B-008

LLM を活用したデータ相互運用性の自動実現について

Achieving Interoperability among Different Systems with LLM aided Approach

山下蘭† 山田正隆† 岩政幹人† 砂川英一† 林晋平‡ 小林隆志‡
Lan Yamashita, Masataka Yamada, Mikito Iwamasa, Eiichi Sunagawa,
Shinpei Hayashi, Takashi Kobayashi

要旨

持続可能な社会を実現するために、製品のカーボンフットプリント（CFP）対応や循環型経済に対応するデジタル製品パスポート（DPP）など企業や業界の垣根を越えたデータ流通が重要となっている。このようなデータ流通を実現するうえで、異なるシステム間でのデータ相互運用性の実現コストが課題となっている。本研究では大規模言語モデル（LLM）を用いて異なるデータモデル間の変換ルールを生成し、得られたルールのバリデーションエラーやデータ変換エンジンからの実行エラー出力に基づいて変換ルールの修正、変換実行を繰り返すことで、標準形式のデータへの自動変換を実現する手法を提案する。ユースケースとして、製品の材料（含有化学物質）宣言に利用する IEC 62474 XSD データモデルから、標準 AAS（Asset Administration Shell）XSD データモデルへの変換と、準拠する AAS XML 形式データの自動生成を検証し、提案手法の有効性を確認した。

キーワード：データ変換、データ相互運用性、データスペース、LLM、デジタル製品パスポート、製品カーボンフットプリント

1. 背景と研究概要

循環型経済においては、製品のライフサイクル全体を通じて、異なるステークホルダー間のデータ連携が必要となる。これに欠かせないデータ相互運用性の確保に向け、我々はルールベースで異なるデータモデル間のデータを変換する技術の研究開発[1][2]を進めてきた。具体的には、データモデルの構文を定めるメタモデルに対してシンタックスレベル（構文レベル）の変換を行ったり、データモデルの定義の意味的類似性に基づいて類似要素間のセマンティックレベル（意味レベル）の変換ルールを作成したりする技術を開発してきた。しかし、変換ルールの作成と修正には専門家の判断や手作業が少なからず含まれており、このコストの抑制が課題であった。

これに対し、生成 AI、特に自然言語処理を行う大規模言語モデル（LLM）の普及に伴い、データ自動変換の効率の向上が期待されている[3]。これまでのデータ変換ルールの生成アプローチにおいては、対象データにおけるデータフォーマットの違い、階層構造の違い、及びスキーマ定義の意味的な違いが要素間の機械的な特定の妨げとなっており、手作業を要していた。LLM はこういった違いを一定程度克服でき、データ変換ルールの自動生成においても効率化が期待できる。

本論文では、低コストに高精度な変換ルールを獲得することを狙いとして、LLM を活用して異なるモデル形式と記述言語によって記述されるデータモデル間のデータ変換ルールの自動生成を行う技術を提案する。Automated Program Repair（APR）におけるソフトウェア更新アプローチ[4]を参考に、チェックツールから得られるエラーメッセージを LLM にフィードバックして生成したデータ変換ルールを修正する、反復的洗練（Iterative Refinement [5][6][7]）のアプローチを用いる。また、データ変換ルールを入力として機能する変換エンジンにも同様にフィードバック機構を組み込み、軽微なエラーを自動修正する。

提案手法では，LLM を用いて直接データ変換を行うのではなく、変換ルールの生成とルールに基づくデータ変換の工程に分解する。変換ルールを明確化することで、LLM の確率的な振る舞いに起因する変換の不確実性を軽減でき、変換先データの説明と根拠の提示も可能になる。これらの特徴は、産業応用の上で重要となる変換の信頼性に大きく寄与することが期待できる。

本論文では、提案手法及び IEC 62474 から AAS への自動変換をモチーフとした検証について詳細に述べる。

2. 提案手法

2.1 概要

異なるシステム間のデータ相互運用を実現するため、これまで我々は Meta-Object Facility（MoF）[8]に従う異なるデータモデル間の変換ルールを生成し、データの相互変換を実現する技術の研究開発を進めてきた。しかし、背景で述べた通り、変換ルール作成に伴うコストが課題であった。

