Automation in Construction 134 (2022) 104041

Contents lists available at ScienceDirect

Automation in Construction

journal homepage: www.elsevier.com/locate/autcon

Modeling multiple space views for schematic building design using space ontologies and layout transformation operations

Georg Suter *

Design Computing Group, Faculty of Architecture and Planning, TU Wien, Karlsplatz 13, A-1040 Vienna, Austria

ARTICLE INFO

Keywords:
Building information modeling
View definition
Model transformation
Semantic enrichment

ABSTRACT

Modeling multiple views of spaces involves mapping or transformation between multiple models. Automated model transformation is challenging as semantic and spatial criteria need to be considered. This paper proposes a novel method and data processing pipeline to define space views and semi-automatically transform room-based building data created in BIM authoring systems into multi-view space models. The method is based on space ontologies and their integration with space layout transformation operations. It is used to define a set of functional views that are relevant to schematic building design. An existing space modeling system is extended with the method and data processing pipeline. Results from a validation study show that the method can cover specific semantic and spatial aspects of space views. Both are relevant for consistent model transformation and accurate analysis. Results further show that it is feasible to fully automate data processing steps, except for space classification, which is semi-automated.

1. Introduction

Spaces enable indoor activities of users. They also meet needs for accessibility, physical comfort, security, or safety. Many building designers need to create models of spaces and carry out domain-specific analysis tasks that rely on space data. In building design, multiple partial models must be maintained that correspond to domain-specific views while being consistent with each other (see, for example, [1,2]). Modeling multiple views of spaces is a sub-problem of this general problem. It typically involves the mapping or transformation of space data from one domain to another. As an illustrative example, consider functional zoning, which is a crucial aspect of schematic building design. Architectural designers tend to group or aggregate connected spaces with similar functional properties into zones. In residential design, they may create zones for communal spaces (e.g., living rooms), private spaces (e.g., bedrooms), and service spaces (e.g., bathrooms) for activity separation, noise control, or improved access for people or technical services [3,4]. In current building information modeling (BIM) authoring systems, the transformation of a room-based into a zone-based model is done manually [5,6]. Conceptually, it involves identifying merge sets of two or more spaces by considering spatial (space connectivity) and semantic (space function) criteria. Spaces that are too different from their neighbor spaces are not merged. That is, they are single-space zones. Spatial and semantic aspects must also be considered when room-based architectural models are transformed to space models for people circulation, lighting, or air circulation analysis, e.g., by decomposing or selecting spaces. It is preferable to automate such complex model transformation tasks in order to minimize manual modeling effort. It further helps to ensure the consistency of resulting models and accuracy of analysis efforts. While this problem has been studied for individual domains, such as building energy or people circulation analysis (e.g., [7,8]), there is a lack of general multi-view space modeling methods that support model transformation.

In previous work by Suter et al. [9], a method is described to define space views and transform room-based source space data into corresponding multi-view space models. A space view comprises spaces and related objects, such as walls, windows, furnishings, or equipment. According to the method, a view is defined by an operation sequence that transforms a source space layout into a layout that models the view. Layout transformation operations include selection, aggregation, or decomposition of spaces in a layout. For example, aggregation can be used to merge architectural spaces with similar functions into functional zones automatically. The method has been used to define functional views that are relevant for schematic building design [10]. These views include architectural, pedestrian space access, natural lighting, natural ventilation, functional zoning, and functional unit views.

* Corresponding author.
E-mail address: georg.suter@tuwien.ac.at.

https://doi.org/10.1016/j.autcon.2021.104041
Received 31 December 2020; Received in revised form 14 October 2021; Accepted 3 November 2021
Available online 8 December 2021

The existing space view definition method is limited in two ways. First, filters on spaces and related objects are restricted to a small set of hard-coded classes that model only the most common space data categories. These filters are input parameters for layout transformation operations. Second, space data are classified mostly manually.

To address these limitations, the work presented in this paper aims to enhance the existing space view definition method with:

(i.) specific and extensible space semantics,
(ii.) semi-automated space classification, and
(iii.) integration of i. and ii. in layout transformation operations.

Specific and extensible semantics influence the accuracy and coverage of the space view definition method. Examples for specific semantics include the distinction between master bedrooms and regular bedrooms (architectural view), communal and private spaces (functional zoning view), main and side entrances (pedestrian space access view), light-transmissive and opaque space enclosing elements (lighting view), and operable and fixed space enclosing elements (natural ventilation view). With extensible semantics, new classes and properties may be modeled to define additional views or to apply existing view definitions to other building types.

There is a need for automated space classification because manual classification is time-consuming and error-prone. Classification effort increases with the size of source space data and the number of views. It is assumed that perfect classifiers are unattainable. Thus some manual classification will still be necessary.

Existing layout transformation operations need to be modified to accept and process filters that are based on specific semantics. Moreover, operation processing needs to be extended to support manual as well as automated instance classification.

In order to address these needs, a method and data processing pipeline are proposed to define space views and semi-automatically transform room-based space data created in BIM authoring systems into multi-view space models.

The remainder of this paper is structured as follows. Related work is reviewed in Section 2. The research methodology is outlined in Section 3. Space ontologies and layout transformation operations are described in Sections 4 and 5. Definitions for a set of functional views are included in Section 6. Implementation details are described in Section 7. A validation study is presented in Section 8. A summary and outlook is given in Section 9.

2. Related work

2.1. View definition

A view of a building may be defined by a set of functions or functional subsystems and modeled by design objects that provide these functions [2]. This approach reflects the notion that objects are multi-functional. For example, a wall may separate spaces visually and acoustically as well as provide stability. Eastman and Siabiris [11] use composition and specialization methods to model space views. In databases, views are query expressions that are defined based on a database schema [12]. Katranuschkow et al. [13] distinguish domain, ad-hoc, and multi-model views. Domain views are sub-schemas of a larger schema. Ad-hoc views are defined at the instance level. They are related to tasks, such as identifying building elements that are part of the building enclosure. Multi-model views include building model and additional data, such as energy analysis data.

Several domain views, or model view definitions (MVD), exist for the Industry Foundation Classes (IFC) data model [14]. An MVD is a subset of the original IFC schema for specific applications and life-cycle phases [15]. An information delivery manual (IDM) describes exchange requirements and provides process maps for the development of an MVD [16]. MVDs that cover *IfcSpace* and related classes include 'Reference', 'Coordination', 'Space Boundary Addon', 'Basic FM Handover', and 'Architectural Design to Building Energy Analysis' views [17].

Weise et al. [18] describe a schema for the definition of arbitrary IFC model views. Objects may be selected from an original model by user-defined filters. Unnecessary attributes or object references may be removed from the model based on a static model view. This approach is elaborated in a filter framework [19]. Filtering is done at schema, class, instance, and reasoning levels. Except for schema filters, filtering is done at runtime. Geometry conversion may be required. An example is the conversion of sweep to Brep geometry representations. A related effort is mvdXML, which is an XML schema for defining IFC MVDs [20]. It supports the definition of rules on IFC entities and attributes. Tools can use such rules to check the quality of IFC models against an MVD or filter models according to an MVD.

Building model query languages support partial model retrieval at the instance level. PMQL provides query constructs for traversal of large, nested object graphs that are common in building models [21]. Examples include spatial structure hierarchies and space connectivity graphs. Similarly, BIMQL supports the derivation of implicit relationship networks from explicitly modeled relationships [22]. BimSPARQL provides queries on spatial relationships between instances based on their geometric properties. Such queries are useful to evaluate the data quality of IFC building models [23]. Declarative rules are implemented as domain-specific, procedural functions, including spatial topology operators and other geometric modeling functions.

In summary, MVDs focus on data at the schema level. While they support model filtering and specific semantics, such as space function classification properties, they do not address model transformation. Space model transformation operations, as exemplified by the functional zoning scenario, can not be specified in MVDs. Instead, required instance data must be edited in content creation software, such as BIM authoring systems. For this work, MVDs are relevant insofar as MVD-based export functions in BIM authoring systems are used to extract and reuse room-based space data, and to transform them into multi-view space models [5,6]. Compared with MVDs, BIM query languages support more granular, instance-level retrieval of partial models. However, model update functions are low-level and limited to instance deletion or property value updates. Higher-level model transformation operations are not supported.

2.2. Model transformation

Model transformation involves, among other things, the insertion, addition of level of detail, or aggregation of objects [24]. It may be supported at modeling language [11] or operational levels [24]. Baznajac and Kiviniemi [25] distinguish three types of model transformations: data set reduction (e.g., wall geometry simplification), data translation (e.g., detection of exterior walls), and data interpretation (e.g., room area calculations). A typical application is the transformation of a room-based architectural model into a model for another domain, such as circulation analysis or facility management. A common goal is to transform semantic, connectivity, and geometry data while minimizing data loss and errors.

Space transformation is relevant for energy analysis. A thermal zone is a group of spaces with similar heating and cooling requirements. Connected spaces with a similar orientation, internal loads, and occupancy schedules are aggregated into thermal zones to transform an architectural model into an input model for building energy simulation [26]. Zone volumes are created by merging space volumes. Volumes of adjacent spaces must touch to ensure that resulting zone volumes are contiguous. Thermal zoning is a non-trivial task that requires engineering expertise and involves the consideration of semantic as well as spatial criteria. A common thermal zoning method is the perimeter and core zoning method [27]. In this method, spaces that are adjacent to the exterior (that is, perimeter spaces) and interior spaces (that is, core spaces) are merged into separate zones. Perimeter zones are subdivided

according to orientation towards the exterior. The thermal zone model for a rectangular floor plan typically consists of a core and four perimeter zones. Raftery et al. describe a zone typing method that considers spatial and non-spatial zoning criteria [7]. The latter include space function, conditioning systems, and available measured data. Compared with the perimeter and core zoning method, the method results in more granular thermal zoning and promises more accurate energy use prediction. In functional zoning for architectural design, only space functions are considered. Although it is relevant in architectural design guidelines (see, for example, [3,4]), the functional zone concept is currently ill-defined.

In the indoor navigation and building evacuation domains, spaces are decomposed or subdivided to generate navigation models with favorable properties, such as paths that avoid obstructions or paths that model people movement. Common decomposition methods use regular grid [28] or irregular cells. Examples for the latter include Delaunay triangulations of space polygons [29], or generalized Voronoi diagrams of navigable area polygons [30]. Lee et al. [8] propose a rule language for evacuation code checking that provides high-level constructs to check building models against people circulation or space requirements. Visibility-based navigation networks are generated automatically from door, vertical access, and concave points that lie on buffered space boundary polygons [31].

BIM authoring systems, such as Autodesk Revit or Graphisoft Archicad, support space-based analysis functions for several domains, including quantity takeoff, energy analysis, lighting, or route analysis [5,6]. Zones are defined manually by selecting spaces or drawing zone contours. In the former case, zone volumes are created by merging space volumes. BIM data quality assurance systems, such as Solibri Model Checker, validate IFC building models by evaluating rules [32]. There are rules to check the containment of spaces in space groups or evacuation routes. Rule execution requires a combination of semantic and geometry data. For example, maximum evacuation distances or the number of alternative routes from spaces are defined by their function. Moreover, door widths, opening directions, and exit doors must be known.

Several limitations of existing model transformation methods are identified. Zoning methods for thermal domains are not formalized [26]. For example, it is unclear how the similarity of space functions is determined. As a result, the creation of zoning models in building energy simulation tools is a manual task. Similarly, zones are defined manually in BIM authoring systems. By contrast, several automated methods exist to decompose architectural spaces for indoor navigation and evacuation analysis. However, these are domain-specific rather than generic. Model transformation and analysis typically requires extensive classification data. These must be edited manually in BIM authoring or data quality assurance systems.

2.3. Space classification systems

Spaces may be classified by form or function. Examples are atria and offices, respectively. In OmniClass and Uniclass classification systems, space classes are organized in multi-level hierarchies and stored in tables [33,34]. Space objects (instances) created in commercial BIM authoring systems may be labeled according to these classification systems, e.g., to support space planning or management. The IFC schema and the buildingSMART data dictionary service [35] support linking of IFC objects to classification systems.

The application potential of OmniClass and Uniclass space classification systems is currently restricted to spaces that are common in contemporary public or commercial buildings. Moreover, these systems lack a modular structure, are proprietary, and are not meant to be modified or extended by their users. Classification systems with a more flexible structure are preferable. These should be modifiable and extensible in order to meet more specific or unforeseen space classification needs for diverse building types or domains. Current space classification systems do not support automated reasoning about classification data. As a result, such data must be edited manually.

2.4. Semantic enrichment

Semantic enrichment methods derive new data from existing building model data. They are useful to improve data interoperability, simplify queries, link data across domains, or enable compliance checking with standards and regulations [36]. A common approach to semantic enrichment is to process ontology-based building models by a semantic reasoner in order to classify objects or to derive new relations between objects [37,38]. Object data (instances) are modeled as assertions (ABox) that are based on a pre-defined vocabulary or terminology (TBox). Rules (RBox) are based on the same ontology. Rules may be defined for objectified relationships in building models that are based on the ifcOWL ontology in order to simplify query access [37,38]. For example, a rule may state that if two spaces are bounded by a door, then they are connected by it [38]. A benefit of this approach is that it leverages existing ontology and rule languages as well as semantic reasoning and query engines. Moreover, existing ontologies may be reused to develop new ontologies.

