Operations on network-based space layouts for modeling multiple space views of buildings

Georg Suter[a,*], Filip Petrushevski[a], Miloš Šipetić[a]

[a]*Design Computing Group, Faculty of Architecture and Planning, Vienna University of Technology, Treitlstrasse 3, A-1040 Vienna, Austria, Phone: +43 1 58801 27220, Fax: +43 1 58801 27299*

Abstract

Space layouts are created by designers to model a building's spaces and related physical objects. Building services designers commonly reuse space layouts created by architectural designers to develop their designs. However, reuse tends to be limited due to differences in designers' space views. In order to address this issue of modeling multiple space views, we define a set of novel operations that can be used by designers to generate new space layouts from existing layouts. Fundamental operations include *select*, *aggregate*, and *decompose* operations. The *select* operation facilitates reuse of space layouts created in building information modeling (BIM) authoring systems. Signatures and processing of these operations are defined. We use an existing schema for network-based space layouts to represent space layouts. In a network-based space layout, specific spatial relations between layout elements are explicitly modeled as a directed, weighted graph or network. Processing of certain operations involves traversal of a spatial relation network with graph algorithms to determine layout modifications. Symmetric difference and *overlay* operations are defined as additional operations. They are composed of *union*, *intersect*, and *subtract* operations, which are fundamental operations. Fundamental and additional layout operations may be composed into expressions to model domain-specific space views. We have extended an existing layout modeling system with implementations of these layout operations. The system relies on geometric and solid modeling as well as graph libraries to represent layouts and process operations. The feasibility of modeling of multiple space views with layout operation expressions is shown with an example in which a security lighting layout of a floor of an existing office building is automatically generated from an architectural layout.

Keywords: Space modeling, Building information modeling, Spatial query languages

*Corresponding author

Email addresses: georg.suter@tuwien.ac.at (Georg Suter), filip.petrushevski@tuwien.ac.at (Filip Petrushevski), milos.sipetic@tuwien.ac.at (Miloš Šipetić)

Nomenclature

A	*"is adjacent to"* relation
le	Layout element
N	*"is near"* relation
O	*"overlaps"* relation
PB	*"partially bounds"* relation
PE	*"partially encloses"* relation
sbe	Space boundary element
se	Space element
se_{c_ws}	Space element contained in a whole space
se_{pe_ws}	Space element partially enclosing a whole space
sr	Spatial relation element
SRN	Spatial relation network
ss	Subspace
T	*"touches"* relation
ws	Whole space

1. Introduction

Space layouts are created by designers to model a building's spaces and related physical objects, such as walls, windows, furnishing and technical equipment elements. Designers increasingly use building information modeling (BIM) authoring systems to develop space layouts [1]. Sharing of space layouts is facilitated by data exchange methods. The space schema in the Industry Foundation Classes (IFC), for example, defines rich data structures for spaces, space boundaries, and space enclosures [2].

It is common practice for building services designers to reuse space layouts created by architectural designers to develop their designs. However, reuse tends to be limited due to substantial differences in designers' space views. This issue of multiple, domain-specific views of objects is well-known in BIM [3, 4]. For an architectural designer, a space typically includes space boundaries, windows, doors, as well as visually relevant furnishing and equipment elements. By contrast, in the space view adopted by an indoor climate control system designer, spaces are often zones that span multiple rooms with similar orientation, function, or thermal requirements. Space enclosure elements with high thermal flux, air inlets/outlets, or indoor air temperature sensors are of concern. A lighting designer in turn is usually interested in zones within rooms that are related to luminaires, shades, walls, windows, illuminance sensors, or occupancy sensors.

While BIM authoring systems and IFC provide general space data modeling and exchange capabilities, support for automated generation of space layouts that model multiple, domain-specific space views, as

outlined above, is limited. Building services designers may thus only partially reuse space layouts created by architectural designers. As a workaround, they need to manually recreate space layouts according to their space views. This existing workflow is inefficient, error-prone, and a significant barrier for seamless collaboration in multi-disciplinary design teams. According to [5], reduced errors, shorter cycle times of workflows, and reduced rework are viewed as highly important for improved quality and productivity of design firms in architecture, engineering, and construction.