そこで本研究は、図 1 に示すデータ相互運用のためのフレームワークを提案する。提案手法は変換元データ（データ A）を別のデータ表現（モデル B）でのデータ（データ

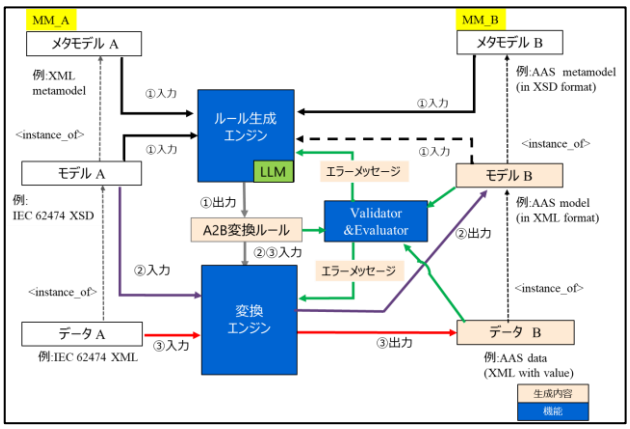

図 1 MoF に従う変換ルールとデータの自動生成フレームワーク

† AI Digital R&D Center, Toshiba Corporate Laboratory.
‡ School of Computing, Institute of Science Tokyo.

B）に変換する際に、それぞれのデータ表現（モデル）と、モデルを規定するモデル（メタモデル）の情報を活用する。データ変換は「ルール生成エンジン」と「変換エンジン」で構成される。各エンジンの出力結果の検証を行う Validator & Evaluator 機能は、実際はそれぞれのエンジン内部に含まれるが、図 1 はエラーメッセージによる修正の流れを示すため、エンジン外部に表示している。「ルール生成エンジン」は、変換元メタモデル A、モデル A、変換先メタモデル B の 3 種類の入力から、LLM を活用してモデル要素間の対応関係を推定し、モデル A からモデル B への変換ルール「A2B 変換ルール」を自動生成する。さらにし、このルールに対する Validator & Evaluator によるバリデーションエラーを用いてルールを修正する。

「変換エンジン」は、A2B 変換ルールを用い、モデル A とこれに基づくデータ A をモデル B とデータ B に変換する。その過程で、モデル B とデータ B に対する Validator & Evaluator のエラーメッセージを利用し、ルールの修正及び変換の実行を繰り返す。

Validator & Evaluator は変換ルール記述及び変換先 B のメタモデルと制約を入力とし、入力された変換ルールないし変換結果の妥当性を検証する。問題があれば、エラーメッセージを出力する。

データ変換は次の①から③のステップで実行される。

- ステップ①：2 つのメタモデルと、変換元のモデル A を入力とし、LLM にデータ変換ルール「A2B 変換ルール」を生成させる。また、過去にステップ②でモデル B が得られている場合、これも変換ルールの生成に利用でき、データ変換ルールの精度向上に貢献する（①の入出力）。
- ステップ②：生成した「A2B 変換ルール」は変換エンジンによって処理され、変換元モデル A から変換先メタモデル B に基づくモデル B が生成される（②の入出力）。
- ステップ③：生成した「A2B 変換ルール」は変換エンジンによって処理され、変換元モデル A またはデータ A から変換先モデル B やデータ B が生成される。データ B のみを生成する際には、ステップ①同様、モデル B も入力となる場合がある（③の入出力）。

以降では、ルール生成エンジン、変換エンジンの内部構成、及び Validator & Evaluator について述べる。

2.2 ルール生成エンジン

2.2.1 システム構成

本研究では、図 2 に示す構成でデータ変換ルールを生成する。下記に手順の概要を述べる。

- 利用する LLM を選定し、「A2B 変換ルール」の生成と修正を行う。
- ツールの設定。(1)生成した変換ルールに対してバリデーションを行うツール、(2)後述する「変換エンジン」で実行する変換機能のツール、(3)生成した変換先 B のモデルとデータの構文チェックを実施するツール、(4)変換先 B のモデルとデータの制約チェックを実施するツールの設定。

