Semantic bSDD: Improving the GraphQL, JSON and RDF Representations of buildingSmart Data Dictionary

Vladimir Alexiev[1,*], Mihail Radkov[1] and Nataliya Keberle[1]

[1] *Ontotext, 79, Nikola Gabrovski Str. Twins Centre, fl.3, Sofia 1700, Bulgaria*

Abstract

The buildingSmart Data Dictionary (bSDD) is an important shared resource in the Architecture, Engineering, Construction, and Operations (AECO) domain. It is a collection of datasets ("domains") that define various classifications (objects representing building components, products, and materials), their properties, allowed values, etc. bSDD defines a GraphQL API, as well as REST APIs that return JSON and RDF representations. This improves the interoperability of bSDD and its easier deployment in architectural Computer Aided Design (CAD) and other AECO software.

However, bSDD data is not structured as well as possible, and data retrieved via different APIs is not identical in content and structure. This lowers bSDD data quality, usability and trust.

We conduct a thorough comparison and analysis of bSDD data related to fulfillment of FAIR (findable, accessible, interoperable, and reusable) principles. Based on this analysis, we suggest enhancements to make bSDD data better structured and more FAIR.

We implement many of the suggestions by refactoring the original data to make it better structured/interconnected, and more "semantic". We provide a SPARQL endpoint using Ontotext GraphDB, and GraphQL endpoint using Ontotext Platform Semantic Objects. Our detailed work is available at https://github.com/Accord-Project/bsdd (open source) and https://bsdd.ontotext.com (home page, schemas, data, sample queries).

Keywords

Linked building data, buildingSMART Data Dictionary, FAIR data

11th Linked Data in Architecture and Construction Workshop, 15–16 June 2023, Matera, Italy
*Corresponding author.
vladimir.alexiev@ontotext.com (V. Alexiev); mihail.radkov@ontotext.com (M. Radkov); nataliya.keberle@ontotext.com (N. Keberle)
0000-0001-7508-7428 (V. Alexiev); 0000-0001-7398-3464 (N. Keberle)

Introduction

bSDD

Reusable data dictionaries are widely used for the electronic exchange of product and component information across industries, improving interoperation between systems. Examples include:

- IEC Common Data Dictionary (IEC CDD): electrical components, units of measure, documents and certificates, etc.
- eCl@ss: a product classification and parts description for a variety of industries.
- ISO 15926 part 4 Reference Data and Services: for digital information across process plant industries (oil & gas).
- buildingSMART Data Dictionary (bSDD): for materials and components in the AECO industry.

The bSDD is a hierarchical dictionary of object concepts (Classifications), their Properties and allowed values used in Building Information Models (BIM). Property sets are predefined by regulation agencies and vendors and extend common property sets of the Industry Foundation Classes (IFC). This allows to describe specific domains (e.g. transportation) and building elements (e.g. doors, windows, stairs). bSDD is organized according to the ISO 23386 (2020) [1] Methodology to describe, author and maintain properties in interconnected data dictionaries. This is a language-independent model used for the development of dictionaries according to ISO 12006-3 (2022) [2] Framework for object-oriented information.

bSDD was initiated to support improved interoperability in the building and construction industry. Palos [3] noted that bSDD is a very comprehensive solution aiming at the provisioning of open complementary product data definitions, identification, and distribution methods.

As of February 2023, bSDD contains descriptions of nearly 80,000 Classifications in 108 domains, ranging from roads and rails to DIN, Omniclass, Uniclass, IFC extensions, etc. It is a widely accepted source of BIM reference data. bSDD uses URLs for nearly all defined entities to enable globalized data use in a variety of AECO applications and structured documents.

Related Work

According to buildingSMART technical roadmap, bSDD service provides output data in various formats and APIs, including RDF, thus making bSDD content higher reusable in the Linked Data ecosystem, particularly with geographical data, regulations, product manufacturer data. However, Pauwels et al. in [4] note that there is no standard method to generate RDF graphs from bSDD API. Starting from [5] where “bSDD vocabulary

has been transformed into a configurable RDF dataset. On the meta-model level several different modelling approaches ranging from OWL to RDFS and SKOS have been implemented to evaluate the advantages and disadvantages of the respective modelling strategies" the work was continued. In [6] a complete translation of IFC from EXPRESS to OWL is presented, however lacking modularity and extensibility inherent to best Semantic Web practices [7]. In [8] it is proposed and implemented to generate from bSDD an OWL representation of a selected IFC element together with its property sets on the fly.

