OPM: An ontology for describing properties that evolve over time

Mads Holten Rasmussen, Maxime Lefrançois, Mathias Bonduel, Christian Anker Hviid, Jan Karlshøj

OPM: An ontology for describing properties that evolve over time

Mads Holten Rasmussen[1],
Maxime Lefrançois[2], Mathias Bonduel[3], Christian Anker Hviid[1], and Jan Karlshøj[1]

[1] Technical University of Denmark, Kgs. Lyngby, Denmark
mhoras@byg.dtu.dk
[2] Univ Lyon, MINES Saint-Étienne, Laboratoire Hubert Curien UMR 5516, France
[3] KU Leuven, Dept. of Civil Engineering, Technology Cluster Construction, Ghent, Belgium

Abstract. The W3C Linked Building Data on the Web community group discusses different potential patterns to associate values to properties of building elements. In this paper, we are interested in enabling a different value association method for these and other properties, to account for changes in time, or to annotate a value association with metadata such as provenance, reliability and origin data. Existing ontologies in the Architecture, Engineering and Construction (AEC) industry are reviewed first and we motivate the use of the Smart Energy-Aware Systems (SEAS) ontology as a starting point. Next, we list new competency questions to represent the aforementioned metadata and develop an extension of SEAS named the Ontology for Property Management (OPM). We illustrate the use of OPM with different scenarios where a value association needs to be annotated or updated in a dataset using SPARQL UPDATE queries.

1 Introduction

The W3C Linked Building Data on the Web Community Group (W3C LBD CG)[1] brings together experts in the area of Building Information Modeling (BIM) and Web of Data technologies to define existing and future use cases and requirements for Linked Data based applications across the life cycle of buildings. Of particular interest to this group (and possibly other domains) is the assignment of properties to any feature of interest (FoI) - in this particular case, building-related elements.

Design is an iterative process, and this is, in particular, the case when designing a building. The iterative nature entails that information which is valid at one point in time might no longer be valid in the future, and keeping an overview of information validity hence becomes a cumbersome task. When change management is furthermore handled in a predominantly manual manner by tracking changes in meeting minutes, mail correspondences or as a worst case, in the heads of the individual project participants it constitutes a serious threat to the project execution [5].

Modeling design changes that occur over time is complex as one must define when some FoI is the same as it was before, only with a changed property, and when it is a completely new FoI. Is a particular door, for instance, the same after the width of it has changed? Linked data provides us with the means to allow a concept defined by one party to be extended by other parties, and this is a useful feature in construction projects where most items have interfaces to several different parties from different domains. The door might have a requirement for thermal capacity defined by one party, whereas another party has

defined the fire rating. In those cases, it will cause complications if a FoI is substituted with a new one, and in this regard, it is preferred that the individual property is changed instead. However, changing a property can also cause problems as there are many inter-dependencies between properties of FoIs in a building. Changing a door width influences the heat loss of the room if the thermal resistance of the door is different from its hosting wall. To a certain degree the consequences might not be significant enough to revise the heating system, but as changes add up it might be necessary. Tracking of property evolution history allows designers to relate any derived property to the particular state of the property on which it was derived. Hence, at any time it is possible to evaluate the significance of the design changes and even assess consequences of a design change. In this work, we suggest a modeling approach which allows properties of any FoI to change over time while still keeping track on the history. The scope of the work is the core functionality of OPM, so dealing with derived properties and classification of a property's reliability is not included, although it is covered by the current version of the OPM ontology[2].

2 Ontologies and Patterns to Model Properties

The following ontologies can be used to describe properties, value assignment for properties, and provenance information. The Smart Energy-Aware Systems (SEAS) ontology [7,8] consists of a set of modules together providing terminology to describe physical systems and their interrelations. The core modules related to property management are the seas:FeatureOfInterestOntology and the seas:EvaluationOntology. Together they describe that some FoI can have a property assigned using the seas:hasProperty predicate, and that different evaluations of a same property can be described using the seas:Evaluation class.