上記事前の選定と設定によって「ルール生成エンジン」を実現する。下記にて本提案構成の詳細を述べる。

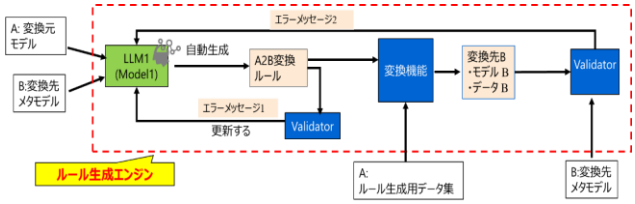

図 2　ルール生成エンジン

2.2.2 初期変換ルール生成のプロンプト

事前調査において、選定する LLM によって、入力 A、B に対して既存知識がある場合は Zero-shot（サンプルなし）で、既存知識がない場合は few-shot（サンプル）を含むプロンプトを提示することにより初期「A2B 変換ルール」を得られるという知見が得られた。また、「A2B 変換ルール」の記述言語に関する指示の与え方も生成精度に影響を与えるという知見も得られた。

そこで本提案では、変換元 A と変換先 B のメタモデルやモデル、変換ルールの記述言語と仕様情報を含むプロンプトのテンプレートを設計した。下記に一例を示す。ルール生成時は、対象のモデルに応じて下記{}部分を具体化することにより初期プロンプトを作成し、LLM に変換を指示することとした。

2.2.3 エラーメッセージによる反復的洗練の実施

```
#Prompt Template
Please generate the transformation rule between A and B using
{#rule_description_language} according to
{#rule_description_language specifications}.

A is using {#A file format} according to {#A metamodel specification}
and {#A file format specifications}.
B is using {#B file format} according to {#B metamodel specification}
and {#B file format specifications}.
##Samles while necessary.
{Sample for A metamodel}
{Sample of B model}
{Sample of {#rule_description_langugae}}
```

ルール生成エンジンは、「A2B 変換ルール」を Validator & Evaluator に与えて構文的なエラーを検出し、出力されたエラーメッセージ（図 2 のエラーメッセージ 1）を用いてルール修正を指示するプロンプトを作成し、これを LLM に与えてルールの修正を行う。

また、変換先 B のモデル B とデータ B に対して、変換先 B メタモデル（制約含む）を用いる制約チェックを行い、得られるエラーメッセージ 2 を用いて、「A2B 変換ルール」の修正を行う。修正プロセスはルール生成エンジン自身によって制御され、Validator & Evaluator からのエラーメッセージがなくなることをゴールとして繰り返される。ただし、LLM によるルール修正では所望のデータ変換に到達できない場合があることも既に確認しており、繰り返しの回数や実行時間を用いた実行のゴール設定や、人間の介入などについて、本提案手法を適用する際に、サンプルデータによる検証の必要がある。

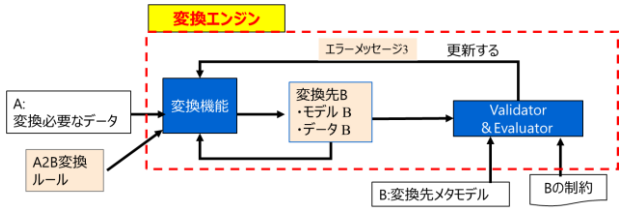

図 3　変換を実施する「変換エンジン」

2.3 変換エンジン

作成した「A2B 変換ルール」を用いて、図 3 に示す構成で変換元モデル A とデータ A をモデル B とデータ B へ変換する。このうち「変換機能」は、図 2 で示した変換機能と同様である。

