9 공유하기 위한 데이터 형식에 대한 매핑 I4.0-준수 정보

9.1 일반

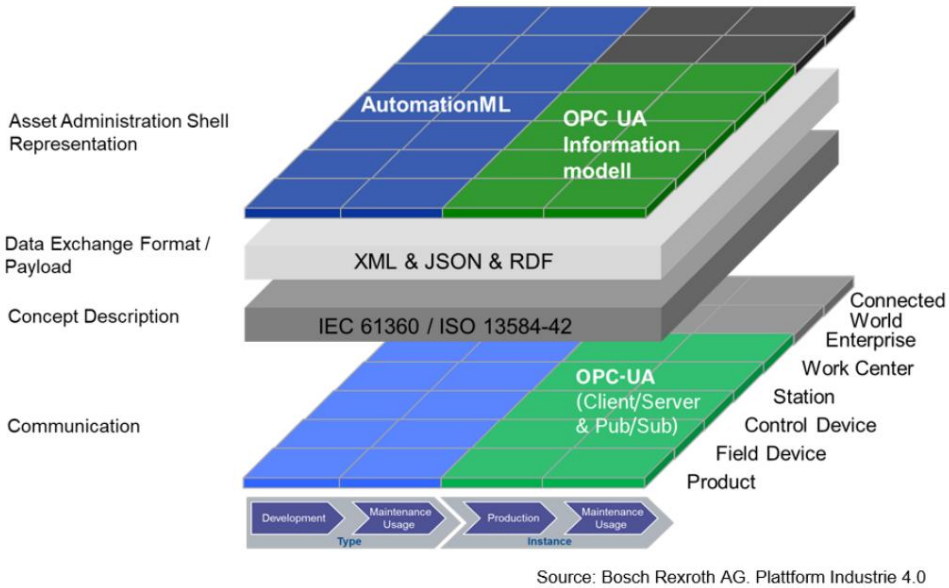
전체 RAMI4.0 모델이 적용되는 영역에 걸쳐 서로 다른 시스템 간에 정보 공유 [1]
[2] 인더스트리 4.0 애플리케이션에 매우 중요합니다. OPC UA는 생산 운영 영역에서 정보 모델의 형식으로 목표로 삼았지만 다른 영역과 이들 간의 상호 관계에 대한 다른 형식이 필요합니다.

이 문서는 자산 관리 셸을 기술 중립 형식인 UML로 지정했습니다. 제품의 다른 라이프 사이클 단계에서 사용하기 위해 다른 데이터 형식이 사용되거나 권장됩니다42. 이러한 각 형식의 직렬화 및 자산 관리 셸 매핑은 전체 수명 주기를 포괄하도록 제공됩니다. 표 12는 OPC UA 정보 모델, AutomationML, XML, JSON 및 RDF와 같은 각 형식의 주요 목적을 설명합니다. 다른 목적은 그림 77에 시각화되어 있습니다.

표 12 AAS에 대한 다양한 데이터 형식의 구별

데이터 형식	목적/동기
OPC UA 정보 모델	관리 데이터의 모든 정보에 액세스하고 프로덕션 작업 내에서 라이브 데이터를 공유합니다. 이 정보에 대한 상위 수준 공장 시스템에 대한 액세스.
자동화ML	특히 엔지니어링 중에 자산에 대한 유형 및 인스턴스 정보 공유. 이 정보를 운영 단계로 전송(참조. OPC UA 및 해당 매핑)
XML, JSON	단계 간 기술 커뮤니케이션을 목적으로 이 정보를 직렬화합니다.
RDF	시맨틱 기술의 장점을 최대한 활용할 수 있도록 이 정보를 매핑합니다.

그림 77 자산 관리 셸43의 Exchange 데이터 형식에 대한 그래픽 보기



Source: Bosch Rexroth AG. Plattform Industrie 4.0

매핑 사양 및 스키마 자체는 더 이상 사양의 일부가 아니지만 유지되는 오픈 소스입니다. 이를 통해 오픈 소스 코드 프로젝트에서 사양과 다양한 형식을 쉽게 사용할 수 있습니다.

---

42 "데이터 형식"이라는 단어의 약칭에는 정보 모델, 체계, 전송 프로토콜 등과 같은 개념적 이점의 사용이 포함됩니다.

43 지금까지 이 문서에서 고려한 데이터 형식만 그림에 언급되어 있습니다.