In the survey [9] discussed the question of where and how the bSDD can fit in the Linked building data (LBD) ecosystem. Authors mention that bSDD has undertaken a new round of development showing a shift towards publishing data classifications and properties as Linked Data.

Admitting the advances of bSDD community at providing data in RDF format we met some issues where accommodating these data for our purposes in the frame of ACCORD project, among them are different results obtained with different APIs, multiple URIs for same entities, various GraphQL implementation errors. In this paper, we discuss these issues and propose a set of technical improvements following the best practices of the Semantic Web and linked data to obtain "a semantically better version" of bSDD. We implemented and made available our solution using the Ontotext Platform.

Unlike [8], we preserve the original bSDD structure (Domain, Classification, Property, etc) and only add specific improvements, described below, and convert the whole bSDD at once, taking more attention to outlining the discrepancies of the current solution and bringing the Semantic Web best practices and FAIR principles.

GraphQL Benefits

GraphQL is an approach to create simplified "facades" over various storages, and to provide schema, uniform query language, API and runtime for handling queries, mutations and subscriptions. It has many benefits over traditional REST APIs:

- Avoid over-fetching by specifying exactly which data and in what nested structure should be returned by the server
- Data is returned in JSON that is precisely congruent to the shape of the query
- Retrieve many resources in a single request; even across storages by using GraphQL Federation
- Schema introspection that allows IDEs and query helpers to offer contextual auto-completion at any point in the query
- Data validation (for both input through mutations and output through queries) that guarantees type and cardinality conformance (optional/mandatory, single/multi-valued)

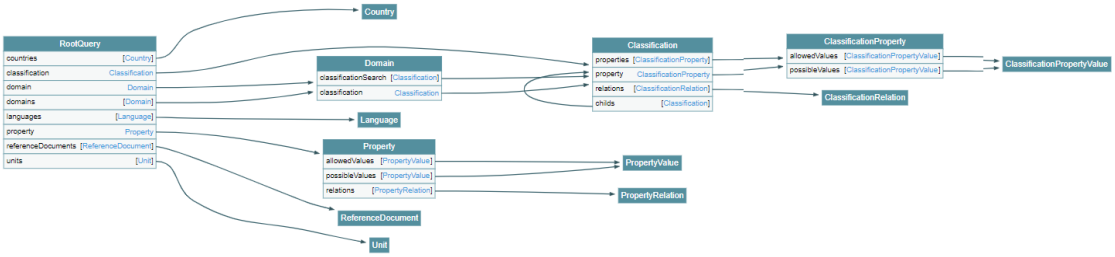


Figure 1: Original bSDD GraphQL Schema: Overview (uncheck "Show leaf fields")

bSDD does offer GraphQL access, moreover, the endpoints use `GraphiQL` - a graphical interactive in-browser GraphQL IDE.:

- Test: https://test.bsdd.buildingsmart.org/graphiql/
- Production: https://api.bsdd.buildingsmart.org/graphqls/ (secured endpoint).
- NOTE: we worked with bSI to get access to the production endpoint, but due to delays all our analysis is done on data from the test endpoint. Nevertheless, we believe that most of our findings also apply to the production data.

Original GraphQL bSDD Schema: Voyager

GraphQL Voyager is a visual application that uses a Schema Introspection query to explore a GraphQL endpoint and displays the schema of the endpoint, allowing the user to search and browse the available types and queries. We used Voyager over the bSDD GraphQL endpoint to investigate the original bSDD schema (see Fig. 1):

As we can see, bSDD has 12 entities (object types):

- Reference entities:
  - `Country`
  - `Language`
  - `ReferenceDocument`, such as a standard
  - `Unit`: unit of measure
- `Domain`: dataset by a single data provider
- `Property`: global property definition
  - `PropertyRelation`: relation between properties
  - `PropertyValue`: allowed property value for enumerated properties
- `Classification`: object, material, component
  - `ClassificationRelation`: relation between classifications
- `ClassificationProperty`: property that is localized to a classification

– `ClassificationPropertyValue`: allowed property value for enumerated properties

Currently bSDD contains 31720 `Classifications`, 111556 `ClassificationProperties`, 214121 `ClassificationPropertyValues` and `PropertyValues` together, because essentially they are the same by structure, 6420 `ClassificationRelations`, 36069 `Properties`, 603 `Units`, 484 `ReferenceDocuments`, 39 `Languages`, 246 `Countries`, 108 `Domains`.