- 生成する変換先 B のモデルまたは B のデータに対して、B のメタモデルを用いた構文などのチェックと検証を行える。また、変換先 B のメタモデルで表現できない追加制約がある場合は、図 3 に示すように、Validator & Evaluator に入力として制約を与え、これを用いるようなチェックも可能とする。
- Validator & Evaluator で得られたエラーメッセージ 3 を用いて、変換先データの修正を行う。修正は外部ツールにおいて行う場合と、LLM による自己修復のアプローチを用いる場合の双方が可能である。

2.4 Validator & Evaluator（ツール設定）

ルール生成エンジンと変換エンジンは、出力となる A2B 変換ルール及びモデル B とデータ B を検証するために、Validator & Evaluator の外部ツールを用いる。検証から得られるエラーメッセージを用いて修正用プロンプトを生成する。修正用プロンプトを LLM にフィードバックし変換ルールの修正を行う必要性がある。本研究の事前調査では、設定する外部ツールにより得られるエラーメッセージの厳密性や正確性により、生成する「A2B 変換ルール」と変換先 B のデータの精度に影響する。変換元 A と変換先 B に応じて外部ツールを事前検証し、実際の適用では影響を多面的に評価したうえで選ぶ必要がある。

3. ケーススタディ

3.1 概要と事前検証

本研究では、既存 LLM はコード生成能力や変換対象に係る事前知識がモデルによってまちまちであることを考慮して、図 1 に示す「ルール生成エンジン」と「変換エンジン」の構成を用いて、異なる LLM にての検証を実施し、それぞれ異なる結果が得られることを確認した。

提案手法の検証として、デジタル製品パスポート（DPP）に含む、化学物質宣言する IEC 62474 の XML データモデル（図 1 のモデル A に該当）と、データスペースで流通する AAS のデータモデル（図 1 のモデル B に該当）を選定し、検証を行った。本章はその詳細内容を述べる。

ユースケース選定における事前検証においては、下記結果が明確に分かり、ユースケースの構成を決定した。

- メタモデル A／モデル A の形式として JSON／JSON Schema 及び XML／XSD を候補として検討した。Gemini 及び Claude 共に、XML／XSD に対してより正確に解釈できることが明確に分かった。従って、本検証は、変換元のメタモデル A／モデル A と変換先メタモデル B／モデル B は、XML／XSD 形式である IEC 62474 XSD データモデルから、標準 AAS XSD データモデルへの変換ルール作成と決定した。
- 変換ルールとして、変換元のメタモデル A が XML の場合に XSLT、JSON-LD の場合に JSONata[1] 及び JSONiq[2]を検証した。初歩的な事前検証の結果では、Gemini において XSLT が JSONota 及び JSONiq より高い精度で変換ルールを生成できることが分かった。従って、本検証は、XML の標準フォーマット変換言語である XSLT を用いた。
- Gemini 2.5 Flash は入力トークン数が 100 万、出力トークン数が 65,000 と大きいため、今回対象となる変換ルールを一度に出力できる。また、Gemini は追加コンテキストなしに AAS XSD のデータ構造を最も正しく解釈できたため、変換ルール生成のモデルとして Gemini を選択した。
- 変換エンジン及び Validator & Evaluator として用いる外部ツールとして、複数の OSS、サンプルデータを用いて検証した。各 OSS ツールから出力するエラーメッセージを LLM にフィードバックして XSLT を修正させるため、その修正に十分な情報を含むエラーメッセージを出力できているかどうかの基準で外部ツールを選定した。結果、XSLT の実行には Saxon-HE[3]、XML のスキーマ検証には XmlValidate[4]、AAS の制約チェックには AAS Compliance Tool[5]を利用することとした。

3.2 ルール生成エンジンの構成

3.2.1 初期 XSLT の生成

最初に XSLT を生成するために用いたプロンプトを以下に示す。

```
Please output the XSLT rules that convert the XML data according to the attached XML Schema into the Asset Administration Shell Submodel XML format. Please output only XSLT rules. When converting XML data, be sure to convert all elements and attributes to Submodels.

=== XML Schema
{IEC_62474_XSD}

=== Asset Administration Shell XML Schema
{AAS_XSD}
```

