이트).
    - PPDU 유형은 L6에서 0x61 (CPC-type PPDU)이어야 한다 (TCP 페이로드의 시작에서 오프셋 11바이트).
    - 확인 서비스(confirmed services)에 대한 MMS 유형은 L7에서 0xa0 (Request) 또는 0xa1 (Response)이어야 한다 (TCP 페이로드의 시작에서 오프셋 20바이트).
- TCP 페이로드의 길이는 140바이트 미만이어야 한다. TCP 헤더는 내용의 길이를 포함하지 않으므로, TCP 페이로드의 시작 부분에 삽입된 2바이트 값인 TPKT 길이를 사용할 수 있다 (TCP 페이로드의 시작에서 오프셋 2바이트). 따라서, TPKT 길이는 136바이트 미만이어야 한다.
  - 사용 가능한 데이터 세트를 분석한 결과, 대다수의 MMS 확인 PDU (약 89%)가 이 범주에 속한다(4.4절 참조).
- MMS 페이로드의 길이가 120바이트 이상인 경우, PPDU 내 TLV 구조의 길이 필드는 1바이트보다 길어진다(그림 26 b 참조). 이러한 경우, TCP 페이로드 시작에서 7바이트 오프셋을 사용하여 PPDU의 시작을 쉽게 찾을 수 있다. 그 후, MMS 메시지의 시작을 찾기 위해 PPDU를 파싱해야 한다.
  - 하지만 이는 더 긴 데이터를 요청할 때만 발생한다. getVariableAccessAttributes, getNamedVariableListAttributes와 같은 요청을 참조하라.
  - 데이터 세트 분석에 따르면, MMS 확인 PDU의 5%만이 120바이트보다 큰 길이를 가진다(4.4절 참조).

다음 예제는 getVariableAccessAttributes에 대한 MMS 요청을 보여준다. 메시지는 120바이트보다 짧으므로 쉽게 파싱할 수 있다. 그러나 응답은 120바이트 이상의 데이터를 전송하므로, PPDU는 고정된 크기를 갖지 않으며 MMS 메시지의 시작을 찾기 위해 각 TLV를 파싱해야 한다.

예제 5: a0 2a 02 02 06 28 a6 24 a0 22 a1 20 1a 0b 4b 4f 43 31 30 34 43 31 4c 44 30 1a 11 4c 4c 4e 30 24 42 52 24 52 65 70 43 6f 6e 43 30 32

- 0xa0은 길이 42바이트(0x2a)인 확인 요청(confirmedRequest) PDU를 참조한다.
- 0x02는 값 1576 (0x06 26)을 가진 InvokeID를 나타낸다.
- 0xa6은 getVariableAccessAttributes 서비스(태그 6)를 참조한다. 서비스 데이터의 길이는 36바이트이다. getVariableAccessAttributes-Request는 시퀀스(sequence)이다.
  - 0xa0은 ObjectName이라는 또 다른 TLV인 choice name (태그 0)을 참조한다.
  - 0xa1은 두 항목(domainID와 itemID)의 구조체(SEQUENCE)인 도메인 특정 유형(domain-specific type)을 나타낸다.
    - 태그 0x1a를 가진 첫 번째 내장된 TLV는 길이 11바이트(0xb)와 값 KOC104C1LD0 를 가진 Visible String

이다.

- 다음 내장된 TLV는 길이 17바이트(0x11)와 값 LLN0BRRepConC02 을 가진 태그 0x1a (Visible String)로 시작한다.

다음 응답은 프레젠테이션 계층(presentation layer)부터 분석된다.

예제 6: 61 81 e2 30 81 df 02 01 03 a0 81 d9 a1 81 d6 02 02 06 28 a6 81 cf 80 01 00 a2 81 c9 a2 81 c6 a1 81 c3 30 0c 80 05 52 70 74 49 44 a1 03 8a 01 bf 30 0c 80 06 52 70 74 45 6e 61 a1 02 83 00 30 0d 80 06 …

- 0x61은 길이 226바이트(0xe2)인 CPC-type PDU를 시작한다. 바이트 0x81은 이 TLV의 길이 필드가 1바이트 길이의 긴 형식(long format)임을 나타낸다.
- 0x30은 PDV-list를 나타내며, 이는 SEQUENCE이다. 그 길이는 223바이트(0xdf)이다.
  - 0x02는 presentation-context-identifier를 나타내며 그 값은 3이다.
  - 0xa0은 페이로드 217바이트(0xd9)의 MMS 메시지를 포함하는 single-ASN1-type 필드를 시작한다. 여기서 MMS PDU 내의 TLV 구조의 길이 필드들도 긴 형식을 갖는 것을 알 수 있다.
  - 0xa1은 확인 응답(confirmed-Response) PDU를 참조한다. PDU의 길이는 214바이트(0xd6)이다.
    - 0x02는 값 1576을 가진 InvokeID 필드를 시작한다.
    - 0xa6은 getVariableAccessAttributes-Response 서비스(태그 6)를 참조한다. 내장된 구조는 SEQUENCE이다.
      - 0x80은 mmsDeletable 플래그(태그 0)를 나타내며 값은 0 (FALSE)이다.
      - 0xa2는 길이 201바이트(0xc9)인 TypeDescription (태그 2)을 참조하며, 이는 또 다른 TLV 구조를 캡슐화한다.
        - 0xa2는 구조체(Structure) 데이터 타입 (태그 2)을 참조하며, 이는 SEQUENCE이다. 0xa1은 componentName과 componentType의 목록인 components를 참조한다. 이 항목들은 새로운 이름과 ASN.1 데이터 타입으로 데이터 타입을 정의하는 데 사용된다.
          - 0x30 (0011 0000)은 유니버설 데이터 타입, 구성형 및 SEQUENCE를 나타낸다. 따라서 각 속성은 이 태그로 시작한다. 시작 태그 0x80을 가진 다음 값은 길이 5의 EXTERNAL 데이터 타입을 참조한다. 값은 RptID (새 데이터 타입)이다. 다음 항목 (componentType)은 0xa1 태그와 길이 3바이트로 시작한다. 0x8a (1000 1010)는 태그 10을 가진 기본 컨텍스트 특정 데이터 타입을 참조하며, 이는 visible-string을 의미한다. 값은 0xbf이다.
          - 다음 components 시퀀스 역시 0x30 바이트, 길이, 첫 번째 항목(componentName, 태그 0x80)으로 시작한다. componentName의 값은 RptEna이다. 두 번째 항목(componentType, 태그 0xa1)은 태그 0x83 (1000 0011)을 가진 내장된 TLV를 참조하며, 이는 길이 0인 Boolean을 나타낸다.
          - 이와 유사하게, 다른 속성 데이터 타입들을 분석할 수 있다.

120바이트보다 큰 크기의 MMS PDU는 일반적으로 L6 (PPDU)와 L7 (MMS)에서 가변적인 데이터 구조를 갖는다. 이는 TLV 구조가 더 긴 형식의 길이 필드를 요구하기 때문이다. 그러나 메시지 파싱은 페이로드 크기에 의존하지 않는 L6 및 L7 헤더의 고정 값을 활용할 수 있다. 따라서 파서는 데이터 구조를 분석할 때 이러한 값들을 마크(mark)로 사용할 수 있다(그림 31 참조).

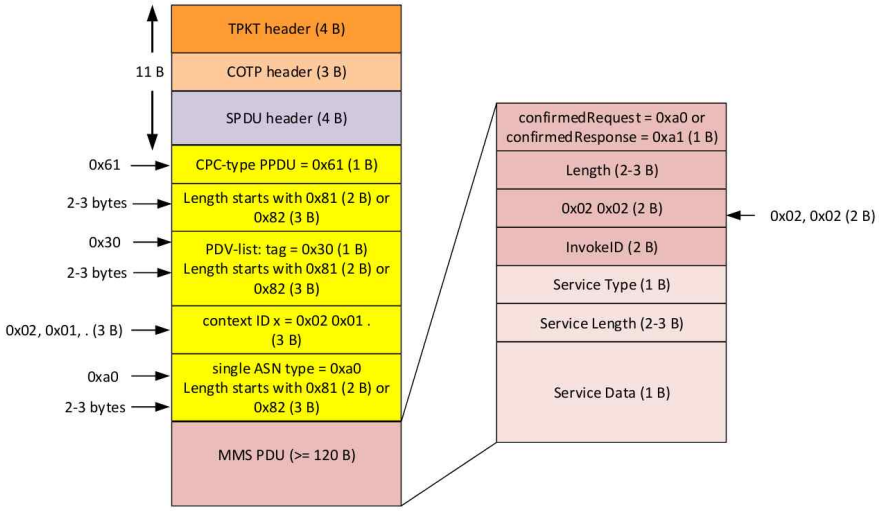


Figure 31: Analyzing MMS packet with payload greater than 120 bytes

4.3.2.3 MMS PDU 분할 및 재조립

구체적인 파싱이 필요한 MMS PDU의 마지막 그룹은 L4의 COTP 프로토콜에 의해 여러 세그먼트로 나누어진 확인 요청(confirmedRequest) 및 확인 응답(confirmedResponse) PDU이다(4.2.1.1절 참조).

분할(Segmenting)은 하나의 TSDU (전송 서비스 데이터 단위)를 하나 이상의 DT TPDU (데이터 전송 프로토콜 데이터 단위)의 순서화된 시퀀스로 매핑하는 송신 엔티티에 의해 시작된다. DT TPDU의 EOT 매개변수는 시퀀스 내에 후속 DT TPDU가 있는지 여부를 나타낸다.

재조립(Reassembling)은 수신자에 의해 수행된다. 첫 번째 세그먼트는 TPKT 및 COTP 헤더와 그 뒤를 잇는 L5, L6, L7 프로토콜을 포함한다. TPKT 길이는 이 세그먼트의 실제 크기를 나타내는 반면, L6 및 L7 길이 필드는 원래 조립된(완전한) PDU의 크기를 나타낸다. 다음 세그먼트들은 TPKT 및 COTP 헤더와 그 뒤를 잇는 해석되지 않은(uninterpreted) 데이터만을 포함한다. 이 데이터는 이전 세그먼트의 완료되지 않은 MMS PDU에 바로 이어진다. 분할 및 재조립의 예는 그림 32에 나타나 있다.

분할은 주로 MMS 응답 PDU, 즉 MMS 확인 응답(ConfirmedResponse) PDU (유형 1, 태그 0xa1) 및 비확인(Unconfirmed) PDU (유형 3, 태그 0xa3)에서 발생한다는 점을 언급해 두는 것이 좋다.

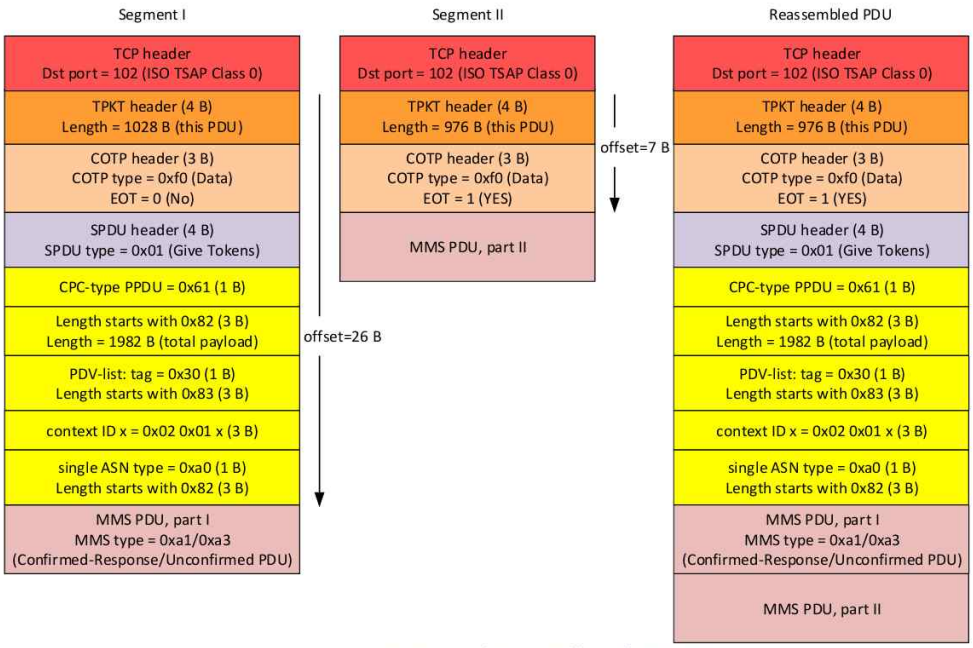


Figure 32: Segmentation and Reassembling of MMS PDUs

4.3.3 비확인 메시지

비확인 서비스(Unconfirmed services)는 서버에 의해서만 호출된다. 이 서비스 클래스는 서버가 클라이언트에게 미리 정의된 이벤트가 발생했음을 알릴 수 있게 한다. 이러한 종류의 서비스를 통해 네트워크를 통한 변수 값의 주기적 읽기와 같은 시간 소모적인 작업을 피할 수 있다. 비확인 서비스를 사용하면 폴링(polling)은 로컬에서 폴링을 수행하는 서버에 위임된다.

MMS는 세 가지 비확인 서비스를 도입한다: UnsolicitedStatus, InformationReport, EventNotification. MMS에는 이벤트 알림이 전송되는 조건을 지정하는 강력한 기능이 정의되어 있다.

비확인 메시지는 비확인 서비스를 제공한다. 비확인 MMS 서비스의 경우, 응답 PDU나 오류 PDU가 발행되지 않는다. 또한, 이 서비스는 취소할 수 없다.

비확인 PDU는 UnconfirmedService와 선택 사항(OPTIONAL)인 UnconfirmedDetail을 포함하는 시퀀스여야 한다. UnconfirmedService 유형은 서비스 유형과 해당 서비스의 인자를 식별한다. 비확인 서비스는 informationReport, unsolicitedStatus, eventNotification이다.

다음 예제는 비확인 PDU의 구조를 분석한다(그림 32 참조).

예제 1: a3 64 a0 62 a1 05 80 03 52 50 54 a0 59 8a 1a 4b 4f 43 31 30 34 43 31 4c 44 30 2f 4c 4c 4e 30 24 52 65 70 43 6f 6e 45 30 31 84 03 06 63 00 86 03 00 00 01 8c 06 02 93 c0 77 31 19 83 01 00 …

- 0xa3은 **비확인 PDU (Unconfirmed-PDU, 태그 3)**인 MMS PDU의 유형을 나타낸다(부록 J 참조). 메시지의 길이는 100바이트이다.