The Provenance Ontology [6] provides classes and properties to describe provenance information such as when a prov:Entity was generated, by what prov:Activity if was generated, and who was the prov:Agent that was associated with that activity.

The schema.org ontology is developed as a collaborative, community activity, initiated by the major search engines [2]. It contains an updated version of the GoodRelations ontology [4], one of the main ontologies regarding e-commerce, which is now deprecated. schema.org allows to define quantitative property values by using the schema:value, schema:minValue and schema:maxValue predicates.

From within the W3C LBD CG, a need for a standardized approach towards building-related properties emerged [1]. Future developments aim at proposing both standardized modeling patterns (e.g. by using one or more levels of complexity as demonstrated in Section 2) and predefined, but expendable, lists of building-related properties.

The CDT Datatypes in [9] leverage the Unified Code of Units of Measures UCUM to define a series of RDF Datatypes to encode quantity values. The value and the unit are defined in the same literal with a custom RDF datatype, e.g. `"115 km.h-1"^^cdt:ucum`, or `"0.27 W/(m2.K)"^^cdt:ucum`.

Let `<wall_A>` be a FoI in a building model. At the moment of writing, three potential Linked Data patterns were proposed to the W3D LBD CG [1], each having a different degree of complexity: Level 1 (L1), Level 2 (L2) and Level 3 (L3). Each level number refers to the number of steps/relations between the FoI and the actual object (literal or individual) that encodes the value of its property. The following paragraphs illustrate how these different levels can be used to model the thermal transmittance of wall

element `<wall_A>`, and its main material. Throughout the paper, we use namespace prefixes as provided by the http://prefix.cc/ service.

Level 1: As illustrated in Listing 1, the FoI is directly linked to the UCUM literal that encodes the quantity value of the thermal transmittance of the wall, using a OWL Datatype property. It is also directly linked to the individual that represents material *concrete*, using an OWL Object property.

Listing 1: Level 1 using a cdt:ucum literal.

```
# ontology
ex:thermalTransmittance a owl:DatatypeProperty . ex:mainMaterial a owl:ObjectProperty .
# data
<wall_A> ex:thermalTransmittance "0.27 W/(m2.K)"^^cdt:ucum ; ex:mainMaterial ex:concrete .
```

Level 2: This level explicitly identifies the thermal transmittance property of `<wall_A>` with an intermediate instance of class seas:Property, following the approach defined in the W3C and OGC Semantic Sensor Networks (SSN) ontology [3]. Using SSN, this property instance may be the object of some observation or actuation activity. The SEAS ontology reuses this pattern, but defines OWL Datatype property seas:simpleValue and OWL Object property seas:value to directly link an instance of seas:Property to a literal that encodes its value, or to an individual that encodes its value, respectively [7].

Listing 2: Level 2 using a cdt:ucum literal.

```
# ontology
seas:thermalTransmittance a owl:ObjectProperty ; rdfs:subPropertyOf seas:hasProperty .
ex:mainMaterial a owl:ObjectProperty ; rdfs:subPropertyOf seas:hasProperty .
# data
<wall_A> seas:thermalTransmittance <wall_A#prop> ; ex:mainMaterial <wall_A#mat> .
<wall_A#prop> seas:simpleValue "0.27 W/(m2.K)"^^cdt:ucum .
<wall_A#mat> seas:value ex:concrete .
```

Level 3: SEAS defines an additional level where the link between a property instance and its value can be qualified. This is done using an intermediary object of class seas:Evaluation. The instance of seas:Evaluation can be used to specify the validity context for the value association (e.g. valid during a certain temporal interval), or the type of evaluation (e.g. the maximal operating value). OWL Datatype property seas:evaluatedSimpleValue and OWL Object property seas:evaluatedValue are then used to link an instance of seas:Evaluation to a literal that encodes the evaluated value for the property, or to an individual that encodes this value, respectively [7].

