Managing and publishing standardized data catalogues to support BIM processes

Christian Clemen, christian.clemen@htw-dresden.de
Faculty of Spatial Information, University of Applied Sciences Dresden, Dresden, Germany

Benjamin Thurm, mail@bentrm.dev
Faculty of Spatial Information, University of Applied Sciences Dresden, Dresden, Germany

Sebastian Schilling, sebastian.schilling@htw-dresden.de
Faculty of Spatial Information, University of Applied Sciences Dresden, Dresden, Germany

Abstract
With every integration step between BIM data sources, there is a certain chance of introducing semantic inaccuracy or losing information. While the Industry Foundation Classes (IFC) provide a huge set of entity types, its semantic expressiveness is limited. The standardization of semantics within a BIM project or an application domain must therefore take place outside of IFC. ISO12006-3:2007 and ISO23387:2020 describe a well-established, language-independent taxonomy model. But due to the abstract nature of the definition, implementing a model is a complex task that lacks tooling. We present our current research, designing a stack of opensource service components. These enable a collaborative design of semantics between construction expert groups. The catalogue itself is published via a application programming interface (API) based on the GraphQL-specification. This API aims to be easy to integrate into tools along the CDE. A practical example of this integration with a customary BIM modeler will be given.

Keywords: data catalogue, data templates, data exchange, semantic web

1 Introduction
BIM aims to be a multilateral approach to optimize planning, execution and management of construction works using software. Ideally, all parties of a construction project are supposed to share a common data environment (CDE). But a CDE will never be one monolithic database but rather a heterogenous collection of data services. With every integration step between these data sources, there is a certain chance of introducing semantic inaccuracy or losing information altogether.

While the Industry Foundation Classes (IFC), the most common openBIM exchange format, provide a huge set of entity types to describe physical and functional components, its semantic expressiveness is limited. With the user-defined property sets, IFC can be extended generically. However, the semantic interoperability is then no longer guaranteed. The standardization of semantics within a BIM project or an application domain must therefore take place outside of IFC.

ISO 12006-3:2007 describes a well-established and language-independent taxonomy model. But due to the abstract nature of the definition, implementing a model is a complex task that lacks tooling. Multiple publications aim to offer guidance in structuring data based on ISO 12006-3:2007 e.g., in the form of data templates (EN ISO 23387:2020). Yet, even utilizing a small subset of the given structural elements of these publications results in an interconnected graph of concepts that is hard to reason about and impractical to work with using conventional software tools like spreadsheets. This hinders knowledge transfer between specialists.

We present our current research, designing a stack of open source service components. These components enable a collaborative design of semantics between construction expert groups. The catalogue itself is published via a novel application programming interface (API) based on the GraphQL-specification. This API aims to be easy to integrate into tools along the CDE and to be a starting point to ontology export routines. A practical example of this integration with a customary BIM modeler to retrieve properties of construction objects will be given.

We will discuss challenges implementing the persistence and API model adapting the taxonomy model of ISO 12006-3:2007. Furthermore, we will present a user-friendly interface to design data templates in a multi-user environment following the recommended structure of ISO 23387:2020, as it is being evaluated by the German buildingSMART e.V. specialist group "Transport Routes".

We present our current research on "datacat", an open source software stack for managing and publishing standardized data catalogues to support BIM processes. With datacat, data catalogues can be collaboratively created, edited, checked and published using the ISO12006-3/IFD standardized meta-concepts for classification systems such as properties, property sets, objects and group of objects (Figure 1).

