2023 European Conference on Computing in Construction
40th International CIB W78 Conference
Heraklion, Crete, Greece
July 10-12, 2023

EXTENDING INFORMATION DELIVERY SPECIFICATION FOR LINKING DISTRIBUTED MODEL CHECKING SERVICES

Noemi Kremer[1], and Jakob Beetz[1]
[1]RWTH Aachen University, Aachen, Germany

Abstract

Since the amount and representation of required information in BIM is extensive, a possible extension of the Information Delivery Specification (IDS) format is presented. This extension is intended to offer the possibility to include further checking resources in the checking process, to perform geometric and topological in addition to semantic checks. The IDS extension uses dynamic links to integrate external calculation services into the checking process. The functions of the calculation service for concept validation were identified based on the requirements for simulation of building energy models. The result is a dynamic extension for requirements checking using the IDS format.

Introduction

Information management and information control are essential for the use of Building Information Models/ Modelling (BIM) in the Architecture, Engineering, and Construction (AEC) industry. A BIM can be structured and exchanged as an IFC data model with semantic properties that are mapped as properties with values in property sets. In addition, the IFC data model provides options to explicitly represent geometry-related properties and topological relationships (e.g. `IfcRelContainedInSpatialStructure` to assign elements to a specific level of the spatial project structure) (Zhang et al., 2018).

However, these options are not mandatory and at the same time, the IFC data model structure is too flexible to ensure consistent data. The example of the `IfcRelContainedInSpatialStructure` (buildingSMART International Ltd., 2020) relation outlines the flexibility since there is a possibility to assign an element to a certain level of the spatial project structure. Which level is relevant for which type depends on the context of the project and may vary. Thus, some geometry-related information is only implicitly included in the IFC data model. IFC data models can become inconsistent and geometry-related information impractical or impossible to map explicitly to the data model.

Geometric relations between building elements play an important role in model checking (Borrmann and Rank, 2009), since the required information in the IFC data model must be checked for accuracy and completeness. To perform automated model checks, information requirements are translated in the form of rules into a machine-readable format. These rule definitions range from simple, requiring single explicit data, to complex, requiring an extension to the data structure (Solihin and Eastman, 2015). To be able to perform rule-based model checking, various tools, and formats have been developed. These developments include commercial software tools such as Solibri Model Checker (SMC)[1] and Autodesk Navisworks[2].

However, both software tools are proprietary systems, hence their applications are less transparent. As a result, defined requirements can only be checked, and rule-saving files are only be exchanged between participants using these software tools or working within the software systems. For complete OpenBIM workflows, such formal definitions for the definition of exchange requirements are desirable to enable a reusable ecosystem of widely agreed specifications.

Current vendor-neutral formats which can be exchanged without a specific native software system are model view definition XML (mvdXML) (Weise et al., 2017) and Information Delivery Specification (IDS)[3]. Even though they both differ in their specific use case, explicit defined information can be checked by using both formats. However, they cannot be used to check implicit geometry-related properties and topological relationships. Currently, no available model-checking tool combines and performs all possible kinds of model checks (Zhang et al., 2018).

In this paper, we present an extension of the IDS structure to expand the capabilities of requesting and checking exchange requirements for model-based exchange. The IDS format becomes a single source of truth that drives and incorporates further checking resources. The extension approach enables a connection between geometric properties, topological relationships, and semantic model checking.

The proposed IDS extension approach has been applied and validated as part of the Moodle Model Check project. In this project, a model checker was developed to sup-

[1]https://www.solibri.com/news/solibri-model-checker-review
[2]https://www.autodesk.com/autodesk-university/class/Navisworks-and-classification-based-model-checking-2014
[3]https://technical.buildingsmart.org/projects/information-delivery-specification-ids/

port students in performing regular model-checking tasks on IFC. The model checker takes an IDS file and an IFC model data file as input and returns the results as a BIM Collaboration Format (BCF) file.

The checker is currently being developed and tested for specific use cases integrated into teaching. The first implemented use case is based on building energy simulation requirements. This use case requires checking explicit and implicit model information. Since an IDS can not provide for implicit model information checking, additional, external geometry-based computation services (functions) for model processing have been implemented. These services offer the calculation of geometry-related properties and topological relationships.