- 0xa0은 CHOICE인 서비스 유형을 나타낸다. 태그 0은 informationReport 서비스를 나타낸다. 이 서비스는 variableAccessSpecification과 listOfAccessResult의 SEQUENCE이다.
- 0xa1은 길이 5바이트인 variableListName (태그 1)을 나타낸다. 내장된 TLV는 vmd-specific object (태그 0)를 나타내는 태그 0x80을 갖는다. 유형은 길이 3의 UTF8String이며 값은 RPT이다.
- 0xa0은 길이 89바이트(0x59)인 listOfAccessResult (태그 0)이다. 이 목록은 AccessResults의 SEQUENCE이다. 성공할 경우 이는 Data를 포함하는 TLV들을 포함한다.
  - 첫 번째 내장된 TLV는 visible-string (태그 10)을 의미하는 0x8a (1000 1010)로 시작한다. 26바이트(0x1a) 길이의 값은 KOC104C1LD0/LLN0$RepConE01이다.
  - 태그 0x84 (비트열)를 가진 다음 TLV는 3바이트 길이와 6개의 미사용 비트를 가지며 값은 99이다.
  - 태그 0x86 (부호 없는 정수)을 가진 다음 TLV는 3바이트 길이와 값 1을 가진다.
  - 0x8c로 시작하는 TLV (태그 12는 바이너리 시간을 의미)는 6바이트 값을 가지며 2018년 5월 31일 12:00:37.495000000 UTC로 표현된다.
  - …

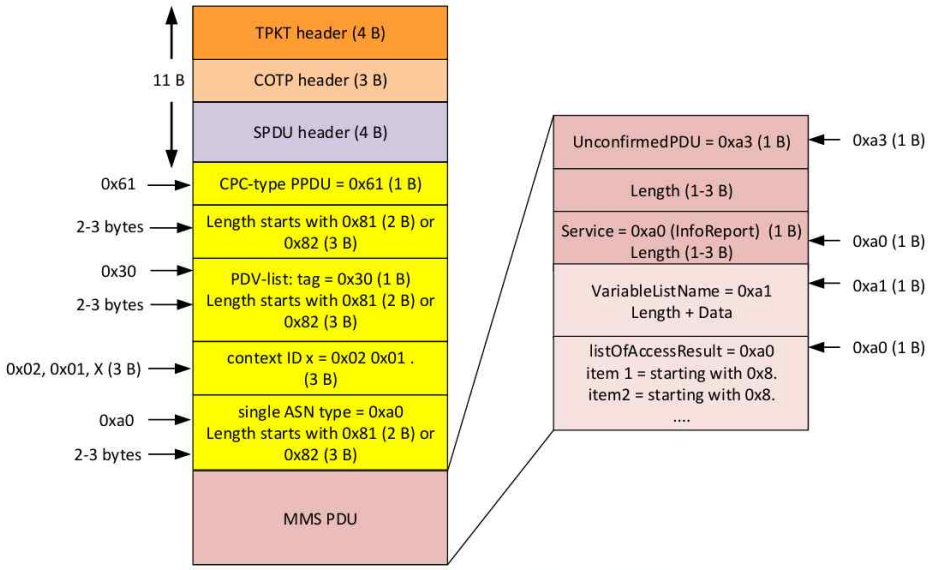


Figure 32: Encapsulation and format of the Unconfirmed PDU

확인 PDU와 마찬가지로, 비확인 PDU도 분할될 수 있다(그림 32 참조).

4.3.4 종료 메시지

MMS 종료(conclude) 메시지는 서버와의 연결을 닫는 데 사용된다. 이는 연결 설정에 사용된 리소스를 적절하게 해제하기 위해 사용된다. 만약 서버가 오류 PDU를 전송하여 종료 요청을 거부하면, 연결은 열린 상태로 유지된다. 이러한 경우, close 또는 abort 요청을 사용하여 연결을 닫을 수 있다.
MMS는 세 가지 유형의 종료 PDU를 정의한다: Conclude-Request, Conclude-Response, Conclude-Error. Conclude-Request와 Conclude-Response의 MMS 페이로드는 비어 있으며, Conclude-Error PDU는 ServiceError 구조체를 포함한다. 이 모든 메시지는 확인 및 비확인 PDU와 유사하게 Given Tokens/DT SPDU에 캡슐화된다. 페이로드가 비어 있기 때문에, 이들은 고정된 크기 형식을 갖는다.
다음 예제는 Conclude-Request PDU와 Conclude-Response PDU의 분석을 보여준다.

예제 1: 8b 00
- 식별자 0x8b (1000 1011)는 태그 번호 11을 가진 기본 형식(primitive form)의 애플리케이션 데이터 타입을 정의하며, 이는 Conclude Request PDU를 의미한다. 길이는 0이다. 즉, 페이로드가 존재하지

않는다.

예제 2: 8c 00

- 마찬가지로, 식별자 0x8c (1000 1100)는 Conclude Response PDU를 나타내며, 길이는 0이다. 즉, 데이터가 존재하지 않는다.

4.4 MMS 통신 예제

이 부분에서는 MMS 통신의 보안 모니터링에 중점을 두고 사용 가능한 MMS 데이터 세트를 분석한다. 두 개의 데이터 세트를 분석할 것이다. 하나는 그르노블 INPG의 전력 시스템 시뮬레이터에서 얻은 mms.pcapng이고, 다른 하나는 상용 파트너로부터 얻은 실제 트래픽이 담긴 mms1.pcapng이다.

4.4.1 데이터 세트 mms.pcapng

이 데이터 세트는 5분 동안 캡처된 506개의 MMS 패킷을 포함한다. 데이터 세트는 Initiate-Request - Initiate-Response PDU의 시퀀스 1개와 504개의 확인 요청(confirmed-Request) 및 확인 응답(confirmed-Response) PDU를 포함한다. PDU는 두 스테이션 간에 전송되며 TCP/IP 프로토콜로 캡슐화되고 A-Profile을 사용한다.

4.4.1.1 통신 개시

연결은 L4, L5, L6 및 L7 프로토콜을 사용하여 설정된다. L4에서는 목적지 포트 102번인 TCP가 연결을 연다. 세션 프로토콜(L5)은 호출(Calling) 및 피호출(Called) 세션 선택자를 포함한 Connect SPDU를 전송한다. 프레젠테이션 계층(L6)은 정의된 프레젠테이션 컨텍스트 목록이 포함된 CP PPDU를 전송한다. 여기에는 식별자 1을 가진 ACSE 추상 구문(OID 2.2.1.0.1)과 식별자 2를 가진 MMS 추상 구문(OID 1.0.9506.1)이 포함된다. 두 컨텍스트 모두 BER 인코딩(OID 2.1.1)을 사용하여 인코딩된다.

L7에서는 MMS initiate-Request 및 initiate-Response 메시지와 함께 AARQ와 AARE가 교환된다. 이 메시지들에서 연결 매개변수가 협상된다. 여기에는 최대 서비스 피어 수, 데이터 구조 중첩 레벨, 프로토콜 버전, 지원되는 적합성 매개변수, 지원되는 서비스 목록이 포함된다(표 11 참조). MMS initiate-Request는 initiate-Response PDU에 의해 확인된다.

4.4.1.2 데이터 세트 초기화

통신의 다음 단계는 확인 요청(confirmed-Request) 및 확인 응답(confirmed-Response) PDU 시퀀스에 의해 제공된다. 요청은 InvokeID 시퀀스 번호를 사용하여 응답과 묶인다. 이 단계는 데이터 초기화와 데이터 액세스를 포함한다.

데이터 초기화는 목적지 물리 디바이스(physical device)에서 사용 가능한 가상 제조 장치(VMD, Virtual Manufacturing Devices)를 검색한다. 각 VMD에 대해 논리 노드(logical nodes), 데이터 객체(data objects), 속성(attributes)의 목록을 getNameList 및 getNamedVariableListAttributes 서비스를 사용하여

획득한다.

- getNameList 서비스 (태그 0xa1) – 5회
  - GetNameList 서비스는 모든 MMS 객체의 이름을 반환한다. 저장된 객체의 이름을 쿼리할 때 어떤 객체 클래스(명명된 변수, 이벤트 조건 등)에서 조회할지 선택적으로 결정할 수 있다. 클라이언트는 VMD(vmd scope) 또는 논리 노드(domain specific scope)를 탐색한 다음 체계적으로 모든 객체의 이름을 쿼리할 수 있다.
  - GetNameList 요청은 각 논리 디바이스(MMS 도메인)에 대해 한 번만 수행되며, 그 결과는 이후 호출을 위해 클라이언트에 캐시된다.
  - 이러한 요청과 응답은 연결이 설정된 직후에 전송되었다.
  - 첫 번째 요청은 주어진 VMD의 모든 도메인(논리 노드)을 쿼리한다. objectClass는 유형 9 (domain)이고 objectScope 유형은 0 (vmdSpecific)이다. 이는 일종의 LN(논리 노드) 검색이다.
  - 응답으로 SIPCTRL, SIPDR, SIPMEAS, SIPPROT의 네 가지 식별자(논리 노드 또는 MMS 도메인) 목록이 반환되었다.
  - 이어지는 getNameList 요청들은 이전에 발견된 논리 노드(MMS 도메인)의 명명된 변수(named variables)를 쿼리한다. 따라서 전송된 요청은 위에서 지정된 각 MMS 도메인, 즉 SIPCTRL, SIPDR, SIPMEAS, SIPPROT에 대해 유형 2 (nameVariableList)의 objectClass를 포함한다. SIPCTRL에 대한 응답은 두 개의 식별자(VMD 이름)인 LLN0$Dataset과 LLN0$Dataset_1_1을 포함했다. SIPDR 및 SIPMEAS에 대한 응답은 어떤 데이터 객체도 포함하지 않았다. SIPPROT에 대한 응답은 식별자 LLN0$Dataset_1을 가진 하나의 VMD 객체를 포함했다.
- getNamedVariableListAttributes (태그 0xac) – 3회
  - getNameList 요청에 이어, 발견된 논리 노드 및 데이터 객체에 대해 getNameVariableListAttributes 요청이 전송된다. 즉, LN SIPCTRL 및 객체 LLN0$Dataset, LN SIPCTRL 및 객체 LLN0$Dataset_1_1, 그리고 LN SIPPROT 및 도메인 LLN0$Dataset_1에 대해 요청이 전송된다.
  - 다음 속성들이 발견되었다: SIPCTRL 및 항목 LLN0$Dataset에 대해 속성 XSWI1$STPosstVal 및 XSWI1STPos$q. 이름은 데이터 객체 참조이며(2.1.6절 참조), 이는 논리 노드 XSWI1(단락 차단 기능이 없는 스위치 [7])을 참조한다. 데이터 객체 Pos(스위치 위치)는 stVal (상태 값) 및 q (품질) 속성을 포함하는 DPC (제어 가능한 이중 포인트)라고 불리는 CDC에 속한다 [5].
  - 마찬가지로, SIPCTRL 및 객체 LLN0$Dataset_1_1에 대해 두 개의 속성 참조가 발견되었다: XCBR1STTripOpnCmd$stVal 및 XCBR1$ST$TripOpnCmd$q. 이들은 XCBR 클래스(단락 차단 기능이 있는 회로 차단기), 기능 그룹 status, 비표준 데이터 객체 TripOpnCmd 및 속성 stVal(상태) 및 q(품질)를 참조한다.
  - 마지막 속성 참조는 SIPPROT 논리 노드 및 VMD LLN0$Dataset_1에서 얻어졌다: ID_PTOC1STStr$general 및 ID_PTOC1$STStrq. 이들은 데이터 객체 Str (시작)과 속성 general 및 quality를 참조한다. 객체 Str은 ACD (방향성 보호 활성화 정보)라고 불리는 CDC에 속한다.

4.4.1.3 데이터 액세스

데이터 초기화에 이어, 발견된 논리 노드들에 대해 업데이트를 쿼리한다. LLN0DCNamPlt$configRev라는 특수 속성이 정기적으로 확인된다. 이 속성은 주어진 VMD를 관리하는 논리 노드 LLN0에 속한다. NamePlt (논리 디바이스의 명판)는 속성 configRev를 포함하는 클래스 LPL (논리 노드 명판)을 참조한다. 이 속성은 논리 디바이스 인스턴스의 구성을 고유하게 식별한다. 속성의 값은 클라이언트에 의한 데이터 해석에 영향을 미칠 수 있는 논리 디바이스 데이터 모델의 의미적 변경이 있을 때 적어

도 변경되어야 한다 [5].

관찰된 통신은 다음과 같은 읽기 요청(Read-Requests) 및 읽기 응답(Read-Responses)만을 포함한다.

- read (태그 0xa4) - 244회
  - 모든 경우에 read 서비스는 객체 LLN0DCNamPlt$configRev를 요청했다.
  - 클라이언트는 네 개의 다른 MMS 도메인(논리 노드), 즉 SIPCTRL, SIPDR, SIPMEAS, SIPPROT에 대해 이 객체를 요청한다.
  - 요청은 주어진 도메인에 대해 5초마다 반복된다.
  - 응답은 동일한 정수 값(예: 636228633875233607)을 가진 하나의 항목(visible string)을 포함한다. 이는 모니터링 동안 구성이 변경되지 않음을 의미한다.

통신 예는 그림 33에 나타나 있다.

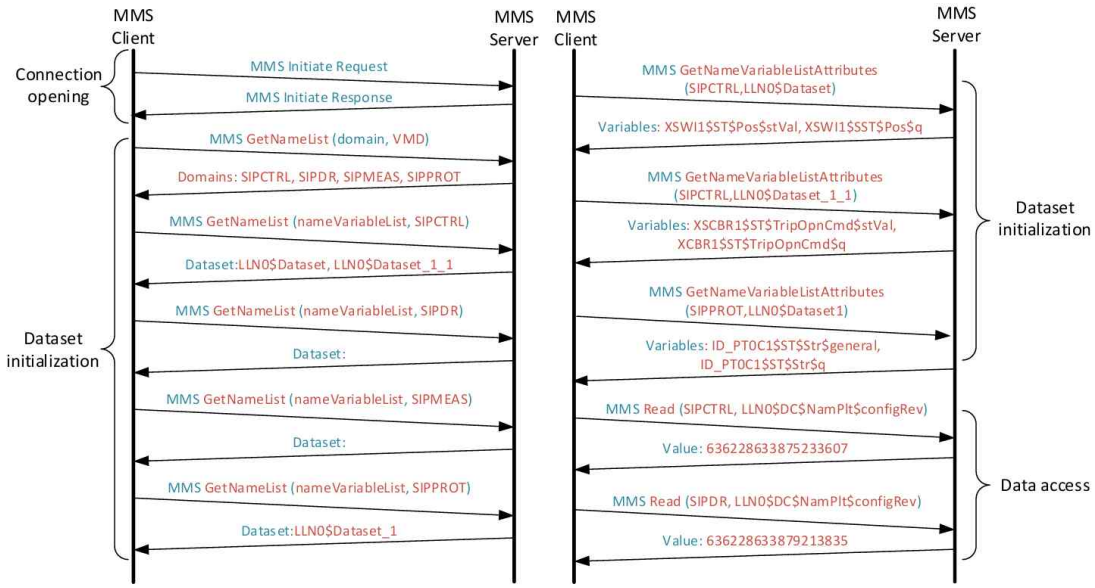


Figure 33: Example of MMS communication

4.4.2 데이터 세트 mms1.pcapng

이 데이터 세트는 프로젝트의 상용 파트너로부터 획득하였다. 이는 5분 동안 전송된 15,840개의 패킷(3.1 MB)을 포함한다. 프로토콜의 분포는 표 16에 나타나 있다.