Original GraphQL bSDD Schema: Problems

Even in the Schema Overview (at low level of detail) we can notice some defects:

- The reference entities (`Country, Language, ReferenceDocument, Unit`) are disconnected from the rest of the schema, i.e. not used by the other entities
- Relation entities have only an incoming link but no outgoing link. This means that if you want to get some data of a `Classification` and all its related `Classifications`, you need to issue two queries because you cannot navigate past `ClassificationRelation`.
- Many entities cannot be queried directly from the root, but have to be reached through their respective "parent" entity.
- There are no backward relations (arrows) to get from a lower-level entity back to its "parent" entity.
- There are a number of parallel relations (arrows). This is not needed in GraphQL because the schema can use parameters to distinguish between the different uses.

At the high level of detail we can notice more defects:

- `Property` and `ClassificationProperty` are very similar, but there's no inheritance/relation between them
- `PropertyValue` and `ClassificationPropertyValue` are exactly the same, so can be reduced to one entity

Refactored GraphQL bSDD Schema: Voyager

The main purpose of this work is to refactor the bSDD data and schema in order to improve them. The results of our work are put into a web page bsdd-graphql-voyager-refact.html (see Fig. 2) that allows to explore the refactored schema.

The proposed improvements are following:

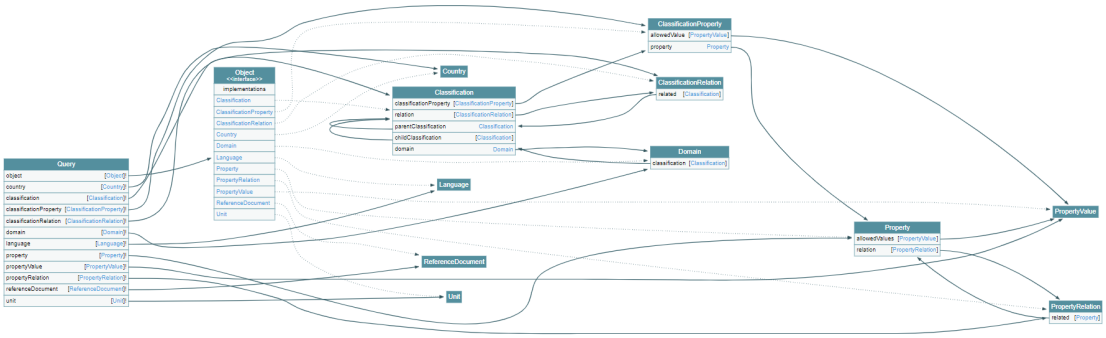


Figure 2: Refactored bSDD GraphQL Schema: Overview (uncheck "Show leaf fields")

- All entities are queryable directly from the root. Note: There's a common interface `Object` that provides functionality common to all entities: the dashed arrows show that each entity implements it. This creates some clutter in the diagram, but doesn't complicate querying and navigation.
- There are no parallel arrows (relations) between entities; each relation is named the same as the target entity, improving predictability and consistency.
- Navigation between entities is bidirectional (e.g. `Domain.classification` but also `Classification.domain`), which is a feature expected of a Knowledge Graph.
  - In particular, the `Classification` hierarchy can be navigated both up and down (`parentClassification, childClassification`)
- A query can traverse a `Relation` entity to get data about the related entity:
  - `Classification.relation -> ClassificationRelation.related -> Classification`
  - `Property.relation -> PropertyRelation.related -> Property`
- A single entity `PropertyValue` is used by both `Property` and `ClassificationProperty`

The solution proposed does not fix all defects noted in the original diagram. The reference entities are still not used by the main entities. To fix that it would require data cleaning work (e.g. to ensure that Unit code strings used in all Properties and ClassificationProperties are in the reference list).

GraphiQL Querying of Original Endpoint

https://test.bsdd.buildingsmart.org/graphiql is the original GraphQL endpoint.