Listing 3: Level 3 using a cdt:ucum literal.

```
# ontology
seas:thermalTransmittance a owl:ObjectProperty ; rdfs:subPropertyOf seas:hasProperty .
ex:mainMaterial a owl:ObjectProperty ; rdfs:subPropertyOf seas:hasProperty .
# data
<wall_A> seas:thermalTransmittance <wall_A#prop> .
<wall_A#prop> seas:evaluation <wall_A#prop-eval1> .
<wall_A#prop-eval1> seas:evaluatedSimpleValue "0.27 W/(m2.K)"^^cdt:ucum.
<wall_A> ex:mainMaterial <wall_A#mat> .
<wall_A#mat> seas:evaluation <wall_A#mat-eval1> .
<wall_A#mat-eval1> seas:evaluatedValue ex:concrete .
```

3 The proposed OPM ontology

This ontology answers a set of competency questions that were identified during interviews with AEC experts. Section 4 lists and answers these competency questions, but for lack of space this section first describes the main terms of the ontology.

Property states. The value of a property can undergo changes over time, e.g. during the building design process or when managing an existing building. The Ontology for Property Management (OPM) enables to describe these changes using L3-modeling of properties of SEAS; reusing concepts from schema.org and PROV-O; and introducing a few classes specific to property management. These classes are all subclasses of opm:PropertyState, which itself is a subclass of seas:Evaluation and defined to differentiate with other types of evaluations as follows. A opm:PropertyState is an evaluation holding the value and metadata about a property that was true for the given time. Metadata must as a minimum be the time of generation stated by prov:generatedAtTime, but preferably also a prov:wasAttributedTo reference to the agent who created the state. Assigning a state to a property is achieved with the OWL Object Property opm:hasPropertyState (sub property of seas:evaluation) which will by its rdfs:range infer that the state is an instance of opm:PropertyState.

Current state and deleted states. So as to ensure efficient management of properties using SPARQL engines, a subclasses of opm:PropertyState is defined to deal with finding the most recent property states: opm:CurrentPropertyState.

PROV-O includes prov:generatedAtTime to indicate the generation time of some resource. Achieving the most recent state can therefore be accomplished by performing a sub-query to first achieve the most recent timestamp and then find the particular opm:PropertyState instance that was generated at this time. However, this query is (a) complex to write and (b) performs poorly. Therefore the opm:CurrentPropertyState class was introduced to explicitly state that a property state is the most recent one. The performance was evaluated by loading 50000 FoIs each having 5 properties with 5 states (5,250,000 triples total) into a triplestore. Two queries (1) by prov:generatedAtTime and (2) by opm:CurrentPropertyState were performed in order to retrieve the latest state of 100 properties. From a cold start (1) returned a result in 6900 ms and (2) in 640 ms, meaning a time reduction of a factor 10. A cold start was also evaluated by redoing each query 10 times and registering the minimum query time. The cold start results were (1) 4780 ms and (2) 630 ms respectively. The tests were performed on local triplestore served on a Lenovo P50 laptop with Intel Core i7-6820HQ 2.70 GHz CPU and 32 GB 2133 MHz DDR ram.

In order to maintain the history of the project and to be able to revert to an earlier state, data should never be removed from the knowledge graph. Using a opm:Deleted marker class enables omission of deleted properties when querying the data store, while they can still be stored in the same database. A deletion is reverted by introducing a new state that inherits the properties of the most recent state.

Property values. OPM does not provide a specific predicate for value assignment, but instead encourages the use of schema:value for single values and, schema:minValue/ schema:maxValue for ranges.

4 Demonstration of property management using OPM

In this section we show how to use OPM for managing properties in combination with SEAS, schema.org, PROV-O and a certain schema defining domain-specific properties, for example the emerging PROPS ontology for the AEC industry. For each of the competency questions below, that have been identified during interviews with AEC experts, a small dataset and example queries were developed and implemented in an online demo.[3]

Competency question 1: How to semantically describe a property such that its value is changeable while its historical record is maintained? Figure 1 illustrates how to assign a property with OPM. When modeling an OPM-compliant L3 property, the property instance must have at least one opm:hasPropertyState relation to a state (entails that the state is an opm:PropertyState class) and the opm:CurrentPropertyState class must be assigned to the most recent state. A state can host any metadata about the property, but should as a minimum have a value and preferably a generation time assigned. In the example (Fig. 1), schema.org is used for the relation between the state and the actual value of the property and PROV-O is used for assigning a generation time and the rdfs:domain of prov:generatedAtTime entails that `<state>` becomes an instance of prov:Entity. Listing 4 shows a complete query to assign an OPM compliant property state to some FoI.