Layer 4		Layer 5		Layer 6	
Connect Request (0x0e)	50	connect (13)	50	CP-PDU (0x31)	50
Connect Confirm (0x0d)	50	accept (14)	50	CPA-PDU (0x31)	50
Data (0x0f), EOT = TRUE	15164	finish (9)	4	CPC-type (0x61)	15064
Data (0x0f), EOT = FALSE	265	disconnect (10)	4		
		give tokens/data (1,1)	15056		
total	15529	total	15164	total	15164

Layer 7		MMS Request services	
confirmed-Requests (0xa0)	7134	status (0xa0)	46
confirmed-Response (0xa1)	7131	getNameList (0xa1)	34
unconfirmed PDU (0xa3)	778	identify (0xa2)	13
initRequest (0xa8)	50	read (0xa4)	339
initResponse (0xa9)	50	write (0xa5)	601
concludeRequest (0xab)	4	getVariableAccessAttributes (0xa6)	5840
concludeResponse (0xac)	4	getNamedVariableListAttributes (0xac)	261
total	15151	total	7134

Table 16: Number of packets and their type in dataset mms1.pcapng

모든 OSI 계층에 대한 요청(COTP Connect Request/Confirm, OSI L5 connect/accept, OSI L6 CP/CPA, L7 initRequest/initResponse)을 포함하여 연결을 개설하는 50개의 패킷이 있었음을 알 수 있다. EOT=FALSE인 L4 PDU는 분할된 PDU를 의미한다. 모든 L6 패킷이 MMS를 전송한 것은 아니다. Wireshark에 의해 제대로 분석되지 않은 MMS PDU가 5개 있었으며, MMS 내용을 캡슐화하지 않는 ACSE Release-Request 및 Response가 8개 있었다.

MMS 패킷의 대다수는 확인 PDU(요청 및 응답)이다. 비확인 PDU는 전체 MMS 패킷의 3.3%를 차지하며 다양한 속성 값을 포함하는 정보 보고서(Information Report)만 전송한다. 종료(Conclude) PDU는 페이로드가 없다(4.3.4절 참조).

4.4.2.1 MMS 요청 및 응답

데이터 세트에는 확인 PDU(Confirmed PDUs) 내에 다음과 같은 MMS 서비스가 존재한다:

- Status Requests (상태 요청, 0xa0)
  - VMD의 논리적 및 물리적 상태를 요청한다.
  - 다음과 같은 값들만이 식별되었다: vmdLogicalStatus = 0 (state-changes-allowed, 상태 변경 허용), vmdPhysical = 0 (operational, 운전 중).
- GetNameList (이름 목록 가져오기, 0xa1)
  - 물리 디바이스의 논리 노드 이름을 요청한다. 예: LN 이름 KOC104C1LD0 및 KOC104C1SES_1.
  - 모든 요청은 objectClass=9 (domain, 도메인) 및 objectScope=0 (VMD specific, VMD 특정)을 포함한다. 이는 mms.pcapng 데이터 세트와는 다르다. 그곳에서는 GetNameList가 두 가지 형태(objectClass=9 (도메인) 및 objectScope=0 (VMD 특정)인 경우와, objectClass=2 (namedVariableList)이며 LN 이름에 해당하는 도메인을 사용하는 경우)로 사용되었다.
- Identify (식별, 0xa2)
  - 요청에는 매개변수가 없다.
  - 응답은 장치의 공급업체 이름(vendorName), 모델 이름(modelName), 리비전(revision)을 반환한다.
  - 이 데이터 세트에서는 다음 값들이 발견되었다: vendor = ABB AB, model = IEC 8-1 Server, revision = V1.80.00.04p.
- Read (읽기, 0xa4)

- 주어진 변수 목록의 값을 읽는다. ReadRequest는 domainID (논리 디바이스 이름, 예: KOC101C1LD0)와 객체 참조로 식별되는 listOfVariables 구조(예: LLN0BRRepConG03$RptEna)를 포함한다.
- ReadRequest는 읽을 변수를 하나 이상 포함할 수 있다. 이 데이터 세트에서 요청들은 1개, 6개 또는 11개의 변수 검색을 포함한다.
- 요청된 각 변수에 대해, ReadResponse는 성공 비트와 변수 값을 포함한다. 예를 들어, 요청된 변수 LLN0BRRepConA03$RptEna에 대해 불리언 값 FALSE가 반환된다.

- Write (쓰기, 0xa5)
  - 이 서비스는 주어진 변수 목록의 값을 설정한다. WriteRequest는 요청된 모든 변수의 도메인 ID와 변수 이름을 포함하는 변수 목록을 전송한다. 그 뒤에는 설정할 값이 포함된 데이터 목록이 이어진다. 예: domainID=KOC104C1LD0, LLN0BRRepConF01$BufTm, value=500 (unsigned int, 부호 없는 정수).
  - WriteResponse는 쓰기 결과의 목록을 반환한다. 예: success (1).
- GetVariableAccessAttributes (변수 접근 속성 가져오기, 0xa6)
  - 요청은 주어진 도메인 및 객체(예: domainID=KOC104C1LD0, data object=SP16GGIO1STInd11)에 대한 MMS 유형 명세를 검색한다.
  - 응답은 이 객체와 관련된 유형 명세를 포함한다. 예: name=stVal, type=boolean, name=q, type=bitstring.
- GetNamedVariableListAttributes (명명된 변수 목록 속성 가져오기, 0xac)
  - 주어진 데이터 세트 내의 사용 가능한 속성(변수)을 요청한다. 요청은 도메인과 논리 노드(데이터 세트)를 지정한다. 예: KOC104C1LD0, LLN0$StatNrmlD.
  - 응답은 요청된 데이터 세트 내의 속성(변수) 목록을 반환한다. 예: SP16GGIO9STInd6, SP16GGIO9STInd7 등.
  - 이 데이터 세트에서 목록은 최대 50개의 변수를 포함한다.

4.4.2.2 대화 (Conversations)

캡처된 통신은 L2 및 L3 계층에서 20개의 엔드포인트를 포함하며, 여기에는 서비스를 요청하는 3개의 MMS 클라이언트와 17개의 MMS 서버(IED)가 포함된다. 각 클라이언트가 여러 MMS 서버에 요청하는 것을 볼 수 있다.

10.164.253.207 (MMS 클라이언트)과 10.164.253.6 (MMS 서버) 간의 대화 예시:

1. MMS initiate Request 및 MMS initiate Response를 사용하여 연결이 개설된다.
2. 데이터 세트 초기화
   i. 클라이언트에 의해 MMS identify 서비스가 요청된다.
   - 응답은 서버에 대한 설명을 포함한다: 공급업체(ABB AB), 모델명(IEC 8-1 server), 리비전(V1.80.00.04p).

   ii. MMS read 변수: domain= KOC171C1LD0, variable= LLN0DCNamPlt$configRev
   - MMS read response: 3311665

   iii. MMS getNamedVariableListAttributes, domain= KOC171C1LD0, LN= LLN0$StatIEDA
   - MMS response: attribute= LPHD1STPhyHealth

   iv. MMS getVariableAccessAttributes, domain= KOC171C1LD0, LN= LPHD1STPhyHealth
   - MMS response: name=stVal, type=integer, name=q, type=q

   v. MMS getNamedVariableAccessAttributes, domain= KOC171C1LD0, LN= LLN0$StatUrgA
   - MMS response: 48개의 변수 (domain ID + variable name), 예: SCSWI1STPos, SCSWI10STPos, SCSWI11STPos, SCSWI2STPos, ...

vi. 이전 변수들에 대한 MMS getVariableAccessAttributes, domain= KOC171C1LD0, LN= SXSWI1STPos

- MMS response: stVal (bit-string)=2, q (bit-string)=13

vii. MMS getNamedVariableListAttributes, domain= KOC171C1LD0, LN= LLN0$StatNrmlA

- MMS response: 50개의 변수 (domain ID + variable name), 예: SP16GGIO1STInd, SP16GGIO1STInd10 등.

viii. MMS getNamedVariableListAttributes, domain= KOC171C1LD0, LN= LLN0$StatNrmlB

- MMS response: 50개의 변수, 예: SP16GGIO14STInd13, SP16GGIO14STInd14 등.

ix. MMS getNamedVariableListAttributes, domain= KOC171C1LD0, LN= LLN0$StatNrmlC

- MMS response: 50개의 변수, 예: SP16GGIO3STInd3, SP16GGIO3STInd4 등.

x. MMS getVariableAccessAttributes, domain= KOC171C1LD0, LN= SP16GGIO6STInd

- MMS response: StVal (boolean), q (bit-string)

xi. ...

3. 데이터 액세스

i. MMS Read (도메인 특정): domain= KOC171C1LD0, LN= LLN0BRRepConA03$RptEna

- MMS response: Boolean (FALSE)

ii. MMS Read (도메인 특정): domain= KOC171C1LD0, 변수 LLN0BRRepConA03의 11개 속성

- MMS response: RptID=MAME_RepConA, DatSet=KOC171C1LD0/LLN0$StatIEDA, ConfRev=6, OptFlds=4

iii. MMS Write, domain= KOC171C1LD0, 변수 LLN0BRRepConA03의 6개 속성: RptID= KOC171C1LD0/LLN0BRRepConA03, IntgPd=4, TrgOps=64, OptFlds=5300, ...

- MMS response: success (6x)

iv. ...

4.5 요약

MMS 통신을 파싱할 때, 확인 및 비확인 MMS PDU에 초점을 맞출 수 있다. Initiate-Request, Initiate-Response 또는 Conclude 메시지와 같은 다른 PDU들은 통신의 시작 또는 종료 단계에서만 사용된다.

모니터링 목적으로 MMS 통신을 분석할 때 수행해야 할 몇 가지 작업이 있다:

- MMS PDU 식별
  - MMS 파싱에서 중요한 작업 중 하나는 MMS PDU를 적절히 식별하는 것이다. 포트 102 (ISO TSAP 클래스 2)가 MMS PDU임을 보장하지 않기 때문에 TCP 레벨에서는 이를 식별하는 것이 간단하지 않다. 그러나 그림 34에서 권장하는 대로 추가적인 확인을 수행할 수 있다. 그림 34의 데이터 오프셋은 MMS 페이로드가 120바이트 미만인 패킷에만 유효하다. 더 긴 패킷의 경우, PPDU 유형 및 MMS 유형은 다른 오프셋을 갖는다(4.3.2.2절 참조).

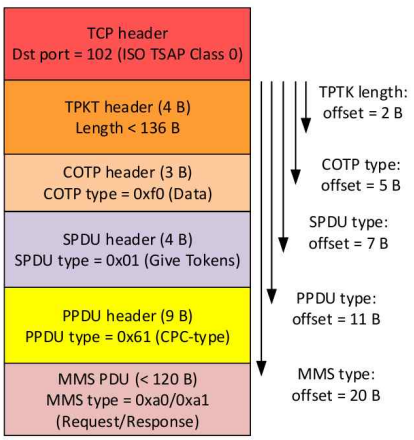


Figure 34: Identifying a MMS PDU with payload < 120 B as encapsulated via OSI protocol over TCP/IP

- MMS 디캡슐레이션 (Decapsulation)
  - 또 다른 중요한 작업은 MMS 데이터가 시작되는 위치를 찾는 것이다. TCP 페이로드가 140바이트 미만인 MMS 확인 PDU의 경우 이는 쉽다. 더 긴 메시지의 경우, PDU 형식은 TLV 구조 내 길이 필드 크기에 따라 가변적이다.
  - 큰 MMS 페이로드의 경우, 분할(segmentation)이 발생할 수 있다. 이 경우 MMS 헤더가 포함된 첫 번째 세그먼트만 분석하는 것으로 충분해 보인다. 전체 MMS 콘텐츠를 파싱하는 것이 목적이 아니라면 나머지 세그먼트는 무시할 수 있다.
- 모니터링 데이터 추출
  - 여러 MMS 데이터 세트 분석을 바탕으로, MMS 헤더에서 다음 데이터를 추출하는 것이 합리적으로 보인다:
    - MMS PDU 유형: initateRequest, initiateResponse, confirmedRequest, confirmedResponse, unconfirmedPDU, conclude 등.
    - 확인 및 비확인 서비스의 경우, 서비스 유형(예: read, getNameList, informationReport 등)을 결정하는 것이 유용할 것이다. 또한 어떤 변수가 요청되었는지 아는 것도 유용할 수 있다. 그러나 수십 개의 변수가 될 수 있다. 예를 들어, mms-kocin1.pcang 데이터 세트는 논리 디바이스 유형에 따라 1개, 6개 또는 11개의 변수에 대한 읽기 요청을 포함한다.
    - 읽기 서비스와 마찬가지로, 쓰기 요청을 탐지하고 어떤 데이터 세트나 변수가 관련되어 있는지 확인하는 것도 흥미로울 수 있다. 또한, 쓰려고 하는 요청 값도 파악하는 것이 좋을 수 있다. 그러나, 그러한 상세한 파싱은 너무 많은 실행 시간을 요구할 것이다. 게다가, 저장해야 할 모니터링 데이터가 너무 많아질 것이다.
    - 특별히 요구되지 않는 한, MMS 메시지에서 데이터 세트나 변수의 이름을 추출하는 것은 실현 가능하지 않다. 그 이유는 메시지가 getNameList, read, write 및 기타 서비스에서 수십 개의 변수를 포함할 수 있기 때문이다.
    - 보안 모니터링에 대한 또 다른 접근 방식은 이러한 작업을 제공하도록 허용된 스테이션(클라이언트)을 모니터링하고, 통신에 대한 잠재적인 공격을 탐지할 수 있도록 이러한 작업이 시간에 따라 어떻게 제공되는지에 대한 기준선(baseline)을 생성하는 것이다.

참고문헌

1. R. E. Mackiewicz, Overview of IEC 61850 and Benefits, 2006 IEEE PES Power Systems Conference and Exposition, Atlanta, GA, 2006, pp. 623-630.
2. Communication networks and systems for power utility automation – Part 7-4: Basic communication structure – Compatible logical node classes and data object classes, IEC 61850-7-4, Edition 2.0, 2010, International Electrotechnical Commission.
3. Communication networks and systems for power utility automation – Part 7-2: Basic information and communication structure – Abstract communication service interface (ACSI), IEC 61850-7-2, Edition 2.0, 2010, International Electrotechnical Commission.
4. David Hanes, et al: IoT Fundamentals. Networking Technologies, Protocols, and Use Cases for the Internet of Things, Cisco Press, 2017.
5. Communication networks and systems for power utility automation – Part 7-3: Basic communication structure – Common data classes, IEC 61850-7-3, Edition 2, 2010, International Electrotechnical Commission.
6. Communication networks and systems in substations – Specific Communication Service Mapping (SCSM) – Mappings to MMS (ISO 9506-1 and ISO 9506-2) and to ISO/IEC 8802-3, IEC 61850-8-1, Edition 2.1, 2017, International Electrotechnical Commission.
7. Olivier Dubuisson: ASN.1 Communication Between Heterogeneous Systems, Morgan Kaufmann, 2000. Available at www.oss.com/asn1/resources/books-whitepapers-pubs/dubuisson-asn1-book.PDF [Jan 2018].
8. Nikunj Patel: IEC 61850 Horizontal Goose Communication and Overview, Lambert Academic Publishing, Saarbruecken, Germany, 2011.
9. Industrial automation systems — Manufacturing Message Specification — Part 2: Protocol specification, ISO standard 9506-2:2003, 2nd Edition, 2003, International Organization for Standardization.
10. Marshall T. Rose, Dwight E. Cass: ISO Transport Service on top of the TCP. Version: 3, IETF RFC 1006, 1987.
11. ITU-T: Information technology – Open Systems Interconnection – Protocol for providing the connection-mode transport service, ITU-T X.224, 11/95.
12. ISO Transport Protocol Specification ISO DP 8073, IETF RFC 905, 1984.
13. ITU-T: Information technology – Open Systems Interconnection – Connection-oriented Session Protocol: Protocol Specification, ITU-T X.225, November, 1995.
14. ITU-T: Information technology – Open Systems Interconnection – Connection-oriented Protocol for the Association Control Service Element: Protocol Specification, ITU-T X.227, April, 1995.
15. ITU-T: Information technology – Open Systems Interconnection – Connection-oriented Presentation Protocol: Protocol Specification, ITU-T X.226, July, 1994.

