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

Vladimir Alexiev1, Mihail Radkov1 and Nataliya Keberle1

1Ontotext, 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. Based on this analysis, we suggest enhancements to make bSDD data better structured. The complete list of suggestions can be found at https://bsdd.ontotext.com/README.html. 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, Linked data best practices

1. Introduction

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

**Proceedings LDAC2023 – 11th Linked Data in Architecture and Construction, June 15–16, 2023, Matera, Italy**

*Corresponding author.

vladimir.alexiev@ontotext.com (V. Alexiev); mihail.radkov@ontotext.com (M. Radkov);
nataliya.keberle@ontotext.com (N. Keberle)

© 2023 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

CEUR Workshop Proceedings (CEUR-WS.org)
- 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.
- buildingSMART Data Dictionary (bSDD): for national, domain, company/projectwide standards and classification systems 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 claimed to be 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 comprehensive solution aiming at the provisioning of open product data definitions, identification, and distribution methods.

bSDD includes 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 many of its defined entities to enable globalized data use in a variety of AECO applications and structured documents.

While the bSDD RDF and GraphQL representations are very useful, they also have various problems, among them are different results obtained with different APIs, multiple URLs for the same entities, various GraphQL implementation errors. In this paper, we discuss these issues and propose improvements that follow Semantic Web and Linked Data best practices to obtain “a semantically improved version” of bSDD.

2. Related Work

According to the buildingSMART technical roadmap, bSDD service provides output data in various formats and APIs, including RDF, thus making bSDD content 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 range from OWL to RDF and SKOS have been implemented to evaluate the advantages and disadvantages of the respective modelling strategies” the work was continued. [6] proposes and implements the solution to generate
from bSDD an OWL representation of a selected IFC element together with its property sets on the fly.

[7] is the survey that discusses 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.

Unlike [6], we preserve the original bSDD structure (Domain, Classification, Property, etc.) and only add specific improvements, described below, and convert the whole bSDD at once. We pay more attention to describe the defects of the current solution, and propose an improved solution that complies better with Semantic Web best practices.

3. GraphQL and bSDD

GraphQL is an approach to create simplified “façades” 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)

bSDD does offer GraphQL access, and an interactive in-browser GraphQL IDE called GraphiQL:

- 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.

3.1. 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, 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 the test GraphQL bSDD endpoint contains 31720 Classifications, 111556 ClassificationProperties, 214121 ClassificationPropertyValue and PropertyValue (represented the same, because they have the same structure), 6420 ClassificationRelations, 36069 Properties, 603 Units, 484 ReferenceDocuments, 39 Languages, 246 Countries, 108 Domains.

### 3.2. 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.
3.3. 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 refactored schema (see Fig. 2) implements the following improvements:

- 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
Figure 3: GraphiQL Querying of Refactoring bSDD Endpoint

- 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).

3.4. GraphiQL Querying of Refactored Endpoint

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

We have deployed a newer version of GraphiQL that keeps all benefits of the original bSDD GraphiQL and adds some more (see Fig. 3):

- 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)
4. Suggested Improvements

In this section, we analyze the 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. The complete list of our findings together with explanations and examples can be found at the bSDD project Web site.

4.1. Representational Shortcomings

- **Return the same data across APIs.** We have compared three representations returned by the bSDD server: JSON from the GraphQL API\(^1\), JSON from the REST (entity) API\(^2\), and RDF from the REST (entity) API\(^3\). 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. 4).

- **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 lowerCase convention. `namespaceUri` is a misnomer since “namespace” means a set of URIs sharing the same prefix, but most bSDD URIs are single URIs. RDF

---

1 https://test.bsdd.buildingsmart.org/graphiql/
2 curl https://identifier.buildingsmart.org/uri/buildingsmart/<domain>/class|prop/<name>
3 curl -H accept:text/turtle \ https://identifier.buildingsmart.org/uri/buildingsmart/<domain>/class|prop/<name>
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/rela tion/def#>.

- **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 in [8]. Almost all domain URLs have the same structure:
The Linked Data Patterns book [9] 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 in the GraphQL schema, to **remove the overlap of En-
tity Classes with classificationType**, to **disambiguate URLs for Property and
ClassificationProperty**, to **provide an URL for all the Entities** in the bSDD schema.