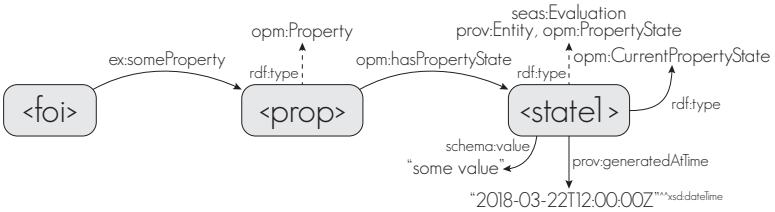

Fig. 1: Modeling a property using states

Listing 4: Insert a new property and an initial property value.

```
INSERT {
  ?foiURI ?prop ?propURI .
  ?propURI opm:hasPropertyState ?stateURI .
  ?stateURI a opm:CurrentPropertyState ;
    prov:generatedAtTime ?now ;
    schema:value ?val .
} WHERE {
  BIND(<wall_A> as ?foiURI)                 # define URI of FoI
  BIND(<wall_A#prop> as ?propURI)           # define URI of Property
  BIND(ex:thermalTransmittance as ?prop)    # define property
  BIND("0.27 W/(m2.K)"^^cdt:ucum as ?val)   # define value
  BIND(<wall_A#state> as ?stateURI)         # define URI of State
  BIND(NOW() as ?now)                       # get current time
  # Do not create a new property instance if the FoI already has it
  MINUS { ?foiURI ?prop ?propURI }
}
```

Competency question 2: How to revise a property value? Making property revisions is done by assigning a new opm:PropertyState to the property instance. The new property must be an instance of opm:CurrentPropertyState and as there cannot be two current states of a property, the class specifying that the previous property state was the current state must be removed (Fig. 2). Listing 5 shows an update query that will handle this.

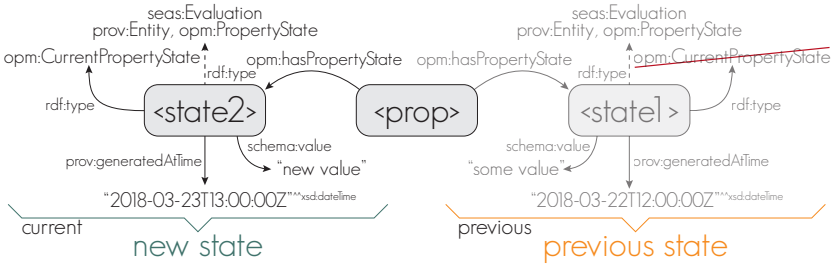

Fig. 2: Revising a property value. Revised state to the left and old property state to the right.

Listing 5: Update a property value.

```
DELETE { ?previousState a opm:CurrentPropertyState }
INSERT {
  ?propURI opm:hasPropertyState ?stateURI .
  ?stateURI a opm:CurrentPropertyState ;
    prov:generatedAtTime ?now ;
    schema:value ?val .
} WHERE {
  BIND(<wall_A#prop> as ?propURI)        # define URI of Property
  BIND("0.25 W/(m2.K)"^^cdt:ucum as ?val) # define new value
  BIND(<wall_A#state2> as ?stateURI)     # define URI for State
  BIND(NOW() as ?now)                    # get current time stamp
  ?propURI opm:hasPropertyState ?previousState .
  ?previousState a opm:CurrentPropertyState ;
    schema:value ?currentVal .           # get value of current state
  FILTER(?val != ?currentVal) # don't update if equal to latest state
}
```

Competency question 3: How to delete a property while still being able to retrieve the history of it and not break all the links to derived properties that depend on it? Deleting a property is done by assigning a new opm:PropertyState to the property instance. The new property state is both an instance of opm:CurrentPropertyState and opm:Deleted, and the opm:CurrentPropertyState class of the previous current state is removed (Fig. 3). Thereby the history is maintained and metadata such as when, why and by whom the property was deleted can be added to the opm:Deleted instance. Listing 6 shows a query for deleting a property in an OPM-compliant way.