Asset Administration Shell 전용 프로젝트는 IDTA(Industrial Digital Twin Association)가 주도하는 Eclipse Foundation[55]의 새로운 최상위 프로젝트 "Digital Twin"에서 호스팅될 예정입니다.

9.2 일반 규칙

9.2.1 소개

모든 직렬화에 적용되거나 다른 직렬화에 사용될 수 있는 몇 가지 일반 규칙이 있습니다.

9.2.2 인코딩

Blob의 경우 base64 문자열 인코딩이 필요합니다.

9.2.3 "참조" 유형 값의 직렬화

일부 매핑 또는 직렬화에서는 "참조" 유형이 단일 문자열로 변환됩니다. 이 경우 다음 직렬화를 사용하는 것이 좋습니다.

<참조> ::= ['['<KeyType>']']<Key>{, <Key>}* <KeyType> ::=
GlobalRef | ModelRef <Key> ::= (<KeyType>)<KeyValue>
<KeyType> ::= AAS:Key/type의 값 <KeyIdType> ::= AAS:Key/.idType
의 값 <KeyValue> ::= 값 AAS:키/값

참고: IRI에는 "(", "," 및 "["와 같은 특수 기호가 포함될 수도 있습니다.
새 키 새 키 또는 값 앞에 공백이 추가됩니다.
참고: KeyType은 키 체인의 첫 번째 키에서 참조가 전역 참조인지 모델 참조인지 명확하기 때문에 선택 사항입니다. 따라서 이 문서의 예에서는 이 접두사를 사용하지 않습니다.

예:

글로벌 참조:

(글로벌참조)0173-1#02-BAA120#008

[GlobalRef](GlobalReference)0173-1#02-BAA120#008

(서브모델)http://example.com/aas/1/1/1234859590, (SubmodelElementList)문서,
(SubmodelElementCollection)0, (MultiLanguageProperty)제목

모델 참조:

(개념설명)0173-1#02-BAA120#008

[모델참조](개념설명)0173-1#02-BAA120#008

(하위 모델)http://example.com/aas/1/1/1234859590, (속성)온도

9.2.4 메타모델 및 데이터 사양에 대한 의미론적 식별자

자산 관리 셸의 메타모델에서 사용 및 정의된 개념의 고유한 식별을 가능하게 하기 위해 이러한 식별자를 생성하기 위한 규칙이 정의됩니다.

다음 문법은 유효한 식별자를 만드는 데 사용됩니다.

<네임스페이스> ::= (<AAS 네임스페이스>|<데이터 사양 네임스페이스>)

<네임스페이스 한정자> ::= <AAS 네임스페이스 한정자>| <데이터 사양 한정자>

<AAS 네임스페이스> ::= <쉘-네임스페이스>”/aas/”<버전>

<데이터 사양 네임스페이스> ::=

<Shell-Namespace>"/DataSpecifications/"<idShort of Data Specification><버전>

<쉘-네임스페이스> ::= “https://admin-shell.io/”

<버전> ::= <자리수>+”/”<자리수>+[”/”<문자>+]

<숫자> ::= 0| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<Character> ::= DIN SPEC 91406에서 허용하는 예약되지 않은 문자

? ::= 0 또는 1

+ ::= 하나 이상

지금까지 두 개의 데이터 사양 템플릿이 정의되어 있습니다. 모든 데이터 사양에 대해 사용할 데이터 사양 네임스페이스를 정의해야 합니다.

<AAS 네임스페이스 한정자> ::= “AAS:”

<데이터 사양 한정자> ::= 데이터 사양에 따라 정의

구체적인 고유 식별자는 다음과 같이 정의됩니다.

<AAS 고유 개념 식별자> ::= (<네임스페이스> | <네임스페이스 한정자>)”/”<AAS 개념 식별자>

<AAS 개념 식별자> ::= <AAS 클래스 이름>[(<AAS 속성>|<AAS 열거>)]

<AAS 속성> ::= “/”<AAS 속성 이름>[{“/”<AAS 속성 이름>}*]

<AAS 열거> ::= [{“/”<AAS 속성 이름>}*]”/”<AAS 열거 값>

유효한 고유 AAS 개념 식별자의 예:

https://admin-shell.io/aas/2/0/AssetAdministrationShell/administration/version

AAS:자산 관리 셸/관리/버전

AAS:자산 정보/자산 종류/인스턴스

패턴의 적용은 다음에서 설명합니다.