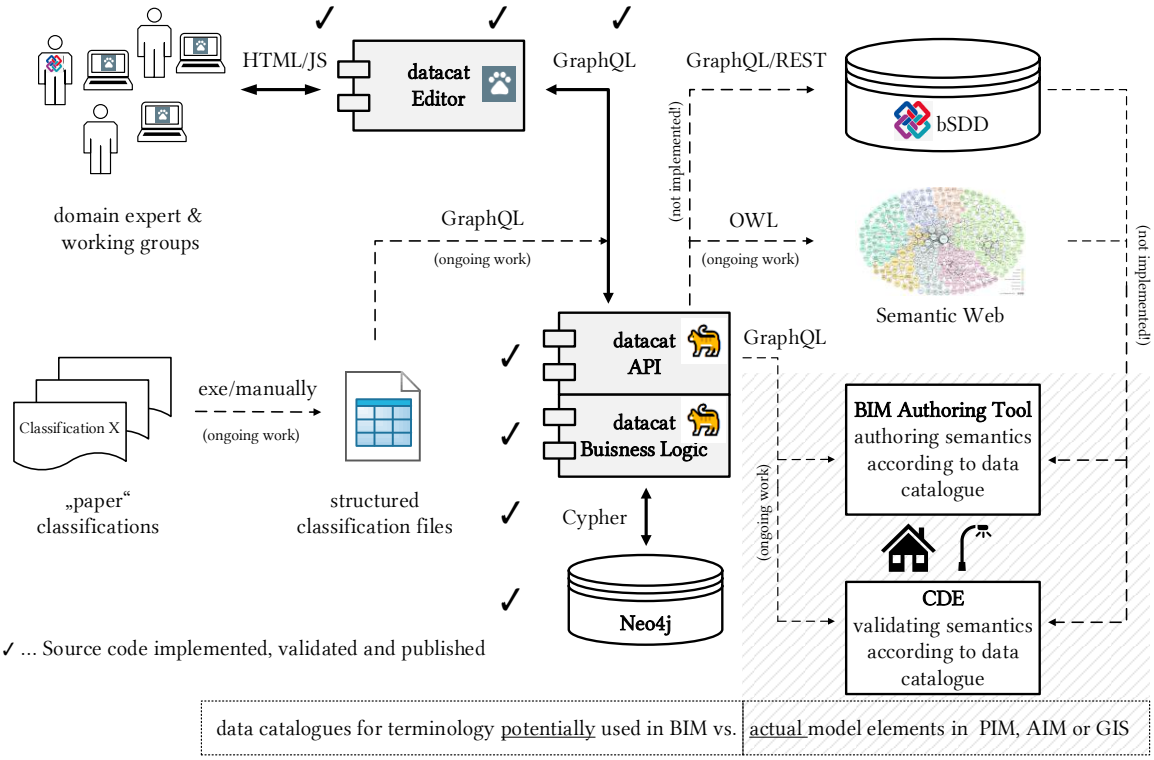

Figure 1. illustrates the usage scenario and information flow of a datacat instance. Please note, that this paper mainly reports about the results of the datacat editor, datacat API, datacat business logic and the Neo4j persistence layer.

Data catalogues are created by domain experts in working groups for pre-standardization or also by professional bodies in order to digitize existing "paper" taxonomies and classification systems. A data catalogue provides a structured terminology that is potentially used in BIM projects. The service may be consumed by BIM/GIS/CAD-software, aiming for "semantic homogeneity". With computer readable data catalogues, the "semantic model" is compatible throughout software systems and along all phases of delivery: Even while executing very different tasks, such as managing object type libraries, authoring BIM models, validating information deliveries or publishing actual models of the built environment as project information model (PIM) or asset information model (AIM), the terminology remains unchanged.

A "data catalogue" is used to file concepts as "catalogue records". In related works, the terms "data dictionary" or "property server" are used almost synonymously. We prefer the term "catalogue" to demarcate systems whose primary focus is to link different records by means of relationship and ID from systems whose main purpose is to list such concepts for advanced querying and editing. The software datacat was developed in several working-package of publicly funded research projects and is currently being validated in the pre-standardization work of the "BIM traffic routes" specialist group of buildingSMART e.V. Germany.

In addition to the academic interest in data catalogues, we have developed datacat to offer modular open source tooling for research and to fulfil our project requirements. Currently, the buildingSMART data dictionary (bSDD) is predominantly used for commercial hosting of pre-prepared data catalogues and possibly chargeable hosting of these. At bSDD, the international publication and consumption is the main focus of operation. However, we are in the need of an agile tool for the collaborative creation and validation of data catalogues. Established commercial solutions such as BIMQ, e.g. described in (Hauer, 2020), or the cobuilder platform (Tune, 2017) would simply be too expensive for our projects needs and could not be adapted for research purposes. Alternatively, the Austrian FreeBIM server (Breuss and Lanzinger, 2021) is also released under a permissive open source license and utilitzes Neo4j as persistence layer. FreeBIM is an Austrian research project by the University of Innsbruck offering a property server to collect, administer and link properties of structural elements and materials. Their goal is to compare properties, e.g. from ÖNORM, with the bSDD and supplement missing properties. The properties in the database then can be mapped to properties of BIM software. However, FreeBIM does not provide a customized GraphQL API nor is it strictly based on the concepts from IFD/ISO12006-3. The main administrative features of datacat are:

