2025 European Conference on Computing in Construction

CIB W78 Conference on IT in Construction

Porto, Portugal
July 14-17, 2025

THE QUEST FOR THE OPENBIM EXCHANGE REQUIREMENTS CHECKING LANGUAGE: COMPARING MVDXML, IDS AND GHERKIN

Štefan Jaud[1], Sylvain Marie[2], and Andreas Pinzenöhler[3]
[1]Jaud IT GmbH, Germany, stefan@jaud-it.com
[2]Eurostep, France, sylvain.marie@eurostep.com
[3]IQ Soft GmbH, Austria, andreas.pinzenoehler@iqsoft.at

Abstract

Information delivery specification (IDS), *model view definition (mvdXML)* and *Gherkin* are the main encodings of quality assurance rules for Industry Foundation Classes (IFC) models governed and/or employed by buildingSMART International (bSI). We critically compare their data models, philosophy, versatility, and supported functional blocks. Of the three, IDS provides fastest time-to-market with its limited vocabulary, Gherkin rules support encoding of the most requirements with most effort, and mvdXML provides the best trade-off between broad support for various requirements while retaining the benefits of a limited vocabulary for fast implementation.

Introduction

Motivation

bSI develops and governs different non-proprietary data schemas important for the digitalization of the architecture, engineering and construction (AEC) industry. The flagship IFC standard – a data model for exchange of modelling data – supports a broad scope of Building Information Modelling (BIM) concepts and requires a proper constraint mechanism to specify and assess the compliance with given exchange information requirements (EIRs). Aligning the legal realm (realized with human-readable contractual requirements) with the engineering realm (realized with machine-readable deliverables) has been facilitated with the International Organization for Standardization (ISO) 19650 standard development. Thus, procurement documents can define and share EIRs which can be checked on IFC deliverables to certify their correctness and relevance.

The main challenge is to capture data requirements from domain experts (usually expressed in their engineering vocabulary) and translate them into an Information Technology (IT) language based on the IFC specification. A combination of both not only supports quality assurance / quality control (QA/QC), but also enables communication with AEC domain experts who may not normally have detailed knowledge of the IFC data model. Ideally, it may also enable control of the data flow between the design team members to deliver just the right amount of data and thus become a central part for information management.

Next to stating the requirements, the EIRs provide the necessary information to derive the rules to check and certify the correctness and relevance of the deliverables. While Tomczak et al. (2022) have focused on the specification possibilities for the different requirements, there is a research gap for the comparison of serialization of these requirements for automatic, machine driven QA/QC (Jaud et al., 2024). In this study, we focus on serialization possibilities of checking rules derived from EIRs which enable automatic compliance testing of IFC deliverables.

Methodology

This contribution takes a closer look at *IDS*, *mvdXML* and *Gherkin* rules – the main encodings of quality assurance rules for IFC models governed and/or employed by bSI. Lacking a formal methodology to compare data models based on eXtensible Markup Language (XML) (IDS and mvdXML) with scripting and programming languages (Gherkin), we relied on our own expertise from previous research projects and development endeavours.

We focus on determining if a selection of requirements is possible to be expressed in each of the formats under review, and not if such an endeavour is economically viable. Furthermore, the IFC data as well as the serialized rules are assumed to be formally correct, i.e. grammatically and language conform.

We critically compare the three data models in focus, looking at their philosophy, versatility, and supported functionality. The criteria chosen for comparison are derived from the need to be useful for the purpose of unambiguously serializing any EIR. Additionally, we stress-test the usage of all three standards under review on fictional examples as well as from the international aIfc4Rail project.

The paper is structured as follows. This section presents our motivation and the research gap at hand. Next section presents the background knowledge necessary to understand the three serialization in focus. Following are the main contributions of the paper: the theoretical analysis as well as the exemplary usage of IDS, mvdXML and Gherkin. We conclude the paper with a short summary of our findings and an outlook.

Background

IFC is a semantically and geometrically rich data model, building on the standard for the exchange of product model data (STEP) legacy. Despite using the EXPRESS & STEP languages (ISO 10303), bSI didn't define IFC to be a STEP Application Protocol (AP). Thus, bSI turned away from related methodologies (i.e. ISO 10303 parts 3x for conformance testing methodology and parts 3xx for Abstract Test Suites). On top of that, the vast majority of IFC files are serialized in STEP Physical Files (SPF) using ISO 10303-21:1994, currently without intention to adopt the newer version 10303-21:2016 which supports referencing, document signing and UTF8. As the schism between the STEP legacy and IFC now seems confirmed, requirement specifications for BIM workflows may not build on other open standards anymore. Thus, a number of languages have been developed by the bSI community in order to ensure EIR compliance. This section summarizes the main elements of the three vocabularies in focus.