클래스의 개념 식별자는 다음 패턴을 따릅니다.

<AAS 클래스 이름>

이는 열거형을 포함한 추상 클래스 및 유형에도 적용됩니다.

예: AAS:Submodel, AAS:Qualifier, AAS:Reference, AAS:ContentType, AAS:AasSubmodelElements

클래스의 속성은 "/"로 구분됩니다. 컨텍스트에서 구체적인 참조 대상이 중요한 경우 상속된 속성도 이와 같이 참조할 수 있습니다.

기본 패턴:

<AAS 클래스 이름>”/”<AAS 속성 이름>

예 44: AAS:Referable/idShort 또는 AAS:Property/idShort 또는 AAS:Qualifier/semanticId

관련된 속성의 카디널리티가 1보다 크지 않은 경우 속성의 속성에도 적용됩니다.

<AAS 클래스 이름>"/"<AAS 속성 이름>[{"/"<AAS 속성 이름>}*]

예: AAS:식별 가능/관리/버전

이것은 열거형 값에도 적용됩니다.

<AAS 클래스 이름>[{"/"<AAS 속성 이름>}*]["/"<AAS 열거 값>]

예: AAS:Key/type/ Submodel 또는 AAS:AasSubmodelElements/Submodel

카디널리티가 1보다 큰 속성의 경우 속성이나 열거 값을 더 이상 추가할 수 없습니다.

참고: UML의 속성 이름은 카디널리티가 > 1인 경우에도 항상 단수입니다.
복수형 "s"로 표시됩니다.

예: AAS:작업/입력변수 또는 AAS:자산 관리 셸/하위 모델 또는
AAS:하위 모델/하위 모델요소

AAS:AssetAdministrationShell/submodels/administration/version 또는 AAS:Submodel/Property/idShort 는 유효한 개념 식별자가 아닙니다.

이러한 의미 체계 식별자는 자산 관리 셸의 AutomationML 매핑에서 RefSemantic 속성 값으로 사용됩니다. 이러한 식별자는 OPC UA에서 의미 체계를 설명하는 데에도 사용됩니다.

OPC UA HasDictionaryEntry 참조 유형을 통한 메타모델.

특정 직렬화 및 매핑의 경우 추가 식별자가 필요할 수 있습니다. 예를 들어 자산 관리 셸 세트 또는 사용 가능한 개념 설명 세트 등. 여기에서 AAS 메타 모델 및 사양은 권장 사항을 제공하지 않습니다.

데이터 사양 처리는 특별합니다. 데이터 사양 템플릿은 AAS의 메타모델에 속하지 않습니다. 그러나 이 사양에 지정된 대로 사전 정의된 데이터 사양 템플릿만 직렬화에서 지원됩니다. 이에 해당하는 이름 공간 한정자가 개별적으로 정의됩니다.

IEC:DataSpecificationIEC61360/unit (6.4절 참조) IEC:DataSpecificationIEC61360/preferredName 예: (보다 절 6.3) 또는

데이터 사양 자체의 경우 AAS 네임스페이스가 사용됩니다. AAS:DataSpecifciationIEC61360

xml 및 JSON에서 데이터 사양은 "embeddedDataSpecification" 속성을 사용하여 스키마 자체에 포함됩니다. 이러한 경우에는 개념 식별자가 사용되지 않습니다. 즉

AAS:ConceptDescription/embeddedDataSpecification

유효한 개념 식별자가 아닙니다. AAS:DataSpecificationContent 는 유효한 개념 식별자입니다.

9.2.5 임베디드 데이터 사양

이 사양은 상호 운용성을 보장하기 위해 AAS 내에서 사용할 수 있는 데이터 사양을 미리 정의합니다.

따라서 일부 직렬화 또는 매핑은 이 사양에 정의된 이러한 데이터 사양을 정확히 지원하지만 메타모델 자체가 더 유연하고 독점 데이터 사양도 지원하지만 다른 것은 지원하지 않습니다.