紙面の都合で割愛しているが、{IEC_62474_XSD}には IEC 62474 の XML Schema、{AAS_XSD}には AAS の XML Schema が含まれる。事前調査では日本語よりも英語のプロンプトの方が高精度であったため、英語を用いた。

[1] https://jsonata.org/
[2] https://www.jsoniq.org/
[3] https://github.com/Saxonica/Saxon-HE
[4] https://github.com/docjason/XmlValidate
[5] https://github.com/rwth-iat/aas-compliance-tool

3.2.2 XSLT のフォーマットエラーによるルール修正

LLM を用いて自動生成した XSLT 形式の変換ルールに対して Validator を適用してフォーマットエラーが発生したときは、LLM を用いて変換ルールを修正する。フォーマットエラーがなくなるまで繰り返し修正する。

得られたエラーメッセージを利用して変換ルール修正プロンプトを作する。XSLT のフォーマットエラーを修正するプロンプトを以下に示す。

```
When I executed the attached XSLT, an error was output. Please correct the XSLT so that the error disappears. Please output only XSLT.

=== ERROR
{ERROR}
=== XSLT
{XSLT}
```

プロンプト中の{ERROR}はデータ変換実行時のエラーメッセージ、{XSLT}はエラーが発生した XSLT に置き換える。

3.2.3 モデルとデータの妥当性検証によるルール修正

変換ルールに対する Validator によるエラーがなくなったら、XSLT による変換を実行して IEC 62474 の XML を AAS 形式に変換する。次に変換した AAS と AAS の XML Schema を用いて妥当性を検証する。妥当性検証を行う Validator にかけて検証エラーが発生したときは、LLM を用いて再度変換ルールを修正する。妥当性検証エラーがなくなるまで繰り返し修正する。

妥当性検証エラーを用いて XSLT の修正を指示するプロンプトを以下に示す。

```
The attached XSLT converts the XML data to the Asset Administration Shell Submodel XML format according to the XML Schema. When I schema validated the output Asset Administration Shell Submodel, the attached ERROR occurred. Please correct the XSLT so that the error disappears. Please output only XSLT rules. When converting XML data, be sure to convert all elements and attributes to Submodels.

=== ERROR
{ERROR}
=== XSLT
{XSLT}
=== XML Schema
{IEC_62474_XSD}
```

プロンプト中の{ERROR}は検証時のエラーメッセージ、{XSLT}はエラーが発生した XML を変換するときに用いた XSLT、{IEC_62474_XSD}は IEC 62474 の XML Schema に置き換える。調査中に得られた知見として、XSLT のルール修正時には、AAS の XML Schema を渡さない方がより正しく修正できたため、ここでは AAS の XML Schema を渡さない場合を掲載している。

3.2.4 制約チェックによるルール修正

妥当性検証の Validator によるエラーがなくなったら、生成された AAS XML 形式のデータに対して、標準 AAS の制約を用いて検証し、制約に合致しないデータを検出する。制約チェックを行う Evaluator がエラーを出力したときは、LLM を用いて変換ルールを修正する。制約チェックエラーがなくなるまで繰り返し修正する。

```
The attached XSLT converts the XML data to the Asset Administration Shell Submodel XML format according to the XML Schema. When I ran the AAS specification constraint check on the translated output Asset Administration Shell Submodel by the XSLT, the attached ERROR occurred. Please correct the XSLT so that the error disappears. Please output only XSLT rules. When converting XML data, be sure to convert all elements and attributes to Submodels.

=== ERROR
{ERROR}
=== XSLT
{XSLT}
=== XML Schema
{IEC_62474_XSD}
```