It provides a number of useful features (see Fig. 3):

- Online searchable documentation of the GraphQL schema

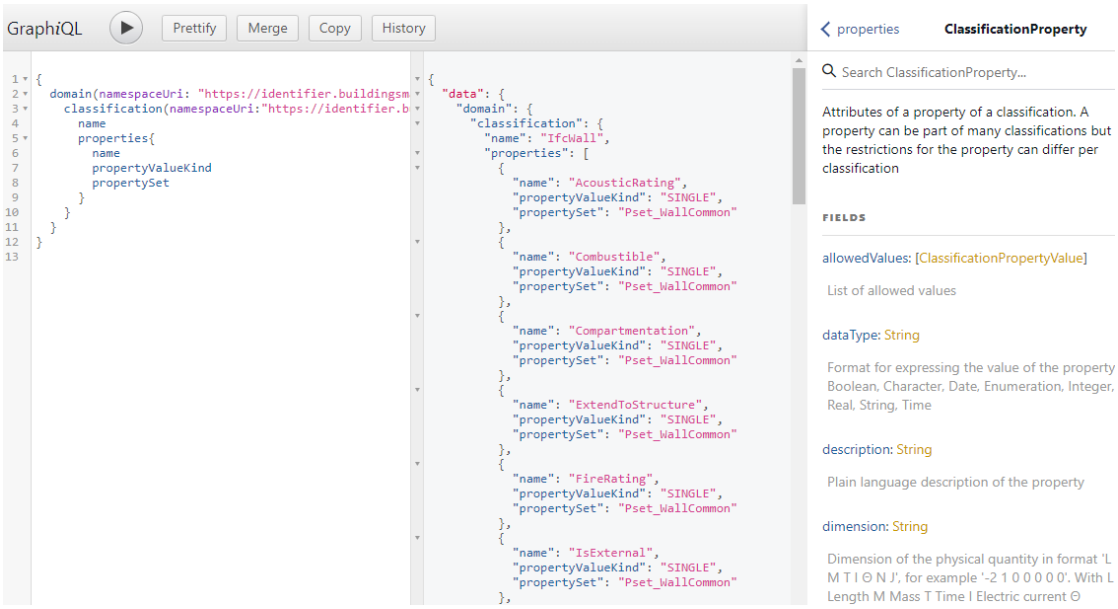


Figure 3: GraphiQL Querying of Original bSDD Endpoint

- Auto-completion of field names and parameters at any point in the query: queries practically "write themselves"!
- Ability to parameterize queries through Query Variables
- Code formatting (Prettifying) of the query
- Syntax highlighting
- History of previous queries
- JSON results that conform exactly to the form of hate query

GraphiQL Querying of Refactored Endpoint

https://bsdd.ontotext.com/graphiql/ is the refactored GraphQL endpoint:

We have deployed a newer version of GraphiQL that has all benefits described in the previous section, and adds some more (see Fig. 4):

- A hierarchical Explorer pane that shows the total schema structure and allows you to select fields by clicking rather than typing. The History and Documentation panes are still present (see toggles at the left edge)
- Useful keyboard shortcuts
- Search in the query text (in addition to search in the Documentation)
- Improved syntax highlighting
- Multiple query tabs so you can easily access several queries at once
- The query response reports errors in addition to returning data (this comes from our GraphQL server implementation, not from the GraphiQL version)