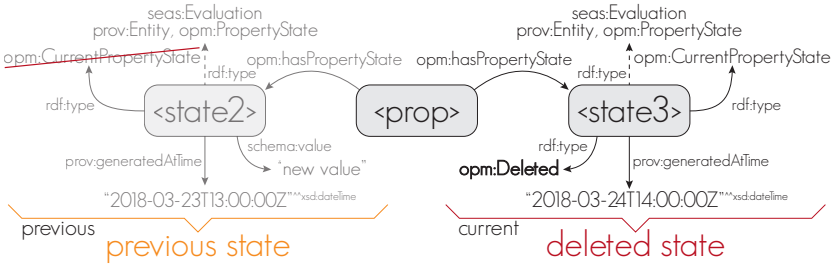

Fig. 3: Deleting a property.

Listing 6: Delete property.

```
DELETE { ?previousState a opm:CurrentPropertyState }
INSERT {
  ?propURI opm:hasPropertyState ?stateURI .
  ?stateURI a opm:CurrentPropertyState , opm:Deleted ;
          prov:generatedAtTime ?now .
} WHERE {
  BIND(<wall_A#prop> as ?propURI)    # define URI of Property
  BIND(<wall_A#state3> as ?stateURI) # define URI of deleted State
  BIND(NOW() as ?now)                # get current time stamp
  ?propURI opm:hasPropertyState ?previousState .
  ?previousState a opm:CurrentPropertyState . # get current state
  # do not delete if the current state is already a opm:Deleted
  MINUS { ?previousState a opm:Deleted }
}
```

Competency question 4: How to restore a deleted property? Restoring a deleted property is done by retrieving the metadata of the most recent property state that is not an instance of opm:Deleted and copy this to a new state (Fig. 4). It requires a sub-query to retrieve the time stamp of such property state (Lst. 7) and as the test in Section 3 revealed, this process is quite resource intensive. However, as it is not an everyday operation it is still acceptable. The reason for creating a new state rather than just deleting the opm:Deleted instance along with its data is to maintain the complete history (incl. deleted states) and record who restored the property, why and when.

Listing 7: Restore property.

```
DELETE { ?previousState a opm:CurrentPropertyState }
INSERT {
  ?propURI opm:hasPropertyState ?stateURI .
  ?stateURI a opm:CurrentPropertyState ;
    prov:generatedAtTime ?now ;
    ?key ?val .
} WHERE {
  BIND(<wall_A#prop> as ?propURI)    # define URI of Property
  BIND(<wall_A#state4> as ?stateURI) # define URI of new State
  BIND(NOW() as ?now)                # get current time stamp
  # get time stamp of most recent property state that was not deleted
  { SELECT ?propURI (MAX(?time) AS ?t)
    WHERE {
          ?propURI opm:hasPropertyState ?s .
            ?s schema:value ?lastVal ;
        prov:generatedAtTime ?time .
      MINUS { ?s a opm:Deleted }
    } GROUP BY ?propURI }
  # get key-value pairs of latest state that is not deleted
  ?propURI opm:hasPropertyState [
    prov:generatedAtTime ?t ;
    ?key ?val ]
  FILTER(?key != prov:generatedAtTime) # filter out time stamps
  # get previous state
  ?propURI opm:hasPropertyState ?previousState .
  ?previousState a opm:CurrentPropertyState .
}
```

Competency question 5: How to retrieve the full history of how the value of a property has evolved over time? The full history is simply retrieved by querying for all seas:PropertyStates of the property. By making it optional for a state to have a schema:value assigned, deleted states are also returned (Lst. 8).