制約チェックエラーを用いて XSLT の修正を指示するプロンプトを以下に示す。プロンプト中の{ERROR}は制約チェックのエラーメッセージに置き換える。

3.3 データ変換エンジンの構成

XSLT によるデータ変換には Saxon-HE XSLT エンジンを用いた。なお、変換先 AAS モデルにしか存在しない要素の生成が難しいことが事前検証によりわかっていたため、XSLT により出力された AAS XML ファイルに対して、あらかじめ人手で策定したテンプレートを後処理として結合して対処した。この出力に対して AAS Compliance Tool を用いて妥当性検証及び制約チェックを実施した。本ツールは 51 種類の制約に対応している。

3.4 データ変換結果及び考察

変換元の IEC 62474 の XML は（モデルとデータを含む）全て同じ内容を用いて、複数回のデータ変換を試行した。

修正を繰り返しても正しい XSLT が生成されない、あるいは結果が空の XML となった場合を除き、正しい AAS 形式への変換に成功した独立の 5 回の試行結果を**表 1** にまとめる。フォーマット修正、妥当性検証、制約チェックの回数は、正しい AAS 形式への変換に成功するまでに繰り返し修正した回数を示している。ここで、正しい AAS 形式とは、各種 Validator & Evaluator がエラーを出力しなかった場合を指す。

表 1　試行の結果（繰り返し修正の総回数）

試行番号(#)	#1	#2	#3	#4	#5
3.2.2 フォーマットエラー	19	7	8	6	4
3.2.3 妥当性検証	6	2	3	4	1
3.2.4 制約チェック	7	1	2	2	1

本例は、提案手法によって IEC 62474 の XML を自動的にエラーのない AAS のデータに変換できることを示唆している。上記の試行でルール生成時の自動修復とルール適用結果に基づく自動修正が実施されていることから、提案手法の構成によって LLM を用いることによる不確実性を軽減できていると考える。

一方で、同一の変換元 XML を用いたにも関わらず、実行のたびに回数は異なり、また修正内容も大きく異なっており、パターン化して整理することは困難であった。例えば、同じエラーでも修正できる場合とできない場合があり、繰り返し回数の違いの要因となっていた。他にも、あるエ

ラーの修正時に、過去のエラーが再発することもあった。また、上記の試行では、全 51 のうち 2 種類の制約エラーのみが発生し、それらは正しく検出され修正できたものの、他の制約については未確認である。これらの傾向の一般化については、今後の課題である。

4. 今後の課題

4.1 LLM の課題

データ変換ルールの生成には LLM を活用するが、選定する LLM によって、変換元及び変換先のメタモデルに対する処理能力に差があることが、本研究開発によって明確になった。今後、汎用性を向上させるための課題として LLM に変換元及び変換先のメタモデルを事前に学習させる手法が必要である。

LLM によって得意とするエラーメッセージの処理に差があり、適切なプロンプトの与え方も異なる。また、LLM 自体は進化が著しいため、プロンプトの与え方を工夫しても次の LLM のバージョンでは有効に機能しないこともある。本研究では、データの構造やセマンティクスに係る制約情報をコンテキストとして与えれば、LLM を用いて適切なデータ変換結果が得られることが分かった。このような結果から、期待する出力を得るためにプロンプトに必要なコンテキストの知見を蓄えることが課題であることがわかった。

4.2 変換ルール記述言語の課題

変換元及び変換先のメタモデル、モデルの記述言語により、データ変換ルールの記述言語は異なる。本研究で記載した XSD と XML には XSLT が広く用いられるが、例えば JSON や RDF など他形式のデータ変換を実施する場合には、それに適した変換ルールの記述言語や、その厳密性に合わせた評価手法について調査・開発する必要がある。

4.3 生成した変換先データのバリデーション、評価の課題（Validator & Evaluator）

自動生成した変換先 B のモデルやデータに対して、記述するファイル形式の構文が明確である場合は、構文レベルのバリデーションとエラー修正は可能である。例えば、本ユースケースの XML は構文が明確であり、このケースに該当する。一方で、例えば、シンプルな CSV ファイルなど、構文情報が自由な場合は、既存 OSS を活用したバリデーションが難しく、追加の外部ツールの開発が必要になる。さらに意味的な評価をシステム的に行うには、意味的な正確性・一貫性を保証する標準データ辞書やオントロジの活用する手法[9]があり、今後の活動に継続して検討を進める。