Semantic enrichment rules may be encoded in different ways. Lee et al. [39] use OWL class expressions to derive work items from work conditions for construction cost estimation. Room properties, such as room usage, room enclosure, and enclosure materials, are examined to determine wall tiling materials and sizes. De Farias et al. [37] use SWRL rules to simplify the formulation of SPARQL queries on ifcOWL models. SWRL is more expressive than OWL regarding rules for relations [40]. A similar solution is adopted by Pauwels et al. [38] to simplify ifcOWL model graphs.

Several efforts explore rule languages that meet specific modeling needs in the building domain. These include the detection of spatial relations between objects. Staub-French et al. [41] describe an ontology to classify building object features for cost estimation. Belsky et al. [42] propose a semantic enrichment engine to infer new facts about objects in IFC-based building models based on domain-specific rules. The rule language supports operators on properties, relationships, geometry, and spatial topology. This approach is extended to object classification with rules on spatial relationships between objects. It has been validated for bridge components [43]. Zhang et al. [44] outline an algorithm that applies a rule set to a building model in order to detect fall hazards and add fall protection installation. Rules are based on semantic and detailed geometric object data. For example, there is a rule that identifies unprotected sides of holes in slabs.

Bloch and Sacks [45] explore the application of rule-based and supervised machine learning (ML) methods to classify space functions for apartment buildings. Explicit rules are developed based on a set of apartment models and processed by a semantic enrichment engine [42]. Pairwise and single feature rules are defined. An example of the former is the comparison of areas for a pair of spaces and for the latter the minimum number of windows in a space. The classification is done by identifying unique string patterns in feature matrices.

While ontology-based semantic enrichment methods are promising because of their openness and reliance on existing technologies that are part of the semantic web, they are limited regarding spatial reasoning and geometry processing, which is relevant for building modeling. The latter is a strength of domain-specific rule languages. However, such languages are still in the early stages of development.

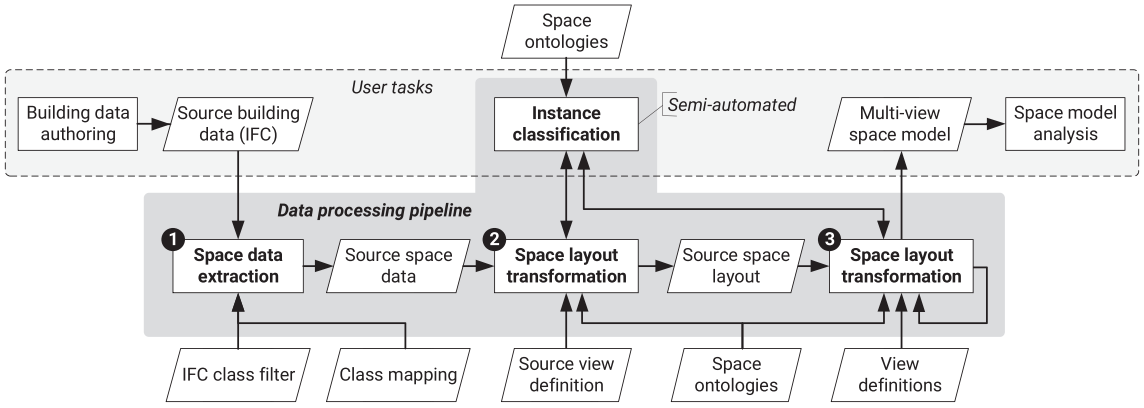


Fig. 1. Data processing pipeline for semi-automated transformation of source building data into multi-view space models.

3. Methodology

A novel method is proposed for defining space views using space ontologies and layout transformation operations. Space view definitions are used in a data processing pipeline to semi-automatically transform source building data into a multi-view space model (Fig. 1). The data processing pipeline is part of a workflow for spatial analysis of building designs. According to this workflow, a user (e.g., a building designer) initially creates room-based source building data using a BIM authoring system and exports these data as an IFC file. The data processing pipeline transforms the content of the IFC file into a multi-view space model and returns the model to the user for analysis. The transformation is automated, except for the instance classification task, which is semi-automated.

In order to support the envisioned data processing pipeline, previously developed layout transformation operations [9] are extended with ontology-based filters that target layout elements or spatial relations between them. This extension enables more specific semantics. As a result, the proposed method can support space views and models that reflect specific spatial and semantic criteria. Moreover, the need for manual instance classification is reduced as layout elements are classified automatically by semantic reasoning.

Compared with existing work, the proposed approach has the following benefits and limitations. In contrast to schema-based view definition methods, such as the MVD method, the proposed view definition method addresses instance-level model transformation. That is, view definitions can be applied to transform room-based source building data into multi-view space models. Unlike space classification systems, space ontologies are modular, extensible, and support automated space classification. Layout operations are generic and may be composed to define different space views, whereas existing automated space transformation methods are limited to specific views, such as evacuation analysis. Layout operations are high-level and are processed based on computational and solid geometry as well as graph search methods. In this work, they are extended with computational logic methods to support specific semantics and instance classification. By comparison, model update operations in BIM query languages are low-level and have limited geometry processing capabilities. Higher-level model transformation operations make it easier to define views than lower-level ones as fewer statements are required. On the other hand, layout transformation operations are restricted to space data, while BIM view definition methods or query languages cover building data more generally.

The data processing pipeline transforms source building data into a multi-view space model in three steps. In the first step, room-based source space data are extracted from source building data. IFC class filters are used to extract geometry data for spaces and related objects from the IFC file. Default class labels are assigned to instances according to a mapping of IFC to space ontologies classes. Relationship data are not extracted because layout operations automatically derive them [46].

In the second step, source space data are transformed into a source space layout based on a source view definition. The transformation involves instance classification, where class labels of layout elements required by targeted views are labeled automatically or manually. Labels are assigned by default according to the IFC to space ontologies class mapping. Additional labels are inferred by semantic reasoning. In case of incorrect labels, a user needs to replace them with the correct ones.

In the third step, the source space layout is transformed into a multi-view space model. This is done by iterating on definitions of targeted views. Each view is defined by a layout operation sequence, which transforms an input layout into an output layout that corresponds to the view. Instance classification may be necessary again in this step to ensure the semantic consistency of the resulting space model.

Key classes and properties of space ontologies are presented for a set of functional views that are relevant for schematic building design and required to model a worked example (Section 4). The ontologies, which comprise a space layout as well as element and function ontologies, are encoded in OWL [47]. Benefits of OWL include expressiveness, semantic reasoner support, and ontology reuse [48]. Alternatively, classes and properties may be encoded in RDF/RDFS and queried using SPARQL [49,50]. OWL is preferred because SPARQL's inference capabilities are limited. Functional views cover architectural, pedestrian circulation, people comfort, and functional zoning domains. Class expression examples highlight the potential for automated classification of layout elements by semantic reasoning. The application of other types of inference rules in OWL or SWRL is left for future work.

Previously developed layout transformation operations are extended to accept and process ontology-based filters. Operation processing is extended with an instance classification step supported by an OWL semantic reasoner (Section 5). Layout operation sequences and corresponding input parameters, including ontology-based filters, are identified for targeted views (Section 6).

Prototypes of space ontologies, definitions of targeted views, extended layout operations, and the data processing pipeline are implemented as extensions of an existing space modeling system (SMS) (Section 7).

The view definition method and data processing pipeline are validated by using SMS to semi-automatically create multi-view space models from room-based source building data authored in BIM systems for targeted views and a sample of floor plans of existing apartment buildings (Section 8). The system's capabilities to support specific space semantics and semi-automatically transform source building data into multi-view space models are measured and evaluated.

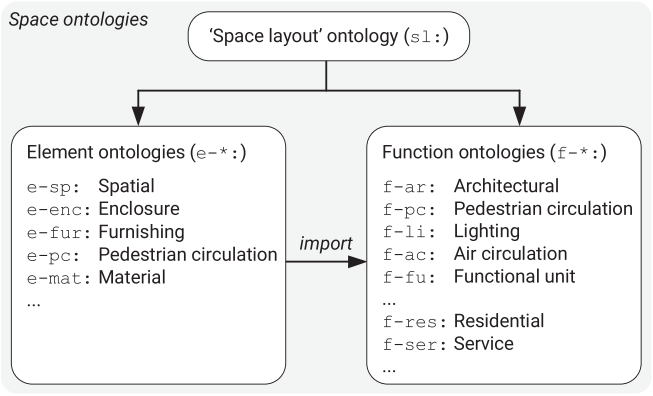


Fig. 2. Space ontologies used to model targeted views and the worked example.

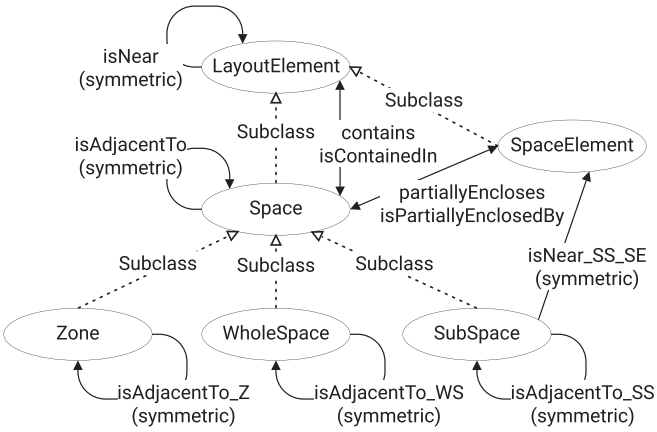


Fig. 3. VOWL diagram [54] of key classes and object properties in the 'Space layout' ontology. The ontology is based on a data model for network-based space layouts [46].

4. Space ontologies

4.1. Overview

Space ontologies are structured as multiple, interrelated ontologies [51]. Fig. 2 shows ontologies that are used to model targeted views and the worked example. Each entity name mentioned in the following has a prefix, which is a shorthand for the entity's ontology. The 'Space layout' ontology (sl: prefix) provides generic entities (classes and properties) for layout elements and spatial relations between them. Layout elements are specialized in element ontologies (e-*: prefix), which model spatial, structural, or material aspects. Functional aspects are covered by function ontologies (f-*: prefix). These reuse entities from 'Space layout' and element ontologies.

Layout elements are multi-functional. For example, a loft space may be used for living, dining, cooking, sleeping, and circulation. Likewise, a glazed door may provide access to a balcony as well as admit natural light. Thus classes in function ontologies generally overlap, that is, they are not disjoint. Overviews of 'Space layout' and element ontologies are given next, focusing on entities required to model target functional views for the worked example. Function ontologies are described together with functional view definitions in Section 6. Key metrics of space ontologies are summarized in Table 1.

4.2. 'Space layout' ontology

The 'Space layout' ontology is based on a data model for network-based space layouts [46]. Formal definitions of layout elements and spatial relations, as well as a comparison with the space data model in IFC, are given in [46]. Key classes and properties of the ontology are shown in Fig. 3. Compared with the 'Building topology' ontology [52,53], the 'Space layout' ontology is more specific regarding subspaces and spatial relations. However, it currently does not cover nested, multi-level spatial structures or alignment with related ontologies, such as ifcOWL.

A space layout consists of layout elements, which are either spaces (sl:Space) or space elements (sl:SpaceElement). Physical objects that either partially enclose spaces, such as walls or windows, or are contained in them, such as furniture, are examples for space elements. A whole space (sl:WholeSpace) is a space that contains zero or more subspaces sl:SubSpace). The latter are similar to partial spaces in IFC, which are modeled as instances of the *IfcSpace* class with attribute value *CompositionType=PARTIAL*.

Multiple types of spatial relations between layout elements are modeled in a layout. These form a spatial relation network (SRN). Each layout element is an SRN node that is located in the layout's local coordinate system. The location typically corresponds to the centroid of the layout element's shape. Spatial relations are edges in the SRN. Fig. 3 includes containment, adjacency, proximity, and partial enclosure relations. Specializations of spatial relations are modeled as subproperties of object properties. For example, the space adjacency relation (sl:isAdjacentTo) relation is specialized into an adjacency relation between whole spaces (sl:isAdjacentTo_WS) and an adjacency relation relation between subspaces (sl:isAdjacentTo_SS).

Table 1
Space ontologies metrics.

Metric	
Class count	184
Class axioms	
SubClassOf count	192
EquivalentClasses count	35
DisjointWith count	5
ObjectProperty count	17
DataProperty count	16
DL expressivity	ALCHIQ(D)

4.3. Element ontologies

Entities for spatial, enclosure, pedestrian circulation, and furnishing elements are defined in element ontologies. A class hierarchy for element ontologies is shown in Listing 1. Indents indicate sub/super class relationships and prefixes the ontologies in which classes are defined (Fig. 2). All classes are sub-classes of either sl:Space or sl:SpaceElement classes.