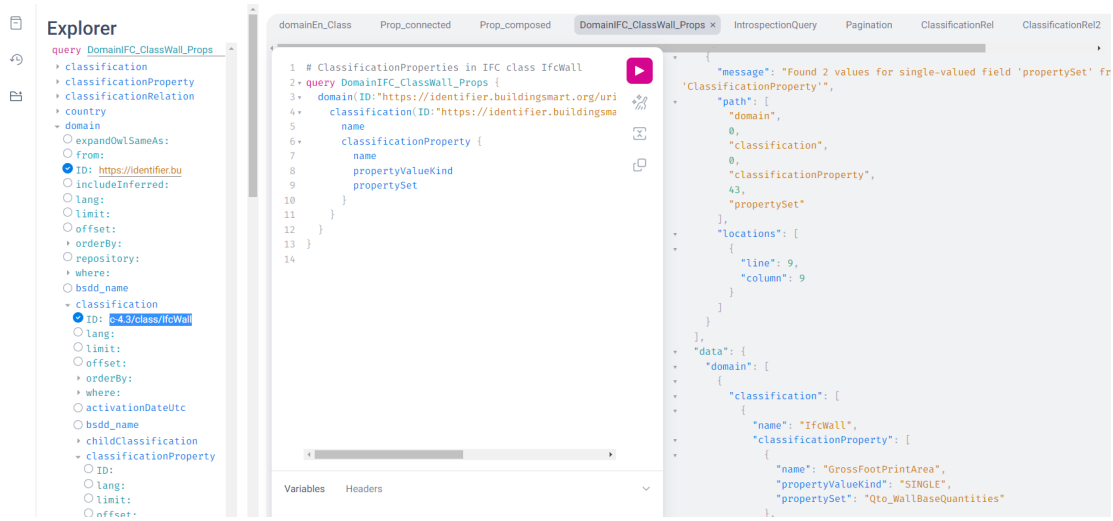


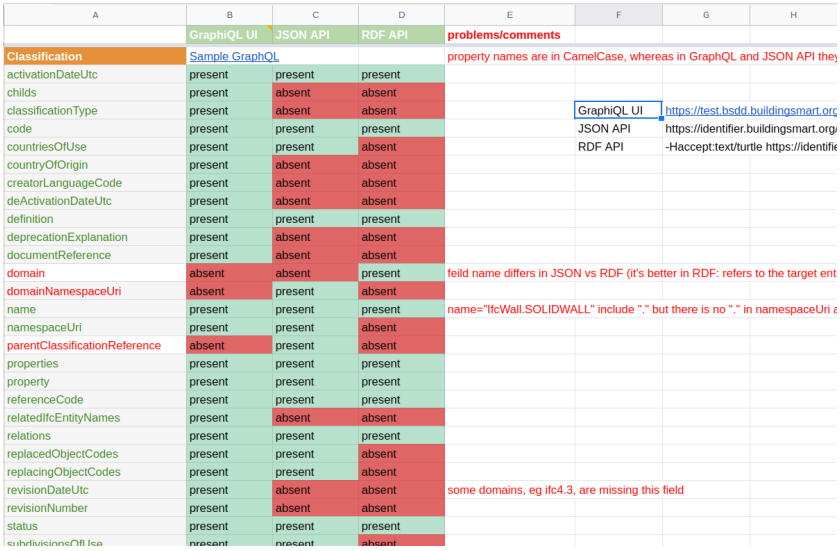
Figure 4: GraphiQL for the Refactored Endpoint

Suggested Improvements

In this section we analyze shortcomings of the original bSDD data structure, and suggest improvements. We briefly present three groups of shortcomings - representational, modelling, and GraphQL implementation related, highlighting the most interesting.

Representational Shortcomings

- **Return the same data across APIs**. We have compared three representations returned by the bSDD server: JSON from the GraphQL API, JSON from the REST (entity) API, and RDF from the REST (entity) API. We selected entities of each class that have the maximum number of filled fields, and compared the results returned by each API. We found a number of detailed differences, as presented in the bSDD data analysis spreadsheet (see Fig. 5)
- **Improve Property Names**. Property names should conform to naming conventions and be spelled consistently. Property (field) names should be spelled in singular, even when they refer to an array. The arity is reflected in the property kind. The GraphQL and JSON field `childs` should be spelled properly as `children`. RDF properties should conform to the `lowerCamelCase` convention. `namespaceUri` is a misnomer since "namespace" means a set of URIs sharing the same prefix, but most bSDD URIs are **single** URIs. RDF properties should use one consistent namespace. Most of them use `bsdd: <http://bsdd.buildingsmart.org/def#>`, except `hasReference`, which uses a different namespace: `<http://bsdd.buildingsmart.org/relation/def#>`.