- GPLv3 License for source code on https://github.com/dd-bim/... (Thurm 2021a,b,c)
- strict orientation towards standardized meta-concepts for compatibility with the bSDD and other catalogues not managed with datacat
- Functionality proven outside of academic projects

The main technical features of datacat are:

- Modularized and web-based-only software components
- Graph database (Neo4j) as persistence level
- datacat API, a developer-friendly, self-documenting GraphQL interface for complex queries on the data catalogue
- datacat editor, a browser-based user tool for navigating and editing the database
- Easy deployment as "docker application stack" to execute the datacat API and editor client application on any server in federated environments

The datacat API is a thin server application that is designed to be hosted decentralized. We believe that the need to maintain data catalogues will be omnipresent with the future application of the BIM method. Also, while other developments focus on offering a centralized, well governed data storage for large-scale operations, we foresee the need for local installations. Fundamental task will be to draft and test new conceptual models, mirror existing data catalogues from official read-only sources and to adapt coexisting models to the current project at hand.

Following this mindset, we decided against a full-fledged versioning and governance scheme, as it would be needed for a centralized service. We aim to realize a service that can be easily adopted during the on-boarding phase of a project, is interoperable with existing authoritative sources and can be easily integrated into a broad BIM-infrastructure.

To ensure interoperability with current and future developments, the fundamental base of our work is the adoption of ISO 12006-3 and the design considerations modelling data templates according to ISO 23387. We followed the guidelines of these standards rather closely, adapting the standards as an implementation specification for the underlying domain layer of the application. This has some influence on the architecture as a whole, as well as on the technologies that we choose as our implementation framework.

As described above, the taxonomy model presented by ISO 12006-3 is highly generic and allows to nest concepts and relationship concepts nearly arbitrarily. This means, that a

fundamental, highly connected concept of a domain might be included in a vast number of relationships. This interconnectedness exceeds the complexity of a tree structure rather quickly resulting in a hard to maintain graph structure. In fact, our practical experience cooperating with the buildingSMART Germany specialized group "Transport routes" has shown, that even a very small draft of a catalogue consisting only of construction subjects that are grouped by domains resulted in a graph structure. This graph will become much more complicated if more complex concepts like property inheritance and composition are included in the domain model.

2 Related Standards

The openBIM standard IFC (ISO 16739-1:2018) includes a vast number of IFC entities that describe real-world concepts and their properties. These entities can be decorated with instances of the type IfcPropertySet, "a container that holds properties within a property tree". While the IFC specification includes a number of IfcPropertySet-definitions, the main purpose of this entity is to allow users to define custom groups of properties that can be assigned to object occurrences and object types, the latter designating static attributes.

Furthermore, IFC includes the definition of an IfcPropertySetTemplate that can be used to define the underlying structure of an IfcPropertySet. Since IFC follows an inheritance strategy, all entities descend from the most general type IfcRoot, it's possible to describe properties and groups of properties by name, type and identifier. This allows to cross reference the definition of these concepts with data catalogues. Additionally, IfcClassification allows to cross reference local and external classification systems to IFC object occurrences. While this offers insights into the semantics of a model's object, no further practical implications are described.

ISO 23386, first released in March 2020, offers a "Methodology to describe, author and maintain properties in interconnected data dictionaries". Its main focus concerns the description of properties and group of properties. Groups of properties can be further categorized as class, domain, composed property and reference document and can be nested into a tree structure, which makes the design of basic inheritance possible. As the main focus is offering a methodology to be adapted by various data catalogue vendors, the standard includes no implementation specification, nor does it define an exchange format but rather a tabular overview of metadata that is needed to administer data dictionary entries. Furthermore, the standard describes an elaborate governance process and dedicated user roles to handle change requests in a coordinated manner. Its main target is to "ensure the quality and the unicity of property descriptions and avoiding the creation of duplicates". The existence of codependent data dictionaries is embraced as a given. Therefore, to achieve unicity, every entry in a data dictionary is not identified by a language dependent string that names the concept but a globally unique identifier. A basic inheritance between groups of properties is supported.