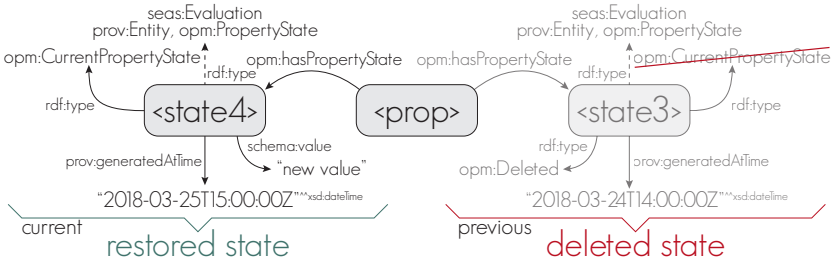

Fig. 4: Restoring a property.

Listing 8: Get property history.

```
SELECT ?dateTime ?value WHERE {
  <wall_A#prop> opm:hasPropertyState ?state .
  ?state prov:generatedAtTime ?dateTime .
  OPTIONAL{ ?state schema:value ?value }
} ORDER BY ?dateTime

### RESULTS
# March 22, 2018 12:00 PM   0.27 W/(m2.K)
# March 23, 2018 1:00 PM    0.25 W/(m2.K)
# March 24, 2018 2:00 PM    -
# March 25, 2018 3:00 PM    0.25 W/(m2.K)
```

Competency question 6: How to retrieve only the latest value of a property? The latest value is retrieved by querying for the opm:PropertyState which is an instance of opm:CurrentPropertyState. The result of the query in Listing 9 is simply `"0.25 W/(m2.K)"^^cdt:ucum`.

Listing 9: Get property value.

```
SELECT ?value
WHERE { <wall_A#prop> opm:hasPropertyState [
   a opm:CurrentPropertyState ; schema:value ?value ] }
```

Competency question 7: How to simplify a complex OPM property (using states) for easier and faster querying? Simplification from L3 to L2 or even L1 can be handled, but will consequently entail some information loss. For both L2 and L1 the property history is lost since only the most recent property state is inferred.

When simplifying to L2 any key-value pair of the most recent state is inferred directly to the property instance node (Fig. 5, yellow). This approach has the advantage that all metadata of the current state of the property such as property unit, provenance data etc. is maintained. It will also still allow for the property value to be specified as a range using schema:minValue and schema:maxValue. The disadvantage is that the property value is still two steps/relations away from the FoI.

When simplifying to L1 the value of the most recent state is inferred directly to the FoI as a datatype property (Fig. 5, red). The advantage is that it becomes very easy and fast to query for the properties of a FoI. Units can still be assigned using custom datatypes but simplifying to L1 comes with some disadvantages. First of all, no metadata can be assigned and hence provenance data is lost and value ranges

are not supported. Further, it will be incorrect to use an owl:ObjectProperty as a owl:DatatypeProperty, and therefore one of the following approaches must be considered: (1) the original property must be described as an rdfs:Property meaning that the dataset becomes less descriptive (RDFS level instead of OWL-DL level) or (2) when simplifying to L1 another predicate (a owl:DatatypeProperty) must be inferred instead. The latter could be handled by adding a suffix to the property URI as illustrated in Fig. 5 and have both an owl:ObjectProperty and owl:DatatypeProperty described in the ontology.

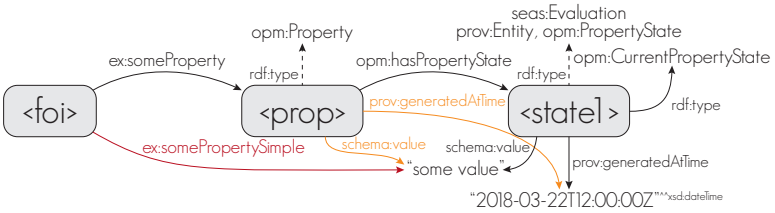

Fig. 5: Simplifying a L3 property to L2 and L1.