State of the art

Improving the various types of model checking is the subject of several research and development efforts. These efforts concern different formats and technologies.

Related work

Especially, the integration of checking and query options of geometric and topological properties is relevant for model checking. Borrmann and Rank (2009) for example, introduced the development of a spatial query language for BIMs enabling their spatial analysis. Part of the spatial query language basis is topological relationships between spatial objects. The relationships are reflected by topological operations calculating. This query language approach focuses on spatial relations and does not include further geometry calculations and extensions and combinations with semantic properties.

A more recent method for extending technology and including geometric and topological calculations is uses semantic web technology. Zhang et al. (2018) developed an approach for extending SPARQL[4] functions for querying IFC data. The functions are modelled as Resource Description Framework (RDF)[5] vocabularies and are partly based on topological relations. The introduced functional extension approach focuses on SPARQL, hence requiring knowledge about semantic web technology. Additionally, the IFC data model to be checked must first be converted to ifcOWL[6]. The authors point out that the knowledge engineering work required is still intensive for domain end users (Zhang et al., 2018).

Jiang et al. (2019) present an approach to integrate green construction code checking in construction processes. The researchers propose an approach for combining mvdXML and semantic web technologies to organise, store and reuse knowledge. Code checking classifies into 4 types based on the difficulty levels of the requirements. This approach combines two different technologies by converting mvdMXL into RDF data. However, according to the authors, parameters that are not explicitly predefined cannot be checked without additional implementations (Jiang et al., 2019).

Another approach presented by Werbrouck et al. (2019) investigates checking distributed linked data building models using the Shapes Constraint Language (SHACL) [7]. To perform a check, the project data has to be fetched and combined into a single graph and the SHACL pattern has to be read. Since the approach relies entirely on semantic web technology, other file formats must first be converted and mapped into ontology.

The research approaches are promising and focus on different approaches to expand and improve model-checking possibilities. However, including additional checking resources in the model checking processes, to perform semantic, topological, and geometric checks, has been less investigated.

Information delivery specification

The IDS format is designed for specifying and checking simple, non-geometrical required information in an IFC data model (buildingSMART International Ltd., 2012). this formatt can be used to define required information and check them, independently of native software format. According to buildingSMART International Ltd. (2012) the purpose of the design is to be a free, lightweight, and standardized approach for model checking.

To define and specify exchange requirements, an IDS consists of specifications. One specification consists of two main structural sections — applicability and requirements. The applicability part provides information to which an entity must comply to apply the specification. Within the requirements part, the information to be checked is defined. In other words, the applicability filters the IFC elements and which must contain and provide the information in the requirements section.

In the applicability and requirements elements, IDS facts are used for defining and describing information. An IDS facet is an element for describing the information that a single model entity (e.g. a wall or a door) can have. These information-defining IDS facets are: `attribute`, `property`, `material`, `entity`, `partOf` and `classification`. Multiple of these facets of information can be combined in either section (applicability or requirements). Thus, each IDS specification facet consists of zero or a combination of information-defining facets. An information-defining facet contains parameters describing its information, for example, name and value. The parameter type is defined as `idsValue`, which can either be a simple value (string data type) or a restriction (further defined in the IDS schema definition).

The IDS format relays strictly on XML markup language and uses IFC Schema modelling approaches, that have wide support in the software industry (Tomczak et al., nd). Geometrical requirements cannot be mapped in the IDS. However, some relations between IFC objects can be described, using the IDS `partOf` facet. One relation

[4]https://www.w3.org/TR/sparql11-query/

[5]https://www.w3.org/RDF/

[6]https://technical.buildingsmart.org/standards/ifc/ifc-formats/ifcowl/

[7]https://www.w3.org/TR/shacl/

is recorded per IDS `partOf` facet by specifying the type of relation (one of 4 predefined relations), as well as the entity. The entity refers to the larger IFC class to which an IFC object should belong. However, in the IDS for `IfcRelContainedInSpatialStructure` only the primary relationship of an IFC object is considered, and all other sub-relationships or assignments are not considered.