Already released in 2007, ISO 12006-3 "consists of [a] specification of a taxonomy model, which provides the ability to define concepts by means of properties, to group concepts, and to define relationships between concepts". Contrary to ISO 23386, it follows a much more object-oriented approach which allows to describe real world phenomena as types like xtdActor, xtdActivity, xtdSubject, xtdProperty and xtdValue. These singular types, relationship types, as well as collection types thereof are all subtypes of the abstract class xtdRoot. The resulting inheritance structure allows to describe all of these concepts by an ID, multilingual names and descriptions.

While the specification itself is given in EXPRESS and EXPRESS-G notation, a concrete implementation of a data structure needs to be derived. Also, due to the highly abstract nature of some of the involved relationship types, it's hard to stipulate a consistent application between data domains and domain users.

ISO 23387 builds on top of ISO 12006-3 and uses the broad definition of the specification's entities to "set out principles and structure for data templates for construction objects". It is meant as an implementation guideline for software developers to improve interoperability between software systems that implement some kind of data templates based on ISO 12003-6.

Furthermore, additional rules for linking data templates to IFC classes and other classification systems design in data dictionary are provided.

The currently developed standard "IFC4.COD.1 Construction Objects Data View Part 1" (CEN, 2020) tries to connect data catalogues with actual IFC models. "The standard defines the syntactic characteristics of a generic structure to transport data about construction objects based on EN ISO 16739-1:2018, prEN ISO 23386 and prEN ISO 23387" (CEN, 2020). The standard also provides an developer friendly XML-Schema, EXPRESS and Java-classes similar to Model View Definitions (MVD). In doing so, this standard will close the gap between the many generic conceptual models (data catalogues) and the actual use of these models in openBIM projects using IFC.

The most-related data catalogue solution might be buildingSMART data dictionary (bSDD) which is offered by buildingSMART International and currently released as version 4. The data model follows ISO 12006-3 closely. The main focus of bSDD is to be a central data store for data catalogues. A tenant system is in place, that allows different domain owners to host data records isolated from all other domains. By linking concepts across data domains, a cross-catalogue relationship can be established. bSDD offers a comprehensive RESTful-API to query and retrieve catalogue records and relationships.

Since 2020, a new version is announced as in active development. The main focus of version 5 will be to revise the API to be more developer friendly, integrate the possibility to publish linked data and a new authorization layer based on the OAuth 2.0 specification. It is unknown to the authors, if bSDD will support ISO 12006-3 to its full extent. The service will still be hosted centrally by buildingSMART International as a data-catalogue-as-a-service. This means, publishers and domain owners pay to host their data catalogue with buildingSMART International and can choose to make it available to the public. Currently, multiple well-known collections are hosted publicly including OmniClass, IFC4 properties sets and also some national classification systems.

3 Methods - datacat solution

The datacat software stack is currently divided into the core components datacat API (Thurm, 2021a), a developer-friendly, self-documenting GraphQL interface for complex queries on the data catalogue and the datacat editor (Thurm, 2021b), a browser-based user tool for navigating and editing the data stock. The datacat API is a thin server application that is designed to be hosted decentralized.

In traditional applications, the persistence layer is based on a relational database. The most prevailing query language to interact with such storage engine is SQL. On the most basic level, this means, that the data storage of such is "table oriented". Entity attributes as the ID are typically represented as columns of these tables. To represent relationships between different entities the ID is referenced either from one side of the relationship or from a designated lookup table. In most cases, querying this store for relationships means to lookup the owning side of the relationship, find all related entity IDs and read all those entities into memory as well. For data in tables with a graph structure, this quickly becomes an expensive operation, as there might be an unknown number of relationships that follow the starting point of a SQL query. While this problem is in part mitigate by some modern implementations supporting recursive queries, a graph structure still is a second-class abstraction for typical relational databases. This is why we choose to favor a graph database to implement the persistence layer of datacat.

In a graph database, more specifically, in a property graph database, the basic storage abstraction are nodes and edges. Both types are identified by an ID and might carry additional arbitrary properties. For our use case, this allows to store administrative data with every edge between data nodes. Also, instead of using SQL to query the data store, we're able to use Cypher, a graph querying language that make it comparably ease to wander and aggregate sub-views of the data catalogue (Francis et al., 2018). A good example where this comes in handy, is a query that aims to aggregate all inherited properties along a nested inheritance tree. Also, since the actual relationship is a first-class citizen of the querying logic, it is possible to define queries based

on properties of the relationship itself. For example, its trivial to query for all relationships of type COMPOSES without defining a concrete type that this relationship originates from.