4.4 変換先の必須モデル要素とその値の付与

本研究では、変換元 A から変換先 B への自動変換を目的とし、変換先 B に必須であるものの変換元 A に存在しないデータモデル要素及び値を付与する必要がある。3.3 節で述べたように、本ケーススタディでは後処理で行ったが、今後、モデル要素と値の付与の自動化の検証を進める。

4.5 制約やエラーメッセージ記述による変換ルール精度への影響

制約の正式表現やエラーメッセージの表現は、LLM を活用した変換ルールの生成精度に影響する。本ケーススタディでは、AAS は国際規格であり、制約が明確に定義されているため、制約検証のエラーを用いて、LLM によるルール更新が可能になった。一般的には、メタモデルの構造や属性に関する形式的な制約が欠如している場合、生成された変換ルール精度が低く、変換先のデータが期待される形式に合致しない可能性が高まる。また、選定した Validator や Evaluator のエラーメッセージの表現に関しても、エラーの原因を正確に特定し、修正するための情報を提供することが求められる。そのため、選定した Validator や Evaluator のエラーメッセージの具体性や正確性の向上が課題である。

4.6 変換漏れ項目の課題

本ケーススタディでは、全てのエラーを解消して正しい AAS XML に変換できたときでも、最終的に出力された変換結果においてデータ変換元の IEC 62474 の XML に存在する 0～28 の要素及び属性が変換先において存在しない変換漏れが発生した。今後、変換漏れの要素及び属性の値が存在するかをチェックし、変換漏れに対して修正させる汎用的な技術開発も必要である。

5. まとめ

本研究で提案した LLM を活用した反復的洗練のアプローチに基づく「ルール生成エンジン」及び「変換エンジン」の有効性を確認した。ケーススタディ検証において IEC 62474 から AAS への自動生成が可能であることも確認した。また、4 章に記述したように課題も明確になった。

今後、得られた課題に対して更なる技術開発を進めると共に、製品 CFP データや DPP データ整備などのケーススタディを進め、製品ライフサイクルを渡った複数システム間や、データスペース間での検証と実用化を目指す。

参考文献

[1] L. Wang et al., “Applying Class Distance to Decide Similarity on Information Models for Automated Data Interoperability,” *International Journal of Software Engineering and Knowledge Engineering*, vol. 31, no. 3, pp. 405-434 (2021)

[2] L. Yamashita et al., “An AAS Generation Tool and Its Application to a Data Ecosystem for Carbon Footprint of Products”, 情報処理学会第 85 回全国大会 2A-06 (2023)

[3] Y. Xia et al., “Generation of Asset Administration Shell with Large Language Model Agents: Towards Semantic Interoperability in Digital Twins in the Context of Industry 4.0”, *IEEE Access*, vol. 12, pp. 84863-84877 (2024)

[4] Q. Zhang et al., “A Systematic Literature Review on Large Language Models for Automated Program Repair”, arXiv:2405.01466 [cs.SE] (2025)

[5] J. D. Zamfirescu-Pereira et al., “Iterative Disambiguation: Towards LLM-Supported Programming and System Design”, AI & HCI Workshop at the 40th International Conference on Machine Learning (ICML) (2023)

[6] S. Russell et al., “Artificial Intelligence: A Modern Approach”, fourth ed. Pearson. (2020)

[7] D. B. Acharya et al. “Agentic AI: Autonomous Intelligence for Complex Goals—A Comprehensive Survey,” *IEEE Access*, vol. 13, pp. 18912-18936, (2025)

[8] ISO/IEC 19502, “Information technology — Meta Object Facility (MOF)” (2005)

[9] N. Braunisch et al., “Software Evolution of I4.0 Digital Twins with Semantic Patching”, the 33rd International Symposium on Industrial Electronics (ISIE) (2024)