부록 A: IEC 61850 논리 노드 그룹 및 클래스

표준 IEC 61850-7-4 Ed. 2, IEC 61850-7-410 및 IEC 61850-7-420은 기능에 따라 논리 노드의 공통 그룹을 정의한다. 총 159개의 서로 다른 전체 LN 클래스 목록은 [2]에서 확인할 수 있다.

코드	논리 노드(LN) 그룹	LN 클래스 수
L	시스템 LN (System LNs)	9
A	자동 제어 (Automatic Control)	5
C	제어 (Control)	6
D	분산 에너지 자원 (Decentralized Energy Resources)	
F	기능 블록 (Functional Blocks)	9
G	범용 (Generic)	4
H	수력 발전 (Hydro Power)	
I	인터페이싱 및 아카이빙 (Interfacing and Archiving)	6
K	기계 및 비전기1차 장비 (Mechanical and Nonelectric Primary Equipment)	5
M	계량 및 계측 (Metering and Measurement)	13
P	보호 기능 (Protection Functions)	30
Q	전력 품질 이벤트 (Power Quality Events)	6
R	보호 관련 기능 (Protection Related Functions)	11
S	감시 및 모니터링 (Supervising and Monitoring)	11
T	계기용 변성기 및 센서 (Instrument Transformers and Sensors)	20
W	풍력 터빈 (Wind Turbines)	
X	스위치기어 (Switchgear)	2
Y	전력 변압기 (Power Transformers)	4
Z	기타 전력 시스템 장비 (Further Power System Equipment)	18
합계		159

각 LN 그룹은 공통 LN 클래스 목록을 포함한다. 시스템 LN 그룹의 예시는 다음과 같다:

시스템 LN (L 그룹) 논리 클래스			
번호	조항(Clause)	설명	이름
1	5.3.2	물리 디바이스 정보 (Physical Device Information)	LPHD
2	5.3.3	공통 논리 이름 (Common Logical Name)	Common LN
3	5.3.4	논리 노드 제로 (Logical Node Zero)	LLN0
4	5.3.5	물리 통신 채널 감시 (Physical Communication Channel Supervision)	LCCH
5	5.3.6	GOOSE 구독 (GOOSE Subscription)	LGOS
6	5.3.7	샘플링된 값 구독 (Sampled Value Subscription)	LSVS
7	5.3.8	시간 관리 (Time Management)	LTIM
8	5.3.9	타임 마스터 감시 (Time Master Supervision)	LTMS
9	5.3.10	서비스 추적 (Service Tracking)	LTRK

부록 B: 공통 데이터 클래스 (CDC)

다음 표는 IEC 61850-7-3, Ed.2 표준에 기초한 CDC 목록을 포함한다. 이 표준은 40개의 서로 다른 데이터 클래스를 정의한다.

공통 데이터 클래스 (Common Data Classes, CDC)				
코드	**설명**		**코드**	**설명**
상태 정보용 CDC			**상태 설정용 CDC**	
SPS	단일 포인트 상태 (Single point of status)		SPG	단일 포인트 설정 (Single point setting)
DPS	이중 포인트 상태 (Double point of status)		ING	정수 상태 설정 (Integer status setting)
INS	정수 상태 (Integer status)		ENG	열거형 상태 설정 (Enumerated status setting)
ENS	열거형 상태 (Enumerated status)		ORG	객체 참조 설정 (Object reference setting)
ACT	보호 활성화 정보 (Protection activation Information)		TSG	시간 설정 그룹 (Time setting group)
ACD	방향성 보호 활성화 정보 (Directional protection activation information)		CUG	통화 설정 그룹 (Currency setting group)
SEC	보안 위반 카운팅 (Security violation counting)		VSG	가시 문자열 설정 (Visible string setting)
BCR	바이너리 카운터 판독 (Binary counter reading)			
HST	히스토그램 (Histogram)			
VSS	가시 문자열 상태 (Visible string status)			
측정 정보용 CDC			**아날로그 설정용 CDC**	
MV	측정값 (Measured value)		ASG	아날로그 설정 (Analogue setting)
CMV	복소수 측정값 (Complex measured value)		CURVE	설정 곡선 (Setting curve)
SAV	샘플링된 값 (Sampled value)		CSG	곡선 형태 설정 (Curve shape setting)
WYE	상-대지/중성점 관련 측정값 (Phase to ground/neutral related measured values)			
DEL	상-상 관련 측정값 (Phase to phase related measured values)			
SEQ	시퀀스 (Sequence)			
HMV	고조파 값 (Harmonic value)			
HWYE	WYE용 고조파 값 (Harmonic value for WYE)			
HDEL	DEL용 고조파 값 (Harmonic value for DEL)			
제어용 CDC			**기술 정보용 CDC**	
SPC	제어 가능한 단일 포인트 (Controllable single point)		DPL	디바이스 명판 (Device name plate)
DPC	제어 가능한 이중 포인트 (Controllable double point)		LPL	논리 노드 명판 (Logical node name plate)
INC	제어 가능한 정수 상태 (Controllable integer status)		CSD	곡선 형태 기술 (Curve shape description)
ENC	제어 가능한 열거형 상태 (Controllable enumerated status)			
BSC	바이너리 제어 스텝 위치 정보 (Binary controlled step position information)			
ISC	정수 제어 스텝 위치 정보 (Integer controlled step position information)			
APC	제어 가능한 아날로그 프로세스 값 (Controllable analogue process value)			
BAC	바이너리 제어 아날로그 프로세스 값 (Binary controlled analog process value)			

부록 C: 속성 유형 및 기능 제약

IEC 61850-7-3은 공통 데이터 클래스(부록 B 참조)에서 사용되는 14개의 구조화된 데이터 속성을 정의한다. 데이터 속성 유형은 12개의 서로 다른 기능 제약(FC)에 속한다.

부록 C1: 공통 데이터 속성 (CDA)

- 품질 (Quality)
- 아날로그 값 (Analogue value)
- 아날로그 값 구성 (Configuration of analog value)
- 범위 구성 (Range configuration)
- 과도 표시 포함 스텝 위치 (Step position with transient indication)
- 펄스 구성 (Pulse configuration)
- 발신자 (Originator)
- 단위 정의 (Unit definition)
- 벡터 정의 (Vector definition)
- 포인트 정의 (Point definition)
- 제어 모델 정의 (CtlModels definition)
- SBO 클래스 정의 (SboClasses definition - 동작 전 선택)
- 셀 (Cell)
- 캘린더 시간 정의 (CalendarTime definition)

부록 C2: 트리거 옵션 (TrgOp)

트리거 옵션은 데이터 속성에 대한 리포팅(reporting)이 트리거될 수 있는 조건을 지정한다.

트리거 옵션 (TrgOp)		
속성 이름	**속성 유형**	**M/O/C**
data-change	BOOLEAN	M
quality-change	BOOLEAN	M
data-update	BOOLEAN	M
integrity	BOOLEAN	M
general-interrogation	BOOLEAN	M

부록 C3: 기능 제약 (Functional Constraints, FC)

기능 제약(FC)은 속성의 특정 용도를 특징짓는 데이터 속성의 속성이다. 이는 특정 데이터 속성에 적용 가능한 서비스를 나타낸다. 애플리케이션 관점에서 데이터 속성은 그 특정 용도에 따라 분류된다. 일부 속성은 제어에 사용되고, 다른 속성은 리포팅 및 로깅, 측정, 설정 그룹, 또는 특정 데이터 속성의 기술(description)에 사용된다.

기능 제약은 IEC 61850-7-3(부록 B 참조)에 정의된 공통 데이터 클래스의 특정 데이터 속성에 적용할 수 있는 서비스를 정의한다는 의미에서 데이터 필터 역할을 한다.

FC	의미(Semantic)	설명(Description)	허용된 서비스 (Services allowed)
ST	상태 속성	데이터 속성은 상태 정보를 나타내야 한다. 초기값은 프로세스에서 가져와야 한다.	GetDataValues, GetDataDefinitions, GetDataDirectory, GetDataSetValues
MX	측정값(아날로그 값)	데이터 속성은 측정 정보를 나타내야 한다.	GetDataValues, GetDataDefinitions, GetDataDirectory, GetDataSetValues
CO	제어		
SP	설정값(Set points)	데이터 속성은 설정 파라미터 정보를 나타내야 한다.	GetDataValues, SetDataValues, GetDataDefinition, GetDataDirectory, GetDataSetValues, SetDataSetValues
SV	대체값(Substituted values)	데이터 속성은 대체(substitution)를 처리하는 데 사용되어야 한다.	GetDataValues, SetDataValues, GetDataDefinition, GetDataDirectory, GetDataSetValues, SetDataSetValues
CF	구성 (Configuration)	데이터 속성은 구성 정보를 나타내야 한다.	GetDataValues, SetDataValues, GetDataDefinition, GetDataDirectory, GetDataSetValues, SetDataSetValues
DC	기술(Description)	데이터 속성은 기술 정보를 나타내야 한다.	GetDataValues, SetDataValues, GetDataDefinition, GetDataDirectory, GetDataSetValues, SetDataSetValues
SG	설정 그룹	데이터 속성은 설정의 설정 멤버에 대한 현재 활성 값을 나타내야 한다. SETTING GROUP CONTROL 참조.	GetDataValues, GetDataDefinitions, GetDataDirectory, GetDataSetValues
SE	편집 가능한 설정 그룹	데이터 속성은 설정 그룹과 관련된 편집 서비스에 속해야 한다. SETTING GROUP CONTROL BLOCK 참조.	GetDataDefinition, GetDataDirectory, GetEditSGValues, SetEditSGValues
SR	서비스 응답	데이터 속성은 동일한 추적 객체(tracking object)를 가진 서로 다른 프로세스 객체의 데이터를 나타내야 한다. 이 속성들은...	GetDataValues, GetDataDefinitions, GetDataDirectory, GetDataSetValues
OR	동작 수신(Operate received)	데이터 속성은 동작(Operate) 실행이 차단되더라도, 동작 요청을 수신하는 데이터 객체에서의 동작 요청 결과를 나타내야 한다.	GetDataValues, GetDataDefinitions, GetDataDirectory, GetDataSetValues
BL	블로킹(Blocking)	데이터 속성은 값 업데이트를 차단하는 데 사용되어야 한다.	GetDataValues, SetDataValues, GetDataDefinition, GetDataDirectory, GetDataSetValues, SetDataSetValues
EX	확장 정의	데이터 속성은 애플리케이션 네임스페이스를 나타내야 한다. IEC 61850-7-1 참조.	GetDataValues, GetDataDefinitions, GetDataDirectory, GetDataSetValues
BR	버퍼링된 리포트		
RP	버퍼링되지 않은 리포트		
LG	로깅(Logging)		
GO	GOOSE 제어		
GS	GSSE 제어		
MS	멀티캐스트 샘플링된 값(9-2)		
US	유니캐스트 샘플링된 값(9-1)		
XX	ACSI에서 와일드 카드로 사용됨		

FC CO, SR, OR 및 BL은 IEC 61850-7-3 Edition 2에서 정의되었다. FC BR, RP, LG 등은 MMS로의 매핑을 위해 재삽입되었다.

부록 D: 속성 유형 및 기능 제약

IEC 61850-7-2, Ed. 2 (2010)는 기본 유형과 공통 ACSI 유형을 정의한다. 다음 표는 IEC 61850-8-1, Ed. 2.1 (2017)에 정의된 MMS 데이터 유형으로의 데이터 유형 매핑도 포함한다.

기본 데이터 유형 (Basic Data Types)		
이름	**설명**	**MMS 데이터 유형**
BOOLEAN	BOOLEAN	Boolean
INT8	8비트 정수	Integer
INT16	16비트 정수	Integer
INT32	32비트 정수	Integer
INT64	64비트 정수	Integer
INT8U	8비트 부호 없는 정수	Unsigned
INT16U	16비트 부호 없는 정수	Unsigned
INT24U	24비트 부호 없는 정수, 타임스탬프에 사용됨	Unsigned
INT32U	32비트 부호 없는 정수	Unsigned
FLOAT32	부동 소수점 값, IEEE 754	Floating-point
ENUMERATED	값의 정렬된 시퀀스	Integer
CODED ENUM	값의 정렬된 시퀀스	Bit-string
OCTET STRING	최대 길이가 정의되어야 함	Octet-string
VISIBLE STRING	최대 길이가 정의되어야 함	Visible-string
UNICODE STRING	최대 길이가 정의되어야 함	MMS
Currency	3문자 국제 통화 코드, ISO 4217	
공통ACSI 데이터 유형 (Common ACSI Data Types)		
이름	**데이터 유형 / 설명**	**MMS 데이터 유형**
Object Name	VISIBLE STRING 64	n/a
ObjectReference	VISIBLE STRING 129	MMS address
PHYCOMADDR	물리 통신 주소 (Physical Communication Address)	Addr-PRI-VID-APPID
ARRAY	Array 0 .. m OF p	MMS array
ServiceError	ENUMERATED	MMS messages
EntryID	OCTET STRING	MMS octet string
Packed List	유형의 시퀀스 (Sequence of types)	MMS bit-string
TimeStamp	1970-01-01 이후의 UTC 타임스탬프: sec, frac, quality	MMS string
EntryTime	1984-01-01 이후의 GMT 시간	Binary-time
TriggerConditions	BOOLEAN의 패킹된 목록 (Packed list of BOOLEANS)	bitstring
ReasonCode (ReasonForInclusion)	BOOLEAN의 패킹된 목록 (Packed list of BOOLEANS)	bitstring

부록 E: IEC 61850 객체 및 서비스의 MMS 매핑

다음 표는 IEC 61850-7-2에 정의된 ACSI 객체 및 서비스와 IEC 61850-8-1 [6]에 따른 MMS 서비스로의 매핑을 보여준다.