Information Delivery Specification (IDS)

The following descriptions are based on the official documentation of IDS standard (bSI, 2024a). An IDS dataset contains the following main elements:

ids represents the top node of an IDS dataset. It includes the *info* metadata object about the dataset itself as well as multiple *specification* elements.

specification is a collection of constraints for a selection of objects.

applicability defines the selection of objects from the model with a combination of facets.

requirements defines the constraint on applicable objects with a combination of facets.

simpleValue allows to specify specific values, ranges and patterns to check against.

The *applicability* and *requirements* nodes both allow to provide cardinality, which allows the author of the IDS to specify exclusions and forbidden content as well. Table 1 lists all currently available facets with their attributes. Each facet used as a *requirement* additionally includes an attribute *instructions*, where the "*author of the IDS can leave instructions for the authors of the IFC*" (bSI, 2024a). Algorithm 1 showcases an exemplary *specification* node serialized in XML, employing *Entity* with *Attribute* facets as well as a *Property* facet as the *applicability* criteria and the *requirement*, respectively. The specification applies to entities of type IfcSpace with the word *Bedroom* in their Description attribute and requires them to have a property set named *Qto_SpaceBaseQuantities* assigned, that contains a property named *NetFloorArea* with an area measure value ≥ 10 assigned.

mvdXML

These descriptions are based on the official documentation of mvdXML standard (bSI, 2024c) as well as based upon Weise et al. (2016). An mvdXML contains the following main elements:

Table 1: IDS facets with their attributes in alphabetical order. Denoted facets allow to specify the source of the requirement by means of an URI (bSI, 2024a).

Facet Name	Attributes	URI
Attribute	Name, Value	✗
Classification	System, Value	✓
Entity	Name, PredefinedType	✗
Material	Value	✓
PartOf	Entity, Relation	✗
Property	BaseName, DataType, PropertySet, Value	✓

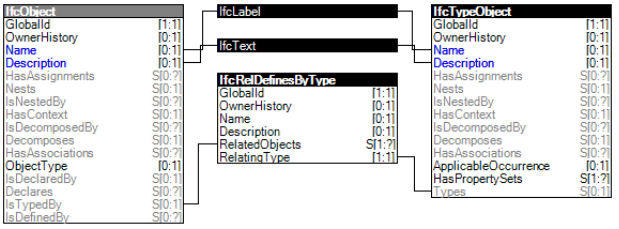

Figure 1: Graphical representation of the concept template Object User Identity *from RV (bSI, 2020). The checkable object is IfcObject, with the attributes relevant for the template drawn with their names in blue and their data types depicted.*

ModelView represents the top node of the mvdXML dataset. It includes multiple *View* elements, which are the main container for *ExchangeRequirement* and *ConceptRoot*.

ExchangeRequirement carries the data that is relevant for a specific use case.

ConceptRoot represents a collection of constraints for a selection of objects of type denoted with the *applicableRootEntity* attribute.

Applicability defines the rules by which objects are selected from the model.

Concept defines an individual constraint on applicable objects and how it is used in exchange scenarios.

ConceptTemplate defines a unit of functionality – a template for rules – that is used by the *Applicability* and *Concept* elements that configure the constraints.

TemplateRules is a collection of *TemplateRule* objects combined freely using boolean logic.

TemplateRule provides the parameters needed to configure the templates and denote the type of checks, following a specialized grammar.

For example, Figure 1 shows the template *Object User Identity*, which provides the means necessary to query an IfcObject's or its assigned IfcObjectType's Name and Description attributes (bSI, 2020). Algorithm 2 lists an excerpt of the XML serialization of the same template. To complete the example, Algorithm 3 presents the checking rules requiring the uniqueness of the door type names within the IFC dataset, following the specialized mvdXML grammar.