Classes that are marked ¶ in Listing 1 are defined by class expressions. That is, membership of instances (individuals) in these classes is derived from existing data. For example, an internal space (e-sp:InternalSpace) is equivalent to a space where the sl:isInternal data property is true. The corresponding class expression in OWL Manchester syntax [55] is included in Listing 2. An internal space is further a spatial element (e-sp:SpatialElement). External air spaces (e-sp:ExternalAirSpace) must be modeled explicitly for people comfort views. Internal enclosure space elements (e-enc:InternalEnclosureSE) enclose internal spaces but are not part of the building enclosure. Rooms, halls, floors, atria, wings, or sections are further e-sp:SpatialElement examples to model space forms or the spatial organization of buildings. As they are not essential for the targeted views, these classes are excluded from

Listing 1.

Listing 1
Classes in element ontologies that are relevant for definitions of targeted views and the worked example. Classes marked ¶ are defined by class expression.

```
sl:Space
  e-sp:SpatialElement
    e-sp:InternalSpace ¶
    e-sp:ExternalSpace ¶
      e-sp:ExternalAirSpace
      e-sp:ExternalGroundSpace
sl:SpaceElement
  e-enc:EnclosureSE
    e-enc:Window
        e-enc:OperableWindow
        e-enc:FixedWindow
    e-enc:Door
        e-enc:RegularDoor
        e-enc:UnitDoor
    e-enc:Opening
    e-enc:InternalEnclosureSE ¶
  e-pc:PedCircSE
    e-pc:FlightOfStairs
    e-pc:Landing
  e-fur:FurnishingSE
    e-fur:Cabinet
    e-fur:SanitaryElement
```

Listing 2
Class expression for internal spaces in OWL Manchester syntax [55].

```
Class: e-sp:InternalSpace
EquivalentTo:
  sl:Space
  and (sl:isInternal value true)
```

5. Space layout transformation

5.1. Operations

Layout selection, aggregation, decomposition, and update operations are used to define targeted views. In each operation, a copy of an input layout (L_{in}) is modified and returned as an output layout (L_{out}). An L_{out} may be passed as an L_{in} to a subsequent operation. Operations may thus be composed into operation sequences.

5.1.1. Filters

Operation input parameters include filters. There are two types of filters: layout element and SRN filters. A layout element filter targets layout element properties. Ontology-based layout element filters may target the *classes* property, which lists a layout element's class labels. For example, filter

$$F_S = \{\forall s \in S \mid \text{e} - \text{sp} : \text{InternalSpace} \in s.\text{classes}\}$$

targets set S of spaces in a layout. A given space s passes F_S if the *classes* property of s includes the e-sp:InternalSpace class. The formal definition of e-sp:InternalSpace is encoded in the 'Spatial element' ontology and referenced by F_S. An SRN filter defines a subnetwork of a layout's SRN. It consists of a node (F_N) and an edge (F_E) filter. Node filters are analogous to layout element filters. Edge filters target spatial relation element properties.

5.1.2. Selection

In the *select* operation, layout elements are selected from an L_{in} based on layout element filters. For example, internal spaces and doors may be selected from an L_{in} using filters F_S and F_{SE} that target spaces or space elements where the *classes* property includes e-sp:InternalSpace or e-enc:Door, respectively.

5.1.3. Aggregation

In the *aggregate* operation, spaces in an L_{in} are merged and replaced by new spaces. They must belong to the same space group, as defined by a grouping criterion, and be connected in a subnetwork of the SRN of L_{in}, as defined by an SRN filter (F_N, F_E). For example, spaces in an L_{in} may be merged if they are circulation spaces (grouping criterion) and if they are connected in L_{in}'s space adjacency network (SRN filter). Each space in L_{out} is labeled as a zone (sl:Zone). Space volumes are modeled as solid boundary representations (Breps). Volumes of adjacent spaces must touch such that they can be merged into larger volumes by solid union. A pair of spaces are connected if there is a path between them in the SRN subnetwork. Moreover, all space nodes in that path must belong to the same group. Group membership of a space s may be determined based on its *classes* property and a given set $G = \{g_1, g_2, \ldots g_n\}$ of grouping space classes. That is, s is assigned to the grouping space class $g \in G$ with the greatest grouping weight (sl:weight_grouping) in $s.classes \cap G$ (grouping criterion). For example, a living room, which also serves as a circulation space because it connects a kitchen and a dining room, would be assigned to the group of living rooms if that group has a greater weight than the group of circulation spaces.

5.1.4. Decomposition

In the *decompose* operation, whole spaces in an L_{in} are divided into smaller spaces. Optionally, the latter either replace existing whole spaces or are inserted as new subspaces that are contained in them. Input parameters include a filter F_{WS_d} on whole spaces that are to be decomposed as well as a decomposition method. For example, whole spaces in an L_{in} may be decomposed into subspaces that are located near doors and openings by a method that is based on Voronoi cells [46]. In general, a Voronoi diagram and its dual, the Delaunay triangulation, encode proximity between given sites [56]. In the above example, positions of subspaces are used as sites. Suppose they already exist in L_{in}. In that case, these subspaces may be targeted explicitly by a filter F_{SS}, or they may be inserted before decomposition, based on a filter F_{S_i} that targets nearby space elements or other subspaces. Subspace volumes are created by intersecting corresponding Voronoi cells with the volume of the containing whole space. A further example for a whole space decomposition method is convex decomposition [57]. According to this method, whole spaces with concave volumes are decomposed into spaces with convex volumes.

5.1.5. Update

In the *update* operation, distances in an SRN subnetwork are derived based on source, destination, and SRN filters. There are two variants of the operation. The $update_{t=distanceNearest}$ operation derives distances in an SRN subnetwork of L_{in} based on source node (F_{N_s}), destination node (F_{N_d}), and SRN filters (F_N, F_E). Distance properties of source nodes targeted by F_{N_s} are updated. For each source node, the distance is computed to the nearest node that passes the destination node filter F_{N_d}. Shortest path search is restricted to an SRN subnetwork of L_{in}, as defined by filters F_N on nodes and F_E on edges. Edge directions are not considered in the search.

The $update_{t=distanceMustPass}$ operation derives distances in an SRN subnetwork of L_{in} based on must-pass node (F_{N_m}), source node, destination node, and SRN filters. Distance properties of must-pass nodes, which are targeted by filter F_{N_m}, are updated. For each must-pass node,

the shortest path is computed between its nearest, distinct source and destination nodes, where paths must pass through that node. The corresponding distance is thus referred to as the must-pass distance. Source and destination nodes are targeted by filters F_{N_s} and F_{N_d}. Shortest path search is restricted to an SRN subnetwork defined by filters F_N and F_E.

5.2. Processing

The processing of a layout transformation operation involves the insertion, removal, or modification of layout elements in L_{out} [46]. Spatial inconsistencies in L_{out} are resolved automatically. For example, when a pair of adjacent spaces are merged by the *aggregate* operation, doors that connect the spaces are excluded from L_{out} because they are considered spatially inconsistent. The SRN of L_{out} is regenerated to reflect such modifications.

An optional instance classification step may be invoked after an operation has been processed to semi-automatically edit class labels of layout elements (instances) in L_{out} (Fig. 4). An OWL semantic reasoner supports instance classification. The reasoner accepts as input an ontology that includes instances as well as facts (ABox) about them regarding class membership and properties according to space ontologies (TBox, RBox). The reasoner realizes the ontology. That is, it computes all instances for named classes [58]. The reasoner exports inferred classes of instances as a new ontology.

In the instance classification step, class labels may be edited manually before and after instance classification by the reasoner. For example, after processing an *aggregate* operation, users may remove existing class labels of merged spaces or insert missing labels to ensure the consistency of reasoner input data. Similarly, after instance classification by the reasoner, inferred class labels that are inconsistent may be removed, or missing labels may be inserted. Label edits after classification by the reasoner may be necessary because class hierarchies, class expressions, and other rules in space ontologies are not expected to cover all conceivable cases.

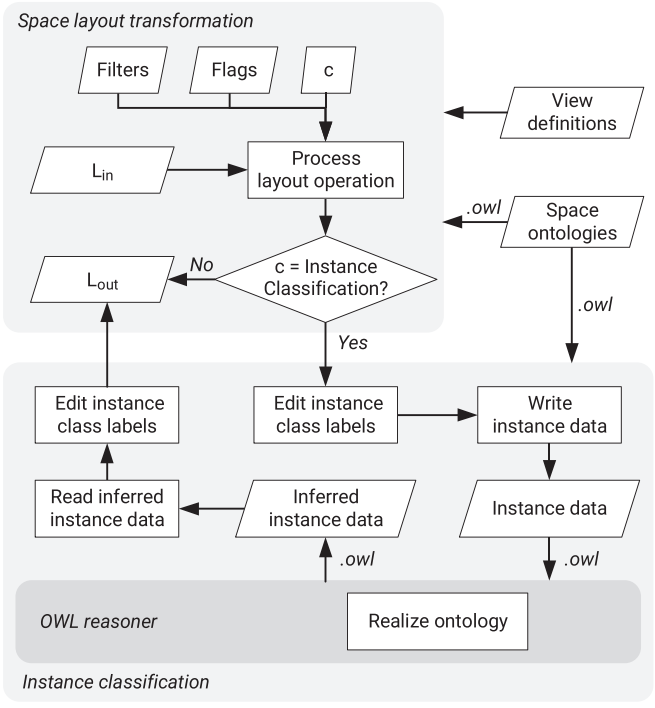


Fig. 4. Flowchart for space layout transformation and semi-automated instance classification. L_{in}: input layout, L_{out}: output layout, c: instance classification flag.

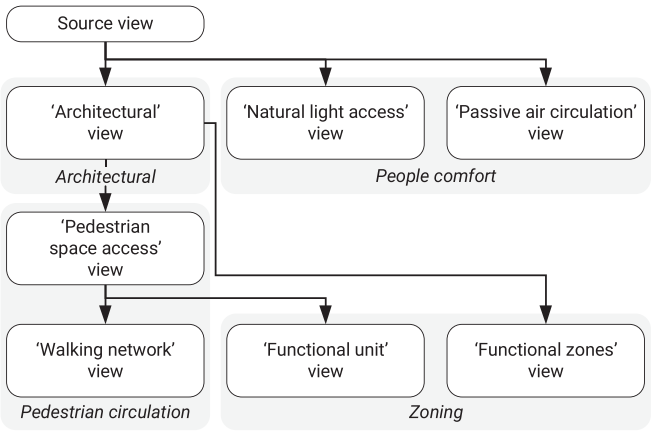


Fig. 5. Dependencies between the source view and targeted views.

6. View definitions

6.1. Overview

Space ontologies and layout transformation operations extended by ontology-based filters are used to develop definitions for a set of functional views that are relevant to schematic building design. These views cover architectural, pedestrian circulation, people comfort, and functional zoning domains. Each view is either directly or indirectly derived from a source view, which is modeled by a source space layout (Fig. 5). For example, the 'Walking network' view is derived from the 'Pedestrian space access' view, which in turn is derived from the 'Architectural' view.

Targeted views are demonstrated with a worked example. Using the SMS system described in Section 7, room-based source building data for a regular floor of an existing, freestanding apartment building were transformed into a space model for the targeted views (Fig. 6, [4]). The example floor has features that are challenging from a modeling perspective. Each of the two apartments on the floor has a main and a side entrance. The latter connects kitchens with a service elevator that is separated from the stairway and the main elevator. There are two small spaces with folding doors near the kitchens. These are interpreted as box rooms, which are private spaces for domestic workers or storage.

6.2. Source view

The source view definition specifies the transformation of source space data into a source space layout L_s in the second step of the data processing pipeline (Fig. 1). Source space data lack an SRN and have only default class labels, which are assigned according to a mapping of IFC to space ontologies classes (Table 2).

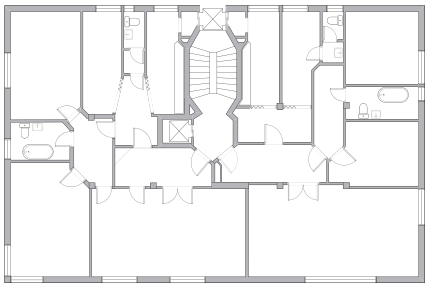


Fig. 6. Example floor of an apartment building designed by L. Caccia Dominioni, Milan, 1961.

Table 2
Mapping of IFC to space ontologies classes. The latter are assigned as default class labels to layout elements. Incorrect default labels may be edited in the instance classification step.