IEC 61850 객체	IEC 61850 서비스	MMS 객체	MMS 서비스	MMS 값
서버 (Server)	GetServerDirectory	가상 제조 장치(VMD)	FileDirectory	4
어소시에이션 (Association)	Associate		initiate	1
	Abort		abort	2
	Release		Conclude	3
논리 디바이스 (Logical Device)	GetLogicalDeviceDirectory	도메인(Domain)	GetNameList	5
논리 노드 (Logical Node)	GetLogicalNodeDirectory	이름 지정 변수 (Named Variable)	GetNameList	55
	GetAllDataValues		Read	6
데이터 (Data)	GetDataValues	이름 지정 변수 (Named Variable)	Read	7
	SetDataValues		Write	8
	GetDataDirectory		GetVariableAccessAttributes	9
	GetDataDefinition		GetVariableAccessAttributes	10
데이터 세트 (Data Set)	GetDataSetValues	이름 지정 변수 목록 (Named Variable List)	Read	11
	SetDataSetValues		Write	12
	CreateDataSet		DefineNamedVariableList	13
	DeleteDataSet		DeleteNamedVariableList	14
	GetDataSetDirectory		GetNameVariableListAttributes	15
설정 그룹 제어 블록 (Setting-Group-Control-Block)	SelectActiveSG	이름 지정 변수 (Named Variable)	Read	16
	SelectEditSG		Read	17
	SetEditSGValue		Write	18
	ConfirmEditSGValues			19
	GetEditSGValue		Read	20
	GetSGCBValues		Read	21
리포트 제어 블록 (Report-Control-Block)	Report	이름 지정 변수 (Named Variable)	InformationReport	22
	GetBRCBValues		Read	23
	SetBRCBValues		Write	24
	GetURCBValues		Read	25
	SetURCBValues		Write	26
로그 (Log)		저널(Journal)		
로그 제어 블록 (Log-Control-Block)	GetLCBValues	이름 지정 변수 (Named Variable)	Read	27
	SetLCBValues		Write	28
	QueryLogByTime		ReadJournal	29
	QueryLogAfter		ReadJournal	30
	GetLogStatusValues		GetJournalStatus	31
GOOSE 제어 블록 (GOOSE-Control-Block)	GetGoCBValues	이름 지정 변수 (Named Variable)	Read	33
	SetGoCBValues		Write	34
	SendGOOSEMessage		Write	32
	GetGoReference			35
	GetGOOSEElementNumber			36
GSSE 제어 블록 (GSSE-Control-Block)	GetGsCBValue	이름 지정 변수 (Named Variable)		
	SetGsCBValue			
제어 (Control)	Select	이름 지정 변수 (Named Variable)	Read	43
	SelectWithValue		Write	44
	Cancel		Write	45
	Operate		Write	46
	CommandTermination		InformationReport	47
	TimeActivatedOperate		InformationReport	48
파일 (Files)	GetFile	파일(Files)	FileOpen/FileRead/FileClose	49
	SetFile		ObtainFile	50
	DeleteFile		FileDelete	51
	GetFileAttributeValues		FileDirectory	52

부록 F: GOOSE용 애플리케이션 프로토콜 사양

이 부분은 IEC 61850-8-1, 부속서 A [6]에 정의된 GOOSE 및 GSE 메시지의 ASN.1 정의를 기술한다.

```
IEC 61850-8-1 Specific Protocol ::= CHOICE {
        mngtPdu     [APPLICATION 0]     IMPLICIT    MngtPdu,
        goosePdu    [APPLICATION 1]     IMPLICIT    IECGoosePdu,
        ...
}

MngtPdu ::= SEQUENCE {
        StateID     [0]         IMPLICIT INTEGER,
        Security    [3]         ANY OPTIONAL, -- 향후 정의를 위해 예약됨
        CHOICE      {           requests    [1]         IMPLICIT    MngtRequests,
                                responses   [2]         IMPLICIT    MngtResponses
                    }
}

MngtRequests ::= CHOICE {
        getGoReference                  [1]         IMPLICIT    GetReferenceRequestPdu,
        getGOOSEElementNumber           [2]         IMPLICIT    GetElementRequestPdu,
        getGsReference                  [3]         IMPLICIT    GetReferenceRequestPdu,
        getGSSEDataOffset               [4]         IMPLICIT    GetElementRequestPdu,
        getMsvReference                 [5]         IMPLICIT    GetReferenceRequestPdu,
        getMSVElementNumber             [6]         IMPLICIT    GetElementRequestPdu,
        getUsvReference                 [7]         IMPLICIT    GetReferenceRequestPdu,
        getUSVElementNumber             [8]         IMPLICIT    GetElementRequestPdu,
        ...
}

MngtResponses ::= CHOICE {
        gseMngtNotSupported             [0]         IMPLICIT NULL, #  이 개정판에서 사용 중단됨
        getGoReference                  [1]         IMPLICIT    MngtResponsePdu,
        getGOOSEElementNumber           [2]         IMPLICIT    MngtResponsePdu,
        getGsReference                  [3]         IMPLICIT    MngtResponsePdu,
        getGSSEDataOffset               [4]         IMPLICIT    MngtResponsePdu,
        getMsvReference                 [5]         IMPLICIT    MngtResponsePdu,
        getMSVElementNumber             [6]         IMPLICIT    MngtResponsePdu,
        getUsvReference                 [7]         IMPLICIT    MngtResponsePdu,
        getUSVElementNumber             [8]         IMPLICIT    MngtResponsePdu,
        ...
}

GetReferenceRequestPdu ::= SEQUENCE {
        ident       [0]         IMPLICIT    VISIBLE-STRING,
                                                -- 크기는 최대 129 옥텟을 지원해야 함
        offset      [1]         IMPLICIT    SEQUENCE OF INTEGER,
        ...
}

GetElementRequestPdu ::= SEQUENCE {
        ident       [0]         IMPLICIT    VISIBLE-STRING,
                                                -- 크기는 최대 129 옥텟을 지원해야 함
        references  [1]         IMPLICIT    SEQUENCE OF VISIBLE-STRING,
        ...
}
```

```
MngtResponsePdu ::= SEQUENCE {
        ident                   [0]     IMPLICIT    VISIBLE-STRING, -- 요청의 값을 그대로 반환함
        confRev                 [1]     IMPLICIT    INTEGER OPTIONAL,
        CHOICE {
                responsePositive        [2]         IMPLICIT SEQUENCE {
                                        datSet      [0]         IMPLICIT VISIBLE-STRING OPTIONAL,
                                        result      [1]         IMPLICIT SEQUENCE OF RequestResults
                },
                responseNegative        [3]         IMPLICIT GlbErrors
        },
        ...
}

RequestResults::= CHOICE {
        offset                  [0]     IMPLICIT INTEGER,
        reference               [1]     IMPLICIT VISIBLE-STRING,
                                                -- 이전에는 IA5STRING,
        error                   [2]     IMPLICIT ErrorReason
}

GlbErrors ::= INTEGER {
        other                           (0),
        unknownControlBlock             (1),
        responseTooLarge                (2),
        controlBlockConfigurationError  (3),
        ...
}

ErrorReason ::= INTEGER {
        other                           (0),
        notFound                        (1),
        ...
}

IECGoosePdu ::= SEQUENCE {
        gocbRef                 [0]     IMPLICIT    VISIBLE-STRING,
        timeAllowedtoLive       [1]     IMPLICIT    INTEGER,
        datSet                  [2]     IMPLICIT    VISIBLE-STRING,
        goID                    [3]     IMPLICIT    VISIBLE-STRING OPTIONAL,
        t                       [4]     IMPLICIT    UtcTime,
        stNum                   [5]     IMPLICIT    INTEGER,
        sqNum                   [6]     IMPLICIT    INTEGER,
        simulation              [7]     IMPLICIT    BOOLEAN DEFAULT FALSE,
        confRev                 [8]     IMPLICIT    INTEGER,
        ndsCom                  [9]     IMPLICIT    BOOLEAN DEFAULT FALSE,
        numDatSetEntries        [10]    IMPLICIT    INTEGER,
        allData                 [11]    IMPLICIT    SEQUENCE OF Data,
}
```

UtcTime ::= OCTET STRING(8)은 1900년 1월 1일 GMT 자정 이후의 경과 시간을 초 단위로 나타낸다. CCIR 권고 460-4 (1986) 참조.

부록 G: ASN.1 및 BER 인코딩

추상 구문 표기법 1(ASN.1, ITU-T X.680에 정의됨)은 다음 범주의 데이터 유형을 지정한다 [7 참조]:

- 기본 데이터 유형 (유니버설 클래스 00)
  - BOOLEAN - 유니버설 클래스 태그 1
  - INTEGER - 유니버설 클래스 태그 2
  - BIT STRING - 유니버설 클래스 태그 3
  - OCTET STRING - 유니버설 클래스 태그 4
  - NULL - 유니버설 클래스 태그 5
  - OBJECT IDENTIFIER - 유니버설 클래스 태그 6
  - ObjectDescriptor - 유니버설 클래스 태그 7
  - EXTERNAL - 유니버설 클래스 태그 8
  - REAL - 유니버설 클래스 태그 9
  - ENUMERATED - 유니버설 클래스 태그 10
  - UTF8String - 유니버설 클래스 태그 12
  - NumericString - 유니버설 클래스 태그 18
  - PrintableString - 유니버설 클래스 태그 19
  - IA5String - 유니버설 클래스 태그
  - UTCTime - 유니버설 클래스 태그 23
  - GeneralizedTime - 유니버설 클래스 태그 24
  - GraphicString - 유니버설 클래스 태그 25
  - VisibleString - 유니버설 클래스 태그 26
  - GeneralString - 유니버설 클래스 태그 27
  - UniversalString - 유니버설 클래스 태그 28
  - CHARACTER STRING - 유니버설 태그 29
- 애플리케이션 전역 데이터 유형 (클래스 01)
  - 표준화되지 않았으나 각 애플리케이션에 의해 정의됨. GOOSE의 경우 부록 F 참조.
- 생성자 데이터 유형 (Constructor data types)
  - SEQUENCE, SEQUENCE OF - 유니버설 클래스 태그 16
  - SET, SET OF - 유니버설 클래스 태그 17
  - CHOICE
  - SELECTION
  - ANY

기본 인코딩 규칙(BER, 표준 ITU-T X.690)은 애플리케이션 간에 전송되는 ASN.1 데이터 구조의 전송 구문을 정의한다. BER은 ASN.1 데이터 값을 옥텟 문자열로 인코딩하는 방법을 기술한다. 이는 데이터 유형의 식별자, 값의 길이, 그리고 값 자체를 포함하는 삼중구조 TLV(유형-길이-값)로 ASN.1 값을 인코딩한다(다음 그림 참조).

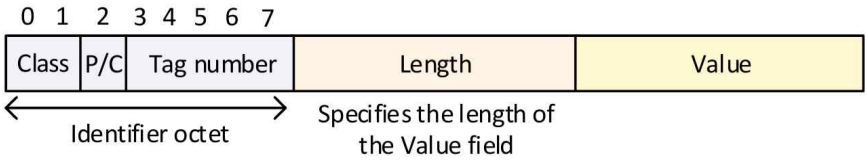
- 유형 (Type) (또는 식별자, Identifier)은 ASN.1 유형을 나타내는 1바이트 값이다 (아래 참조).
- 길이 (Length)는 실제 값 표현의 길이를 나타낸다.

- 값 (Value)은 옥텟 문자열로서 ASN.1 유형의 값을 나타낸다. 구성형 유형(constructed types)의 경우, 값은 내장된 TLV 삼중구조가 될 수 있다.

예시 1: 시퀀스 41 02 3F 22 (hex)

- 유형 41 = 0100 0001 (2진수)은 클래스 01 (애플리케이션), 기본 유형 (0)을 나타내며, 애플리케이션 태그는 1 (00001)이다(아래 참조). SNMP에서 애플리케이션 태그 1은 Counter32 데이터 유형이다.
- 02는 데이터의 길이를 나타낸다(예: 2바이트).
- 3F 22는 Counter 32 유형 변수의 값이다. 이 값은 10진수로 16,162이다.

식별자는 ASN.1 데이터 유형, 유형의 클래스 및 인코딩 방식을 지정한다(다음 그림 참조).



- 처음 두 비트는 ASN.1 데이터 유형의 클래스를 결정한다:
  - 유니버설 (00) - 표준에 의해 주어지는 유니버설 데이터 유형 (기본 또는 구성형),
  - 애플리케이션 (01),
  - 컨텍스트 특정 (10),
  - 사설(Private) (11).
- 세 번째 비트는 인코딩 방식을 기술한다: 기본(0) 또는 구성(1) 형식.
  - 1로 설정되면, SEQUENCE, SET, CHOICE 등과 같은 구성 데이터 유형이 사용된다. 이는 또한 다른 TLV 삼중구조가 값으로 내장됨을 의미한다.
- 마지막 5비트는 데이터 유형을 식별한다. 이를 태그라고 부른다. 유니버설 데이터 유형 태그는 위에 나열되어 있다. 애플리케이션, 컨텍스트 특정 및 사설 태그는 애플리케이션에 의해 정의된다.

길이는 다음 세 가지 형태로 인코딩될 수 있다:

- 짧은 확정 길이 (기본 형식): MSB(최상위 비트)가 0인 경우 1바이트 값. 즉, 길이 0-127 B인 경우이다.
- 긴 확정 길이 (기본 형식): 첫 번째 바이트(선행 1 제외)는 길이 필드의 길이를 나타낸다. 즉, 길이를 인코딩하는 데 필요한 옥텟 수이다.
- 부정 길이 (구성 형식): 첫 번째 바이트 1000 0000은 이 형식을 나타내며, 다음 옥텟들은 값(value)을 나타내고, 값 인코딩 뒤에 두 개의 0 옥텟(0x 00 00)이 추가된다.

예시 2: 시퀀스 61 81 83 80 1f 53 … (hex)

유형 61 (2진수로 0110 0001)은 식별자 옥텟으로, 데이터 유형 1을 가진 구성(1) 형식의 애플리케이션 클래스(01)를 기술한다. 애플리케이션 유형 01은 goosePDU를 의미한다(부록 F 참조).

길이 81 83은 확장 길이 필드이다. 여기서 0x81 (1000 0001)은 1 옥텟을 사용하는 길이의 긴 확정 형식을 기술하며, 0x83은 길이 값, 즉 131바이트이다.

유형 80 (1000 0000)은 내장된 TLV 삼중구조를 시작한다. 이는 컨텍스트 특정 클래스(10), 기본 형식(0)이며 유형은 0으로, 이는 gocbRef이다(부록 F 참조).

길이 1f는 gocbRef 필드 내 VISIBLE STRING의 길이이다.

BIT STRING 값 인코딩

BIT STRING 값의 인코딩 형식은 기본형(primitive) 또는 구성형(constructed)일 수 있다.
기본 형식에서 문자열은 옥텟 단위로 잘리며, 끝부분에 사용되지 않고 남은 비트 수를 0에서 7 사이의 정수로 식별할 수 있도록 선행 옥텟이 추가된다. 이 옥텟이 0이면 모든 비트가 사용되었음을 의미한다.

예를 들어, BIT STRING 1011 0111 0101 1 (13비트)은 2개의 옥텟(16비트)에 정렬된다. 즉, 1011 0111 0101 1000이 되며, 따라서 비트 문자열을 옥텟에 맞추기 위해 3개의 0 비트가 추가된다. BER 인코딩은 0000 0011 1011 0111 0101 1000이 된다. 여기서 첫 번째 옥텟(이탤릭체)은 추가된 0 비트의 수(3)를 나타내며, 이어지는 두 옥텟은 비트 문자열을 나타낸다(마지막 3개의 0 비트는 제외됨). 이에 대한 자세한 내용은 [8]을 참조한다.