In order to address the issue of modeling multiple space views, we describe a set of novel operations that can be used by designers to generate new space layouts from existing layouts. We build on and extend previous work, in which we have developed initial specifications for these operations [6, 7]. Layout operations may be composed into expressions to model domain-specific space views. For example, a layout operation expression may be defined to generate an indoor climate control layout from an architectural layout. We use an existing schema for network-based space to represent space layouts [8]. In a network-based space layout, spatial relations between layout elements are explicitly modeled as a directed, weighted graph or network. Processing of certain operations involves tracing of this spatial relation network with graph algorithms to determine layout modifications.

The remainder of this paper is organized as follows. A survey of related work is included in Section 2. Network-based space layout concepts are reviewed in Section 3. Fundamental layout operations are defined in Section 4. Additional layout operations that are compositions of fundamental operations are introduced in Section 5. The implementation of fundamental and additional layout operations in an existing layout modeling system is described in Section 6. Section 7 presents an example of a layout operation expression. Open issues and future work are discussed in Section 8.

2. Related work

Related work is reviewed, including IFC model view definitions, modeling and query languages, and ontology-based reasoning. We highlight aspects that deal with spatial modeling.

2.1. IFC model view definitions

An IFC model view definition is a subset of the original IFC schema that is relevant for specific applications and life-cycle phases [9]. The IFC Coordination View defines a view for architectural, mechanical, and structural design coordination [10]. It includes $IfcSpace$, $IfcSpatialStructureElement$, and $IfcZone$ classes. A building may be hierarchically divided into $IfcSpatialStructureElement$ levels, such as sections, floors and rooms. $IfcSpace$s and building elements may be attached to these levels. Alternatively, $IfcSpace$s may be aggregated into $IfcZone$s. In contrast to $IfcSpatialStructureElement$s, the latter do not have a shape. Their structure does not need to be hierarchical, that is, an $IfcSpace$ may belong to one or more $IfcZone$s.

Weise et al. [11] describe a schema for the definition of arbitrary IFC model views. A data subset is retrieved in two steps from an IFC model instance. In the first step, objects are selected dynamically from the original model by filtering based on criteria supplied by users at runtime. An example is the selection of objects on a particular floor. In the second step, attribute data or references to objects that are not needed are removed based on a pre-defined, static model view.

2.2. Modeling languages

GLIDE and EDM are early examples of modeling languages [12, 13]. Eastman and Siabiris [14] use composition and specialization in EDM to model views of constructed spaces and use spaces. Constraints verify if required relations between model entities are satisfied. An example is a constraint that checks for correct alignment of faces of constructed space and use space polyhedrons.

Zamanian et al. [15] describe a framework for modeling spatial abstractions of buildings. Decomposition, selection, and composition operations are applied to spatial configurations that are based on a spatial representation scheme. In this scheme, disjoint geometric elements (vertices, faces, solids) are generated by user-defined carrier elements (points, lines, planes). Domain-specific spatial views are defined by selection and composition operations and generated in three steps. First, Euclidean space is partitioned recursively until the desired level of granularity in a spatial configuration is reached. Second, atomic elements in the configuration are selected based on user-defined criteria. Third, selected atomic elements are composed to create new elements. The composition operation ensures that new elements are disjoint.

Stouffs et al. [16, 17] define the SORTS language for manipulating nested property structures. Although the language is not domain-specific, its development is motivated by the need for representational flexibility in building design. Property structures are composed with addition and sum operations. Furthermore, they may be combined using addition, subtract, and product operations. Primitive data constructs have specific, built-in behaviors under these operations. For example, the addition of two properties that model boundaries of solids corresponds to a merge (union) of these solids.