	GraphiQL UI	JSON API	RDF API	problems/comments			
Classification	Sample GraphQL			property names are in CamelCase, whereas in GraphQL and JSON API they			
activationDateUtc	present	present	present				
childs	present	absent	absent				
classificationType	present	absent	absent		GraphiQL UI	https://test.bsdd.buildingsmart.org	
code	present	present	present		JSON API	https://identifier.buildingsmart.org/	
countriesOfUse	present	present	absent		RDF API	-Haccept:text/turtle https://identifie	
countryOfOrigin	present	absent	absent				
creatorLanguageCode	present	absent	absent				
deActivationDateUtc	present	absent	absent				
definition	present	present	present				
deprecationExplanation	present	absent	absent				
documentReference	present	absent	absent				
domain	absent	absent	present	feild name differs in JSON vs RDF (it's better in RDF: refers to the target ent			
domainNamespaceUri	absent	present	absent				
name	present	present	present	name="IfcWall.SOLIDWALL" include "." but there is no "." in namespaceUri a			
namespaceUri	present	present	absent				
parentClassificationReference	absent	present	absent				
properties	present	present	present				
property	present	present	present				
referenceCode	present	present	present				
relatedIfcEntityNames	present	absent	absent				
relations	present	present	present				
replacedObjectCodes	present	present	absent				
replacingObjectCodes	present	present	absent				
revisionDateUtc	present	absent	absent	some domains, eg ifc4.3, are missing this field			
revisionNumber	present	absent	absent				
status	present	present	present				
subdivisionsOfUse	present	present	absent				

Figure 5: Analysis of bSDD data returned with different APIs

- **Use the Same URL for Data and for Web Pages**. bSDD has implemented "entity URLs", i.e. for each kind of entity it can return its data in JSON or RDF. The same URL can be used to get a static web page in the browser. However, the interactive bSDD Search UI uses a different URL that returns slightly different information. There is not really a need for two different web pages showing nearly the same info.
- **Improve URL Structure and Consistency**. To facilitate the accessibility of digital artifacts available from bSDD, their URLs should be designed uniformly according to Linked Data Principles. Recommendations on ontology URI design, including versioning and opaque URIs to maintain evolution and multilingualism inherent to bSDD, are described at [7]. Almost all domain URLs have the same structure: `https://identifier.buildingsmart.org/uri/<org>/<domain>-<version>`. The Linked Data Patterns book describes a pattern of Hierarchical URIs, that make URLs more "hackable", allowing users to navigate the hierarchy by pruning the URI. bSDD URLs could become more hierarchical if they **all** follow a structure `https://identifier.buildingsmart.org/uri/<org>/<domain>/<version>` which is not a case now. bSDD uses dash not slash to separate the version, in some cases, the `<org>` is repeated in the `<domain>` part, in some cases, the `<org>` name doesn't quite mesh with the domain name, perhaps due to the way bSDD allocates `<org>` identifiers to bSDD contributors.

We recommend also to **explicate domain versions**, to **declare URLs to be** `ID` and use a mandatory field `id`, to **remove the overlap of Entity Classes with** `classificationType`, to **disambiguate URLs for** `Property` and `ClassificationProperty`, to **provide an URL for all the Entities** in the bSDD schema.

Modelling Issues

In addition to the technical recommendations above (to ease findability and accessibility of data in bSDD by improving URls), we have noticed several modelling issues:

- **Unify different solutions in modelling of Complex Properties**: the bSDD data model allows the modelling of complex properties that are composed of other properties, the key attribute `propertyValueKind` has values COMPLEX and COMPLEX_LIST used in combination with `connectedProperties`. These key values are defined for `Property` and `ClassificationProperty`, however, `connectedPropertyCodes` is defined only for `Property`
- **Improve modelling of Dynamic Properties**, which is also done partially - while 12385 `Properties` are declared as `isDynamic` (135250 are not), the field `dynamicParameterPropertyCode` is always empty.
- **Improve relations between entities**, as shown in Fig. 2.
- **Add more entities** for e.g.`PhysicalQuantity` to govern allowed `Units`.
- **Use class inheritance** for `ClassificationProperty` and `Property` and whenever possible.
- **Improve representation of PropertyValues**: `PropertyValue` and `ClassificationPropertyValue` are structured values with rich fields: `code, value, namespaceUri, description, sortNumber`. However, most structured values we've seen have only `code, value`.
- **Improve representation of** `predefinedValue`.
- **Improve multilingual support**: while bSDD is advertised as a multilingual dictionary, most domains are unilingual.

GraphQL Improvements

In this subsection we outline the problems with the original bSDD GraphQL API.