Listing 1: A specification *node serialized in XML. Some attributes of nodes left out for brevity.*

```
<ids:specification name="Bedroom␣minimum␣
↪ size">
 <ids:applicability>
  <ids:entity>
   <ids:name>
    <ids:simpleValue>IFCSPACE</ids:
    ↪ simpleValue>
   </ids:name>
  </ids:entity>
  <ids:attribute>
   <ids:name>
    <ids:simpleValue>Description</ids:
    ↪ simpleValue>
   </ids:name>
   <ids:value>
    <xs:restriction>
     <xs:pattern value=".*Bedroom.*"/>
    </xs:restriction>
   </ids:value>
  </ids:attribute>
 </ids:applicability>
 <ids:requirements>
  <ids:property cardinality="required"
  ↪ dataType="IFCAREAMEASURE">
   <ids:propertySet>
    <ids:simpleValue>
    ↪ Qto_SpaceBaseQuantities</ids:
    ↪ simpleValue>
   </ids:propertySet>
   <ids:baseName>
    <ids:simpleValue>NetFloorArea</ids:
    ↪ simpleValue>
   </ids:baseName>
   <ids:value>
    <xs:restriction>
     <xs:minInclusive value="10"/>
    </xs:restriction>
   </ids:value>
  </ids:property>
 </ids:requirements>
</ids:specification>
```

Listing 2: XML excerpt from the template Object User Identity *from Figure 1 (bSI, 2020). Some attributes of nodes left out for brevity.*

```
<ConceptTemplate uuid="c19ec186-9cfd-47fc-
↪ a4d4-9fb35008d04a" name="Object␣User␣
↪ Identity" applicableSchema="IFC4"
↪ applicableEntity="IfcObject">
 <Rules>
  <AttributeRule RuleID="ObjectName"
  ↪ AttributeName="Name"/>
  <AttributeRule RuleID="ObjectDescription
  ↪ " AttributeName="Description"/>
  <AttributeRule AttributeName="IsTypedBy"
  ↪ >
   <EntityRules>
    <EntityRule EntityName="
    ↪ IfcRelDefinesByType">
     <AttributeRules>
      <AttributeRule AttributeName="
      ↪ RelatingType">
       <EntityRules>
        <EntityRule EntityName="
        ↪ IfcTypeObject">
         <AttributeRules>
          <AttributeRule RuleID="TypeName"
          ↪  AttributeName="Name"/>
          <AttributeRule RuleID="
          ↪ TypeDescription"
          ↪ AttributeName="Description"/>
 <!-- some closing tags not shown -->
</ConceptTemplate>
```

Listing 3: XML excerpt of a ModelView *node. Since the node* ConceptRoot.Applicability *is empty, the* ConceptRoot.Concepts *apply to all* IfcDoor *objects within the IFC dataset. The* uuid *reference* c19e... *points to the* ConceptTemplate *defined in Algorithm 2. The* TemplateRule.Parameters *attribute holds the checking settings, where the keys to the attributes have been defined with* RuleID *attributes in Algorithm 2. Some attributes of nodes left out for brevity.*

```
<ModelView applicableSchema="IFC4" uuid=
↪ ...>
 <ExchangeRequirements>
  <ExchangeRequirement applicability="both
  ↪ " uuid="375beddf-a19a-5acd-1759-9121
  ↪ ea23e312" name.../>
 </ExchangeRequirements>
 <Roots>
  <ConceptRoot applicableRootEntity="
  ↪ IfcDoor" uuid...>
   <Applicability/>
   <Concepts>
    <Concept uuid...>
     <Template ref="c19ec186-9cfd-47fc-
     ↪ a4d4-9fb35008d04a" />
     <Requirements>
      <Requirement exchangeRequirement="
      ↪ 375beddf-a19a-5acd-1759-9121
      ↪ ea23e312" requirement="
      ↪ recommended" />
     </Requirements>
     <TemplateRules operator="or">
      <TemplateRule Parameters="
      ↪ TypeName[Unique]=TRUE" />
 <!-- some closing tags not shown -->
</ModelView>
```

Gherkin

These descriptions are based on the official documentation of Gherkin language (Cucumber, 2024) as well as from Wynne and Hellesoy (2012). A Gherkin file is a simple textual file, which is by design an executable test script as well. All instructions in the file are written on individual lines and consist of a recognized Gherkin keyword followed by a string.

Feature definition represents the top node of the *feature* file and describes the intent of the feature file.

Scenario (alias **Example**) illustrates a specific business rule, consisting of (multiple) steps.

Given is a step describing an initial context – the scene of the *scenario*.

When is a step describing an event, e.g. a user action.

Then is a step describing the expected outcome, given the context and event steps. Here, the assertion of the state of the system occurs.