Our main focus is to offer versatile querying capabilities to application developers to request sub-views of the data catalogue. In many cases, this means to export a data template of a selected concept or a list of concepts – maybe construction objects. To aggregate such data view, many relationships originating from a catalogue record needs to be queried at once, recursively along the user-defined inheritance structure. Practically, this could be achieved through a RESTful interface that offers a representation for each of the participating entity types as a HTTP resource. This may pose some practical difficulties consuming the API: The resources follow a predefined structure determined by the interface developer. If the predetermined server response does not satisfy all data requirements for the use case of the requesting application, more requests need to be sent to the server's API to retrieve the missing information. This lays a burden on the server hardware that needs to handle additional requests as well as on the application developer who needs to combine multiple roundtrips in the application logic. If the API developer foresees this, one might be inclined to include more information in the most prominent resources of the service. But this bares the risk to process information on the server side that might never be used on the client. Since datacat does not dictate a schema for any data catalogue other than the rules defined by ISO 12006-3, it's impossible to predetermine the shape of all HTTP resources needed for all use-cases.

To compensate for this, we implemented the datacat API following the GraphQL-specification as it has been published by Facebook. A GraphQL-API allows the application developer to define type-safe queries along the published domain model. "'Plain GraphQL is both a query language and a specification. It focuses on minimising and optimising data loads over the web, by connecting to a GraphQL API" (Werbrouck et al.,2019). An example is given in Figure 2. Incidentally, this model resembles the taxonomy of the underlying ISO specification, improving interoperability and understanding.

Figure 2. A simple GraphQL-query (left) selecting id, name and related collections of a construction object. The term „subject" is derived from the ISO 12006-3 entity „xtdSubject". More properties can be selected by extending the query structure. The server response (right) includes the fields selected by the query.

The datacat software stack is organized into components. Main focus of our research has been the development of the datacat API (Figure 3). The datacat API is based on a Neo4j graph database in which all data is stored. Among other things, the API consists of components of the Spring Framework, the GraphQL API and its integrated development environment (IDE) GraphiQL, which provides a graphical interface for interaction. Through the datacat API, users can directly make complex queries or make changes to the data with the appropriate permission. Alternatively, the datacat API can be accessed via a client, for example the datacat editor developed for this purpose. In the future, BIM authoring systems will also be able to access datacat via plug-ins.

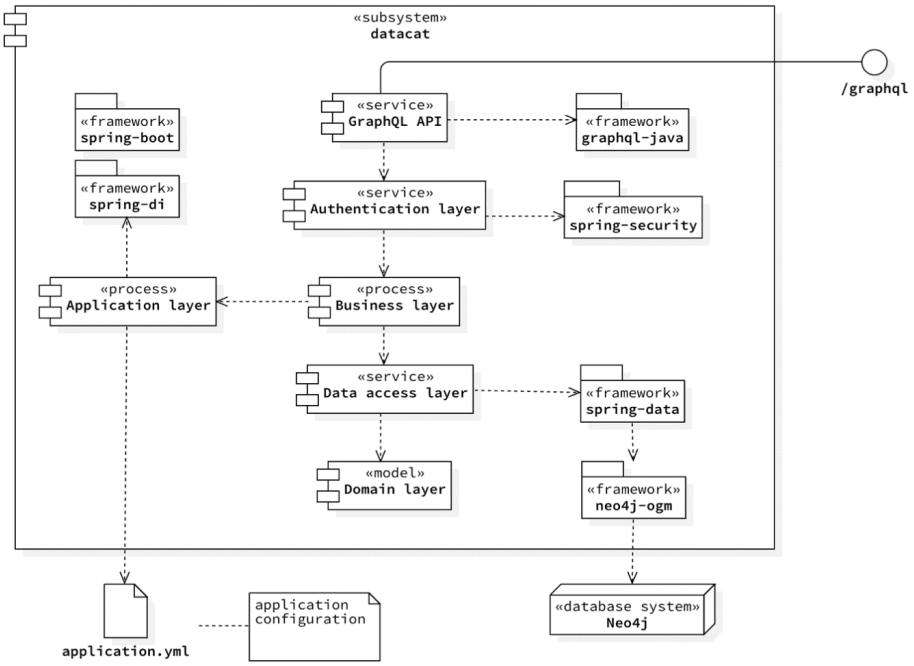

Figure 3. Basic software packages of the datacat API component.