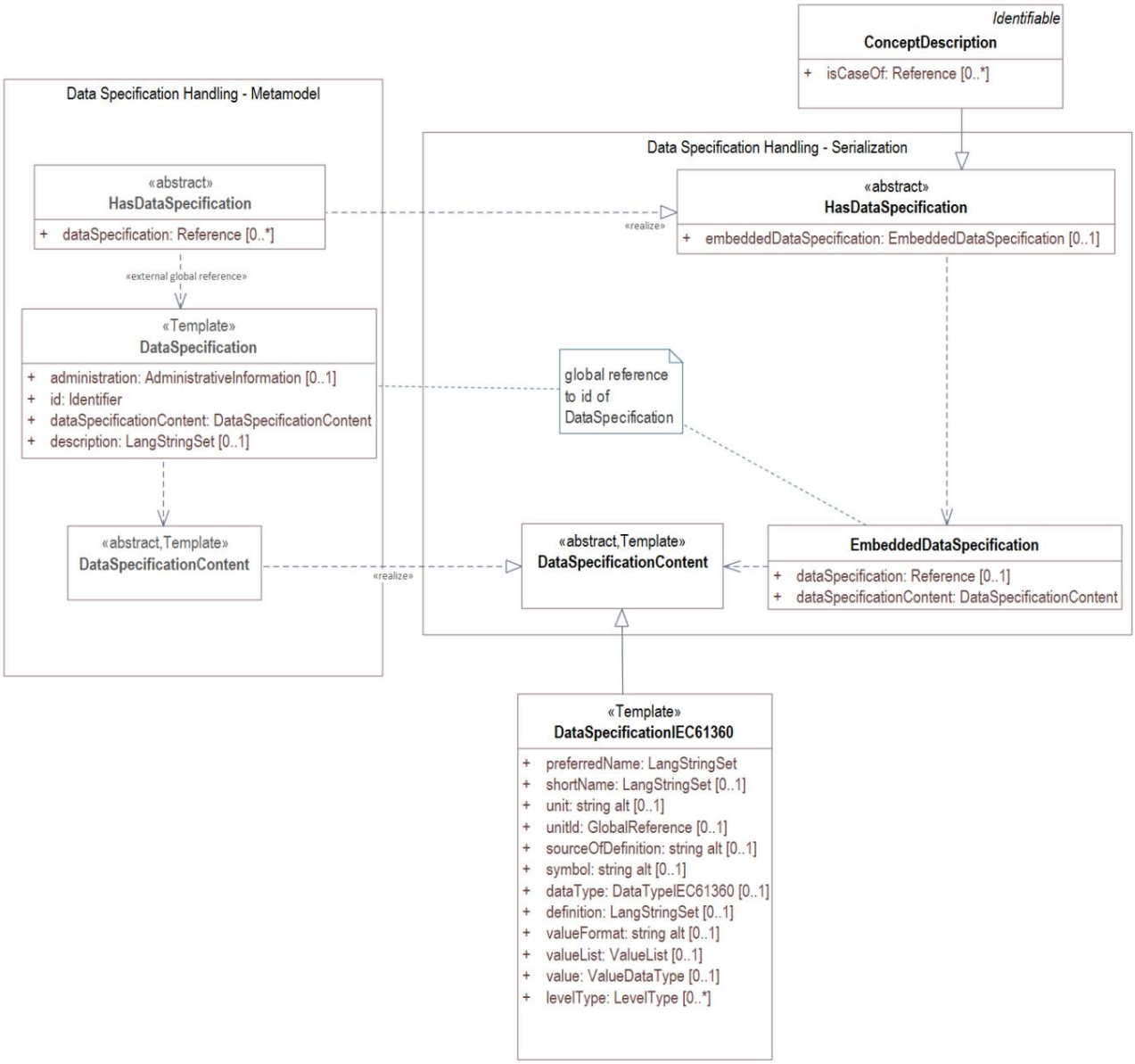
제한된 데이터 사양의 경우 사용되는 표기법은 "임베디드 데이터 사양"입니다.
그림 78에서 실현이 설명됩니다. 외부에서 정의된 데이터 사양에 대한 외부 글로벌 참조 세트 대신 데이터 사양에 대한 외부 글로벌 참조와 데이터 사양 콘텐츠 자체로 구성된 쌍 세트가 직접 "포함"됩니다. 이 실현에서 데이터 사양 내용은 스키마 등에 속하지만 일반적인 개념에서는 내용을 포함하는 데이터 사양이 아닙니다.

---

44 단순함을 위해 대부분의 예에서는 네임스페이스의 전체 경로가 아니라 네임스페이스 한정자를 사용합니다.

스키마의 일부입니다. 이것은 semanticIds 의 개념과 유사합니다 . 외부 개념 사전에 대한 외부 글로벌 참조이거나 스키마 내의 개념 설명에 대한 참조입니다. 그러나 semanticId 의 경우 정확히 하나의 참조만 허용하는 반면 데이터 사양의 경우 데이터 사양 참조 세트가 허용됩니다.