Inferring the simplified properties along with the more complex property states makes it easier to query the dataset, and there is no problem in having the data in the same data store. Listing 10 shows an update query that will automatically update all L1 simplifications and a similar approach can be used for L2 simplifications. These queries could be run as a routine job (backward chaining). As an alternative, the same dependency could simply be defined in SWRL rules (forward chaining). The latter has the advantage that there will never be a situation where an outdated property is returned, but it has the cost of a reduced query performance.

Listing 10: Simplify from OPM to simple datatype property.

```
DELETE { ?foi ?p ?simpleValOld }
INSERT { ?foi ?p ?simpleValNew }
WHERE {
  ?foi ?p ?prop .
  ?prop opm:hasPropertyState ?state .
  ?state a opm:CurrentPropertyState ;
    schema:value ?simpleValNew .
  # Get old simplified value (if any)
  OPTIONAL {
    ?foi ?p ?simpleValOld .
    FILTER(?simpleValOld != ?prop) # don't delete L2 property
    FILTER(?simpleValNew != ?simpleValOld) # don't update if unchanged
  }
}
```

5 Conclusions and Future Work

With this work, we propose an extension of the SEAS evaluation ontology with terms specific to tracking properties that evolve over time. We use existing ontologies to describe how to manage property changes of a building element; a wall instance, but OPM is also relevant in any other domain that deal with properties that change over time. The construction industry is rather fragmented, and in a construction project, there are many interdependencies between properties. OPM could be a good foundation for working with derived properties as it allows a derived property to be linked directly to the specific state of its arguments. Further investigation of the potential of OPM in relation to property interdependencies is therefore a future research topic of interest.

OPM can also be used to keep track of changes in BIM models received from other project participants. Communicating with an OPM-compliant SPARQL endpoint directly from a BIM authoring tool to store only state changes of properties could save space and allow insights that comprises an interesting research subject. For legal applications it would be interesting to investigate the use of blockchain technologies to document traceable state changes using OPM. It would also be worth investigating the possibility of having complex and simplified representations of properties co-existing, and using any for answering queries.

Notes

[1]W3C LBD CG - https://www.w3.org/community/lbd
[2]OPM - https://w3id.org/opm
[3]http://www.student.dtu.dk/~mhoras/ldac2018/

References

1. Bonduel, M.: Towards a PROPS ontology (2018), https://github.com/w3c-lbd-cg/lbd/blob/gh-pages/presentations/props/presentation_LBDcall_20180312_final.pdf
2. Guha, R.V., Brickley, D., Macbeth, S.: Schema. org: evolution of structured data on the web. Communications of the ACM **59**(2), 44–51 (2016)
3. Haller, A., Janowicz, K., Cox, S.J.D., Le Phuoc, D., Taylor, K., Lefrançois, M.: Semantic Sensor Network Ontology. W3C Recommendation, W3C (Oct 19 2017), https://www.w3.org/TR/vocab-ssn/
4. Hepp, M.: Goodrelations: An ontology for describing products and services offers on the web. In: International Conference on Knowledge Engineering and Knowledge Management. pp. 329–346. Springer (2008)
5. Kiviniemi, A., Fischer, M.: Requirements management interface to building product models (2004)
6. Lebo, T., Sahoo, S., McGuiness, D.: PROV-O: The PROV Ontology. W3C Recommendation, W3C (Apr 13 2013), https://www.w3.org/TR/prov-o/
7. Lefrançois, M.: Planned ETSI SAREF Extensions based on the W3C&OGC SOSA/SSN-compatible SEAS Ontology Patterns. In: Proceedings of Workshop on Semantic Interoperability and Standardization in the IoT, SIS-IoT, (July 2017)
8. Lefrançois, M., Kalaoja, J., Ghariani, T., Zimmermann, A.: SEAS Knowledge Model. Deliverable 2.2, ITEA2 12004 Smart Energy Aware Systems (2016), 76 p.
9. Lefrançois, M., Zimmermann, A.: The unified code for units of measure in RDF: cdt:ucum and other UCUM datatypes. In: Extended Semantic Web Conference (2018), demonstration paper