EXTERNAL 유형

EXTERNAL 데이터 유형은 사용자가 프레젠테이션 컨텍스트(presentation context)를 변경할 수 있게 한 최초의 유형이다. 이는 다른 추상 구문으로 정의되거나 활성 프레젠테이션 컨텍스트와 다른 전송 구문으로 인코딩된 값을 의미한다는 점에서, 현재 사양의 외부(external) 값을 모델링한다. direct-reference 구성 요소는 데이터 유형 구문을 식별한다. indirect-reference 구성 요소는 협상된 프레젠테이션 컨텍스트 중 하나를 참조하는 정수이다. data-value-descriptor는 데이터의 추상 구문을 기술하는 문자열이지만 실제로는 사용되지 않는다. 인코딩 구성 요소에 값을 내장하기 위해, 추상 구문이 ASN.1 유형이고 데이터가 활성 프레젠테이션 컨텍스트와 동일한 전송 구문으로 인코딩된 경우 single-ASN1-type 항목이 선택된다.

```
EXTERNAL ::=            [UNIVERSAL 8]           IMPLICIT SEQUENCE {
        direct-reference                        OBJECT IDENTIFIER       OPTIONAL,
        indirect-reference                      INTEGER                         OPTIONAL,
        data-value-descriptor                   ObjectDescriptor                OPTIONAL,
        encoding                CHOICE {
                single-ASN1-type                [0]             ANY,
                octet-aligned                   [1]             IMPLICIT        OCTET STRING,
                arbitrary                       [2]             IMPLICIT        BIT STRING
        }
}
```

EXTERNAL 유형은 예를 들어 OSI 스택을 사용하는 모든 애플리케이션에 의해 호출되는 ACSE(Association Control Service Element)의 PDU에서 사용된다. 부록 H를 참조하라.

부록 H: ACSE APDU

ACSE APDU의 추상 구문은 표준 X.227 []에 지정되어 있으며 ASN.1을 사용하여 표현된다. MMS는 두 가지 ACSE APDU만을 사용한다: AARQ (Association Request APDU)와 AARE (Association Response APDU)로, 이들은 각각 MMS Initiate Request와 MMS Initiate Response를 캡슐화한다.

```
AARQ-apdu ::= [APPLICATION 0] IMPLICIT SEQUENCE {     --- APPLICATION 0 = 60H = 96
        protocol-version                [0]     IMPLICIT BIT STRING { version 1(0)} DEFAULT {version 1},
        application-context-name        [1]     Application-context-name,
        called-AP-title                 [2]     AP-title OPTIONAL,
        called-AE-qualifier             [3]     AE-qualifier OPTIONAL,
        called-AP-invocation-id         [4]     AP-invocation-identifier OPTIONAL,
        called-AE-invocation-ide        [5]     AE-invocation-identifier OPTIONAL,
        calling-AP-title                [6]     AP-title OPTIONAL,
        calling-AE-quantifier           [7]     AE-qualifier OPTIONAL,
        calling-AP-invocation-id        [8]     AP-invocation-identifier OPTIONAL,
        calling-AE-invocation-id        [9]     AE-invocation-identifier OPTIONAL,
        sender-acse-requirements        [10]    IMPLICIT ACSE-requirements OPTIONAL,
        mechanism-name                  [11]    IMPLICIT Mechanism-name OPTIONAL,
        calling-authentication-value    [12]    EXPLICIT Authentication-value OPTIONAL,
        application-context-name-list   [13]    IMPLICIT Application-context-name-list OPTIONAL,
        implementation-information      [29]    IMPLICIT Implementation-data OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}

AARE-apdu ::= [APPLICATION 1] IMPLICIT SEQUENCE {     --- APPLICATION 1 = 61H = 97
        protocol-version                [0]     IMPLICIT BIT STRING {version 1(0)} DEFAULT {version 1},
        application-context-name        [1]     Application-context-name,
        result                          [2]     Association-result,
        result-source-diagnostic        [3]     Associate-source-diagnostic,
        responding-AP-title             [4]     AP-title OPTIONAL,
        responding-AE-qualifier         [5]     AE-qualifier OPTIONAL,
        responding-AP-invocation-id     [6]     AP-invocation-identifier OPTIONAL,
        responding-AE-invocation-id     [7]     AE-invocation-identifier OPTIONAL,
        responder-acse-requirements     [8]     IMPLICIT ACSE-requirement OPTIONAL,
        mechanism-name                  [9]     IMPLICIT Mechanism-name OPTIONAL,
        responding-authentication-value [10]    EXPLICIT Authentication-value OPTIONAL,
        application-context-name-list   [11]    IMPLICIT Application-context-name-list OPTIONAL,
        implementation-information      [29]    IMPLICIT Implementation-data OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}

RLRQ-apdu ::= [APPLICATION 2] IMPLICIT SEQUENCE {     --- APPLICATION 2 = 62H = 98
        reason                          [0]     IMPLICIT Release-request-reason OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}

RLRE-apdu ::= [APPLICATION 3] IMPLICIT SEQUENCE {     --- APPLICATION 3 = 63H = 99
        reason                          [0]     IMPLICIT Release=response-reason OPTIONAL,
        user-information                [30]    IMPLICIT Association-information OPTIONAL
}

ACSE-requirements    ::= BIT STRING {
        authentication                  (0),
        application-context-negotiation (1),
}
```

```
Application-context-name-list ::= SEQUENCE OF Application-context-name
Application-context-name ::= OBJECT IDENTIFIER
Implementation-data ::= GraphicString
Association-information ::= SEQUENCE OF EXTERNAL -- EXTERNAL 데이터 유형은 부록 G 참조

Mechanism-name ::= OBJECT IDENTIFIER

Association-result ::= INTEGER {
        accepted                (0),
        rejected-permanent      (1),
        rejected-transient      (2)
}

Associate-source-diagnostic ::= CHOICE {
        acse-service-user       [1]     INTEGER {
                null                                                    (0),
                no-reason-given                                         (1),
                application-context-name-not-supported                  (2),
                calling-AP-title-not-recognized                         (3),
                calling-AP-invocation-identifier-not-recognized         (4),
                calling-AE-qualifier-not-recognized                     (5),
                calling-AE-invocation-identifier-not-recognized         (6),
                called-AP-title-not-recognized                          (7),
                called-AP-invocation-identifier-not-recognized          (8),
                called-AE-qualifier-not-recognized                      (9),
                called-AE-invocation-identifier-not-recognized          (10),
                authentication-mechanism-name-not-recognized            (11),
                authentication-mechanism-name-required                  (12),
                authentication-failure                                  (13),
                authentication-required                                 (14)
        },
        acse-service-provider   [2]     INTEGER {
                null                                    (0),
                no-reason-given                         (1),
                no-common-acse-version                  (2)
        }
}
```

부록 I: 프레젠테이션 프로토콜 데이터 단위(PPDU)의 형식

표준 ISO/IEC 8823-1 또는 ITU-T X.226은 여러 유형의 프레젠테이션 프로토콜 데이터 단위(PPDU)를 정의한다. MMS는 단지 세 가지 유형, 즉 CP PPDU (Connect Presentation), CPA PPDU (Connect Presentation Accept), 그리고 사용자 데이터만을 전송하는 CPC 유형을 사용한다.

```
CP-type ::= SET {
        mode-selector                       [0]     IMPLICIT Mode-selector,
        normal-mode-parameters              [2]     IMPLICIT SEQUENCE {
                protocol-version                    [0]     IMPLICIT Protocol-version DEFAULT {version-1},
                calling-presentation-selector       [1]     IMPLICIT Calling-presentation-selector OPTIONAL,
                called-presentation-selector        [2]     IMPLICIT Called-presentation-selector OPTIONAL,
                presentation-context-definition-list [4]    IMPLICIT Presentation-context-definition-list OPTIONAL,
                default-context-name                [6]     IMPLICIT Default-context-name OPTIONAL,
                presentation-requirements           [8]     IMPLICIT Presentation-requirements OPTIONAL,
                user-session-requirements           [9]     IMPLICIT User-session-requirements OPTIONAL,
                user-data                           User-data OPTIONAL
        } OPTIONAL
}
CPA-PPDU ::= SET {
        mode-selector                               [0]     IMPLICIT Mode-selector,
        normal-mode-parameters                      [2]     IMPLICIT SEQUENCE {
                protocol-version                            [0] IMPLICIT Protocol-version DEFAULT {version 1},
                responding-presentation-selector            [3] IMPLICIT Responding-presentation-selector
                                                                OPTIONAL,
                presentation-context-definition-result-list [5] IMPLICIT
                                                            Presentation-context-definition-result-list OPTIONAL,
                presentation-requirements                   [8] IMPLICIT Presentation-requirements OPTIONAL,
                user-session-requirements                   [9] IMPLICIT User-session-requirements OPTIONAL,
                user-data                           User-data OPTIONAL
        } OPTIONAL
}
CPC-type ::= User-data
Mode-selector ::= SET {
        mode-value          [0]     IMPLICIT    INTEGER {
                            x410-1984-mode      (0),
                            normal-mode         (1)
        }
}
Protocol-version ::= BIT STRING {
        version-1           (0)
}
Calling-presentation-selector ::= Presentation-selector

Called-presentation-selector ::= Presentation-selector

Presentation-context-definition-list ::= Context-list

Context-list ::= SEQUENCE OF SEQUENCE {
        presentation-context-identifier     Presentation-context-identifier,
        abstract-syntax-name                Abstract-syntax-name,
        transfer-syntax-name-list           SEQUENCE OF Transfer-syntax-name
}
Default-context-name ::= SEQUENCE {
        abstract-syntax-name                [0]     IMPLICIT Abstract-syntax-name,
        transfer-syntax-name                [1]     IMPLICIT Transfer-syntax-name
```

```
}
Presentation-requirements ::= BIT STRING {
        context-management              (0),
        restoration                     (1)
}
User-session-requirements ::= BIT STRING {
        half-duplex                     (0),
        duplex                          (1),
        expedited-data                  (2),
        minor-synchronize               (3),
        major-synchronize               (4),
        resynchronize                   (5),
        activity-management             (6),
        negotiated-release              (7),
        capability-data                 (8),
        exceptions                      (9),
        typed-data                      (10),
        symmetric-synchronize           (11),
        data-separation                 (12)
}
User-data ::= CHOICE {
        simply-encoded-data             [APPLICATION 0]     IMPLICIT    Simply-encoded-data,
        fully-encoded-data              [APPLICATION 1]     IMPLICIT    Fully-encoded-data
}
Responding-presentation-selector ::= Presentation-selector

Presentation-context-identifier-list ::= SEQUENCE OF SEQUENCE {
        presentation-context-identifier         Presentation-context-identifier,
        transfer-syntax-name                    Transfer-syntax-name
}
Presentation-selector                   ::= OCTET STRING
Presentation-context-identifier         ::= INTEGER
Transfer-syntax-name                    ::= OBJECT IDENTIFIER
Abstract-syntax-name                    ::= OBJECT IDENTIFIER
Simply-encoded-data                     ::= OCTET STRING
Fully-encoded-data                      ::= SEQUENCE OF PDV-list
PDV-list ::= SEQUENCE {
        transfer-syntax-name            Transfer-syntax-name            OPTIONAL,
        presentation-context-identifier Presentation-context-identifier,
        presentation-data-values        CHOICE {
                        single-ASN1-type    [0]     ABSTRACT-SYNTAX.&Type (CONSTRAINED BY{
                                                    -- Type corresponding to presentation context identifier --}),
                octet-aligned               [1]     IMPLICIT OCTET STRING,
                arbitrary                   [2]     IMPLICIT BIT STRING
        }
}
```

부록 J: MMS 프로토콜 데이터 단위(PDU)의 형식

이 부분은 표준 ISO 9506-2:2003 [9]에 명시된 MMS 프로토콜을 운영하는 데 사용되는 PDU를 설명한다. MMS에는 14가지 유형의 PDU가 있다. 가장 빈번하게 사용되는 PDU는 initiate-Request, initiate-Response, confirmed-Request, confirmed-Response, conclude-Request, conclude-Response, 그리고 unconfirmed-PDU이다. MMS PDU의 형식은 ASN.1 표기법을 사용하여 기술되며, 전송을 위해 기본 인코딩 규칙(BER)을 사용하여 인코딩된다(부록 G 참조). 전자 버전은 이곳[4]에 있다.

```
MMSpdu ::= CHOICE {
        confirmed-RequestPDU        [0]     IMPLICIT Confirmed-RequestPDU,      -- 0xa0
        confirmed-ResponsePDU       [1]     IMPLICIT Confirmed-ResponsePDU,     -- 0xa1
        confirmed-ErrorPDU          [2]     IMPLICIT Confirmed-ErrorPDU,        -- 0xa2
        unconfirmed-PDU             [3]     IMPLICIT Unconfirmed-PDU,           -- 0xa3
        rejectPDU                   [4]     IMPLICIT RejectPDU,                 -- 0xa4
        cancel-RequestPDU           [5]     IMPLICIT Cancel-RequestPDU,         -- 0xa5
        cancel-ResponsePDU          [6]     IMPLICIT Cancel-ResponsePDU,        -- 0xa6
        cancel-ErrorPDU             [7]     IMPLICIT Cancel-ErrorPDU,           -- 0xa7
        initiate-RequestPDU         [8]     IMPLICIT Initiate-RequestPDU,       -- 0xa8
        initiate-ResponsePDU        [9]     IMPLICIT Initiate-ResponsePDU,      -- 0xa9
        initiate-ErrorPDU           [10]    IMPLICIT Initiate-ErrorPDU,         -- 0xaa
        conclude-RequestPDU         [11]    IMPLICIT Conclude-RequestPDU,       -- 0xab
        conclude-ResponsePDU        [12]    IMPLICIT Conclude-ResponsePDU,      -- 0xac
        conclude-ErrorPDU           [13]    IMPLICIT Conclude-ErrorPDU          -- 0xad
}

Confirmed-RequestPDU ::= SEQUENCE {
        invokeID                    Unsigned32,
        listOfModifiers             SEQUENCE OF Modifier            OPTIONAL,
        service                     ConfirmedServiceRequest,
        service-ext                 [79]    Request-Detail OPTIONAL
}

Confirmed-ResponsePDU ::= SEQUENCE {
        invokeID                    Unsigned32,
        service                     ConfirmedServiceResponse,
        service-ext                 [79]    Response-Detail OPTIONAL
}

Confirmed-ErrorPDU ::= SEQUENCE {
        invokeID                    [0]     IMPLICIT Unsigned32,
        modifierPosition            [1]     IMPLICIT Unsigned32             OPTIONAL,
        serviceError                [2]     IMPLICIT ServiceError
}

Unconfirmed-PDU ::= SEQUENCE {
        service                             UnconfirmedService,
        service-ext                 [79]    Unconfirmed-Detail      OPTIONAL
}

RejectPDU ::= SEQUENCE {
        originalInvokeID            [0]     IMPLICIT    Unsigned32 OPTIONAL,
        rejectReason                CHOICE {
                confirmed-requestPDU                [1]     IMPLICIT INTEGER {
                other                                               (0),
```