And , **But** and * can replace successive repeated step keywords for a more fluidly structured scenario.

The rest of the lines are considered comments and are ignored by the interpreter.

However, the feature files only represent the business logic and requirements (cf. Algorithm 4). These need to be connected to executable code in a chosen programming language (cf. Algorithm 5), which then runs the system under test, given the provided parameters in the individual steps. Thus, even a short statement like *Given an IfcGroup* might entice multiple function calls, resulting in 25 code

Listing 4: Acyclic group rule written in Gherkin (bSI, 2024b).

```
Feature: GRP001 - Acyclic groups

The rule verifies that an IfcGroup does
↪ not reference itself, not even through
↪ intermediary entities.

 Scenario: Agreement on IfcGroup (and
 ↪ hence systems) being acyclic

  Given an IfcGroup
  Then  It must not be referenced by
  ↪ itself directly or indirectly
```

Listing 5: Acyclic requirement (cf. Then *line in Algorithm 4) as implemented with Python (bSI, 2024b).*

```
@gherkin_ifc.step('It␣must␣not␣be␣
↪ referenced␣by␣itself␣directly␣or␣
↪ indirectly')
def step_impl(context, inst):
 relationship = {'IfcGroup': ('
 ↪ HasAssignments', 'IfcRelAssignsToGroup
 ↪ ', 'RelatingGroup')}
 def get_memberships(inst):
  for rel in filter(misc.is_a(ent),
  ↪ getattr(inst, inv, [])):
   container = getattr(rel, attr)
   yield container
   yield from get_memberships(container)
 inv, ent, attr = relationship[inst.is_a()
 ↪ ]
 if inst in get_memberships(inst):
  yield ValidationOutcome(inst=inst,
  ↪ expected=None, observed=None,
  ↪ severity=OutcomeSeverity.ERROR)
```

lines manually implemented in bSI (2024b).

Examples

We put the three serializations to the test with requirements collected from various sources. We consider the theoretically possible serializations as positive results of our study, even though an implementation may still be necessary to empirically prove the correctness of our assessment. To determine the possible serialization we relied on our own expertise from previous research projects and development endeavours.

IDS Documentation

The IDS documentation lists some illustrative examples of requirements supported and unsupported by the standard (bSI, 2024a). We list them in Table 2 where our success with serialization is marked for each technology.

aIfc4Rail project

The *Applying IFC 4.3 for Rail* (aIfc4Rail) project was the third in a sequence of bSI-projects dedicated to the extension of IFC for the railway domain. A substantial number of railway engineers fulfilled the role of domain experts and together elaborated requirements considered important for the introduction of IFC based data exchanges into the typically highly standardised business processes of rail construction. The business requirements were categorised into three groups: i) requirements of the appointed party, ii) required software feature, and iii) requirements in IFC. At the end of requirements detection work package approximately 150 detailed requirements were available.

Table 2: Requirement examples from IDS documentation with denoted support for the three serializations (bSI, 2024a).

Requirement	IDS	mvdXML	Gherkin
External load bearing walls need to have a fire rating property.	✓	✓	❍
Bedrooms should have a minimum area of 10 m^2.	✓[1]	✓	❍
All brick wall types must be classified and follow a naming convention.	✓	✓	❍
There must be no clashes between structural beams and pipes.	✗	✗	❍
All walls need to be 3 m away from the site boundary.	✗	✗	❍
The names of all door types must be unique.	✗	✓[2]	✓
All pumps need to have a nominated supplier and manufacturer.	✗	✓	❍
The total area of all office spaces must be more than 300 m^2.	✓[3]	✓	❍
All air handling units must have sensors assigned with trigger events.	✗	✓	❍
Saturday and Sunday must be a holiday in all work schedules.	✗	❍	❍
All models shall load in under 3 minutes by major software vendors.	✗	✗	✗
Associated drawings in the model must match the latest revisions.	✗	❍	❍
All rebar should be modeled as parametric swept disks.	✗	✓	❍
The model must match the as-built state of construction.	✗	✗	✗

✗ not possible ❍ technically possible
✓ possible and available
[1] cf. Algorithm 1 [2] cf. Algorithm 3 [3] Realized with a precalculated area measure attached to a storey or a building.

We present an example from the track domain of the aIfc4Rail project as schematically presented in Figure 2:

> *The railway signal* Signal 01 *shall be linearly placed along the alignment* 0001R *at station* 69.395 m.