Grabska et al. [18, 19, 20] describe a visual design language that uses attributed, hierarchical hypergraphs to represent designs. Designers create hypergraphs representations of space layouts, for example, with operations that include hypergraph development and suppression. These operations add or remove levels of detail in a spatial hierarchy. Szuba [21] combines the hypergraph representation with a graph rewriting system to reason about spatial requirements. Activity requirements that are modeled as paths are matched against access graphs that are extracted from a space layout created in a BIM authoring system. This matching is non-trivial as it considers indirect access relations between rooms via circulation spaces.

The BERA language developed by Lee [22] provides high-level constructs to check building models against pedestrian circulation or space requirements. Users define simplified but extensible views of a building model that is populated with IFC objects. Space groups or paths may be included in a view. Paths are groups

of connected spaces. Pedestrian circulation networks are generated from door, vertical access, and concave points that lie on buffered space boundary polygons [23]. Edges in a circulation network relate nodes with line-of-sight.

2.3. Query languages

Several query languages have been developed to retrieve data from building models. Some are defined as extensions of SQL or XML, such as PMQL, GTCIS2SQL, and PMQ [24, 25, 26]. PMQL supports recursive object selection using a *cascades* operator. BimQL simplifies access to objects, attributes, and relations in the IFC schema by hiding its nested structure from users [27]. An IFC object graph is recursively traced, e.g. to derive the name of a space.

A spatial query language developed by Borrmann et al. [28, 29, 30] supports queries on implicit topological, directional, and metric spatial relations between objects. The language is designed as an extension of SQL. Topological spatial query operators derive *equal*, *contain*, *within*, *touch*, and *overlap* relations. Their processing involves evaluation of the 9-intersection model, which uses point-set topology as geometry representation [31].

Nepal et al. [32] describe query operations to derive location, alignment, spacing, and variation of objects such as columns, openings, or penetrations. These relations are required for construction planning but are typically not included in building models, nor are they easily derived from other relations. Object features such as faces, center points, and center lines are relevant for queries. Query operations are based on object geometry data as well as construction planning knowledge.

2.4. Ontology-based reasoning

Ontology-based reasoning has the potential to simplify access to building models for users and applications [33] and provide feedback to designers. Bhatt et al. [34] combine description logic with spatial logics to develop an ontology for the architectural design domain. Topological spatial relations between objects are determined by region connection calculus, which is a qualitative spatial reasoning method ([35]). Design requirements are modeled as spatial constraints that are based on this reasoning method. An example for a constraint is a range space of an occupancy sensor that must contain the functional space of a door.

Implicit relations in an ontology may be derived based on user-defined inference rules. In rule languages such as SWRL or SPARQL, rules are defined as subgraph patterns which a reasoner matches against an object graph. Pauwels et al. [36] define rules to derive a view of space boundaries and their sound reduction properties from a building model and construction element type data. The resulting partial model is processed with a second rule set that encodes a sound control standard. A building model can thus be checked for compliance with this standard.

2.5. Discussion

The following observations are made based on the above review. First, while IFC model view definitions provide a common reference for modeling multiple space views, operations to generate models are limited to selections [11]. Second, decomposition, selection, and composition operations described by Zamanian and Fenves [15] as well as space group and path operations in the BERA language [22] are high-level spatial operations that are conceptually similar to the operations described in this paper. By contrast, operations in other modeling and query languages tend to be low-level but are more generally applicable. Third, in case of the BERA language, operations use a space layout representation that is specific to circulation and space planning domains. Our operations are based on a layout representation that can be used in multiple domains (Section 3.2). Fourth, recursive object graph traversal, which is required to process operations such as the space path operation, is difficult to implement in the reviewed query languages and ontology-based reasoning methods. In summary, our operations provide novel, high-level space layout modeling functionality. They may be composed to automatically generate richly structured space layouts that model space views of multiple domains.

3. Review of network-based space layout concepts

3.1. Background

Space layout representations developed by Steadman [37] and Grason [38] for the space planning domain are among the first representations that model spatial relations between spaces, walls, and openings as graphs or networks. Space adjacency and access graphs are relevant for the development of space planning algorithms [39]. Flow networks are used to simulate air circulation behavior in buildings [40]. Several recent efforts explore layout representations whose spatial relation graphs are derived from IFC space data. Application domains include code compliance checking of pedestrian circulation systems, indoor navigation, construction planning, and case-based design [23, 41, 42, 43, 44].