IFC class	Space ontologies class
IfcSpace	sl:WholeSpace, sl:InternalSpace
IfcDoor	e-encl:RegularDoor
IfcWindow	e-encl:OperableWindow
IfcStairFlight	e-pc:FlightOfStairs
IfcSlab	e-pc:Partition
IfcFurnishingElement	e-fur:Cabinet
IfcFlowTerminal	e-fur:SanitaryElement
IfcSanitaryTerminalType	e-fur:SanitaryElement

Source space data are semi-automatically transformed into an L_s that is semantically and spatially consistent to generate space models according to definitions of targeted views. The source view is defined as follows:

$L_s \leftarrow \text{select}_{(F_S,\ F_{SE}),\ c}\ (\text{SSD})$

where

— *SSD* are source space data,
— L_s is the source space layout,
— $F_S = \{\forall\ s \in S\}$ is a filter on spaces,
— $F_{SE} = \{\forall\ se \in SE\}$ is a filter on space elements, and
— c = *InstanceClassification* is an enumeration type flag.

Sets S of spaces and SE of space elements in source space data are targeted by layout element filters F_S and F_{SE} to create L_s. Instance classification is invoked for manual editing of layout element class labels and automated inference of additional labels by semantic reasoning (Section 5.2, Fig. 4). In general, it is sufficient for users to label only the most specific child (or leaf) classes in class hierarchies as their parent classes are inferred. Moreover, certain classes are inferred based on class expressions.

In order to model specific semantics for L_s for the example floor, it was necessary to insert 19 new labels and remove 9 default labels. For example, two e-enc:UnitDoor labels were inserted and two e-enc:RegularDoor labels were removed. The semantic reasoner processed 121 input labels and returned 1,225 labels. That is, label edits are 23% of input labels, and the reasoner output to input label ratio is approximately 10:1.

Table 3
Functional view definitions. Input parameters for *update* operations are defined in Tables 4 and 5.

View/Layout transformation	Input parameters
'Architectural' [Fig. 7]: $L_{\text{ar}} \leftarrow \text{select}_{(F_S,F_{SE}),\ c}\ (L_s)$	L_s: Source layout $F_S = \{\forall\ s \in S \mid \text{f} - \text{ar} : \text{ArchitecturalSpace} \in s\ .\ classes\}$: filter on spaces $F_{SE} = \{\forall\ se \in SE \mid \text{f} - \text{ar} : \text{ArchitecturalSE} \in se\ .\ classes\}$: filter on space elements c = *NoInstanceClassification*: enumeration type flag
'Pedestrian space access' [Fig. 8a]: $L_{\text{psa}} \leftarrow \text{select}_{(F_S,F_{SE}),\ c}\ (L_{\text{ar}})$ $L_{psa} \leftarrow update_{t=distanceNearest,\ldots}\ (L_{psa})$ [Table 4]	L_{ar}: 'Architectural' layout, L_{psa}: 'Pedestrian space access' layout $F_S = \{\forall\ s \in S \mid \text{f} - \text{pc} : \text{PedCircSpace} \in s\ .\ classes\}$: filter on spaces $F_{SE} = \{\forall\ se \in SE \mid \text{f} - \text{pc} : \text{PedCircSE} \in se\ .\ classes\}$: filter on space elements c = *NoInstanceClassification*: enumeration type flag
'Walking network' [Fig. 8b]: $L_{\text{wn}} \leftarrow \text{decompose}_{F_{\text{WS}_d},\ m_1,\ i,\ c}\ (L_{\text{psa}}) L_{\text{wn}} \leftarrow \text{decompose}_{F_{\text{WS}_d},F_{S_i},\ m_2,\ i,\ c}\ (L_{\text{wn}})$ $L_{wn} \leftarrow update_{t=distanceNearest,\ldots}\ (L_{wn})$ [Table 4]	L_{psa}: 'Pedestrian space access' layout, L_{wn}: 'Walking network' layout $F_{\text{WS}_d} = \{\forall \text{ws} \in \text{WS}\}$: filter on whole spaces to be decomposed $F_{S_i} = (F_{\text{SE}},\ F_{\text{SS}})$: space insertion filters —$F_{SE} = \{\forall\ se \in SE\}$: filter on space elements —$F_{SS} = \{\forall\ ss \in SS\}$: filter on subspaces m_1 = *ConvexVolumes*, m_2 = *VoronoiCells*, i = *InsertAsSubSpaces*, c = *NoInstanceClassification*: enumeration type flags
'Natural light access' [Fig. 8c]: $L_{\text{nla}} \leftarrow \text{select}_{(F_S,F_{SE}),\ c}\ (L_s)$ $L_{nla} \leftarrow update_{t=distanceNearest,\ldots}\ (L_{nla})$ [Table 5]	L_s: Source layout, L_{nla}: 'Natural light access' layout $F_S = \{\forall\ s \in S \mid (\text{f} - \text{li} : \text{NatLightSpace} \in s\ .\ classes) \wedge (\text{f} - \text{li} : \text{NatLightSpaceException} \notin s\ .\ classes)\}$: filter on spaces $F_{SE} = \{\forall\ se \in SE \mid \text{f} - \text{li} : \text{NatLightSE} \in se\ .\ classes\}$: filter on space elements c = *NoInstanceClassification*: enumeration type flag
'Passive air circulation' [Fig. 8d]: $L_{\text{pac}} \leftarrow \text{select}_{(F_S,F_{SE}),\ c}\ (L_s)$ $L_{pac} \leftarrow update_{t=distanceMustPass,\ldots}\ (L_{pac})$ [Table 5]	L_s: Source layout, L_{pac}: 'Passive air circulation' layout $F_S = \{\forall\ s \in S \mid (\text{f} - \text{ac} : \text{PassiveAirCircSpace} \in s\ .\ classes) \wedge (\text{f} - \text{ac} : \text{PassiveAirCircSpaceException} \notin s\ .\ classes)\}$: filter on spaces $F_{SE} = \{\forall\ se \in SE \mid (\text{f} - \text{ac} : \text{PassiveAirCircSE} \in se\ .\ classes) \wedge (\text{f} - \text{ac} : \text{PassiveAirCircSEException} \notin se\ .\ classes)\}$: filter on space elements c = *NoInstanceClassification*: enumeration type flag
'Functional units' [Fig. 8e]: $L_{\text{fu}} \leftarrow \text{aggregate}_{F_N,\ F_E,\ g,\ c}(L_{\text{psa}})$	L_{psa}: 'Pedestrian space access' layout $F_N = (F_S, F_{SE})$: SRN node filters —$F_S = \{\forall\ s \in S \mid \text{f} - \text{pc} : \text{PedCircSpace} \in s\ .\ classes\}$ on spaces —$F_{SE} = \{\forall\ se \in SE \mid (\text{f} - \text{pc} : \text{PedCircSE} \in se\ .\ classes) \wedge (\text{f} - \text{pc} : \text{FUAccessSE} \notin se\ .\ classes)\}$ on space elements $F_E = \{\forall\ pe_{se,s} \in PE_{SE,S}\}$: SRN edge filter on the sl : partiallyEncloses relation between space elements and spaces g = *NoGrouping*, c = *InstanceClassification*: enumeration type flags
'Functional zones' [Fig. 8f]: $L_{\text{fz}} \leftarrow \text{aggregate}_{F_N,\ F_E,\ g,\ c}(L_{\text{psa}})$	L_{psa}: 'Pedestrian space access' layout $F_N = \{\forall\ s \in S\}$: SRN node filter on spaces $F_E = \{\forall\ a_s \in A_S\}$: SRN edge filter on the sl : isAdjacentTo relation between spaces g = *GroupingByClassesProperty*, c = *NoInstanceClassification*: enumeration type flags

Listing 3
Class hierarchy for the 'Architectural' view.

```
sl:Space
  f-ar:ArchitecturalSpace
    f-res:ResidentialSpace
      f-res:CommunalSpace
        f-res:LivingRoom
      f-res:PrivateSpace
        f-res:Bedroom
        f-res:BoxRoom
      f-res:Hallway
    f-ser:ServiceSpace
      f-ser:Kitchen
      f-ser:Bathroom
    f-pc:PedCircSpace
      f-pc:Stairway
      f-pc:Elevator
      f-pc:Hallway
sl:SpaceElement
  f-ar:ArchitecturalSE
    e-enc:EnclosureSE
    e-pc:PedCircSE
    e-fur:FurnishingSE
```

6.3. 'Architectural' view

Architectural spaces (f-ar:ArchitecturalSpace) and space elements (f-ar:ArchitecturalSE) are selected from L_s to create a layout L_{ar} that models the 'Architectural' view. The view definition is included in Table 3. Classes that are required by this view to model the example floor are included in Listing 3. A visualization of L_{ar} for the floor is shown in Fig. 7. Spaces and space elements are labeled according to function and element ontologies, respectively. Each layout element is a member of multiple classes. For clarity, only primary classes are shown in Fig. 7. The primary class of a layout element corresponds to the class in its *classes* property with the greatest class weight (sl:weight_class).

Architectural spaces and space elements, which are targeted by filters F_S and F_{SE} in the *select* operation, are inferred from their sub-classes in the class hierarchy in Listing 3. These inferences are made when instance classification is invoked to transform source space data into L_s (Section 6.2). For example, f-ar:ArchitecturalSpace is inferred from f-res:

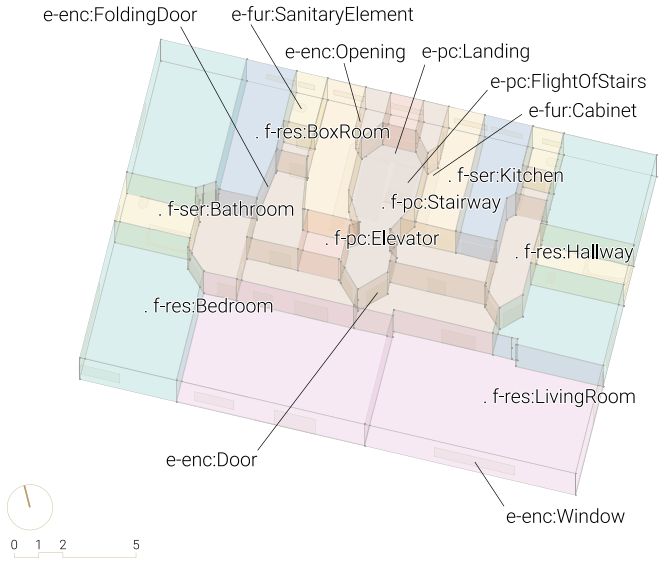


Fig. 7. Space model for targeted views and example floor. Layout L_{ar} models the 'Architectural' view. Sample class labels of spaces and space elements are shown from function (f-*:) and element (e-*:) ontologies. L_{ar} was generated using the SMS system (Sections 7 and 8.1).

LivingRoom, whereas f-ar:ArchitecturalSE is inferred from subclasses of e-enc:EnclosureSE, such as e-enc:Window. Other views similarly rely on inferring class membership of layout elements from class hierarchies.

6.4. Pedestrian circulation views

6.4.1. 'Pedestrian space access' view

Pedestrian circulation spaces (f-pc:PedCircSpace) and space elements (f-pc:PedCircSE) are selected from L_{ar} to create a layout L_{psa} that models the 'Pedestrian space access' view (Table 3, Fig. 8a, Listing 4). An L_{psa} has a space access network, which is a subnetwork of its SRN. Its nodes are pedestrian circulation spaces or space elements, and its edges elements of the partial enclosure relation (sl:partiallyEncloses) between space elements and spaces.

The space access network of the example is colored according to the distance from each space or space element to the nearest main entrance (f-pc:MainEntranceSE). The path length is used as a distance measure. Distances are computed by an *update* operation of the space access network. Details about filters and additional parameters used in this operation are given in Table 4. There are two main entrances in the example that provide access to residential units. The maximum distance or depth [59] from the most remote space to the nearest main entrance is 9 (5 spaces). Hallways in each apartment form cycles, which imply alternative paths between communal, private, and domestic spaces [59]. When distances are computed, side entrances (f-pc:SideEntranceSE) are considered as inaccessible in order to exclude paths that, for example, pass from residential through main circulation spaces to reach a main entrance. Inaccessibility is indicated by the ∞ symbol in Fig. 8a.

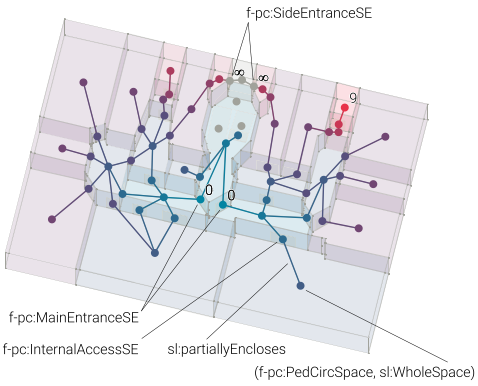
Main or side entrances must be labeled explicitly in L_s. On the other hand, internal access space elements (f-pc:InternalAccessSE) are inferred (Listing 5). A door or opening (e-enc:Opening) must meet two conditions to be classified as an internal access space element. First, it must be an internal enclosure space element. This excludes doors or openings that are connected to the exterior. Second, elevations relative to the spaces connected by an internal access space element must not exceed a certain threshold.