While the focus of IDS is on defining non-geometric exchange requirements, the main purpose of MVD is on software implementation certification(Weise et al., 2017). However, the combination of MVD and information delivery manual (IDM)[8] are the base for an information management system defined by buildingSmart (Weise et al., 2017). While the IDM contains the essential agreements on processes and responsibilities, the MVD defines the implementation details (Weise et al., 2017).

On top of the MVD, exchange requirements can be defined. MVDs are encoded in the mvdXML format, it allows defining values at particular attributes of particular data types (buildingSMART International Ltd., 2023). Since the introduction of mvdXML, the application has expanded from documenting an MVD in machine-readable format to information validation (Weise et al., 2017).

The structure of an mvdXML consists of a concept templates element that represents arbitrary IFC instance graphs and template rules, defined to constrain this graph pattern (Tomczak et al., nd). These template rules are connected to a specific concept template by referencing their identifier. Thus, the rule definition is closely linked to the IFC concept and leaves little scope for extensions outside this structure. In addition, the definition of checking rules in a mvdXML is built according to its grammar specification and therefore, independent of the XML structure. These checking rules are integrated into XMl as an attribute for an element (mvdXML element `TemplateRule`).

Both formats IDS and mvdXML do not map information dependencies between two different IFC objects and their properties. Furthermore, both formats are not designed for checking geometric properties and topological relationships. However, in comparison to mvdXMl, the existence of objects of an IFC class can be checked in an IDS. An IDS can be extended by adding more facets with XML-based methods. In the case of mvdXML, the specially defined grammar would have to be adapted, which does not provide any extension methods. In our approach, IDS 0.9 has been chosen for performing model checking due to its lightweight structure and the exclusively XML-based structure.

XLink and dynamic links

XML Linking Language (XLink) [9] is a World Wide Web Consortium (W3C) Recommendation, used to describe links between different resources. The use of XLink requires the definition of an additional namespace. The XLink syntax provides attribute references to further specify when and how the resource should appear.

`href` is the locator attribute in XLink. `XLink:href` specifies a locator (Uniform Resource Locator (URL)) in an XML file to find a remote resource. A URL does can be either static or dynamic implemented. The URL contains information about the location of the resource, the domain, and the path, as well as dynamic parameters that contain and transfer further information to the server.

These parameters are stored at the end of a dynamic URL. Each parameter can either be generated on the fly or prepared in advance. The use of parameters makes the URL more flexible and ensures that applications are adapted to the needs of the clients. Web pages are not stored on the server as a whole (cf. static URL) but are generated from server data and an application (database) when queried.

Concept and method

In this research, we have developed an approach to extend IDS in such a way that further checking resources can be integrated into the checking process.

IDS extension facet

As described in Section 2, an IDS can define how objects, classifications, properties, values, and units need to be delivered and exchanged. However, for some use case information requirements, the existing IDS options are not sufficient. Therefore, we introduce an IDS extension for integrating external functions for calculating implicit or difficult-to-access information from the IFC model, using the W3C Recommendation XLink.

Since external resources shall be integrated into the model-check routine, a suitable position in the IDS format must be found. A new IDS type is introduced called *extension*. The structure of this IDS extension is based on the structure of other IDS facets and contains parameters for further information definition. These parameters of an IDS extension facet are *resource* and *value*.

The resource parameter contains an XLink attribute `href` for referencing an external service resource. Thus, the IDS schema has to be extended to integrate the XLink namespace and a resource type definition. The value element consists of an `idsValue` type, that already exists in the IDS XML Schema Definition (XSD). The IDS extension attributes are the same as those of an IDS property facet.

The property's value is calculated externally in the extending geometric computation service. The value definition in IDS and the process of comparing the calculated value with the required value remain the same. This concept enables integrating functions for geometric-topological model checks into IDS non-geometric checks by extending the XSD in a standardised way.