Network-based space layouts extend the network approach to space layout representation to facilitate multi-domain space modeling. Concepts are reviewed in the following. For a detailed description, the reader is referred to [8]. The schema for network-based space layouts incorporates aspects of existing space schemas [45, 14, 46, 2]. In contrast to the IFC space schema, whole spaces and subspaces are modeled as separate classes in the layout schema. Flexible subspace modeling is supported to meet the needs of multiple domains. Furthermore, certain spatial relations in the layout schema are not modeled explicitly in IFC.

3.2. Layout schema

The principal classes in the schema for network-based space layouts are shown in Fig. 1. Layout elements (*le*'s) include whole spaces (*ws*'s), subspaces (*ss*'s), space boundary elements (*sbe*'s), and space elements

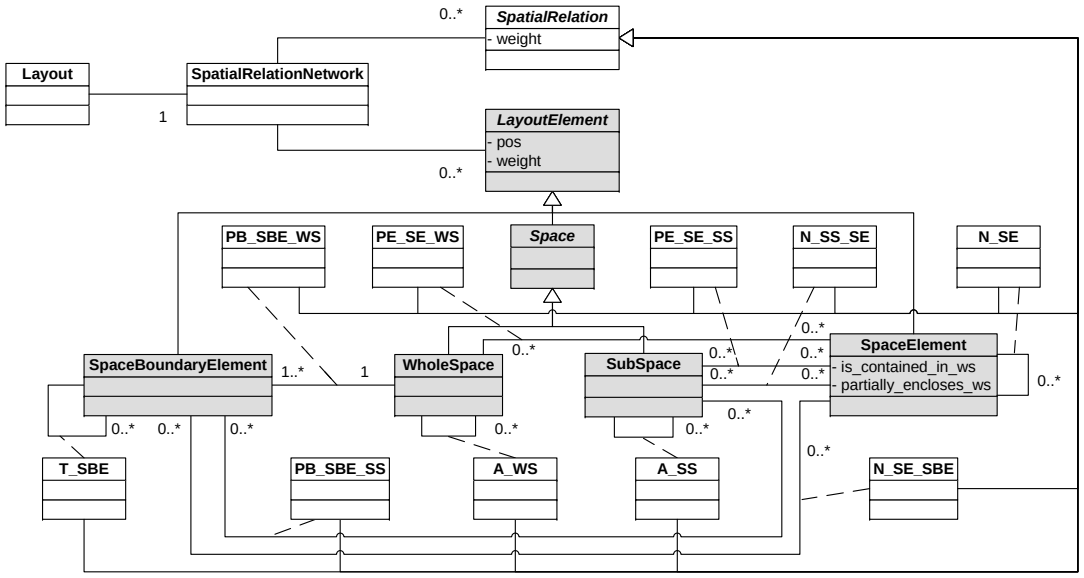

Figure 1: UML class diagram of the principal classes in the network-based space layout schema [8, 47]. *SpatialRelation* subclasses: *A: "is adjacent to"*, *PB: "partially bounds"*, *PE: "partially encloses"*, *T: "touches"*, *N: "is near"*.

(*se*'s). A whole space is a space which is bounded on all sides by *sbe*'s. An *sbe* is an immaterial surface that is part of the boundary of a whole space. A partial space or subspace is a space which is contained in a whole space. It is typically only partially bounded by *sbe*'s. Subspaces are relevant e.g. to model access zones of doors, intensity distributions of luminaires, or coverage of occupancy sensors. Volumes of whole spaces and subspaces are modeled as boundary representation (Brep) solids ([48, 49, 50]). Space elements (*se*'s) are physical objects, such as windows, tables, luminaires, or sensors, that are either contained in or partially enclose a whole space. An *se* has an attribute that specifies if it is contained in or partially encloses a whole space (se_{c_ws} or se_{pe_ws}). A desk is an example for an se_{c_ws} and a door for an se_{pe_ws}.