Listing 4
Class hierarchy for 'Pedestrian space access' and 'Walking network' views. Classes marked ¶ are defined by class expression.

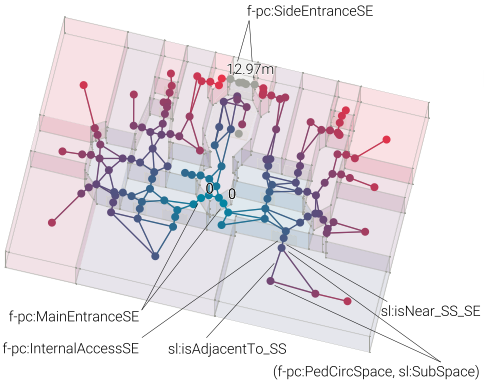
```
sl:Space
  f-pc:PedCircSpace
    e-sp:InternalSpace
sl:SpaceElement
  f-pc:PedCircSE
    e-pc:PedCircSE
    f-pc:PedAccessSE
      f-pc:EntranceSE
        f-pc:MainEntranceSE
        f-pc:SideEntranceSE
      f-pc:InternalAccessSE ¶
```

Listing 5
Class expression for internal access space elements.

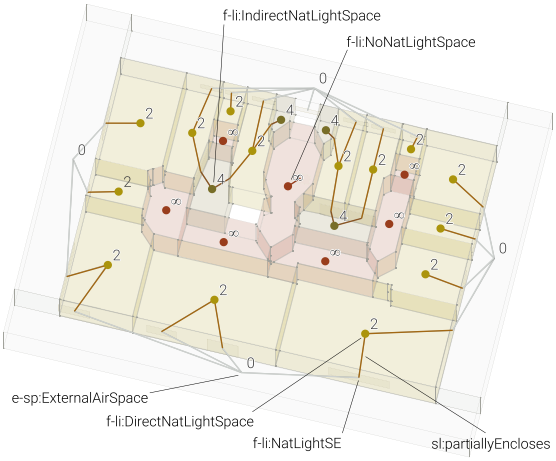
```
Class: f-pc:InternalAccessSE
EquivalentTo:
  (e-enc:Door or e-enc:Opening)
  and e-enc:InternalEnclosureSE
  and (sl:elevationFromFloorSpace1 some
    xsd:float[>= -0.2f , <= 0.2f])
  and (sl:elevationFromFloorSpace2 some
    xsd:float[>= -0.2f , <= 0.2f])
```



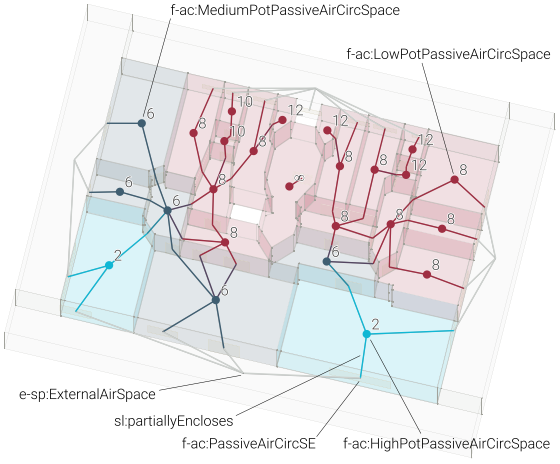
(a) 'Pedestrian space access' layout L_{pac}.



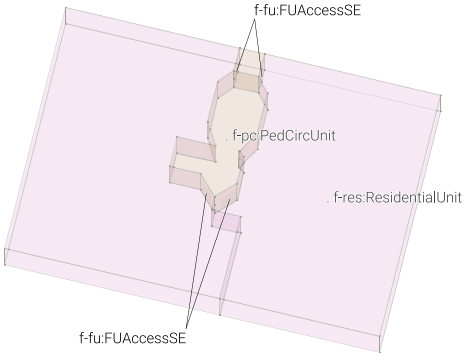
(b) 'Walking network' layout L_{wn}.



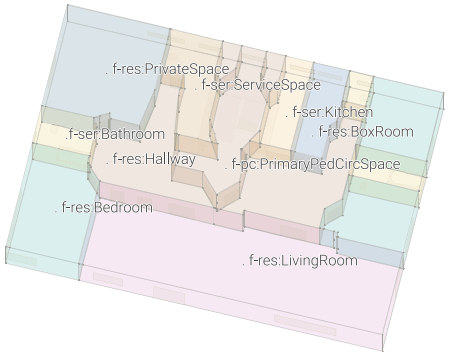
(c) 'Natural light access' layout L_{nla}.



(d) 'Passive air circulation' layout L_{pac}.



(e) 'Functional unit zones' layout L_{fuz}.



(f) 'Functional zones' layout L_{fz}.

Fig. 8. Space model for targeted views and example floor. (a)-(f): Each layout models a view. Sample class labels of layout elements and spatial relations are shown from space layout (sl:), element (e-*:), and function (f-*:) ontologies. The model was created in the SMS system by transformation of room-based source building data of the example floor (Sections 7 and 8.1). (a) 'Pedestrian space access' layout L_{pac}. (b) 'Walking network' layout L_{wn}. (c) 'Natural light access' layout L_{nla}. (d) 'Passive air circulation' layout L_{pac}. (e) 'Functional unit zones' layout L_{fuz}. (f) 'Functional zones' layout L_{fz}.

Table 4
Updates of pedestrian circulation views.

View/Layout update	Input parameters
'Pedestrian space access' [Fig. 8a]: $L_{psa} \leftarrow$ update$_{t,\ F_{N_s},\ F_{N_d},\ F_N,\ F_E,\ c,\ m}$ (L_{psa})	L_{psa}: 'Pedestrian space access' layout $t = DistanceNearest$: enumeration type flag $F_{N_s} = (F_S,\ F_{\text{SE}_1})$: SRN source node filters —$F_S = \{\forall\ s \in S\}$: filter on spaces —$F_{\text{SE}_1} = \{\forall \text{se} \in \text{SE}\}$: filter on space elements $F_{N_d} = \{\forall \text{se} \in \text{SE} \mid \text{f} - \text{pc} : \text{MainEntranceSE} \in \text{se.classes}\}$: SRN destination node filter on space elements $F_N = (F_S,\ F_{\text{SE}_2})$: SRN node filters —$F_S = \{\forall\ s \in S\}$: filter on spaces —$F_{\text{SE}_2} = \{\forall \text{se} \in \text{SE} \mid \text{f} - \text{pc} : \text{SideEntranceSE} \notin \text{se.classes}\}$: filter on space elements $F_E = \{\forall\ pe_{se,s} \in PE_{SE,S}\}$: SRN edge filter on the sl: partiallyEncloses relation between space elements and spaces $c = NoInstanceClassification$: enumeration type flag $m = PathLength$: enumeration type flag
'Walking network' [Fig. 8b]: $L_{wn} \leftarrow$ update$_{t,\ F_{N_s},\ F_{N_d},\ F_N,\ F_E,\ c,\ m}$ (L_{wn})	L_{wn}: 'Walking network' layout $t = DistanceNearest$: enumeration type flag $F_{N_s} = (F_{\text{SS}},\ F_{\text{SE}_1})$: SRN source node filters —$F_{SS} = \{\forall\ ss \in SS\}$: filter on subspaces —$F_{\text{SE}_1} = \{\forall \text{se} \in \text{SE}\}$: filter on space elements $F_{N_d} = \{\forall \text{se} \in \text{SE} \mid \text{f} - \text{pc} : \text{MainEntranceSE} \in \text{se.classes}\}$: SRN destination node filter on space elements $F_N = (F_{\text{SS}},\ F_{\text{SE}_2})$: SRN source node filters —$F_{SS} = \{\forall\ ss \in SS\}$: filter on subspaces —$F_{\text{SE}_2} = \{\forall \text{se} \in \text{SE} \mid \text{f} - \text{pc} : \text{SideEntranceSE} \notin \text{se.classes}\}$: filter on space elements $F_E = (F_{E_1},\ F_{E_2})$: SRN edge filters —$F_{E_1} = \{\forall n_{\text{ss,se}} \in N_{\text{SS,SE}}\}$: filter on the sl : isNear _ SS _ SE relation between subspaces and space elements —$F_{E_2} = \{\forall a_{\text{ss}} \in A_{\text{SS}} \mid a_{\text{ss}}.\text{lineOfSight} = \text{true}\}$: filter on the sl : isAdjacentTo _ SS relation between subspaces $c = NoInstanceClassification$: enumeration type flag $m = PathWeight$: enumeration type flag

Table 5
Updates of people comfort views.

View/Layout update	Input parameters
'Natural light access' [Fig. 8c]: $L_{nla} \leftarrow$ update$_{t,\ F_{N_s},\ F_{N_d},\ F_N,\ F_E,\ c,\ m}$ (L_{nla})	L_{nla}: 'Natural light access' layout $t = DistanceNearest$: enumeration type flag $F_{N_s} = \{\forall s \in S\}$: SRN source node filter on spaces $F_{N_d} = \{\forall s \in S \mid \text{e} - \text{sp} : \text{ExternalAirSpace} \in s.\text{classes}\}$: SRN destination node filter on spaces $F_N = (F_S, F_{SE})$: SRN node filters —$F_S = \{\forall\ s \in S\}$: filter on spaces —$F_{SE} = \{\forall\ se \in SE\}$: filter on space elements $F_E = \{\forall\ pe_{se,s} \in PE_{SE,S}\}$: SRN edge filter on the sl : partiallyEncloses relation between space elements and spaces $c = InstanceClassification$: enumeration type flag $m = PathLength$: enumeration type flag
'Passive air circulation' [Fig. 8d]: $L_{pac} \leftarrow$ update$_{t,\ F_{N_m},\ F_{N_s},\ F_{N_d},\ F_N,\ F_E,\ c,\ m}$ (L_{pac})	L_{pac}: 'Passive air circulation' layout $t = DistanceMustPass$: enumeration type flag $F_{N_m} = \{\forall s \in S \mid \text{e} - \text{sp} : \text{InternalSpace} \in s.\text{classes}\}$: SRN must-pass node filter on spaces $F_{N_s} = \{\forall s \in S \mid \text{e} - \text{sp} : \text{ExternalAirSpace} \in s.\text{classes}\}$: SRN source node filter on spaces $F_{N_d} = F_{N_s}$ $F_N = (F_S, F_{SE})$: SRN node filters —$F_S = \{\forall\ s \in S\}$: filter on spaces —$F_{SE} = \{\forall\ se \in SE\}$: filter on space elements $F_E = \{\forall\ pe_{se,s} \in PE_{SE,S}\}$: SRN edge filter on the sl : partiallyEncloses relation between space elements and spaces $c = InstanceClassification$: enumeration type flag $m = PathLength$: enumeration type flag

6.4.2. 'Walking network' view

Whole spaces in L_{psa} are decomposed to create a layout L_{wn} that models the 'Walking network' view (Table 3, Fig. 8b). An L_{wn} has a walking network, which is a subnetwork of its SRN. Its nodes are pedestrian circulation subspaces (f-pc:PedCircSpace, sl:SubSpace) or space elements. Its edges are elements of the proximity relation between space elements and subspaces (sl:isNear_SS_SE) or the subspace adjacency relation (sl:isAdjacentTo_SS). In comparison with a space access network, a walking network is more detailed, and its edges do not intersect space boundaries. It thus supports more accurate estimates of walking distances between locations.

A walking network is generated by two consecutive layout decomposition operations. In the first decomposition, whole spaces, which are targeted by filter F_{WS_d}, are decomposed by the convex decomposition method (Section 5.1). A new subspace is inserted for each volume created by convex decomposition and each existing convex whole space

($i = InsertAsSubSpaces$). The second decomposition uses the Voronoi cell decomposition method (Section 5.1). New spaces are inserted as subspaces ($i = InsertAsSubSpaces$) according to space insertion filter F_{S_i}. Subspaces are inserted near space elements, which are targeted by filter F_{SE}. They are inserted based on space element types [46]. These are templates for recurring space elements and include locations of nearby spaces. For example, door types have spaces near their fronts and backs. Similarly, subspaces are inserted near flights of stairs and landings. Moreover, subspaces that were created by the preceding convex decomposition are targeted by filter F_{SS} to insert new subspaces at locations where volumes of targeted subspaces touch. The latter must be contained in the same whole space. Resulting subspace volumes are not shown in Fig. 8b to avoid visual clutter.

The walking network of the example is colored according to the distance from each subspace or space element to its nearest main entrance. Path weight is used as a distance measure, where weights correspond to edge lengths or the Euclidean distance between nodes related by an edge. Distances are computed by an *update* operation of the walking network (Table 4). Side entrances are considered inaccessible. The maximum distance from the most remote subspace to the nearest main entrance is 12.97 m.