XLink extension modelling

The IDS extension facet definition is used to introduce the XLink method for extending the original IDS file, as

[8] https://technical.buildingsmart.org/standards/information-delivery-manual/

[9] https://www.w3.org/TR/xlink11/

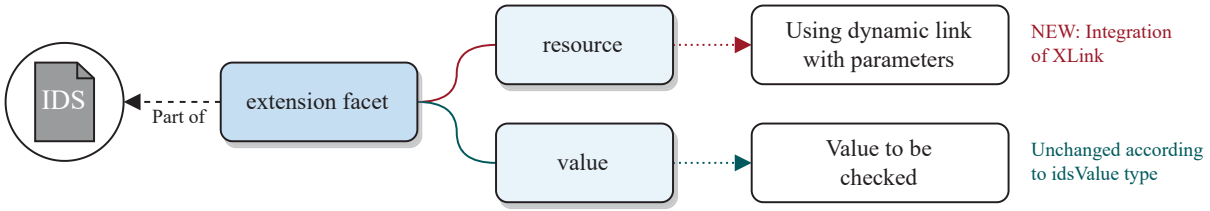

Figure 1: IDS property extension

shown in Figure 1. XLink provides with *href* (*xlink:href*) the dynamic URL for linking and addressing the additional resources. This resource can be an external calculation service that performs a specific calculation, that the original checking software cannot process. An example of an additional calculation for IDS is a function for processing geometry data.

The external resource requires an access point that can be represented by an interface implemented as an application programming interface (API). The API structure enables services to be made available locally for the computation of events. Implemented as representational state transfer API (RestAPI), they enable distributed service structures on different computers.

Listing 1: Example IDS property facet XML with text-based extension for an IfcWall entity

```
<ids:extension minOccurs="1" maxOccurs="1"
    measure="IfcBoolean">
  <ids:resource
  xlink:href="https://www.localhost.de/
      touches/ifcelementadd=IFCSITE&
      ifcelementori=1C_xd4xVP5R8Y8yHywtYQL"
      />
  <ids:value>
    <ids:simpleValue>True</ids:simpleValue>
  </ids:value>
</ids:extension>
```

Calculating functions are arranged behind the interface of an external resource. These functions require parameters and a data basis on which the calculation can be performed. These parameters including information and specifications for the external calculation service are provided and transferred as parameters in a dynamic URL. The data basis for calculation is the IFC data model, which is always required and passed when an interface function is called.

The possible parameters are determined depending on the structure of the functions implemented in the external calculation modules. A function that is supposed to calculate a relation between two IFC objects may need two different IFC classes or IFC objects for the calculation. The first IFC class can be defined in IDS. The other object or class must be defined additionally since the IDS specification allowed a maximum of one entity per IDS applicability/ requirements object. As the example in Listing 1 shows, these parameters are an IFC class additional (URL parameter *ifcelementadd=IFCSLAB*) and IFC class the original or the IFC object original (URL parameter *ifcelementori=1C_xd4xVP5R8Y8yHywtYQL*). Additionally, the IFC data file is passed, however not included in the URL parameters, since these parameters are text-based information, while an IFC file is its data model.

Functions do not have to be reinvented, existing functions can be adapted and reused. It depends on the use case which functions are needed, and which results are expected. A function called by a checker implementation also returns a value that is compared with the expected value in the IDS. The possibility of calculating a specific value depends on the function's return value. Thus, the return value of the selected function must be known by the user.

Implementation

For the implementation of the method, two core components must be present. On one side, the IDS reading and processing modules must be adapted or implemented, in our case, the ifctester tool. On the other side, the extending calculation services must be implemented and made available, in our case a local implementation of different functions.

Using IDS to verify IFC models

In the Moodle Model check project, IDS 0.9 has been chosen for performing model-checking due to the existence of an open-source model checker tool - ifctester[10]. An advantage of the application in the Moodle Model Check project is that the module is openly accessible and flexibly adaptable to the needs of our teaching-oriented model check. The checker consists of two parts: an XML schema checker and the IFC model checker. Checking results are returned to the console, and web page, as JSON and BCF file(Moult, 2022). Since IDS is an ongoing project, the ifctester tool is constantly being adapted to new developments.

The ifctester tool is used to implement the model-checking process. The tool performs the model-checking process using an IDS and an IFC model file (see Figure 2). The tool is openly available and can be flexibly adapted. However, it is still being developed and can therefore contain development-related problems.