The datacat editor (Figure 4) offers an opinionated view to manage a data catalogue structured after ISO 12006-3. With this browser-based editor the user can easily navigate through the data catalogue and insert, update or delete data.

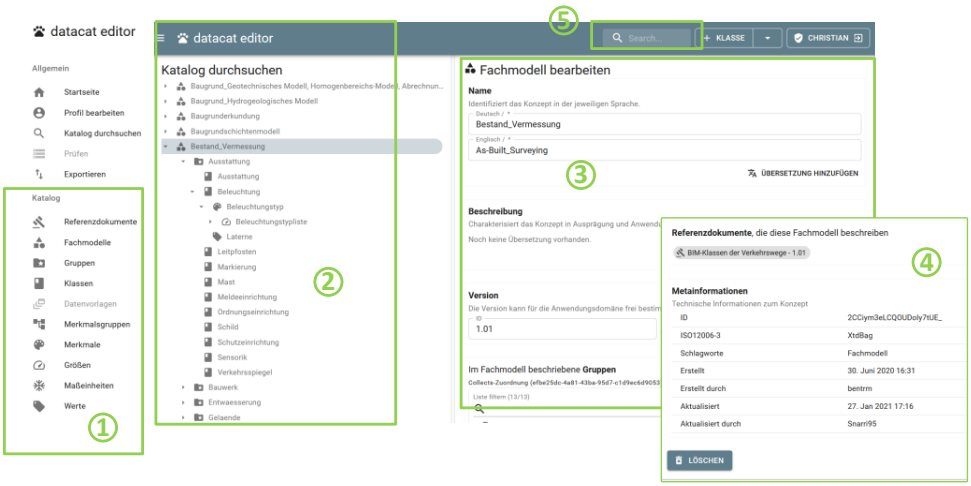

Figure 4. Illustration of the customized editor (German): (1) Meta-concepts of ISO12006-3 understandable for non-IT-experts (2) a interactive tree-view on the catalogue graph (3) UI for simple editing (4) some metadata (5) free text search.

Using e.g. “pagination” the queries and UI-updates after editing the catalogue are optimized and user friendly. The authentication system controls who is allowed to perform which actions. For a simple and complete installation, the datacat docker stack (Thurm, 2021c) can be used, which combines the components datacat API, datacat editor and Neo4j database.

4 Conclusion

4.1 Discussion

As already mentioned, the relevant standards for data catalogues are very generic. Therefore, numerous design decisions have to be made in the programmatic implementation:

GUID. While it is possible to use arbitrary unique identifiers specified while creating new catalog records, in practice, most concepts will be assigned an auto-generated ID by the system. We use a standardized UUID algorithm to generate IDs as 128bit-numbers. As of now, we do not compress these IDs as defined in the IFC and ISO 12006-3 specification as the advantages in space efficiency are marginal. Future plans include an encoding mechanism that allows to output contemporary UUIDs in the compressed IFC identifier format for support with legacy systems.

Optional Properties. In ISO12006-3 there are no rules or attributes in place that allow to designate optional properties and property groups. Until now this feature is also not implemented in datacat because we need to check, if this would violate the compatibility with other catalogue servers.

Inheritance of xtdRelationship. For usability reasons handling return types of API methods, datacat relationships do not inherit directly from xtdRoot as described in ISO 12006-3. All properties of xtdRoot are also assigned to all relationship types, but -in contrast to the standard- our relationship records cannot be a member of other relationship definitions. We did not find a practical relevant application of this. Nevertheless, this might change in the future and would mean only a rather small change in the codebase.

Multilingualism. One core feature of the ISO 12006-3 specification is multilingualism of names and descriptions of all concepts. This allows to define a name of a concept even in closely related language families like German, Austrian German or Swiss German. While a rather versatile approach, this leads to complexity for application developers that consume the data catalog. datacat implements this feature as specified but extends on this by offering additional API methods to interact with language representations. In addition to querying all language representations of a concept, a language priority list can be provided (['de-at', 'de', 'en']) that will be interpreted by the service selecting the most appropriate language representation.