To achieve an IFC exchange, a series of different IFC entities and concepts have to be orchestrated in a correct manner. This exchange requirement defines some key-value constraints (e.g. on the station's linear measurement) but also some structural constraints to ensure the correct relationships between objects to achieve the domain-specific semantics. The example can be modelled in IFC as presented in Table 3. Here, we additionally list the necessary functional blocks to facilitate IDS and mvdXML checks with each (sub)branch of the IFC dataset. Let it be noted here that a Gherkin check is possible in any manner, since it employs open vocabulary. That is – depending on the underlying implementation – either as this exact sentence

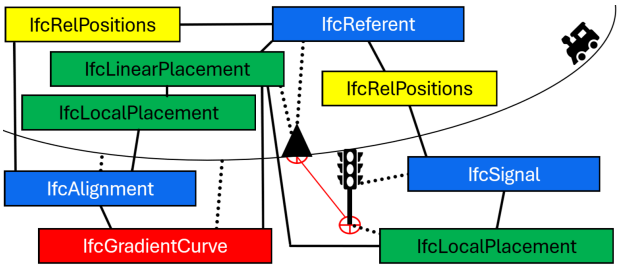

Figure 2: Schematic representation of a signal positioned along an alignment, with required IFC entities from Table 3 for realization shown. Full lines depict connections in the data, dotted lines connect the entities to their representation.

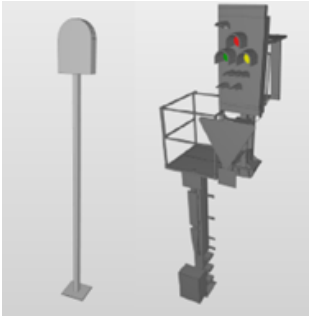

Figure 3: Different geometric representation for a signal: as a basic, atomic view (left) and as an assembly of many (potentially individual) construction parts (right) (DB Infra GO, 2025).

calling a specialized function, or as individual steps.

Infrastructure Networks and Data Driven Checks

Traditional vertical BIM models typically possess clear spatial boundaries and a clear assignment to a specific process phase. This also allows for rather simple structured quality assurance tests. One can safely assume that within one digital work unit (i.e. an IFC file) all instances of a specific asset category adhere to identical structure. For example, within one file the description of a window is the same as for all windows. The windows have the same structure (atomic vs. made of parts), the same property sets, the same material description and the same type of geometric representation. Therefore the methods for quality assurance of such homogeneous IFC models can be quite simple.

The assumptions above cannot be applied as-is to rail infrastructure, which has consequences for necessary capabilities of digital QA/QC methods. Contrary to the traditional vertical BIM models there are no clear spatial boundaries. Even greenfield projects connect somewhere to the existing network, making the boundary fuzzy with at least some existing elements being affected by the undertaking. Moreover, in mature networks most design and construction work deals with replacement and reconstruction of existing assets. Additionally, spatial boundaries defining areas of work tend to be different for different assets.

The second assumption of clear homogeneous process phase assignments is also questionable. Typical brownfield replacement and reconstruction activities in the rail domain take place during continued operation of the line or station. This again means that a specific asset category might be modelled with high granularity, whereas the neighbouring objects are absolutely sufficiently modelled using simplified structures and representations. Figure 3 presents a rail signal in both a sketchy and a very detailed geometric level of detail. A similar scenario could be found in a project with a road bridge residing next to a railway bridge, where – although both included in the IFC exchange – the former shall be modelled in far less detail than the latter.

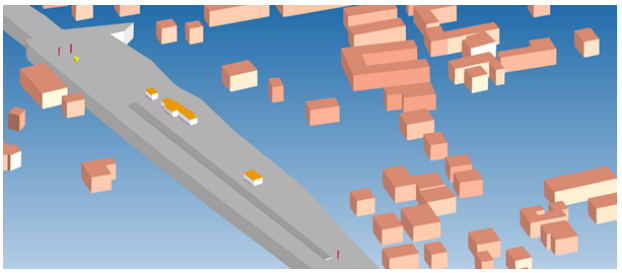

Figure 4: The scenery of the spatial-structure-centred example with the colour scheme following the requirements of the asset owner: white and orange are owned buildings, red are belonging to anybody else.