The algorithm of the ifctester tool performs the model-checking process by iteratively filtering the information content of an IFC data file and comparing it with the requirements from the IDS. Thus, each IFC object corre-

[10]https://github.com/IfcOpenShell/IfcOpenShell/tree/v0.7.0/src/ifctester

sponding to the IFC class specified in the IDS entity facet is checked by comparing it and its properties and values with the information in the IDS. The IFC objects are therefore checked individually and consecutively for all IDS requirements. A basic ifctester model-checking process is carried out in 5 steps:

1. User passes the required data files: IDS and IFC model.
2. Checking for the IDS XML file to be according to the IDS XSD.
3. In case IDS applicability is given: Check for each IFC element, that corresponds to the information defined in the IDS applicability section. If it matches, the IFC object element is filtered out, otherwise, the next element is checked.
4. Check for each (filtered) IFC element whether it fulfils the required information in the Requirements section. If the requirements are not met by the IFC object, save an error message for output.
5. Check results can be returned as JSON and BCF ZIP formats.

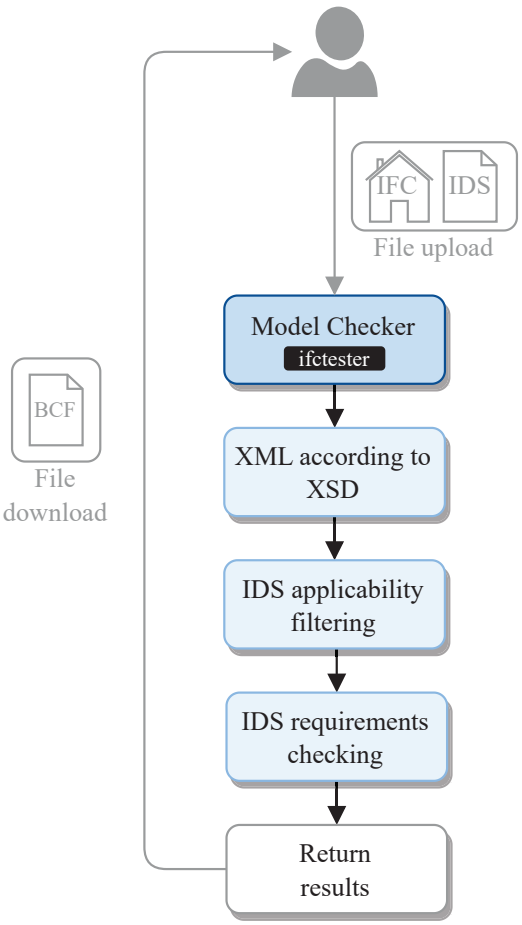

Figure 2: ifctester module checking procedure

Integrating the IDS extension

These five model-checking steps remain the same, regardless of whether an IDS file contains an extension. In case, the IDS file contains an extension, the initial checking process is interrupted, and an external calculation service is called, as shown in Figure 3.
The interruption leads to an external interface. The interface establishes the connection, it combines the ifctester module with the external calculation service, providing one or more functions (see Figure 3) and therefore, allows keeping the extensions flexible and independent.
The information that is calculated using an external calculation service and the result value that is returned, are specifically predefined through the structure of the implemented functions. The parameters used by the function for the calculation are stored in a dynamic URL. One parameter is the globally unique identifier (GUID) of the currently checked IFC object. Since the ifctester module processes one individual IFC object entity at a time, it pre-filters information from the model for the external calculation service. This GUID is extracted on-the-fly, during the model-checking process, and set as a dynamic URL parameter. The other parameter, the additional IFC class, is already embedded when the IDS file is generated. In addition to these parameters, the IFC model is passed.
The calculation result is returned to the ifctester module and compared to the given IDS property type value requirement. With the comparison between the calculated value and the value expected in the IDS, the check process implemented in the ifctester is continued, by moving on to the next requirement or the next IFC object.
Finally, all checking results are available as BCF file reports and JSON. Since model data is processed on-the-fly, no geometric or topological data is stored in the IDS file. In the BCF and JSON issue report, no distinction is made between the different checking methods.