6.5. People comfort views

6.5.1. 'Natural light access' view

Natural light spaces (f-li:NatLightSpace) and space elements (f-li:NatLightSE), as well as external air spaces are selected from L_s to create a layout L_{nla} that models the 'Natural light access' view (Table 3, Fig. 8c, Listing 6). An L_{nla} has a natural light access network, which is a subnetwork of its SRN. Its nodes are natural light spaces or space elements, or external air spaces. Its edges are elements of the partial enclosure relation between space elements and spaces.

The natural light access network of the example is colored according to the distance from each natural light space to its nearest external air space. Path length is used as a distance measure. Distances are computed by an *update* operation of the natural light access network (Table 5). Access to natural light is classified based on these distances by instance classification, which is invoked after the *update* operation. A natural light space with distance $d = 2$ has direct access to natural light (f-li:DirectNatLightSpace, Listing 7), whereas one with $d > 2$ has indirect access (f-li:IndirectNatLightSpace). If it is disconnected from external air spaces in the natural light access network, then it has no access to natural light (f-li:NoNatLightSpace). This is indicated by the ∞ symbol in Fig. 8c. Most hallways as well as the stairway lack access to natural light.

Certain enclosure space elements, such as glazed doors, transmit light, but they are neither windows nor openings. The class expression in Listing 8 covers space elements with a light-transmissive material (f-li:LightTransmissiveMaterial). In the example, folding doors of box rooms and kitchens are interpreted as being transparent. As a result, two hallways at the core of the floor, accessed through these doors, are classified as having indirect access to natural light.

Assessment of natural light access may not be relevant for elevators. In order to capture this situation, the f-pc:Elevator class is defined as a subclass of f-li:NatLightSpaceException. Thus elevators are not selected from L_s. The corresponding logical negation is encoded in a more straightforward manner in layout operation filters than in space ontologies, as is discussed in Section 6.7.

Listing 6
Class hierarchy for the 'Natural light access' view. Classes marked ¶ are defined by class expression.

```
sl:Space
  f-li:LightSpace
    f-li:NatLightSpace
      e-sp:ExternalAirSpace
      e-sp:InternalSpace
      f-li:NatLightAccessSpace
        f-li:DirectNatLightSpace ¶
        f-li:IndirectNatLightSpace ¶
        f-li:NoNatLightSpace ¶
    f-li:NatLightSpaceException
      f-pc:Elevator
sl:SpaceElement
  f-li:LightSE
      f-li:NatLightSE
        f-li:LightTransmissiveSE ¶
          e-enc:Window
          e-enc:Opening
e-mat:Material
  f-li:LightTransmissiveMaterial
    e-mat:Glass
```

Listing 7
Class expression for spaces with direct access to natural light.

```
Class: f-li:DirectNatLightSpace
EquivalentTo:
  f-li:NatLightSpace
  and (sl:distance some xsd:float[= 2.0f])
```

Listing 8
Class expression for light transmissive space elements.

```
Class: f-li:LightTransmissiveSE
EquivalentTo:
  sl:SpaceElement
  and (e-mat:hasMaterial some
    f-li:LightTransmissiveMaterial)
```

6.5.2. 'Passive air circulation' view

Passive air circulation spaces (f-ac:PassiveAirCircSpace) and space elements (f-ac:PassiveAirCircSE), as well as external air spaces are selected from L_s to create a layout L_{pac} that models the 'Passive air circulation' view (Table 3, Fig. 8d, Listing 9). Since they are subclasses of corresponding exception classes, elevators and entrances are excluded from this view. An L_{pac} has a passive air circulation network, which is a subnetwork of its SRN. Its nodes are passive air circulation spaces or space elements, or external air spaces. Its edges are elements of the partial enclosure relation between space elements and spaces.

The passive air circulation network of the example is colored according to the must-pass distance d_{mp} of each internal space. Values for d_{mp} correspond to the length of the shortest path between the two nearest external air spaces that passes through an internal space. These are computed by an *update* operation of the passive air circulation network (Table 5). The first and last edge in a path is ignored to

determine d_{mp} because each is related to an external air space. Short paths are considered as having a greater potential to support passive air circulation than long ones. Passive air circulation potential of spaces is classified by instance classification, which is invoked after the *update* operation. Spaces with $d_{mp} = 2$, $4 <= d_{mp} <= 6$, and $d_{mp} >= 8$ are classified as having high (f-ac:HighPotPassiveAirCircSpace, Listing 10), medium (f-ac:MediumPotPassiveAirCircSpace Space), and low (f-ac: LowPotPassiveAirCircSpace) potential for passive air circulation, respectively. Obstructing elements and opening properties, such as size, orientation, or cracks, which are relevant for detailed analysis, are not considered by this classification. In the example, a living room and a bedroom have high passive air circulation potential due to windows that are located diagonally from each other. There are four spaces with medium and 19 spaces with low passive air circulation potential. The stairway is disconnected from external air spaces. This is indicated by the ∞ symbol in Fig. 8d.

Listing 9
Class hierarchy for the 'Passive air circulation' view. Classes marked ¶ are defined by class expression.

```
sl:Space
  f-ac:AirCircSpace
    f-ac:PassiveAirCircSpace
      e-sp:ExternalAirSpace
      e-sp:InternalSpace
      f-ac:PassiveAirCircPotSpace
        f-ac:HighPotPassiveAirCircSpace ¶
        f-ac:MediumPotPassiveAirCircSpace ¶
        f-ac:LowPotPassiveAirCircSpace ¶
    f-ac:PassiveAirCircSpaceException
      f-pc:Elevator
sl:SpaceElement
  f-ac:AirCircSE
    f-ac:PassiveAirCircSE
      f-ac:OperableWindow
      e-enc:Opening
      f-pc:InternalAccessSE
    f-ac:PassiveAirCircSEException
      f-pc:EntranceSE
```

Listing 10
Class expression for spaces with high passive air circulation potential.

```
Class: f-ac:HighPotPassiveAirCircSpace
EquivalentTo:
  f-ac:PassiveAirCircSpace
  and (sl:mustPassDistance some
      xsd:float[= 2f])
```

6.6. Functional zoning views

6.6.1. 'Functional units' view

Layout L_{psa} is aggregated to create a layout L_{fu} that models the 'Functional units' view (Table 3, Fig. 8e, Listing 11). Each space in L_{fu} is labeled as a zone (Section 5.1). Aggregation is based on L_{psa}'s space access network without entrances. An entrance is equivalent to a functional unit (FU) access space element (f-fu:FUAccessSE). There is no grouping based on space properties. FU access space element nodes typically form a node cut set that partitions a space access network into multiple components. Spaces in each component are merged into an FU (f-fu:FunctionalUnit). Instance classification is invoked to infer specific FU classes.

In the example there are four FUs, including two residential units (f-res:ResidentialUnit) and two pedestrian circulation units (f-pc:PedCircUnit). One pedestrian circulation unit is a single-space zone which contains the service elevator. Functions of FUs are determined based on the sl:contains relationship between zones (sl:Zone) and spaces. For example, a pedestrian circulation unit is defined as a zone that contains a pedestrian circulation space, such as a stairway (Listing 12). Similarly, if a zone contains a kitchen, then it is classified as a residential unit.

Listing 11
Class hierarchy for the 'Functional units' view. Classes marked ¶ are defined by class expression.

```
sl:Space
  sl:Zone
    f-fu:FunctionalUnit
      f-res:ResidentialUnit ¶
      f-pc:PedCircUnit ¶
sl:SpaceElement
  f-fu:FUAccessSE ≡ f-pc:EntranceSE
```

Listing 12
Class expression for pedestrian circulation units.

```
Class: f-pc:PedCircUnit
EquivalentTo:
  sl:Zone
  and (
    (sl:contains some f-pc:Elevator)
    or (sl:contains some f-pc:Stairway)
  )
```

6.6.2. 'Functional zones' view

Layout L_{ar} is aggregated to create a layout L_{fz} that models the 'Functional zones' view (Table 3, Fig. 8f). Each space in L_{fz} is labeled as a zone. Aggregation is based on the space adjacency network of L_{ar}, and grouping on the *classes* property. Instance classification is not required for this view. Classes that are marked ¶ in Listing 13 are in grouping class set G to determine group membership of spaces (Section 5.1). Grouping weights of marked classes decrease from top to bottom of Listing 13. Primary and secondary pedestrian circulation spaces (f-pc:PrimaryPedCircSpace and f-pc:SecondaryPedCircSpace) are assigned to different groups. An example for the former are stairways, and for the latter hallways in residential units (f-res:Hallway).

In the example, two adjacent living rooms in L_{ar} are merged into a communal space (f-res:CommunalSpace) zone in L_{fz}. The zone is labeled as a living room because it is more specific than a communal space. Similarly, seven internal hallways from both residential units form a contiguous secondary pedestrian circulation space zone. There is a private space (f-res:PrivateSpace) zone that contains a bedroom and a box room. Three single-space bedroom zones are adjacent to single-space bathroom zones. On average, a functional zone contains 1.8 architectural spaces.

Listing 13
Class hierarchy for the 'Functional zones' view. Classes marked ¶ are included in set *G* to group spaces.

```
sl:Space
  f-res:ResidentialSpace
    f-res:CommunalSpace ¶
      f-res:LivingRoom
    f-res:PrivateSpace ¶
      f-res:Bedroom
      f-res:BoxRoom
  f-ser:ServiceSpace ¶
    f-ser:Kitchen
    f-ser:Bathroom
  f-pc:PedCircSpace
    f-pc:PrimaryPedCircSpace ¶
      f-pc:Hallway
      f-pc:Stairway
      f-pc:Elevator
    f-pc:SecondaryPedCircSpace ¶
      f-res:Hallway
```

6.7. Discussion

As shown above, semantic and spatial aspects of space views may be defined using space ontologies and layout transformation operations. At most, three layout operations are required to define each targeted view. All filters are simple Boolean expressions, as, for the most part, specific semantics are captured by space ontologies. However, statements that involve negation, such as *'all spaces that are not elevators'* (Section 6.5.1) are difficult to encode in OWL. This is because of the open-world assumption that underlies OWL. As layout operations assume a closed world, it was decided to encode negations in filters instead. Another challenge was the encoding of grouping logic for the *aggregate* operation (Section 5.1). Grouping corresponds to a classification of spaces concerning a set of disjoint classes. However, this is inconsistent with the notion of overlapping classes in order to model multiple functions of layout elements (Section 4.1). Thus it was decided to implement grouping logic as part of *aggregate* operation processing based on grouping weights defined in space ontologies. Primary functions of layout elements are similarly determined based on class weights (Section 6.3).

Since space ontologies are structured as multiple, interrelated ontologies, they may be extended more readily when compared with a single ontology. For example, to make targeted views applicable to office buildings, a function ontology for office spaces and an element ontology for office furniture may be added without requiring modifications of existing ontologies or view definitions. View definitions are currently hard-coded in the SMS system. Thus the definition of new views requires changes in source code. Instead, a space modeling language is favored that supports view definitions independent of the source code.

7. Implementation

The existing SMS system was extended to support the proposed view definition method and the data processing pipeline (Fig. 1). Extensions include the extraction of space data from IFC files and the integration of space ontologies in space layout transformation operations. The data processing pipeline was implemented as a collection of Windows 10 OS batch files that are executed sequentially. They invoke SMS system components and third-party applications to transform given IFC-based source building data into a multi-view space model.

7.1. SMS components

SMS consists of a kernel and a viewer component. The SMS kernel component reads source space data and creates the source space layout L_s as well as the multi-view space model in the space layout transformation steps of the data processing pipeline. The SMS kernel is written in C++ and implements a schema for network-based space layouts [46]. It uses the CGAL computational geometry library, the Acis solid modeler, and the BGL graph library to process layout operations [60–62,9]. The SMS viewer component converts between required geometry data formats and supports the visualization of space model data. It is implemented as a custom ObjectARX plug-in for Autocad Core Console [63,64]. The latter is a command-line version of Autocad for automated processing of geometry and drawing data.

7.2. Space data extraction

Users create room-based source building data in Revit or Archicad BIM authoring systems. These data are exported to an IFC file according to IFC Reference or Coordination MVDs [65]. The IfcConvert application, which is part of the IfcOpenShell IFC toolkit [66], is used to extract space data based on IFC class filters and the IFC to space ontologies class mapping (Table 2). Extracted geometry data are saved in the STEP format for each selected IFC class. The SMS viewer was extended to convert STEP to SAT data. SAT is the native file format for the Acis modeler. The SMS viewer was further extended to simplify space element geometries. This minimizes model size and reduces clutter in space model visualizations.

7.3. Integration of space ontologies

Ontology-based layout element filters were implemented in the SMS kernel. The HermiT OWL semantic reasoner was chosen to support automated instance classification (Section 5.2, [67]). The SMS kernel accesses the reasoner's API through a wrapper application that was written in Java. A parser was developed for the SMS kernel to load space ontologies and instantiate ontology-based filters. The parser further reads and writes instance data that are exchanged with the reasoner.