그림 78 임베디드 데이터 사양의 구현



9.3 XML

가져오기 및 내보내기 시나리오의 경우 자산 관리 셸의 메타모델을 직렬화해야 합니다. 직렬화 형식은 XML입니다.

XML45(eXtensible Markup Language) 는 IT 시스템에서 정보를 추출하는 데 매우 적합하며, 이를 수동으로 처리한 다음 다른 IT 시스템에 제공할 수 있습니다. 따라서 이는 4절의 주요 그림에 정의된 정보 공유 시나리오의 요구를 충족합니다. XML은 각 단계에서 표현된 정보를 구문적으로 검증하는 데 사용할 수 있는 체계 정의의 가능성을 제공합니다.

xml 스키마(.xsd 파일)는 github 프로젝트 admin-shell-io [41]: https://github.com/admin-shell-io/aas-specs/tree 의 "aas-spec" 리포지토리에서 유지 관리됩니다. /마스터/스키마/xml
이 사양에 정의된 기술 중립적 메타 모델에서 xml 스키마를 파생시키는 매핑 규칙은 https://github.com/admin-shell-io/aas-specs/tree/master/schemas/xml 에서 찾을 수 있습니다. #xml 매핑 규칙.

예제 파일은 https://github.com/admin-shell-io/aas-specs/tree/master/schemas/xml/examples에서 찾을 수 있습니다.

9.4 JSON

가져오기 및 내보내기 시나리오의 경우 자산 관리 셸의 메타모델을 직렬화해야 합니다. ㅏ
직렬화 형식은 JSON46 (JavaScript Object Notation)입니다.

또한 JSON 형식은 활성 자산 관리 셸에 대한 http/REST API의 페이로드를 설명하는 데 사용됩니다[49].

JSONschema(.json 파일)는 github 프로젝트 admin-shell-io [41]의 "aas-spec" 리포지토리에서 유지 관리됩니다.
https://github.com/admin-shell-io/aas-specs/tree/master/schemas/json
이 사양에 정의된 대로 기술 중립적 메타 모델에서 JSON 스키마를 파생시키는 매핑 규칙은 https://github.com/admin-shell-io/aas-specs/tree/master/schemas/json 에서 찾을 수 있습니다. #json 매핑 규칙

예제 파일은 https://github.com/admin-shell-io/aas-specs/tree/master/schemas/json/examples에서 찾을 수 있습니다.

9.5 RDF

RDF(Resource Description Framework)[44]는 의미 데이터를 명확하게 모델링하고 표시하기 위해 W3C에서 권장하는 표준입니다. RDF 문서는 주체, 관계, 객체로 구성된 3중 구조로 되어 있다. 결과 모델은 종종 주제 및 개체 요소를 노드로, 관계를 그래프 가장자리로 사용하는 그래프로 해석됩니다.

RDF는 모든 요소가 (HTTP-
)URI. 일반적으로 RDF 엔터티의 참조 위치에 추가 정보를 제공 하면 웹을 기반으로 엔터티47를 직접 연결할 수 있습니다 . 관련 정보를 찾기 위해 링크를 따라가는 이 프로세스를 리소스 역참조라고 하며 모든 브라우저 또는 웹 클라이언트에서 지원됩니다. 설명된 방식으로 웹을 통해 분산 데이터 소스를 연결하는 것을 링크드 데이터라는 용어로 참조합니다. Linked Data의 사용 가능한 리소스와 기능을 Asset Shell의 표현력과 연결하는 것이 RDF 직렬화의 동기 중 하나입니다.

또한 RDF는 광범위한 논리적 추론 및 추론 기술의 기초입니다. RDF 스키마(RDFS) 및 웹 온톨로지 언어(OWL)와 같은 어휘는 RDF의 그래프 기반 구문과 형식 정의 및 공리를 결합합니다. 이를 통해 자동화된 추론자는 엔터티 간의 관계를 이해할 수 있습니다.

어느 정도 결론을 내립니다.

두 기능을 결합한 자산 관리 셸의 RDF 매핑은 복잡한 쿼리 및 요청에 대한 기반을 제공할 수 있습니다. Semantic Web의 표준 쿼리 언어인 SPARQL은 추론을 결합할 수 있습니다.