Use case requirements

As part of the Moodle Model Check project, only use cases for tools and processes integrated into teaching at the RWTH Aachen were considered for practical implementation. One of these tools is Enercalc [11]. The Enercalc tool balances the energy demand for a building based on DIN v 18599 [12].
Enercalc is suitable as a use case since it has several useful features. The tool is free for use in teaching, is still being developed, and is therefore not abandoned, easy to use, as calculations have been partially simplified. In addition, the number of necessary information can be limited to a manageable level through simplified calculations.
This required information is (Every other value can be assumed):

- Outer surface of the façade (including walls) and windows for every geographical direction

[11] https://ingefo.de/Werkzeuge/EnerCalC/
[12] https://www.din.de/de/mitwirken/normenausschuesse/nabau/auslegungendinv18599-68632

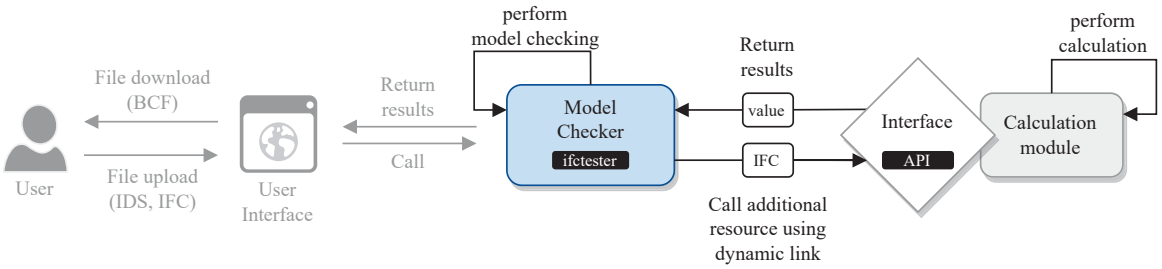

Figure 3: Model check process with extension

- Outer surface of roof and slab (against unheated earth)
- General building volume
- Outer surface of walls against unheated/ earth
- Profile of use per building zone
- Net floor space per building zone
- Zone height

Most predefined values are based on average assumptions and calculations. Some required values, for example, standards, are already predefined and are not to be changed by the user. A few values are not pre-filled and must be supplemented by properties of the IFC data model, such as façade areas for different geographic directions.

The properties and their values were categorised in explicitly and implicitly available model data. Most of the Enercalc requirements are values for properties that can be recorded in a property set as explicit information. A few values for properties considering topological relationships and geometric properties are only implicitly available in the IFC data model. These values have to be calculated. For the values that are implicit or cannot be checked with IDS, corresponding functions (compare Table 1) were implemented. These functions are generalized to be reusable.

Table 1: Implemented functions for the Enercalc use case

Function name	Description
hasOrientation	Calculate the geographical direction according to the entity's placement in the project.
touches	Identify connection relationships between building elements. (Zhang et al., 2018)
externalSurface	Calculate the external surface area without openings.
buildingVolume	Calculate volume of the hole building.

A specification in an IDS for Enercalc states that there should be no glass facade areas facing south that are larger than a certain area. To check this, entities were assumed in the applicabilities `IfcCurtainWall` that have the properties `IsExternal` and the orientation south. `IsExternal` is an explicit property and the orientation is an implicit property that must be calculated. Another implicit property is in the requirements of the IDS specification. The requirements specify the maximum size of the façade area that may not be exceeded.

Another specification checks whether walls abut against the unheated ground. For this specification, the IDS prefilters for exterior walls in the applicability part and checks the filtered walls for contact with the unheated ground in the requirements. It is calculated whether an `IfcWall` element touches an `IfcSite` element.

Implementation of extended calculation modules

In this research, only services running locally are implemented and tested. Therefore, no RestAPI has been used. All services are implemented as Python functions based on the Topologic tool[13].

Topologic is an open-source software modelling library, it enables the topological and hierarchical representation of architectural components based on non-manifold topology (NMT). The functions implemented with Topologic are based on the requirements of the Enercalc tool, as part of the building energy model simulations.