A label editor was developed to support manual label edits for the instance classification step. The editor is based on HTML and scalable vector graphics (SVG, [68]). Space classification data are embedded in SVG-based space model visualizations as custom data attributes that a user may search and modify. The SMS viewer generates content and scripts for the label editor from space model data.

Definitions of targeted views were implemented as routines in the SMS kernel. Each routine executes a layout operation sequence that corresponds to its view definition. Ontology-based filters are used as input parameters for layout operations. Intermediate output layouts are passed as input layouts to subsequent operations.

8. Validation

8.1. Sample models

The capabilities of the SMS system to support specific space semantics and semi-automatically transform room-based source building data into multi-view space models are evaluated for the targeted views and a sample of 22 regular floors of existing apartment buildings in

Table 6
Data summary for sample models and selected metrics (n=22).

Metric	Min.	Med.	Max.	Sum
Source data				
Space count7	10	44.0	239	1,134
Space element count	34	88.0	394	2,593
Label edit count	17	56.5	481	1,878
Semantic reasoner				
Input label count	64	165.5	642	4,407
Output label count	560	1,576.0	8,033	44,125
Space merge count	3	16.5	97	430

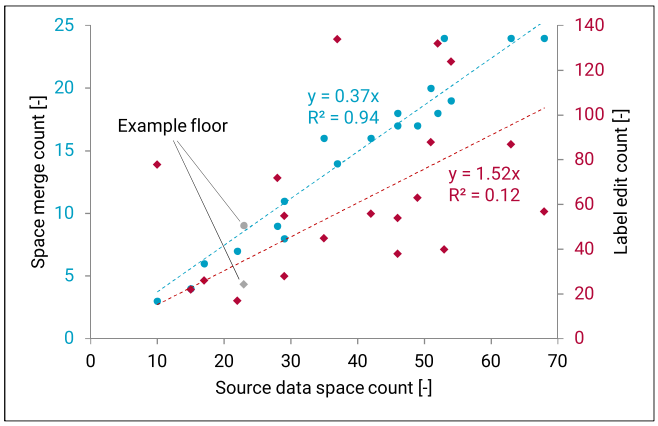


Fig. 9. Space merge (left) and label edit (right) counts relative to source data space count (n=20).

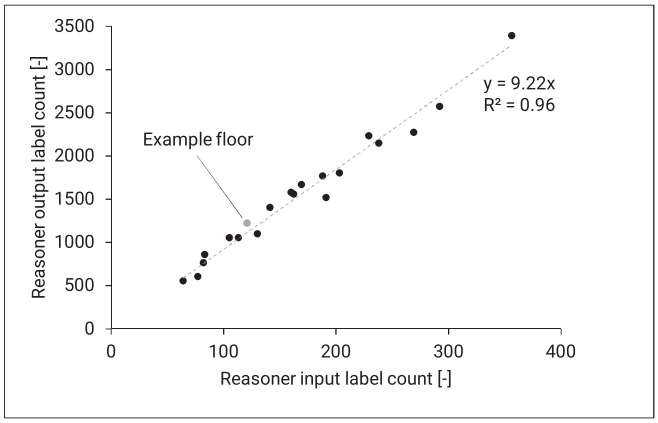


Fig. 10. Reasoner input and output label counts (n=20).

Europe (19), Mexico (2), and United States (1). Twenty floors in the sample were selected and modeled by 14 Master of Architecture students at TU Wien for a course project. Floors of two buildings, including the example floor, were selected and modeled by the author. Following the data processing pipeline (Fig. 1), room-based source building data were prepared in Archicad 24 (10 floors) or Revit 2021 (12 floors) [6,5]. IFC files were exported according to 'IFC 2x3 Reference View 1.2' (Archicad) or 'IFC 2x3 Coordination View 2.0' (Revit) MVDs. The SMS label editor was used to edit default labels (Table 2). Where necessary, the author resolved inconsistencies in source space data (e.g., missing doors) or labeling (e.g., missing unit door labels or mislabeling). The semantic and spatial consistency of generated models was verified by visual inspection and label search supported by the label editor.

8.2. Metrics

Sample models are analyzed according to the following metrics. *'Source data space count'* and *'Source data space element count'* refer to the size of source space data. The former includes internal as well as external spaces. *'Label edit count'* measures manual labeling. A label edit is defined as the manual insertion of a new label or the removal of an existing, default label. *'Input label count'* is defined as the number of labels (OWL types) that are passed to the semantic reasoner as input. It includes labels that are inserted manually as well as unmodified default labels. *'Output label count'* is defined as the number of labels that the reasoner returns. *'Space merge count'* is related to automated space layout transformation. It is defined as the number *aggregate* operations that are executed by the SMS system to merge two or more connected spaces from L_{ar} and L_{psa} into zones in layouts L_{fz} and L_{fu}, respectively (Table 3).

8.3. Results

Room-based source building data for the sample floors were transformed by SMS into space models for the targeted views (Figs. 7 and 8, Tables 3–5). Resulting models were spatially and semantically consistent. Whereas instance classification was semi-automated, the three steps in the data processing pipeline were fully automated (Fig. 1).

Data collected from resulting models for the selected metrics are summarized in Table 6. Label edits to reasoner input label percentage varies between 11.2% and 101.3% (median: 33.8%). Reasoner output to input label ratio varies between 7.9:1 and 12.5:1 (median: 9.5:1).

From space ontologies and the IFC to space ontologies class mapping (Table 2), it follows that labels for external spaces, living rooms, bedrooms, and unit doors need to be inserted manually for each model. For the sample models, 672 such labels (35.8% of all label edits) were inserted. Labels were inserted in certain models to model specific semantics. For example, five f-pc:SideEntranceSE labels (0.3%) were inserted in three models, including the model of the example floor, for space access analysis. Similarly, 162 f-li:LightTransmissiveSE labels (8.6%) were inserted in ten models to model glass doors for daylight access analysis, and 36 e-enc:FixedWindow labels (1.9%) in three models for assessment of natural ventilation potential of spaces.

Label edit and space merge counts are plotted relative to source space counts in Fig. 9. There are two exceptionally large models in the sample with source space counts of 126 and 239, respectively. These are considered outliers and excluded from Fig. 9 and regression analysis. Space merge counts equal 37% of space counts ($R^2 = 0.94$). The correlation between space counts and label edits is weak ($R^2 = 0.12$). The relationship between reasoner input and output label counts is shown in Fig. 10. The ratio of output to input labels is 9.22 ($R^2 = 0.96$).

8.4. Discussion

Results from sample models confirm the feasibility of the proposed multi-view space modeling method. It supports specific space semantics as well as the integration of semantic and spatial aspects of space views. At the same time, it is feasible to automate the main steps in the data processing pipeline fully and instance classification partially.

Consistent space data were extracted from source building data authored in Archicad or Revit based on IFC MVDs. A difference between these systems concerns the creation of space volumes. Revit supports the automated generation of space volumes that meet at wall centers. In Archicad, it is necessary to manually adjust generated space volumes to meet at wall centers.

Since they account for 35.8% of label edits for sample models, extending space ontologies to automatically classify external spaces, living rooms, bedrooms, and unit doors could significantly reduce the need for manual label edits. These appear to be high overall but varied considerably among sample models (Fig. 9). There are two explanations for this observation. First, certain models have more specific semantics than others, such as side entrances, glass doors, or fixed windows. Second, some source space data included detailed furnishing elements, whereas other included none. In the former case, it was necessary to replace default labels with labels for chairs, tables, and so on. When considering label insertions and removals, the label edits to reasoner input label percentage was 101.3% in one exceptional instance (median: 33.8%). Improved reuse of IFC semantic data, e.g., by more granular IFC class mappings or processing of property data, could reduce the need for such edits.

A reasoner input to output label ratio of approximately 1:9 for sample models suggests substantial automation of the instance classification step by semantic reasoning. Higher output to input label ratios and improved automated labeling is feasible by extending space

ontologies with additional logic. Rules that rely on rich spatial relations in space layouts were not explored in this study.

A high correlation between space merge and space counts in Fig. 9 reflects the use of functional zoning strategies and the repetitive layout of similarly sized FUs in the design of apartment buildings. Space merge operations were invoked 430 times to generate zone views for the sample models. Automation of such complex and frequently used layout transformation operations is likely to result in significant user productivity improvements and fewer modeling errors. However, this validation study did not measure manual modeling effort and quality without automated transformation operations.

9. Conclusion

A method and a data processing pipeline have been introduced to define space views with specific semantics and semi-automatically transform room-based source building data into corresponding multi-view space models. The presented work highlights the potential of automated model enrichment approaches that combine computational logic with computational geometry and graph search methods to support the creation and analysis of richly structured building models.

Future work is envisioned in two directions. First, as a part of BIM requirements engineering methodologies, such as IDM [16], the described space views definition method and space ontologies could be validated, revised, and extended further in order to meet specific space modeling needs of building design practitioners. For example, thermal zoning views could be developed in this manner. Second, there is a need to reduce the need for manual space classification further. Towards this end, the application of ML methods to space classification could be explored. As results from a related study suggest [45], the use of machine learning in the instance classification step may be more effective than rule-based methods. Automated, shape-based classification of space elements could be helpful to minimize related label edits [69]. More accurate classification of space elements may also enable improved classifiers for spaces. Supervised ML methods require large training data sets. To address this need, the SMS system is used in an ongoing effort at TU Wien to develop a data set for space models with richly structured semantic and spatial content and high data quality.

Declaration of Competing Interest

The author declares that he has no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgements

The author gratefully acknowledges Mihael Barada and participants in '259.428-2021S Architectural Morphology' at the Faculty of Architecture and Planning, TU Wien for their contributions to the validation study. He thanks Pieter De Wilde and the anonymous reviewers for their comments and suggestions.