4) https://www.nettedautomation.com/standardization/ISO/TC184/SC5/WG2/mms_syntax/index.html 참고 [2018.6]

```
                    unrecognized-service            (1),
                    unrecognized-modifier           (2),
                    invalid-invokeID                (3),
                    invalid-argument                (4),
                    invalid-modifier                (5),
                    max-serv-outstanding-exceeded   (6),
                    max-recursion-exceeded          (8),
                    value-out-of-range              (9)
}
confirmed-responsePDU       [2]     IMPLICIT INTEGER {
                    other                           (0),
                    unrecognized-service            (1),
                    invalid-invokeID                (2),
                    invalid-result                  (3),
                    max-recursion-exceeded          (5),
                    value-out-of-range              (6)
}
confirmed-errorPDU          [3]     IMPLICIT INTEGER {
                    other                           (0),
                    unrecognized-service            (1),
                    invalid-invokeID                (2),
                    invalid-serviceError            (3),
                    value-out-of-range              (4)
unconfirmedPDU              [4]     IMPLICIT INTEGER {
                    other                           (0),
                    unrecognized-service            (1),
                    invalid-argument                (2),
                    max-recursion-exceeded          (3),
                    value-out-of-range              (4)
pdu-error                   [5]     IMPLICIT INTEGER {
                    unknown-pdu-type                (0),
                    invalid-pdu                     (1),
                    illegal-acse-mapping            (2)
}
cancel-requestPDU           [6]     IMPLICIT INTEGER {
                    other                           (0),
                    invalid-invokeID                (1)
}
cancel-responsePDU          [7]     IMPLICIT INTEGER {
                    other                           (0),
                    invalid-invokeID                (1)
}
cancel-errorPDU             [8]     IMPLICIT INTEGER {
                    other                           (0),
                    invalid-invokeID                (1),
                    invalid-serviceError            (2),
                    value-out-of-range              (3)
}
conclude-requestPDU         [9]     IMPLICIT INTEGER {
                    other                           (0),
                    invalid-argument                (1)
}
conclude-responsePDU        [10]    IMPLICIT INTEGER {
                    other                           (0),
                    invalid-result                  (1)
}
conclude-errorPDU           [11]    IMPLICIT INTEGER {
                    other                           (0),
                    invalid-serviceError            (1),
```

```
                                        value-out-of-range          (2)
                    }
            }
}
Cancel-RequestPDU ::= Unsigned32 – originalInvokeID

Cancel-ResponsePDU ::= Unsigned32 -- originalInvokeID

Cancel-ErrorPDU ::= SEQUENCE {
            originalInvokeID        [0]     IMPLICIT    Unsigned32,
            serviceError            [1]     IMPLICIT    ServiceError
}
Initiate-RequestPDU ::= SEQUENCE {
            localDetailCalling                          [0]     IMPLICIT    Integer32   OPTIONAL,
            proposedMaxServOutstandingCalling           [1]     IMPLICIT    Integer16,
            proposedMaxServOutstandingCalled            [2]     IMPLICIT    Integer16,
            proposedDataStructureNestingLevel           [3]     IMPLICIT    Integer8    OPTIONAL,
            initRequestDetail                           [4]     IMPLICIT    SEQUENCE {
                        proposedVersionNumber           [0]     IMPLICIT    Integer16,
                        proposedParameterCBB            [1]     IMPLICIT    ParameterSupportOptions,
                        servicesSupportedCalling        [2]     IMPLICIT    ServiceSupportOptions ,
                        additionalSupportedCalling      [3]     IMPLICIT
            AdditionalSupportOptions
                        additionalCbbSupportedCalling   [4]     IMPLICIT    AdditionalCBBOptions,
                        privilegeClassIdentityCalling   [5]     IMPLICIT    VisibleString
            }
}
Initiate-ResponsePDU ::= SEQUENCE {
            localDetailCalled                           [0]     IMPLICIT    Integer32   OPTIONAL,
            negotiatedMaxServOutstandingCalling         [1]     IMPLICIT    Integer16,
            negotiatedMaxServOutstandingCalled          [2]     IMPLICIT    Integer16,
            negotiatedDataStructureNestingLevel         [3]     IMPLICIT    Integer8    OPTIONAL,
            initResponseDetail                          [4]     IMPLICIT    SEQUENCE {
                        negotiatedVersionNumber         [0]     IMPLICIT    Integer16,
                        negotiatedParameterCBB          [1]     IMPLICIT    ParameterSupportOptions,
                        servicesSupportedCalled         [2]     IMPLICIT    ServiceSupportOptions,
                        additionalSupportedCalled       [3]     IMPLICIT
            AdditionalSupportOptions
                    additionalCbbSupportedCalled        [4]     IMPLICIT    AdditionalCBBOptions,
                    privilegeClassIdentityCalled        [5]     IMPLICIT    VisibleString
            }
}
Initiate-ErrorPDU ::= ServiceError

Conclude-RequestPDU ::= NULL

Conclude-ResponsePDU ::= NULL

Conclude-ErrorPDU ::= ServiceError

UnconfirmedService ::= CHOICE {
            informationReport           [0]     IMPLICIT    InformationReport
            unsolicitedStatus           [1]     IMPLICIT    UnsolicitedStatus
            eventNotification           [2]     IMPLICIT    EventNotification
}
InformationReport ::= SEQUENCE {
            variableAccessSpecification     VariableAccessSpecification,
            listOfAccessResult          [0]     IMPLICIT    SEQUENCE OF AccessResult
}
```

```
ConfirmedServiceRequest ::= CHOICE {
        status                          [0]     IMPLICIT    Status-Request
        getNameList                     [1]     IMPLICIT    GetNameList-Request
        identify                        [2]     IMPLICIT    Identify-Request
        rename                          [3]     IMPLICIT    Rename-Request
        read                            [4]     IMPLICIT    Read-Request
        write                           [5]     IMPLICIT    Write-Request
        getVariableAccessAttributes     [6]     GetVariableAccessAttributes-Request
        defineNamedVariable             [7]     IMPLICIT    DefineNamedVariable-Request
        defineScatteredAccess           [8]     IMPLICIT    DefineScatteredAccess-Request
        getScatteredAccessAttributes    [9]     GetScatteredAccessAttributes-Request
        deleteVariableAccess            [10]    IMPLICIT    DeleteVariableAccess-Request
        defineNamedVariableList         [11]    IMPLICIT    DefineNamedVariableList-Request
        getNamedVariableListAttributes  [12]    GetNamedVariableListAttributes-Request
        deleteNamedVariableList         [13]    IMPLICIT    DeleteNamedVariableList-Request
        defineNamedType                 [14]    IMPLICIT    DefineNamedType-Request
        getNamedTypeAttributes          [15]    GetNamedTypeAttributes-Request
        deleteNamedType                 [16]    IMPLICIT    DeleteNamedType-Request
        input                           [17]    IMPLICIT    Input-Request
        output                          [18]    IMPLICIT    Output-Request
        takeControl                     [19]    IMPLICIT    TakeControl-Request
        relinquishControl               [20]    IMPLICIT    RelinquishControl-Request
        defineSemaphore                 [21]    IMPLICIT    DefineSemaphore-Request
        deleteSemaphore                 [22]    DeleteSemaphore-Request
        reportSemaphoreStatus           [23]    ReportSemaphoreStatus-Request
        reportPoolSemaphoreStatus       [24]    IMPLICIT    ReportPoolSemaphoreStatus-Request
        reportSemaphoreEntryStatus      [25]    IMPLICIT    ReportSemaphoreEntryStatus-Request
        initiateDownloadSequence        [26]    IMPLICIT    InitiateDownloadSequence-Request,
        downloadSegment                 [27]    IMPLICIT    DownloadSegment-Request,
        terminateDownloadSequence       [28]    IMPLICIT    TerminateDownloadSequence-Request
        initiateUploadSequence          [29]    IMPLICIT    InitiateUploadSequence-Request,
        uploadSegment                   [30]    UploadSegment-Request,
        terminateUploadSequence         [31]    TerminateUploadSequence-Request
        requestDomainDownload           [32]    RequestDomainDownload-Request
        requestDomainUpload             [33]    IMPLICIT    RequestDomainUpload-Request
        loadDomainContent               [34]    IMPLICIT    LoadDomainContent-Request
        storeDomainContent              [35]    IMPLICIT    StoreDomainContent-Request
        deleteDomain                    [36]    IMPLICIT    DeleteDomain-Request
        getDomainAttributes             [37]    IMPLICIT    GetDomainAttributes-Request
        createProgramInvocation         [38]    IMPLICIT    CreateProgramInvocation-Request
        deleteProgramInvocation         [39]    IMPLICIT    DeleteProgramInvocation-Request
        start                           [40]    IMPLICIT    Start-Request
        stop                            [41]    IMPLICIT    Stop-Request
        resume                          [42]    IMPLICIT    Resume-Request
        reset                           [43]    IMPLICIT    Reset-Request
        kill                            [44]    IMPLICIT    Kill-Request
        getProgramInvocationAttributes  [45]    IMPLICIT    GetProgramInvocationAttributes-Request
        obtainFile                      [46]    IMPLICIT    ObtainFile-Request
        defineEventCondition            [47]    IMPLICIT    DefineEventCondition-Request
        deleteEventCondition            [48]    DeleteEventCondition-Request
        getEventConditionAttributes     [49]    GetEventConditionAttributes-Request
        reportEventConditionStatus      [50]    ReportEventConditionStatus-Request
        alterEventConditionMonitoring   [51]    IMPLICIT    AlterEventConditionMonitoring-Request
        triggerEvent                    [52]    IMPLICIT    TriggerEvent-Request
        defineEventAction               [53]    IMPLICIT    DefineEventAction-Request
        deleteEventAction               [54]    DeleteEventAction-Request
        getEventActionAttributes        [55]    GetEventActionAttributes-Request
        reportEventActionStatus         [56]    ReportEventActionStatus-Request
```

```
        defineEventEnrollment                [57]    IMPLICIT    DefineEventEnrollment-Request
        deleteEventEnrollment                [58]    DeleteEventEnrollment-Request
        alterEventEnrollment                 [59]    IMPLICIT    AlterEventEnrollment-Request
        reportEventEnrollmentStatus          [60]    ReportEventEnrollmentStatus-Request
        getEventEnrollmentAttributes         [61]    IMPLICIT    GetEventEnrollmentAttributes-Request
        acknowledgeEventNotification         [62]    IMPLICIT    AcknowledgeEventNotification-Request
        getAlarmSummary                      [63]    IMPLICIT    GetAlarmSummary-Request
        getAlarmEnrollmentSummary            [64]    IMPLICIT    GetAlarmEnrollmentSummary-Request
        readJournal                          [65]    IMPLICIT    ReadJournal-Request
        writeJournal                         [66]    IMPLICIT    WriteJournal-Request
        initializeJournal                    [67]    IMPLICIT    InitializeJournal-Request
        reportJournalStatus                  [68]    ReportJournalStatus-Request
        createJournal                        [69]    IMPLICIT    CreateJournal-Request
        deleteJournal                        [70]    IMPLICIT    DeleteJournal-Request
        getCapabilityList                    [71]    IMPLICIT    GetCapabilityList-Request
        fileOpen                             [72]    IMPLICIT    FileOpen-Request
        fileRead                             [73]    IMPLICIT    FileRead-Request
        fileClose                            [74]    IMPLICIT    FileClose-Request
        fileRename                           [75]    IMPLICIT    FileRename-Request
        fileDelete                           [76]    IMPLICIT    FileDelete-Request
        fileDirectory                        [77]    IMPLICIT    FileDirectory-Request
        additionalService                    [78]    AdditionalService-Request
        getDataExchangeAttributes            [80]    GetDataExchangeAttributes-Request
        exchangeData                         [81]    IMPLICIT    ExchangeData-Request
        defineAccessControlList              [82]    IMPLICIT    DefineAccessControlList-Request
        getAccessControlListAttributes       [83]    GetAccessControlListAttributes-Request
        reportAccessControlledObjects        [84]    ReportAccessControlledObjects-Request
        deleteAccessControlList              [85]    DeleteAccessControlList-Request
        changeAccessControl                  [86]    IMPLICIT    ChangeAccessControl-Request
}
ConfirmedServiceResponse ::= CHOICE {
        status                               [0]     IMPLICIT    Status-Response,
        getNameList                          [1]     IMPLICIT    GetNameList-Response,
        identify                             [2]     IMPLICIT    Identify-Response,
        rename                               [3]     IMPLICIT    Rename-Response,
        read                                 [4]     IMPLICIT    Read-Response,
        write                                [5]     IMPLICIT    Write-Response,
        getVariableAccessAttributes          [6]     IMPLICIT    GetVariableAccessAttributes-Response,
        defineNamedVariable                  [7]     IMPLICIT    DefineNamedVariable-Response
        defineScatteredAccess                [8]     IMPLICIT    DefineScatteredAccess-Response,
        getScatteredAccessAttributes         [9]     IMPLICIT    GetScatteredAccessAttributes-Response,
        deleteVariableAccess                 [10]    IMPLICIT    DeleteVariableAccess-Response,
        defineNamedVariableList              [11]    IMPLICIT    DefineNamedVariableList-Response,
        getNamedVariableListAttributes       [12]    IMPLICIT    GetNamedVariableListAttributes-Response,
        deleteNamedVariableList              [13]    IMPLICIT    DeleteNamedVariableList-Response,
        defineNamedType                      [14]    IMPLICIT    DefineNamedType-Response,
        getNamedTypeAttributes               [15]    IMPLICIT    GetNamedTypeAttributes-Response,
        deleteNamedType                      [16]    IMPLICIT    DeleteNamedType-Response,
        input                                [17]    IMPLICIT    Input-Response,
        output                               [18]    IMPLICIT    Output-Response,
        takeControl                          [19]    TakeControl-Response,
        relinquishControl                    [20]    IMPLICIT    RelinquishControl-Response,
        defineSemaphore                      [21]    IMPLICIT    DefineSemaphore-Response,
        deleteSemaphore                      [22]    IMPLICIT    DeleteSemaphore-Response,
        reportSemaphoreStatus                [23]    IMPLICIT    ReportSemaphoreStatus-Response,
        reportPoolSemaphoreStatus            [24]    IMPLICIT    ReportPoolSemaphoreStatus-Response,
        reportSemaphoreEntryStatus           [25]    IMPLICIT    ReportSemaphoreEntryStatus-Response,
        initiateDownloadSequence             [26]    IMPLICIT    InitiateDownloadSequence-Response,
        downloadSegment                      [27]    IMPLICIT    DownloadSegment-Response,
```

```
terminateDownloadSequence [28] IMPLICIT TerminateDownloadSequence-Response,
initiateUploadSequence [29] IMPLICIT InitiateUploadSequence-Response,
uploadSegment [30] IMPLICIT UploadSegment-Response,
terminateUploadSequence [31] IMPLICIT TerminateUploadSequence-Response,
requestDomainDownload [32] IMPLICIT RequestDomainDownload-Response,
requestDomainUpload [33] IMPLICIT RequestDomainUpload-Response,
loadDomainContent [34] IMPLICIT LoadDomainContent-Response,
storeDomainContent [35] IMPLICIT StoreDomainContent-Response,
deleteDomain [36] IMPLICIT DeleteDomain-Response,
getDomainAttributes [37] IMPLICIT GetDomainAttributes-Response,
createProgramInvocation [38] IMPLICIT CreateProgramInvocation-Response,
deleteProgramInvocation [39] IMPLICIT DeleteProgramInvocation-Response,
start [40] IMPLICIT Start-Response,
stop [41] IMPLICIT Stop-Response,
resume [42] IMPLICIT Resume-Response,
reset [43] IMPLICIT Reset-Response,
kill [44] IMPLICIT Kill-Response,
getProgramInvocationAttributes [45] IMPLICIT GetProgramInvocationAttributes-Response,
obtainFile [46] IMPLICIT ObtainFile-Response,
defineEventCondition [47] IMPLICIT DefineEventCondition-Response,
deleteEventCondition [48] DeleteEventCondition-Response,
getEventConditionAttributes [49] GetEventConditionAttributes-Response,
reportEventConditionStatus [50] ReportEventConditionStatus-Response,