An IFC model can be converted to a Topologic tool class: vertex, edge. Face, cell, and cellcomplex. Other Topologic tool classes are not relevant to the project objective and are not considered. Before the IFC file is converted, the data is filtered to reduce the computational effort. The GUID parameter is important, as a calculation only needs to take place for this one IFC object.

In the implemented Enercalc use case, the touch function requires an additional IFC class definition. This function is required to check if two elements are touching. In the Enercalc IDS, all walls are checked to see whether they touch an unheated surface. Consequently, for each `IfcWall` object labelled `IsExternal` it has to be checked if it touches an IfcSite object representing an exterior surface. To keep the functions as generic as possible, this second IFC class element is additionally specified in the IDS file and not implemented in the function.

A called function returns a specific value type according to the required calculated result type. *HasDirection* for

[13]https://topologic.app/software/

example returns a string, while *ExternalSurface* returns a float value. Since the implemented functions focus on requirements of the Enercalc tool, particularly exterior walls, and model zones are processed.

In the case of Enercalc, the model checker first filters for building elements (walls, windows, slabs, roofs) which are labelled *IsExternal*. The filtered wall elements are then further filtered for the applicability definition of geographical filtering for a specific direction. Since the outer surface function is used within requirements, only wall elements facing a specific direction have to fulfil these requirements. In addition, external wall objects that touch the unheated ground are filtered if they are above or under a specific outside surface area. Since geometry-based checks were included in the model-checking process, all exchange requirements for the Enercalc use case can be integrated and checked using an IDS file.

Discussion and result analysis

The practical implementation already shows some advantages and disadvantages, even if only one use case was used for the testing. In the long run, more use cases and different services for calculations are needed.

XLink extension

The XLink-based extension is generic since extending functions can be provided on web servers for different applications. The XLink syntax allows further specification of the resources to be linked. However, the IDS format has to be extended to define XLink as an attribute of an XML element. When using dynamic URL addresses for extension, possible security risks must be considered during implementation, since the checking tool does not know the called extending service. The development of IDS is a continuing process, future IDS versions may contain extension facets.

Function evaluation

The implemented functions use topological data structures. These structures offer further possibilities for information retrieval and processing. The Topologic library proved to be suitable for the implementation of extending Python functions. However, Topologic is only one possibility, there are a variety of other tools and implementation options.

The implemented functions for Enercalc are independent of other functions. Dependencies between two functions can cause issues, as dependency leads to structural confusion. Multi-depending functions could lead to non-transparent and highly complicated structures. The waiver of dependencies corresponds to the IDS definition of requirement content.

Extending an IDS file using functions shifts process information and requirement definition to the implementation background. The actual information processing remains a black box. The checking process becomes less transparent as a function name is given instead of explicit information, traceable in an IFC model. Contractors may not be able to identify the required information by reading the IDS file. To clarify the requirements, the extension should be described. Simplification of the extension may lead to not having enough information for other participants to know what information to provide.

Implementation results

During the validation of the model-checking tool, the checking performance depends on the handling of vertices, edges, and faces. The number of faces should be reduced to the needed ones, before performing further calculations. The face reduction implies a reduction of processing time when running a model check.

The extension with XLink in the ifctester has a clear disadvantage in that an extending external resource is called for each IFC object to be checked. This is not noticeable in small models, but in larger models with many individual IFC objects, this external resource is called for each one. This can make the checking algorithm inefficient. To realise more extensive check calculations other algorithms must be implemented, for example, to collect IFC objects for joint processing in functions.

The fact that only one IFC object element is checked at a time has further disadvantages for the calculation speed. More extensive models require significantly more time, since, for example, the additional external function had to be called for each IFC object of the IfcWall class.

One possibility for scalability of this approach is to call external resources not for each IFC object, but for IFC classes. This significantly reduces the number of calls to external resources, but more calculation content is outsourced from the ifctester tool. It would be no longer a value that is returned, but the IFC objects that did not pass the test.