References

[1] G. van Nederveen, F. Tolman, Modelling multiple views on buildings, autom. construct. 1 (1992) 215–224, https://doi.org/10.1016/0926-5805(92)90014-B.
[2] M. Rosenman, J. Gero, Modelling multiple views of design objects in a collaborative environment, Comput. -Aided Des. 28 (1996) 193–205, https://doi.org/10.1016/0010-4485(96)86822-9.
[3] W. Stamm-Jeske, K. Fischer, T. Haag, Raumpilot Wohnen, Karl Krämer, Stuttgart, Germany and Zurich, Switzerland, 2012. ISBN: 978-3-7828-1554-3.
[4] O. Heckmann, F. Schneider, Grundrissatlas Wohnungsbau, Birkhäuser, Basel, 2017. ISBN: 3035611424.
[5] Autodesk, Inc., Revit 2021 User's Guide, 2021. https://knowledge.autodesk.com/support/revit-products [Accessed 6 July 2021].
[6] Graphisoft, Inc., Archicad 14 User's Guide, 2021. https://helpcenter.graphisoft.com/user-guide-chapter/85451/ [Accessed 6 July 2021].
[7] P. Raftery, M. Keane, J. O'Donnell, Calibrating whole building energy models: an evidence-based methodology, Energy Build. 43 (2011) 2356–2364, https://doi.org/10.1016/j.enbuild.2011.05.020.
[8] J.-K. Lee, C. Eastman, Y. Lee, Implementation of a BIM domain-specific language for the building environment rule and analysis, J. Intell. Robot. Sys.: Theory Appl. 79 (2015) 507–522, https://doi.org/10.1007/s10846-014-0117-7.
[9] G. Suter, F. Petrushevski, M. Sipetic, Operations on network-based space layouts for modeling multiple space views of buildings, Adv. Eng. Inf. 28 (2014) 395–411, https://doi.org/10.1016/j.aei.2014.06.004.
[10] G. Suter, Definition of views to generate, visualize, and evaluate multi-view space models of schematic building designs, in: J. Beetz, L. van Berlo, T. Hartmann (Eds.), 22nd International Workshop: Intelligent Computing in Engineering, EG-ICE, Eindhoven, The Netherlands, 2015. ISBN: 9781510809567.
[11] C. Eastman, A. Siabiris, A generic building product model incorporating building type information, autom.construct. 3 (1) (1995) 283–304, https://doi.org/10.1016/0926-5805(94)00028-L.
[12] A. Silberschatz, H. Korth, S. Sudarshan, Database System Concepts, McGraw-Hill, New York, NY, USA, 2006. ISBN: 0073523321.
[13] P. Katranuschkov, M. Weise, R. Windisch, S. Fuchs, R.J. Scherer, BIM-based generation of multi-model views, in: W. Thabet (Ed.), 27th CIB W78 - Information Technology for Construction, CIB, Cairo, Egypt, 2010.
[14] BuildingSmart, Industry Foundation Classes IFC 4, 2020. https://standards.buildingsmart.org/IFC/RELEASE/IFC4/ADD2_TC1/HTML/ [Accessed 5 December 2020].
[15] BuildingSmart, Model View Definition (MVD) - an Introduction, 2020. https://technical.buildingsmart.org/standards/ifc/mvd/ [Accessed 5 December 2020].
[16] BuildingSmart, An Integrated Process for Delivering IFC Based Data Exchange, 2012, in: https://standards.buildingsmart.org/documents/IDM/IDM_guide-IntegratedProcess-2012_09.pdf [Accessed 13 July 2021].
[17] BuildingSmart, MVD Database, 2012. https://technical.buildingsmart.org/standards/ifc/mvd/mvd-database/?sfw=pass1626187175 [Accessed 13 July 2021].
[18] M. Weise, P. Katranuschkov, R. Scherer, Generalised model subset definition schema, in: R. Amor (Ed.), 20th CIB W78 - Information Technology for Construction, CIB, Auckland, New Zealand, 2003. ISBN: 0-908689-71-3.
[19] R. Windisch, P. Katranuschkov, R. Scherer, A generic filter framework for consistent generation of BIM-based model views, in: A. Borrmann, P. Geyer (Eds.), 19th International Workshop: Intelligent Computing in Engineering, EG-ICE, Munich, Germany, 2012. ISBN: 9781634395489.
[20] T. Chipman, T. Liebich, M. Weise, Specification of a Standardized Format to Define and Exchange Model View Definitions with Exchange Requirements and Validation Rules, 2016. https://standards.buildingsmart.org/MVD/RELEASE/mvdXML/v1-1/mvdXML_V1-1-Final.pdf [Accessed 13 July 2021].
[21] Y. Adachi, Overview of partial model query language, in: Proceedings of the 10th International Conference on Concurrent Engineering, 2003, pp. 549–555. ISBN: 9789058096234.
[22] W. Mazairac, J. Beetz, BIMQL - an open query language for building information models, Adv. Eng. Inf. 27 (2013) 444–456, https://doi.org/10.1016/j.aei.2013.06.001.
[23] W. Solihin, C. Eastman, Y.-C. Lee, Toward robust and quantifiable automated ifc quality validation, Adv. Eng. Inf. 29 (2015) 739–756, https://doi.org/10.1016/j.aei.2015.07.006.
[24] M. Fischer, F. Aalami, R. Akbas, Formalizing product model transformations: case examples and applications, in: I. Smith (Ed.), Artificial Intelligence in Structural Engineering, Information Technology for Design, Collaboration, Maintenance, and Monitoring, 1998, pp. 113–132, https://doi.org/10.1007/BFb0030438. ISBN: 978-3-540-64806-2.
[25] V. Bazjanac, A. Kiviniemi, Reduction, simplification, translation and interpretation in the exchange of model data, in: D. Rebolj (Ed.), 24th CIB W78 - Information Technology for Construction, CIB, Maribor, Slovenia, 2007.
[26] M. Shin, J.S. Haberl, Thermal zoning for building HVAC design and energy simulation: a literature review, Energy Build. 203 (2019) 109429, https://doi.org/10.1016/j.enbuild.2019.109429.
[27] ASHRAE, Standard 90.1-2016. energy standard for buildings except low-rise residential buildings, Am. Soc. Heat., Refrig. Air-Cond. Eng. 278 (2016). https://www.ashrae.org/technical-resources/bookstore/standard-90-1 [Accessed 13 October 2021].
[28] Y.-H. Lin, Y.-S. Liu, G. Gao, X.-G. Han, C.-Y. Lai, M. Gu, The IFC-based path planning for 3D indoor spaces, Adv. Eng. Inf. 27 (2013) 189–205, https://doi.org/10.1016/j.aei.2012.10.001.
[29] W.Y. Lin, P.H. Lin, Intelligent generation of indoor topology (i-GIT) for human indoor pathfinding based on IFC models and 3D GIS technology, autom.construct. 94 (2018) 340–359, https://doi.org/10.1016/j.autcon.2018.07.016.
[30] J.O. Wallgr“un, Autonomous construction of hierarchical voronoi-based route graph representations, in: C. Freksa, M. Knauff, B. Krieg-Br”uckner, B. Nebel, T. Barkowsky (Eds.), Spatial Cognition IV. Reasoning, Action, Interaction, Berlin, Heidelberg, 2005, pp. 413–433, https://doi.org/10.1007/978-3-540-32255-9_23.
[31] J.-K. Lee, C. Eastman, J. Lee, M. Kannala, Y. Jeong, Computing walking distances within buildings using the universal circulation network, Environ. Plann. B 37 (2010) 628–645, https://doi.org/10.1068/b35124.
[32] Solibri, Inc., Solibri Model Checker, 2021. https://help.solibri.com/hc/en-us/categories/1500000260302-Using-Solibri [Accessed 6 July 2021].
[33] CSI, OmniClass Construction Classification System (OCCS), Table 13: Spaces by Function, 2010. https://www.csiresources.org/standards/omniclass/standards-omniclass-about [Accessed 5 December 2020].

[34] NBS, UniClass, Table SL - Spaces/locations, 2020. https://www.thenbs.com/-/media/uk/files/xls/uniclass/2020-10/uniclass2015_sl_v1_18.xlsx?la=en [Accessed 5 December 2020].
[35] BuildingSmart, BuildingSMART Data Dictionary 4.4.1, 2020. https://www.buildingsmart.org/users/services/buildingsmart-data-dictionary/ [Accessed 5 December 2020].
[36] P. Pauwels, S. Zhang, Y.-C. Lee, Semantic web technologies in AEC industry: a literature overview, autom.construct. 73 (2017) 145–165, https://doi.org/10.1016/j.autcon.2016.10.003.
[37] T.M. Farias, A. Roxin, C. Nicolle, A rule based system for semantical enrichment of building information exchange, in: CEUR Proceedings of RuleML (4th Doctoral Consortium), 2014, pp. 2–9. http://ceur-ws.org/Vol-1211/paper2.pdf [Accessed 13 October 2021].
[38] P. Pauwels, T.M. de Farias, C. Zhang, A. Roxin, J. Beetz, J.D. Roo, C. Nicolle, A performance benchmark over semantic rule checking approaches in construction industry, Adv. Eng. Inf. 33 (2017) 68–88, https://doi.org/10.1016/j.aei.2017.05.001.
[39] S.-K. Lee, K.-R. Kim, J.-H. Yu, BIM and ontology-based approach for building cost estimation, autom.construct. 41 (2014) 96–105, https://doi.org/10.1016/j.autcon.2013.10.020.
[40] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, SWRL: A Semantic Web Rule Language Combining OWL and RuleML, 2004. http://www.w3.org/Submission/SWRL/ [Accessed 5 December 2020].
[41] S. Staub-French, M. Fischer, J. Kunz, K. Ishii, B. Paulson, A feature ontology to support construction cost estimating, Artif. Intell. Eng. Des. Anal. Manufact. 17 (2003) 133–154, https://doi.org/10.1017/S0890060403172034.
[42] M. Belsky, R. Sacks, I. Brilakis, Semantic enrichment for building information modeling, Comput. -Aided Civ. Infrastruct. Eng. 31 (2016) 261–274, https://doi.org/10.1111/mice.12128.
[43] R. Sacks, L. Ma, R. Yosef, A. Borrmann, S. Daum, U. Kattel, Semantic enrichment for building information modeling: procedure for compiling inference rules and operators for complex geometry, J. Comput. Civ. Eng. 31 (2017) 04017062, https://doi.org/10.1061/(ASCE)CP.1943-5487.0000705.
[44] S. Zhang, J. Teizer, J.-K. Lee, C.M. Eastman, M. Venugopal, Building information modeling (BIM) and safety: automatic safety checking of construction models and schedules, autom.construct. 29 (2013) 183–195, https://doi.org/10.1016/j.autcon.2012.05.006.
[45] T. Bloch, R. Sacks, Comparing machine learning and rule-based inferencing for semantic enrichment of BIM models, autom.construct. 91 (2018) 256–272, https://doi.org/10.1016/j.autcon.2018.03.018.
[46] G. Suter, Structure and spatial consistency of network-based space layouts for building and product design, Comput. -Aided Des. 45 (2013) 1108–1127, https://doi.org/10.1016/j.cad.2013.04.004.
[47] World Wide Web Consortium, OWL 2 Web Ontology Language, 2012. https://www.w3.org/TR/owl2-overview [Accessed 5 December 2020].
[48] P. Hitzler, M. Krötzsch, B. Parsia, P.F. Patel-Schneider, S. Rudolph, OWL 2 Web Ontology Language Primer, Second Edition, 2012. https://www.w3.org/TR/owl2-primer [Accessed 5 December 2020].
[49] A. Schreiber, Y. Raimond, RDF 1.1 Primer, W3C Working Group Note, World-Wide Web Consortium, 2014. https://www.w3.org/TR/rdf11-primer/ [Accessed 6 December 2020].
[50] World Wide Web Consortium, SPARQL 1.1 Query Language, 2013. https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ [Accessed 18 July 2021].
[51] G. Suter, Space ontologies, 2021. http://spacepatterns.com/ontologies/space_ontologies_002_a_in_c_20/_ontology_directory.htm [Accessed 6 July 2021].
[52] M. Rasmussen, P. Pauwels, M. Lefrançois, G. Schneider, Building Topology Ontology, 2020. https://w3c-lbd-cg.github.io/bot/ [Accessed 5 December 2020].
[53] M. Rasmussen, M. Lefrançois, G. Schneider, P. Pauwels, BOT: the building topology ontology of the W3C linked building data group, Semant. Web J. 12 (2021) 143–161, https://doi.org/10.3233/SW-200385.
[54] S. Lohmann, S. Negru, F. Haag, T. Ertl, VOWL 2: user-oriented visualization of ontologies, in: K. Janowicz, S. Schlobach, P. Lambrix, E. Hyvönen (Eds.), Knowledge Engineering and Knowledge Management 2014, volume 8876, Linköping, Sweden, 2014, pp. 266–281, https://doi.org/10.1007/978-3-319-13704-9_21.
[55] M. Horridge, P. Patel-Schneider, OWL 2 Web Ontology Language. Manchester Syntax, Second Edition, 2012. http://www.w3.org/TR/owl2-manchester-syntax/ [Accessed 5 December 2020].
[56] F. Aurenhammer, R. Klein, Chapter 5 - voronoi diagrams, in: J.-R. Sack, J. Urrutia (Eds.), Handbook of Computational Geometry, North-Holland, Amsterdam, 2000, pp. 201–290, https://doi.org/10.1016/B978-044482537-7/50006-1. ISBN: 978-0-444-82537-7.
[57] A. Eftekharian, M. Campbell, Convex decomposition of 3D solid models for automated manufacturing process planning applications, in: ASME Design Engineering Technical Conference, volume 2, Chicago, IL, USA, 2012, pp. 727–735, https://doi.org/10.1115/DETC2012-70278.
[58] B. Parsia, N. Matentzoglu, R.S. Gonçalves, B. Glimm, A. Steigmiller, The OWL reasoner evaluation (ORE) 2015 competition report, J. Autom. Reason. 59 (2017) 455–482, https://doi.org/10.1007/s10817-017-9406-8.
[59] B. Hillier, J. Hanson, The Social Logic of Space, Cambridge University Press, Cambridge, UK, 1989. ISBN: 9780521367844.
[60] CGAL, Open Source Project, Computational Geometry Algorithms Library 4.12, 2018. https://doc.cgal.org/4.12 [Accessed 5 December 2020].
[61] Spatial Technologies, Inc., 3d ACIS Modeler 2019.1.0.2, 2019. http://doc.spatial.com [Accessed 5 December 2020].
[62] Boost, Boost Library 1.59, 2015. https://www.boost.org/doc/libs/1_59_0 [Accessed 5 December 2020].
[63] Autodesk, Inc., AutoCAD 2019 Object Modeling Framework, 2019. https://www.autodesk.com/developer-network/platform-technologies/autocad/objectarx [Accessed 5 December 2020].
[64] Autodesk, Inc., AutoCAD 2019 User's Guide, 2019. http://help.autodesk.com/view/ACD/2019/ENU/ [Accessed 5 December 2020].
[65] BuildingSmart, Reference view, 2019. https://standards.buildingsmart.org/MVD/RELEASE/IFC4/ADD2_TC1/RV1_2/HTML/schema/views/reference-view/index.htm [Accessed 5 December 2020].
[66] T. Krijnen, IfcOpenShell, 2021. http://ifcopenshell.org/ [Accessed 6 July 2021].
[67] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: an OWL 2 reasoner, J. Autom. Reason. 53 (2014) 245–269, https://doi.org/10.1007/s10817-014-9305-1.
[68] World Wide Web Consortium, Scalable Vector Graphics (SVG) 1.1, Second Edition, 2002. https://www.w3.org/TR/SVG11/ [Accessed 5 December 2020].
[69] C. Emunds, N. Pauen, V. Richter, J. Frisch, C. van Treeck, IFCNet: a benchmark dataset for IFC entity classification, in: J. Abualdenien, A. Borrmann, L.-C. Ungureanu, T. Hartmann (Eds.), 28th International Workshop: Intelligent Computing in Engineering, EG-ICE, Berlin, Germany, 2021. ISBN: 978-3-7983-3211-9.