To explore requirement sets for data driven checks in diverse sceneries, we chose a spatial-structure-centred example shown in Figure 4. The model describes a railway station and its environment. It is expected that all surrounding buildings are included independent of ownership. Buildings belonging to the asset owner should be modelled with a detailed spatial structure using building storeys and spaces. The visualizations for these buildings is built on representations for IfcWall and IfcSlab entities in white as well as IfcRoof entities in orange. Buildings belonging to anybody else should be modelled without a detailed spatial structure as IfcBuilding entities and a visualization of their hulls in bricklike red colour is expected. To distinguish between these buildings we use a custom property named *ownedBy* with values *[Company, Subsidiary, Neighbour]*. The example can be modelled in IFC as presented in Table 4.

Comparison of the Requirements Models

This Section compares the IDS, mvdXML and bSI's Gherkin + Python possibilities for serialization, exploring the differences and similarities while focusing on the checking rules. An overview in presented in Table 5.

Governance

First and foremost, some non-technical statistics. Both IDS and mvdXML are bSI standards. Of the two, mvdXML is older, with the first commits on the development repository from 2013, while the IDS is younger, with the initial commit from 2020. mvdXML is currently in version 1.1, with version 1.2 in development, while IDS pub-

Table 3: IFC requirements for the aIfc4Rail example, together with denoted necessary facets and concept templates for successful serialization in IDS and mvdXML, respectively. The complete documentation of the facets and templates can be found in bSI (2024a) and in chapter 4 of ISO (2018), respectively. The colour of boxes on Figure 2 is provided for reference.

Requirement	Figure 2	IDS	mvdXML
There shall be an alignment IfcAlignment with name *0001R* and	blue	entity, attribute	Object User Identity
…a signal IfcSignal with name *Signal 01*.	blue	entity, attribute	Object User Identity
The alignment has an IfcGradientCurve as a geometric representation assigned.	red	✗	Product Representation
There is a station IfcReferent with predefined type STATION	blue	entity	Object Predefined Type
…which is linearly placed along the alignment's geometry at 69.395 m using IfcLinearPlacement.	green	✗	Product Linear Placement
The station is connected with the alignment using IfcRelPositions.	yellow	✗	Product Relative Positioning
The signal is placed relatively to the station using IfcLocalPlacement	green	✗	Product Relative Placement
…and connected to it with another IfcRelPositions relationship.	yellow	✗	Product Relative Positioning

Table 4: IFC requirements for the example from Figure 4, together with denoted necessary facets and concept templates for successful serialization in IDS and mvdXML, respectively. The complete documentation of the facets and concept templates can be found in bSI (2024a) and in chapter 4 of ISO (2018), respectively.

Requirement	IDS	mvdXML
Each IfcBuilding instance has an IfcProperySet named *Company Data* assigned, with an IfcPropertySingleValue named *ownedBy* included. It can have one of the three values *[Company, Subsidiary, Neighbour]*.	entity, property	Property Sets for Objects
Each IfcBuilding with *ownedBy* in *[Company, Subsidiary]*	entity, property	Property Sets for Objects
…and shall contain an IfcRoof element	✗	Spatial Container
…shall be decomposed in at least one IfcBuildingStorey element.	✗	Spatial Decomposition
Each such IfcBuildingStorey	✗	Spatial Composition with Property Sets for Objects
…shall contain an IfcSlab and at least one IfcWall.	✗	Spatial Container
Each IfcBuilding with *ownedBy* being *Neighbour*	entity, property	Property Sets for Objects
…shall not be decomposed into IfcBuildingStorey	✗	Spatial Decomposition
…and shall have a volumetric geometry assigned.	✗	Product Body Geometry

Table 5: Comparison of the three serializations at hand.

Criterion	IDS	mvdXML	Gherkin + Python
Governing entity	bSI	bSI	bSI
Development platform	GitHub	GitHub	GitHub
Checking scope	limited (Table 1)	full, no geometry	full
Controlled vocabulary	✓	✓	✗ (natural language)
Checking criteria	existence, values, (types)	existence, values, cardinality, types, uniqueness	anything
Combinations	AND	AND, OR, XOR, NAND, NOR, NXOR	AND, (OR)
Commercial product support	many	some	bSI only

lished version 1.0 with achieving final standard status in June 2024. Gherkin is managed and developed by SmartBear Software company, together with their commercial tool Cucumber. The idea can be traced back to the article about behaviour-driven development from North (2006). The implementation of bSI chose Python as the programming language executing the business rules in the background. IDS and Gherkin are still continuously developed further by the bSI technical team, while mvdXML has been shelved and archived.