The model checker has been developed based on the use case Enercalc, only four required extending functions have been identified in developing the Enercalc IDS file. The extending function implementation depends on the person performing model-checking, therefore, a function can be very specific to a problem. Alternatively, a function can be implemented generally. This leads to an undefined amount of possible functions. Furthermore, different IFC entities utilize the same functions. For example, the IFC entities `IfcWall`, `IfcSlab`, and `IfcRoof` all use the function calculating the outer wall surface. Therefore, a function implementation should not contain any predefined IFC element in its code structure.

Conclusions

This paper presents an approach for extending IDS by introducing a new IDS facet containing XLink. The extension allows performing geometric and topological in addition to semantic checks in model-checking procedures.

The purpose of the extension presented is to link different types of model checking to make the functions of the IDS model checker more versatile. The implemented IDS

model checker extension was verified by performing several testing iterations. We checked the IFC data model content according to the Enercalc information requirements. However, further use cases are required and will reveal further options for improvements and expansions. Especially concerning the scalability and structure of extension options and function implementations.

For some required information, the question arises as to what extent it is explicit or implicit information. The ability to check implicit information outlines the possibility of reducing explicit IFC data model content. However, the question arises, of how functions are implemented that check this implicit, geometric properties and topological relationship information. The type of modelling plays a decisive role in the classification of the information. Further research is needed to determine what and how information is best represented in an IFC data model.

The lack of generally applicable requirements for the additional functions, their implementation, and their use leads to a very individual application. In future research, these functions and their use in model checking must be further reviewed and structured. However, geometric and topological calculations enable more comprehensive open-source-based checking and have the potential to strengthen the position of open formats among model-checking tools.

Model checks could be controlled and managed more centrally and requirements formulated more coherently. This way, dependencies verification between different information requirements, and model checks can be made more user-friendly. Connecting various model-checking tools depending on the properties to be checked makes it possible to optimally use the focus properties of different software in rule-based checking. To implement this, more research in the area of distributed model checking services, their implementation, and their exchange possibilities is necessary. Additionally, the exchange possibilities should be expanded between existing model-checking tools and become more transparent.

Acknowledgments

We would like to thank Dr. Markus Lichtmeß for providing us with Enercalc access.

References

Borrmann, A. and Rank, E. (2009). Topological analysis of 3D Building Models using a Spatial Query Language. Advanced Engineering Informatics, 23:370–385.

buildingSMART International Ltd. (2012). Information Delivery Specification IDS. https://technical.buildingsmart.org/projects/information-delivery-specification-ids/ [Accessed: 2022-11-07].

buildingSMART International Ltd. (2020). IfcRelContainedInSpatialStructure. https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD1/HTML/link/ifcrelcontainedinspatialstructure.htm [Accessed: 2023-04-15].

buildingSMART International Ltd. (2023). mvdXML. https://technical.buildingsmart.org/standards/ifc/mvd/mvdxml/ [Accessed: 2023-04-15].

Jiang, S., Wu, Z., Zhang, B., and Cha, H. S. (2019). Combined MvdXML and Semantic Technologies for Green Construction Code Checking. Applied Sciences, 9(7):1463.

Moult (2022). Information Delivery Specification IDS. https://github.com/IfcOpenShell/IfcOpenShell/tree/v0.7.0/src/ifctester/ifctester [Accessed: 2022-11-18].

Solihin, W. and Eastman, C. M. (2015). Classification of rules for automated BIM rule checking development. Automation in Construction, 53:69–82.

Tomczak, A., van Berlo, L., and Borrmann, A. (n.d.). A review of methods to specify information requirements in digital construction projects. In n.b., page n.p. cib w78.

Weise, M., Liebich, T., Nisbet, N., and Benghi, C. (2017). IFC model checking based on mvdXML 1.1. In eWork and eBusiness in Architecture, Engineering and Construction, pages 19–26. Mahdavi, Ardeshir and Martens, Bob and Scherer, Raimar.

Werbrouck, J., Senthilvel, M., Beetz, J., and Pauwels, P. (2019). A Checking Approach for Distributed Building Data. In 31st Forum Bauinformatik, Berlin: Universitätsverlag der TU Berlin, pages 173–181.

Zhang, C., Beetz, J., and de Vries, B. (2018). BimSPARQL: Domain-specific functional SPARQL extensions for querying RDF building data. Semantic Web, 9:829–855.