Direct relations. ISO 12006-3 describes multiple ways to designate the relationship between an arbitrary measure and the concept of values and units. The specification is inconsistent by including an optional unit component of type xtdUnit as well as a direct relation to a list of xtdValue entities as entity properties of the type xtdMeasureWithUnit. This deviates from all other relationship characterizations of the specification that are reified as instances of xtdRelationship entities. Furthermore, the two concrete relationship types of name xtdAssignsValues and xtdAssignsUnit can also be used to describe the very same concept and are more in line with the rest of the spec. The datacat implementation of this concept follows the latter, more consistent approach and ignores the entity components of xtdMeasureWithUnit.

Import and Export. Instead of implementing designated import and export routines, we decided to lay focus on implementing a versatile API layer. Application and plug-in developers can use this API to implement use-case oriented solutions that map their data format to the API entities, hence to the underlying ISO specifications. However, a very simple export of the data as CSV is also possible in datacat editor.

4.2 Limitations of the technical solution

As already indicated in **Figure 1**, the datacat API and the datacat editor are already very mature and can be used proactively. However, some "typical IT"-functionalities are still missing:

- Until now, a new instance of datacat had to be created for each additional data catalogue to be managed. This will become obsolete with multi-catalogue support and multi-tenant support.
- The user administration has to be done by GraphQL - there is no UI in datacat editor.
- Until now users are registered by password and automatic but simple Email authentication. A proper "Authentication as a Service" should be implemented.
- We have started with a continuous testing, integration and deployment pipeline (CI/CD) but it is not yet fully operable until now.
- An activity feed (email notification, slack integration) would be very useful for the domain expert groups, that are asynchronously editing data catalogues.

- Other components, not covered in this paper, are datacat CLI for using datacat via the console and datacat explorer, an initial implementation of the datacat editor interface. With the datacat importer excel templates can be imported via the CLI. However, these components are currently not used nor maintained.

4.3 Future research work

Ongoing Validation. The user-facing UI of datacat, datacat editor, has been a tremendous success in user-communication and has allowed us to make more informed decisions in designing the datacat platform as a whole. We will continue our efforts in supporting buildingSMART Germany to improve this part of the system as well. Also, other domains (e.g. facilities management, GIS) and other project types (project setup, existing paper standard, …) will create further demands on usability. We are currently investigating how datacat may be used to create homogenous documents for building permit and landscape planning including BIM, GIS and surveying information. To convince practitioners about the benefit of machine-readable data catalogues we developed a rudimentary Plug-In for Autodesk Revit, that is able to query datacat, annotated any Revit model element with a template for properties and allows the user to fill in the desired values of any property.

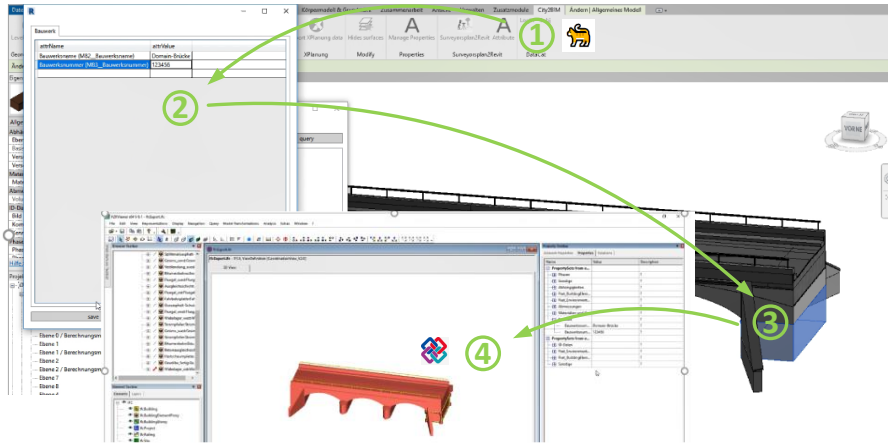

Figure 5. Illustration of the datacat Revit Plugin: (1) connect “read only” to server and browse catalogue (2) link xtdSubject catalogue element to a selected Revit element (3) and type in the desired properties (4) export annotated model to IFC.

Import, Checking, Export. Many more standards of different domains and different standard types should be analysed and imported to datacat. To prove the usability of datacat, the usage of IfcPropertySetTemplate and Construction Objects Data View (CEN,2020) might be feasible to exchange a subset of information that is stored in the datacat catalogue with a number of modelling tools. It allows to delegate the test for validity of a model regarding property assignment to external tools. We will also add checking tools for the catalogue.