alterEventConditionMonitoring [51] IMPLICIT AlterEventConditionMonitoring-Response,
triggerEvent [52] IMPLICIT TriggerEvent-Response,
defineEventAction [53] IMPLICIT DefineEventAction-Response,
deleteEventAction [54] IMPLICIT DeleteEventAction-Response,
getEventActionAttributes [55] IMPLICIT GetEventActionAttributes-Response,
reportEventActionStatus [56] IMPLICIT ReportEventActionStatus-Response,
defineEventEnrollment [57] IMPLICIT DefineEventEnrollment-Response,
deleteEventEnrollment [58] IMPLICIT DeleteEventEnrollment-Response,
alterEventEnrollment [59] IMPLICIT AlterEventEnrollment-Response,
reportEventEnrollmentStatus [60] IMPLICIT ReportEventEnrollmentStatus-Response,
getEventEnrollmentAttributes [61] IMPLICIT GetEventEnrollmentAttributes-Response,
acknowledgeEventNotification [62] IMPLICIT AcknowledgeEventNotification-Response,
getAlarmSummary [63] IMPLICIT GetAlarmSummary-Response,
getAlarmEnrollmentSummary [64] IMPLICIT GetAlarmEnrollmentSummary-Response,
readJournal [65] IMPLICIT ReadJournal-Response,
writeJournal [66] IMPLICIT WriteJournal-Response,
initializeJournal [67] IMPLICIT InitializeJournal-Response,
reportJournalStatus [68] IMPLICIT ReportJournalStatus-Response,
createJournal [69] IMPLICIT CreateJournal-Response,
deleteJournal [70] IMPLICIT DeleteJournal-Response,
getCapabilityList [71] IMPLICIT GetCapabilityList-Response,
fileOpen [72] IMPLICIT FileOpen-Response,
fileRead [73] IMPLICIT FileRead-Response,
fileClose [74] IMPLICIT FileClose-Response,
fileRename [75] IMPLICIT FileRename-Response,
fileDelete [76] IMPLICIT FileDelete-Response,
fileDirectory [77] IMPLICIT FileDirectory-Response,
additionalService [78] AdditionalService-Response,
getDataExchangeAttributes [80] GetDataExchangeAttributes-Response,
exchangeData [81] IMPLICIT ExchangeData-Response,
defineAccessControlList [82] IMPLICIT DefineAccessControlList-Response,
getAccessControlListAttributes [83] IMPLICIT GetAccessControlListAttributes-Response,
reportAccessControlledObjects [84] IMPLICIT ReportAccessControlledObjects-Response,
deleteAccessControlList [85] IMPLICIT DeleteAccessControlList-Response,
changeAccessControl [86] IMPLICIT ChangeAccessControl-Response,
reconfigureProgramInvocation [87] IMPLICIT ReconfigureProgramInvocation-Response
```

```
}

GetNameList-Request ::= SEQUENCE {
        objectClass                     [0]     ObjectClass,
        objectScope                     [1]     CHOICE {
                        vmdSpecific             [0]     IMPLICIT NULL,          -- whole VMD
                        domainSpecific          [1]     IMPLICIT Identifier, -- only domain (log.node)
                        aaSpecific              [2]     IMPLICIT NULL           -- application association
                        },
        continueAfter           [2]
}
ObjectClass ::= CHOICE {        IMPLICIT Identifier OPTIONAL
        basicObjectClass        [0]     IMPLICIT INTEGER {
                                        namedVariable                   (0),
                                        scatteredAccess                 (1),
                                                namedVariableList               (2),
                                                namedType                       (3),
                                                semaphore                       (4),
                                                eventCondition                  (5),
                                                eventAction                     (6),
                                                eventEnrollment                 (7),
                                                journal                         (8),
                                                domain                          (9),
                                                programInvocation               (10),
                                                operatorStation                 (11),
                                                dataExchange                    (12),
                                                accessControlList               (13),
                                        }
        csObjectClass                   [1]     IMPLICIT INTEGER {
                                                eventConditionList              (0),
                                                unitControl                     (1)
                                        }
}

GetNameList-Response ::= SEQUENCE {
        listOfIdentifier                [0]     IMPLICIT SEQUENCE OF            Identifier,
        moreFollows                     [1]     IMPLICIT BOOLEAN                DEFAULT TRUE
}

Identifier ::= UTF8String (SIZE(1..maxIdentifier))

maxIdentifier INTEGER ::= 32

Read-Request ::= SEQUENCE {
        specificationWithResult         [0]     IMPLICIT BOOLEAN DEFAULT FALSE,
        variableAccessSpecification     [1]     VariableAccessSpecification
}

Read-Response ::= SEQUENCE {
        variableAccessSpecification     [0]     VariableAccessSpecification     OPTIONAL,
        listOfAccessResult              [1]     IMPLICIT SEQUENCE OF AccessResult
}

VariableAccessSpecification ::= CHOICE {
        listOfVariable                  [0]     IMPLICIT SEQUENCE OF SEQUENCE {
                                        variableSpecification VariableSpecification,
                                        alternateAccess         [5] IMPLICIT AlternateAccess OPTIONAL
                                        }
        variableListName                [1]     ObjectName
```

```
}

VariableSpecification ::= CHOICE {
        name                            [0]     ObjectName,
        address                         [1]     Address,
        variableDescription             [2]     IMPLICIT SEQUENCE {
                                                address             Address,
                                                typeSpecification   TypeSpecification
                                        },
        scatteredAccessDescription      [3]     IMPLICIT ScatteredAccessDescription,
        invalidated                     [4]     IMPLICIT NULL
}

GetVariableAccessAttributes-Request ::= CHOICE {
        name                            [0]     ObjectName,
        address                         [1]     Address
}

ObjectName ::= CHOICE {
        vmd-specific                    [0]     IMPLICIT Identifier,
        domain-specific                 [1]     IMPLICIT SEQUENCE {
                                        domainID Identifier,
                                        itemID  Identifier
                                        },
        aa-specific                     [2]     IMPLICIT Identifier
}

GetVariableAccessAttributes-Response ::= SEQUENCE {
        mmsDeletable                    [0]     IMPLICIT BOOLEAN,
        address                         [1]     Address     OPTIONAL,
        typeDescription                 [2]     TypeDescription,
        accessControlList               [3]     CHOICE {
                                                        basic BasicIdentifier,
                                                        extended ExtendedIdentifier
                                                } OPTIONAL,
        meaning                         [4]     ObjectName OPTIONAL
}

Address ::= CHOICE {
        numericAddress                  [0]     IMPLICIT Unsigned32,
        symbolicAddress                 [1]     MMSString,
        unconstrainedAddress            [2]     IMPLICIT OCTET STRING
}

AccessResult ::= CHOICE {
        failure                         [0]     IMPLICIT DataAccessError,
        success                         Data
}

TypeDescription ::= CHOICE {
        array                           [1]     IMPLICIT    SEQUENCE {
                                        packed                  [0]     IMPLICIT BOOLEAN DEFAULT FALSE,
                                        numberOfElements        [1]     IMPLICIT Unsigned32,
                                        elementType             [2]     TypeSpecification
                                        },
        structure                       [2]     IMPLICIT    SEQUENCE {
                                        packed                  [0]     IMPLICIT BOOLEAN DEFAULT FALSE,
                                        components              [1]     IMPLICIT SEQUENCE OF SEQUENCE {
                                                componentName   [0] IMPLICIT Identifier OPTIONAL,
```

```
                                componentType       [1] TypeSpecification
                                }
                        },
        boolean         [3]     IMPLICIT NULL,      -- BOOLEAN
        bit-string      [4]     IMPLICIT Integer32, -- BIT-STRING
        integer         [5]     IMPLICIT Unsigned8, -- INTEGER
        unsigned        [6]     IMPLICIT Unsigned8, -- UNSIGNED
        floating-point  [7]     IMPLICIT SEQUENCE {
                                    format-width        Unsigned8,
                                    exponent-width      Unsigned8
                                },
        octet-string    [9]     IMPLICIT Integer32
        visible-string  [10]    IMPLICIT Integer32,
        generalized-time [11]   IMPLICIT NULL,
        binary-time     [12]    IMPLICIT BOOLEAN,
        bcd             [13]    IMPLICIT Unsigned8,
        objId           [15]    IMPLICIT NULL,
        mMSString       [16]    Integer32
}

TypeSpecification ::= CHOICE {
        typeName        [0]     ObjectName,
        typeDescription         TypeDescription
}

Data ::= CHOICE {
        array           [1]     IMPLICIT SEQUENCE OF Data,
        structure       [2]     IMPLICIT SEQUENCE OF Data,
        boolean         [3]     IMPLICIT BOOLEAN,
        bit-string      [4]     IMPLICIT BIT STRING,
        integer         [5]     IMPLICIT INTEGER,
        unsigned        [6]     IMPLICIT INTEGER, -- shall not be negative
        floating-point  [7]     IMPLICIT FloatingPoint,
        octet-string    [9]     IMPLICIT OCTET STRING,
        visible-string  [10]    IMPLICIT VisibleString,
        generalized-time [11]   IMPLICIT GeneralizedTime,
        binary-time     [12]    IMPLICIT TimeOfDay,
        bcd             [13]    IMPLICIT INTEGER, -- shall not be negative
        booleanArray    [14]    IMPLICIT BIT STRING,
        objId           [15]    IMPLICIT OBJECT IDENTIFIER,
        mMSString       [16]    IMPLICIT MMSString
}

TimeOfDay ::= OCTET STRING (SIZE(4|6))    -- a relative day since January 1, 1984.

DataAccessError ::= INTEGER {
        object-invalidated              (0),
        hardware-fault                  (1),
        temporarily-unavailable         (2),
        object-access-denied            (3),
        object-undefined                (4),
        invalid-address                 (5),
        type-unsupported                (6),
        type-inconsistent               (7),
        object-attribute-inconsistent   (8),
        object-access-unsupported       (9),
        object-non-existent             (10),
        object-value-invalid            (11)
}
```

```
ServiceSupportOptions ::= BIT STRING {
        status                          (0),
        getNameList                     (1),
        identify                        (2),
        rename                          (3),
        read                            (4),
        write                           (5),
        getVariableAccessAttributes     (6),
        defineNamedVariable             (7),
        defineScatteredAccess           (8),
        getScatteredAccessAttributes    (9),
        deleteVariableAccess            (10),
        defineNamedVariableList         (11),
        getNamedVariableListAttributes  (12),
        deleteNamedVariableList         (13),
        defineNamedType                 (14),
        getNamedTypeAttributes          (15),
        deleteNamedType                 (16),
        input                           (17),
        output                          (18),
        takeControl                     (19),
        relinquishControl               (20),
        defineSemaphore                 (21),
        deleteSemaphore                 (22),
        reportSemaphoreStatus           (23),
        reportPoolSemaphoreStatus       (24),
        reportSemaphoreEntryStatus      (25),
        initiateDownloadSequence        (26),
        downloadSegment                 (27),
        terminateDownloadSequence       (28),
        initiateUploadSequence          (29),
        uploadSegment                   (30),
        terminateUploadSequence         (31),
        requestDomainDownload           (32),
        requestDomainUpload             (33),
        loadDomainContent               (34),
        storeDomainContent              (35),
        deleteDomain                    (36),
        getDomainAttributes             (37),
        createProgramInvocation         (38),
        deleteProgramInvocation         (39),
        start                           (40),
        stop                            (41),
        resume                          (42),
        reset                           (43),
        kill                            (44),
        getProgramInvocationAttributes  (45),
        obtainFile                      (46),
        defineEventCondition            (47),
        deleteEventCondition            (48),
        getEventConditionAttributes     (49),
        reportEventConditionStatus      (50),
        alterEventConditionMonitoring   (51),
        triggerEvent                    (52),
        defineEventAction               (53),
        deleteEventAction               (54),
        getEventActionAttributes        (55),
        reportEventActionStatus         (56),
        defineEventEnrollment           (57),
```

```
        deleteEventEnrollment           (58),
        alterEventEnrollment            (59),
        reportEventEnrollmentStatus     (60),
        getEventEnrollmentAttributes    (61),
        acknowledgeEventNotification    (62),
        getAlarmSummary                 (63),
        getAlarmEnrollmentSummary       (64),
        readJournal                     (65),
        writeJournal                    (66),
        initializeJournal               (67),
        reportJournalStatus             (68),
        createJournal                   (69),
        deleteJournal                   (70),
        getCapabilityList               (71),
        fileOpen                        (72),
        fileRead                        (73),
        fileClose                       (74),
        fileRename                      (75),
        fileDelete                      (76),
        fileDirectory                   (77),
        unsolicitedStatus               (78),
        informationReport               (79),
        eventNotification               (80),
        attachToEventCondition          (81),
        attachToSemaphore               (82),
        conclude                        (83),
        cancel                          (84),
        getDataExchangeAttributes       (85),   -- shall not appear in minor version one
        exchangeData                    (86),   -- shall not appear in minor version one
        defineAccessControlList         (87),   -- shall not appear in minor version one
        getAccessControlListAttributes  (88),   -- shall not appear in minor version one
        reportAccessControlledObjects   (89),   -- shall not appear in minor version one
        deleteAccessControlList         (90),   -- shall not appear in minor version one
        alterAccessControl              (91),   -- shall not appear in minor version one
        reconfigureProgramInvocation    (92)    -- shall not appear in minor version one
}

ParameterSupportOptions ::= BIT STRING {
        str1    (0),    -- array support
        str2    (1),    -- structure support
        vnam    (2),    -- named variable support
        valt    (3),    -- alternate access support
        vadr    (4),    -- unnamed variable support
        vsca    (5),    -- scattered access support
        tpy     (6),    -- third party operations support
        vlis    (7),    -- named variable list support
        cei     (10),   -- condition event support
        aco     (11),
        sem     (12),
        csr     (13),
        csnc    (14),
        csplc   (15),
        cspi    (16)
}

AdditionalSupportOptions ::= BIT STRING {
        vMDStop     (0),
        vMDReset    (1),
        select      (2),
```

```
        alterProgramInvocationAttributes    (3),
        initiateUnitControlLoad             (4),
        unitControlLoadSegment              (5),
        unitControlUpload                   (6),
        startUnitControl                    (7),
        stopUnitControl                     (8),
        createUnitControl                   (9),
        addToUnitControl                    (10),
        removeFromUnitControl               (11),
        getUnitControlAttributes            (12),
        loadUnitControlFromFile             (13),
        storeUnitControlToFile              (14),
        deleteUnitControl                   (15),
        defineEventConditionList            (16),
        deleteEventConditionList            (17),
        addEventConditionListReference      (18),
        removeEventConditionListReference   (19),
         getEventConditionListAttributes    (20),
        reportEventConditionListStatus      (21),
        alterEventConditionListMonitoring   (22)
}
```