- **Improve searchability and pagination**. Currently, the user is limited to very basic fetching of data: all entities of a class, entity by `namespaceUri`, or basic full-text search (`classificationSearch`). There is no pagination, the user cannot get only a portion of the results, and iterate through pages with `limit/offset`.
- **Eliminate parallel links between entities**, see Fig.1.
- **Improve GraphQL arrays and nullability**: e.g., there is no way to enforce a **non-empty** array in GraphQL.
- **Null classifications error**. Although `classificationSearch` is declared as nullable, a GraphQL error is returned whenever the backend returns `null`.
- **Null classification childs error**. `Classification.childs` is defined as nullable with type `[Classification]` However, unless `includeChilds: true` is provided as input argument in `classification`, queries return NULL_REFERENCE errors, thus breaking GraphQL specification compliance.

- **Null ClassificationProperty name error**. Some ClassificationProperties have no `name`. Although that field is declared nullable, bSDD does not return such properties and instead returns `NULL_REFERENCE` errors.
- **Missing domains**. The GraphQL root field `domains` used to return some domains that are not available individually through the field `domain`.
- **Deprecated properties**. The field `possibleValues` is described as "deprecated". However, the GraphQL specification section Field Deprecation shows that a specific `@deprecated` directive should be used for this purpose.

We found also many data quality problems in bSDD, but due to the page limits we leave them outside, and redirect a curious reader to the bSDD project Web site.

Implementing Improvements

We implemented a lot (but not all) of the improvements suggested above by using the following process:

- **Fetch bSDD data as JSON** with the help of a script bsdd2json.py developed to get all the data.
- **Convert it to RDF** using SPARQL Anything. We developed two scripts: rdfize.sparql and rdfize-zip.sparql to rdfize either single file or an archive (zip) of files.
- **Load it to GraphDB**
- Refactor the RDF using SPARQL Update

As far as raw RDF has drawbacks, we wrote the SPARQL Update script transform.ru which does the following:

- Cut out fractional seconds from date-times, and add datatype `xsd:dateTime`
- Convert strings to URIs, and shorten props as appropriate
- Drop redundant information of a referenced resource
- Drop deprecated property `bsdd:possibleValues`, since `bsdd:allowedValue` is used instead;
- Multi-valued properties: skip a level (`rdfs:member`) and change property name to singular
- Shorten the path `bsdd:parentClassificationReference/bsdd:namespaceUri` to just `bsdd:parentClassification`;
- Add `rdf:type` based on GraphQL `__typename`;
- Drop parasitic `rdf:type fx:root`;
- Because link `ClassificationProperty.namespaceUri` refers to a `Property` rename it to `ClassificationProperty.property`;
- Add meaningful URIs to blank nodes whenever possible. In particular (here `+` indicates concatenation):

  - `ClassificationProperty` gets URI: `Classification.uri+"/"+propertyCode` ;
  - `ClassificationPropertyValue` gets URI: `Classification.uri+"/"+ClassificationProperty.propertyCode +"/"+value`. This class has `namespaceUri`, but that is optional and is rarely filled;
  - `PropertyValue` gets URI: `Classification.uri+"/"+Property.propertyCode +"/"+value`. This class has `namespaceUri`, but that is optional and is rarely filled.

- The following remain blank nodes:
  - `ReferenceDocument`: no id field (only `name, title, date`);
  - `ClassificationRelation`: is just a pair of `related` Properties, no own URI;
  - `PropertyRelation`: is just a pair of `related` Properties, no own URI;

- Remove redundant `namespaceUri` when equal to the node's URI.

GraphQL to SOML and Back

The major goal of this work is to improve the bSDD RDF representation and GraphQL API. To achieve this, in addition to refactoring RDF:

- The original GraphQL schema was fetched with GraphQL introspection: `bsdd-graphql-schema-orig.json`, 116kb
- Then it was converted to a prototypical SOML schema using the script `graphql2soml.py`: `bsdd-graphql-soml-orig.yaml`, 22kb. This SOML schema has issues inherited from the original GraphQL schema. The purpose of the generated SOML schema is to serve as a starting point (instead of starting from scratch).
- The schema was refactored by hand, using similar steps as the RDF refactoring above: `bsdd-graphql-soml-refact.yaml`, 20kb.
- The results were loaded to Ontotext Platform Semantic Objects to generate a refactored GraphQL schema: `bsdd-graphql-schema-refact.json`, 867k. The reason it is so much bigger is that it includes a comprehensive `where` query language

Conclusions and Future Work

Admitting the advances of bSDD community at providing data in RDF format we met some issues where accommodating these data for our purposes in the frame of ACCORD project, among them are different results obtained with different APIs, multiple URIs for same entities, various GraphQL implementation errors. In the presented work, we highlighted these issues and proposed a set of technical improvements following the best

practices of the Semantic Web and linked data to obtain "a semantically better version" of bSDD. We implemented and made available our solution using the Ontotext Platform.