Scope

The first version of IDS is a simple solution focused to cover the most wanted data requirements from the praxis as resulting from its requirement analysis. This can be observed in the limited number of facets from Table 1. Next to the intended usage for serialization of exchange requirements as presented by Tomczak et al. (2022), IDS files can be used to check the delivered IFC datasets against the rules themselves. There are already many commercial implementations with checking functionality available.

While IDS sets as a goal to encode level of information need (LOIN) as defined in ISO 7817 (Beyer et al., 2022), mvdXML addresses the needs for certification of IFC export and import within software products as well as providing the corresponding documentation. As such, mvdXML is used to encode the *General Usage* as well as other model view definitions (MVDs) in use by the industry (e.g. bSI, 2020). The standard has been successfully employed for model checking (Weise et al., 2016) and certification activities. Both IDS and mvdXML do not consider any requirement needing geometric calculations to be in scope.

Gherkin feature rules rely heavily on the Python implementation of the background logic, and can thus support any functionality that can be implemented by the programming language. Here, however, bSI is not only specifying the rules in Gherkin, but is implementing their parsing and execution by themselves as well. This is not the case for IDS or mvdXML, where bSI only governs and publishes schema specification and does not implement anything themselves.

Functional Blocks

The most important difference between the three standards is handling of different functional blocks. All three languages allow for the selection and the requirements sections to be specified, i.e. defining the applicability of a check (e.g. all IfcWall objects) as well as the checking rules (e.g. have to have a material assigned) independently from one another.

IDS foresees 6 predetermined facets (cf. Table 1), while mvdXML includes a mechanism to define the functional blocks within the data model itself. This is achieved with *ConceptTemplate*-s, which can arbitrarily define any subgraph within the data model, complete with constraints and predefined checking positions (cf. Algorithm 2). bSI (2020) predefines around 90 *ConceptTemplate*-s used throughout the specification.

The Gherkin rules have full flexibility of e.g. English language, however every specified rule needs to have corresponding Python implementation and account for different grammar peculiarities. Current bSI implementation offers around 70 different statements with corresponding code logic available. In the end, the rules can be defined by filling in the blanks in the IDS facets, or by filling in the predefined checking positions with rule grammar in mvdXML and Gherkin.

An important aspect of checking are also combinations of rules. On the one hand, this is still quite limited within IDS and Gherkin, where only a simultaneous agreement of defined rules is possible. That is, only the entity fulfilling *all* specified rules (i.e. rule1 AND rule2 AND …) is considered applicable or passing. On the other hand, mvdXML allows for many boolean operations between the rules (e.g. AND, OR, XOR, NAND, …) as showcased by Jaud et al. (2024).

Universality

On the one hand, IDS leaves a lot of interpretation options open when checking an IFC dataset. For example, the property set from Algorithm 1 may be fulfilled with either an IfcPropertySet or an IfcElementQuantity. Additionally, the property may be any of the subtypes of either the IfcProperty or the IfcPhysicalQuantity. There is no way of limiting or specifying which option is required in the current version of IDS. A similar problem occurs when considering the material facet – does a window with wooden frame fulfils the requirement of wood material? This might be perceived irrelevant to the more abstract requirement writer, but presents an additional burden on the implementers as well as a barrier for more advanced requirements.

Additionally, IDS interpretation of IFC schema is only written in the documentation, which can change without the change to the version of the serialization. Similarly, bSI's Gherkin rules are also only interpretable by looking through the associated implementation, which is currently only available in Python.

On the other hand, the interpretation of mvdXML is independent of its implementation. Its limitation is – similarly to IDS and Gherkin – that the checking algorithm needs to be aware of the IFC schema. However, the necessary graph to parse and check is provided within the mvdXML dataset itself, whereas this is hard-coded within the documentation and thus checking algorithms for IDS and Gherkin + Python implementation of bSI.

Conclusions

In this study, we focus on serialization possibilities of EIR which enable automatic compliance testing of IFC deliverables. We take a closer look at *IDS*, *mvdXML* and *Gherkin rules* with Python – the main encodings of quality assurance rules for IFC models governed by bSI. We compare their data models, philosophy and supported func-

Table 6: Recommendation for the AEC community as resulting from Tables 2, 3, 4 and 5.