45 참조: https://www.w3.org/TR/2008/REC-xml-20081126/

46 참조: https://tools.ietf.org/html/rfc8259 또는 https://www.ecma-international.org/publications/standards/Ecma-404.htm

47 참고: 일반 용어로서의 엔티티와 특정 하위 모델 요소 하위 유형으로서의 엔티티는 구별되어야 합니다.

외부 데이터 소스의 통합 기능. 이러한 능력을 활용하기 위해 AAS는 RDF 표현에 대한 명확한 계획이 필요합니다.

RDF 체계/OWL 파일(.ttl 파일)은 github 프로젝트 admin-shell의 "aas-spec" 저장소에서 유지 관리됩니다.
io [41]: https://github.com/admin-shell-io/aas-specs/tree/master/schemas/rdf
이 사양에 정의된 대로 기술 중립적 메타 모델에서 RDF 스키마를 파생시키는 매핑 규칙은 https://github.com/admin-shell-io/aas-specs/tree/master/schemas/json 에서 찾을 수 있습니다. #json 매핑 규칙.
예제 파일은 https://github.com/admin-shell-io/aas-specs/tree/master/schemas/rdf/examples에서 찾을 수 있습니다.

9.6 자동화ML

가져오기 및 내보내기 시나리오의 경우 자산 관리 셸의 메타모델을 직렬화해야 합니다. 직렬화 형식으로서 AutomationML(IEC 62714)은 특히 엔지니어링 단계에 적합합니다.

일반적으로 직렬화 접근 방식은 자산 관리 셸 메타모델의 각 개체를 AutomationML 역할 클래스 또는 AutomationML 인터페이스 클래스와 함께 제공되는 AutomationML 역할 클래스에 매핑하는 것입니다. 이 역할 클래스 및 (적용되는 경우) 인터페이스 클래스는 AutomationML의 필수 속성도 정의합니다.

자산 관리 셸 자체는 유형 및 인스턴스의 종류 정보에 따라 AutomationML 시스템 단위 클래스 또는 인스턴스 계층 내의 내부 요소로 모델링됩니다.

직렬화에 필요한 역할 클래스 및 인터페이스 클래스의 경우 AutomationML 역할 클래스 라이브러리 resp. 인터페이스 클래스 라이브러리가 정의되어 대중에게 제공됩니다.

목표 중 하나는 자산 관리 셸의 AutomationML 모델을 독립 실행형 AutomationML 모델로 사용할 수 있을 뿐만 아니라 향후 AutomationML 구성 요소 설명과 같은 기존 AutomationML 모델과 함께 사용할 수 있도록 하는 것입니다. 따라서 이 절에 정의된 직렬화 접근 방식의 정의는 AutomationML 정의와 인터리브되고 https://www.automationml.org/o.red.c/dateien.html 에서 AutomationML 기술 정의를 광범위하게 적용합니다.

[37]은 자산 관리 셸(AR AAS)에 대한 AutomationML 애플리케이션 권장 사항입니다. 이 부록은 정보용입니다.

Asset Administration Shell을 AutomationML에 매핑하는 작업은 공동 작업에서 수행됩니다.
AutomationML eV와 Platform Industry 4.0 간의 그룹.
결과 애플리케이션 권장 사항(AR 004E) "자산 관리 셸(AAS) 표현"[37]은 .aml 파일과 함께 https://www.automationml.org/download-archive/에서 찾을 수 있습니다.

9.7 OPC UA

OPC UA는 자산 관리 셸의 운영 단계에 적합하며 특히 기계 간 통신의 경우에 적용 가능합니다. 정보 모델[57]은 소위 OPC UA 정보 모델 또는 OPC UA Companion Specifications[58]의 정의를 위한 기초입니다.

OPC 통합 아키텍처에 대한 매핑 작업은 공동 작업 그룹에서 수행됩니다. [48] OPC Foundation,
ZVEI 및 VDMA 간의 "I4AAS" (https://opcfoundation.org/markets-collaboration/I4AAS/[39].
I4 Asset Administration Shell용 OPC UA Companion 사양의 다른 버전은 여기에서 찾을 수 있습니다. 예:
릴리스 1.00 의 경우 https://reference.opcfoundation.org/ <버전>/I4AAS/<버전>/docs /,
https://reference.opcfoundation.org/v104/I4AAS/v100/docs/.

[48] 참조: https://opcfoundation.org/collaboration/i4aas/