Acknowledgements

This work is partially funded by the European Union's Horizon Europe research and innovation programme under grant agreement no 101056973 (ACCORD).

Author contributions:

- VA conceived the work, described bSDD shortcomings, implemented GraphQL and RDF refactoring.
- MK wrote GraphQL queries, fetched bSDD data, and deployed GraphDB and Ontotext Platform Semantic Objects.
- NK performed statistics and comparisons of bSDD data and wrote the final paper.

We thank Léon van Berlo and Erik Baars from buildingSmart International for their help with accessing bSDD.

References

[1] "ISO 23386:2020 Building information modelling and other digital processes used in construction — Methodology to describe, author and maintain properties in interconnected data dictionaries," international standard [Online]. Available: https://www.iso.org/standard/75401.html. [Accessed: Jan. 25, 2023]

[2] "ISO 12006-3:2022 Building construction — Organization of information about construction works — Part 3: Framework for object-oriented information," international standard [Online]. Available: https://www.iso.org/standard/74932.html. [Accessed: Jan. 25, 2023]

[3] S. Palos, "State-of-the-art analysis of product data definitions usage in BIM," in *eWork and eBusiness in Architecture, Engineering and Construction: European Conference on Product and Process Modelling 2012, ECPPM 2012*, 2012, pp. 397–403.

[4] P. Pauwels, T. Krijnen, and J. Beetz, "Making sense of building data and building product data," Mar. 2016 [Online]. Available: https://pdfs.semanticscholar.org/93f0/278821cf554be5a6f6e2667b24cb39096fe4.pdf

[5] J. Beetz, W. Coebergh Van Den Braak, R. Botter, S. Zlatanova, and R. De Laat, "Interoperable data models for infrastructural artefacts : A novel IFC extension method using RDF vocabularies exemplified with quay wall structures for harbors," in *eWork and eBusiness in Architecture, Engineering and Construction - Proceedings of the 10th*

European Conference on Product and Process Modelling, ECPPM 2014, 2014, pp. 135–136, doi: 10.1201/b17396-26 [Online]. Available: http://www.scopus.com/inward/record.url?scp=84907300060&partnerID=8YFLogxK. [Accessed: Jan. 25, 2023]

[6] P. Pauwels and W. Terkaj, "EXPRESS to OWL for construction industry: Towards a recommendable and usable ifcOWL ontology," *Automation in Construction*, vol. 63, pp. 100–133, 2016, doi: https://doi.org/10.1016/j.autcon.2015.12.003. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580515002435

[7] D. Garijo and M. Poveda-Villalón, "Best practices for implementing FAIR vocabularies and ontologies on the web," in *Applications and practices in ontology design, extraction, and reasoning*, vol. 49, G. Cota, M. Daquino, and G. L. Pozzato, Eds. IOS Press, 2020 [Online]. Available: http://ebooks.iospress.nl/doi/10.3233/SSW200034. [Accessed: Feb. 15, 2023]

[8] J. Oraskari, "Live Web Ontology for buildingSMART Data Dictionary," in *Forum Bauinformatik*, 2021, pp. 166–173 [Online]. Available: https://www.researchgate.net/profile/Jyrki-Oraskari/publication/355425683_Live_Web_Ontology_for_buildingSMART_Data_Dictionary/links/616fcce1718a2a7099e4d86b/Live-Web-Ontology-for-buildingSMART-Data-Dictionary.pdf

[9] R. Kebede, A. Moscati, H. Tan, and P. Johansson, "Integration of manufacturers' product data in BIM platforms using semantic web technologies," *Automation in Construction*, vol. 144, p. 104630, Dec. 2022, doi: 10.1016/j.autcon.2022.104630. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0926580522005003. [Accessed: Jan. 25, 2023]