For	Select
a limited LOIN (i.e. simple key-value pairs)	IDS
full control over IFC structure as well as individual values	mvdXML
geometric evaluations	Gherkin[1]

[1] If each rule is already implemented and validated.

tional blocks in Table 5. Our findings agree with those of Jaud et al. (2024) and additionally show that while there is some overlap between the three approaches, each of them addresses the serialization of EIR rules in its own specific way.

On the one hand, IDS is the youngest and simplest among them, with a (intentionally) very limited functionality as presented in Table 1. On the other hand, *Gherkin* – being a more general-purpose technology using natural language as input – first needs to be implemented in a programming language of choice, which results in a bespoke implementation as showcased in Algorithm 4 and 5. mvdXML provides the best trade-off between the flexibility of the data model to encode various requirements while retaining the benefits of a limited vocabulary for fast implementation in commercial workflows and products.

Additionally, we test the versatility of all three technologies on fictitious examples from the IDS documentation (bSI, 2024a) as well as on real world examples of EIRs from the the aIfc4Rail project and an European railway manager. We exemplary list the necessary functional blocks for encoding a fairly common EIR of a signal placed along an alignment as well as a differentiation in representation of surrounding buildings of interest.

Tables 2, 3, 4 and 5 support our recommendation for the AEC community as presented in Table 6. Another observation of our study is that the IDS seems like a lightweighted mvdXML, with a few hard coded functional blocks from the bSI (2020).

Future Work

In the future, our study can be continued with more thorough examples from the rail domain, or other domains within AEC community. For example, the IFC representation of the topology of the railway network and the fuzzy model boundaries could be explored in more detail. The comparison was limited to the three technologies as governed by bSI. These could be compared to technologies outside of the bSI community, e.g. STEP part 3x.

Additional research shall be also devoted to the important question: who checks the checker? That is, a sound and transparent process shall be established which ensures that various checker implementations are correct.

References

Beyer, J., Mellenthin Filardo, M., Borrmann, A., Beetz, J., Eckardt, S., Freund, H.-P., and Scherer, R. J. (2022). LOIN in der Anwendung: Whitepaper zur Beschreibung der Informationsbedarfstiefe (LOIN) nach DIN EN 17412 / ISO 7817-1.

bSI (2020). Industry Foundation Classes 4.0.2.1: Reference View 1.2. `https://standards.buildingsmart.org/MVD/RELEASE/IFC4/ADD2_TC1/RV1_2/HTML/` Accessed: 2024-02-15.

bSI (2024a). IDS Online Specification. `https://github.com/buildingSMART/IDS` Accessed: 2024-04-09.

bSI (2024b). IFC Gherkin Rules. `https://github.com/buildingSMART/ifc-gherkin-rules` Accessed: 2024-04-29.

bSI (2024c). mvdXML Online Specification. `https://github.com/buildingSMART/mvdXML` Accessed: 2024-04-09.

Cucumber (2024). Gherkin Online Specification. `https://cucumber.io/docs/gherkin/reference` Accessed: 2024-04-29.

DB Infra GO (2025). LST-Bauteilbibliothek. `https://www.dbinfrago.com/web/schienennetz/dienstleistende/planpro/lst-bauteilbibliothek-11161510` Accessed: 2025-01-09.

ISO (2018). ISO 16739-1:2018: Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries – Part 1: Data schema. Standard, International Organization for Standardization, Geneva, CH.

Jaud, Š., Weise, M., and Krischler, J. (2024). Information Management with IDS: Status, Potential Use Cases and Relationship to Other Specifications. In Proceedings of the 15th European Conference on Product & Process Modelling (ECPPM 2024).

North, D. T. (2006). Introducing BDD. `https://dannorth.net/introducing-bdd/` Accessed: 2024-04-29.

Tomczak, A., van Berlo, L., Bolpagni, M., Krijnen, T., and Borrmann, A. (2022). A review of methods to specify information requirements in digital construction projects. IOP Conference Series Earth and Environmental Science, 1101.

Weise, M., Nisbet, N., Liebich, T., and Benghi, C. (2016). IFC model checking based on mvdXML 1.1. In Christodoulou, S. and Scherer, R., editors, eWork and eBusiness in Architecture, Engineering and Construction – Proceedings of the 11th European Conference on Product and Process Modelling, ECPPM 2016.

Wynne, M. and Hellesoy, A. (2012). The Cucumber Book: Behaviour-Driven Development for Testers and Developers. The pragmatic programmers. Pragmatic Bookshelf, Dallas, Texas [u.a.].