Review conceptual model. The ISO12006-3 is currently under development and will be updated. Very similar, but until now, not part of our research is the ISO19110 “Geographic Information – Methodology for feature cataloguing”. This standard is well integrated in the ISO19xxx series but does not perfectly fit BIM requirements. However, its structure and technical specifications on implementing catalogues is worth further investigation.

Semantic Web. In the next stage of research, our main goal for datacat will be a more in-depth examination of the feasibility of a Semantic Web export. We believe that our practical approach in offering tools to amplify and collect expert’s knowledge will be a good foundation in converting this information into RDF graphs. Also, we would like to broaden our efforts in integration of the datacat API in modelling tools to improve usability and performance of the services and give further way for practical application of the components. As a first step we have tested first ontologies using the concepts and OWL-ontologies from (Beetz and de Vries, 2009). In our approach the data is already available in a graph structure (labelled directed graph, Neo4j)

and can be used in the Semantic Web as an ontology to describe real objects. Therefore, a first OWL exporter was written that queries the data from the database via the datacat API and stores it as an ontology in the Turtle syntax. For example, in the context of transport routes, the ontology provides a knowledge base for semantic description and linking of roads and traffic facilities and enables the practical application of the data catalogue.

Acknowledgements

The development of datacat has been financially supported from Federal Ministry for Economic Affairs and Energy (BMWi) during the project "LandBIM" and the project "TerrainTwin". We would also like to thank the German buildingSMART e.V. specialist group "Transport Routes" for insights into their domain model and their continued interest in datacat during this first stage of research and development.

References

Beetz, Jakob and Vries, Bert de (2009): Building product catalogues in the semantic web, 26th International Conference on Information Technology in Construction CIB W78 (pp. 221–226)

Breuss, Raine and Lanzinger, ASI - Propertyserver Patrick (2021): https://github.com/asi-propertyserver/

CEN - European Committee for Standardization (2020), IFC4.COD.1 Construction Objects Data View Part – Candidate, Link: buildingSMART.github.io/ProductData

Francis, Nadime, Alastair Green,Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, Andrés Taylor, Cypher (2018), An Evolving Query Language for Property Graphs, SIGMOD'18, June 10-15, 2018, Houston, TX, USA

FreeBIM (2021): https://www.freebim.at

Thurm, Benjamin, DD-BIM (2021a): https://github.com/dd-bim/datacat

Thurm, Benjamin, DD-BIM (2021b): https://github.com/dd-bim/datacat-stack

Thurm, Benjamin, DD-BIM (2021c): https://github.com/dd-bim/datacat-editor

Tune, Nick (2017). "Manufacturers: BIM objects don't make you BIM ready." Bimplus.co.uk. Chartered Insitute of Building, Bracknell, U.K. Link: https://www.bimplus.co.uk/people/manufactu6rers-bim-objects-dont-ma5ke-bi4m-ready/

Hauer, S.; Murschetz, J.; Bres, A.; Sporr, A.; Schöny, M.; Monsberger, M. (2020) metaTGA: a chance for BIM in the field of MEP. Bau-physik 42, no. 6, pp. 345–351.

International Organization for Standardization (2007). Building construction — Organization of information about construction works — Part 3: Framework for object-oriented information (ISO standard no. 12006-3:2007)

International Organization for Standardization (2016). Geographic information – Methodology for feature cataloguing (ISO standard no. 19110:2016)

International Organization for Standardization (2020a). Building information modelling and other digital processes used in construction — Methodology to describe, author and maintain properties in interconnected data dictionaries (ISO standard no. 23386:2020)

International Organization for Standardization (2020b). Building information modelling (BIM) — Data templates for construction objects used in the life cycle of built assets — Concepts and principles (ISO standard no. 23387:2020)

Werbrouck, J., Senthilvel, M., Beetz, J., Bourreau, P., & Van Berlo, L. (2019). Semantic query languages for knowledge-based web services in a construction context. In P. Geyer, K. Allacker, M. Schevenels, F. De Troyer, & P. Pauwels (Eds.), *Proceedings of the 26th International Workshop on Intelligent Computing in Engineering (EG-ICE)* (Vol. 2394). Leuven, Belgium.