### 4.2. Modelling Issues

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

- **Unify different solutions in the 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:** bSDD includes numerous string attributes (codes or URLs) that should be converted to relations (object fields) to improve the connectedness of the bSDD GraphQL graph. E.g., there are several entities (Country, Language, ReferenceDocument, Unit) that are not used anywhere. Instead of relations pointing to these types, the other types have properties (e.g., countryOfOrigin, countriesOfUse) representing the same information as String. Also, ClassificationRelation and PropertyRelation use strings (e.g., relatedPropertyUri) instead of outgoing relations, thus blocking further GraphQL navigation, see one possible solution in the Fig. 2.

• **Add more entities:** there are reusable components of the bSDD GraphQL graph that can become entities. E.g., PropertySet, the important concept in IFC and bSDD, PhysicalQuantity, which can govern allowed Units and subjugate to all dimension-related fields, CountrySubdivision as lookup for subdivisionsOfUse, subjugated to Country.

• **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 of them have only code, value.

• **Improve representation of predefinedValue:** while allowedValues property is structured, its 'sibling' property predefinedValue is a String, it does not have structure and cannot be identified with its URL. The possible structure can resemble the one for PropertyValue.

• **Improve multilingual support:** while bSDD is advertised as a multilingual dictionary, most domains are unilingual.

4.3. GraphQL Improvements

In this subsection, we outline GraphQL implementation problems.

• **Improve searchability and pagination.** Currently, the user is limited to very basic fetching of data. Also, there is no pagination, the user cannot get only a portion of the results, and iterate through pages with \texttt{limit/offset}.

• **Eliminate parallel links between entities.** A number of parallel relations in the original GraphQL schema, e.g., \texttt{Root.domain,domains} can be eliminated.

• **Improve GraphQL arrays and nullability.** It is better to use \texttt{[Type!]} – optional array of mandatory elements.

• **Null classifications error.** Although \texttt{classificationSearch} is declared as nullable, a GraphQL error is returned whenever the backend returns null.

• **Null classification childs error.** \texttt{Classification.childs} is defined as nullable. However, unless \texttt{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 page limits we leave them out, see more details online.

## 5. 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.
- **Load it to GraphDB.**
- **Refactor the RDF** using SPARQL Update.

The refactoring update 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:
  \text{Classification.uri}+"/"+\text{propertyCode} ;
- ClassificationPropertyValue gets URI:
  \text{Classification.uri}+"/"+\text{ClassificationProperty.propertyCode} +"/"+\text{value}.
  This class has namespaceUri, but that is optional and is rarely filled;
- PropertyValue gets URI:
  \text{Classification.uri}+"/"+\text{Property.propertyCode} +"/"+\text{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.

5.1. 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: bsd
dd-graphql-schema-orig.json, 116kb
- Then it was converted to a prototypical SOML schema using the script
graphql2soml.py: bsd
dd-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: bsd
dd-graphql-soml-refact.yaml, 20kb.
- The results were loaded to Ontotext Platform Semantic Objects to generate a refactored GraphQL schema: bsd
dd-graphql-schema-refact.json, 867k. The reason it is so much bigger is that it includes a comprehensive where query language.

6. 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 the same entities, various GraphQL implementation errors. In the presented work, we highlighted these issues and proposed a set of both technical and modelling improvements.
following the best practices of the Semantic Web and Linked Data to obtain “a semantically better version” of bSDD. Namely, we revealed new potential entities, improved relations, making bSDD GraphQL graph more connected thus more navigable. We implemented and made available our solution using the Ontotext Platform.

In the future, we plan to work with bSI to implement more of the suggested improvements and to deploy the official bSDD using Ontotext software. Among the directions are: improvement of bSDD ontology, implementation of more radical data model refactoring to convert "strings" (countries, reference documents, etc.) into 'things', linkage of bSDD units of measure to QUDT ontology. We also plan to perform deeper data quality analysis using SHACL shapes generation and validation provided by Ontotext Platform Semantic Objects; to address and resolve more data quality issues, including seeking correlation between dimension vectors, units of measure and physical quantity, parsing out enumeration values from Property/ClassificationProperty descriptions and creation of corresponding PropertyValue lists.

7. 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