A layout has a spatial relation network (SRN), which is a directed, weighted graph with *le*'s as nodes and spatial relation elements (*sr*'s) as edges. An SRN is a geometric network and embedded in 3-space. That is, spatial relation edges are line segments whose end points correspond to positions of related layout elements. An SRN may thus be traversed based on Euclidean distances and line-of-sight between *le*'s. This is useful, for example, to determine shortest paths in a circulation layout. As *sr*'s have a weight and other attributes, corresponding *SpatialRelation* classes are modelled as association classes in the schema.

Spatial relations include selected *"is adjacent to"* (A), *"partially bounds"* (PB), *"partially encloses"* (PE), and *"is near"* (N) relations between *le*'s. For example, A_{WS} refers to the *"is adjacent to"* relation between whole spaces. The subscript indicates that the set WS of whole spaces in a layout participates in the relation. The A_{WS} relation corresponds to the A_WS class in the schema. Similarly, sets SS, SBE and SE are, respectively, subspace, space boundary element, and space element sets in a layout. There are

six base relations in an SRN that are derived from layout element geometries. They include $PB_{SBE,WS}$, $PB_{SBE,SS}$, T_{SBE}, A_{SS}, $N_{SE,SBE}$, and $N_{SS,SE}$ relations. Relations A_{WS}, N_{SE}, $PE_{SE,WS}$, and $PE_{SE,SS}$ are derived from these base relations. For example, the A_{WS} relation is derived from base relations $PB_{SBE,WS}$ and T_{SBE}, which are, respectively, the *"partially bounds"* relation between *sbe*'s and whole spaces, and the *"touches"* relation between *sbe*'s.

3.3. Example layout

The structure of network-based space layouts and analysis of their SRNs is illustrated with an example layout (Figs. 2 – 5). Each figure features selected spatial relations between *le*'s that form a subnetwork of the example layout's SRN. Such a subnetwork is referred to as a filtered SRN. Throughout this paper, figures show filtered rather than entire SRNs in order to simplify visualizations. Similarly, whole space volumes are offset.

In Fig. 2, the filtered SRN includes base relations $PB_{SBE,WS}$, T_{SBE} and relation A_{WS} that is derived from these relations. The weight of each whole space node corresponds to the distance from that node to the nearest *sbe* node with a degree (that is, number of incident edges) of 1. Such an *sbe* is part of the (not explicitly modeled) building perimeter. Distances correspond to the length (that is, the number of edges) of the shortest path between node pairs in the filtered SRN. Two whole spaces are perimeter spaces because they have a weight of 1. Conversely, the whole space with weight 2 is a non-perimeter space. The distinction of perimeter and non-perimeter spaces is relevant for thermal design.

In Fig. 3, the filtered SRN includes base relations A_{SS} (*"is adjacent to"*) and $N_{SS,SE}$ (*"is near"*) between subspaces and *se*'s. Subspace node weights correspond to the distance to the nearest door se_{pe_ws} node with degree 1. Such a door is an exit door. The two most distant subspaces have weights of 4. This information is useful for pedestrian circulation design. Subspace volumes correspond to geodesic Voronoi cells [51] that are derived from whole space boundaries (used as obstacles) and subspace positions (used as sites). Other types of subspace volume supported by the schema include sphere and cone volumes.

In Fig. 4, the filtered SRN includes base relations $N_{SE,SBE}$ (*"is near"*), $PB_{SBE,SS}$ (*"partially bounds"*), and derived relation $PE_{SE,SS}$ (*"partially encloses"*) between *se*'s, *sbe*'s, and subspaces. Subspace node weights correspond to the distance to the nearest window se_{pe_ws} node. Only paths that do not include doors are considered to compute distances. Subspaces with weight 1 are near windows, whereas those with greater weights are more remote. This information is useful for lighting design. One subspace has an undefined weight as it is partially enclosed only by a door and there is no path to windows that does not include doors in the filtered SRN.

In Fig. 5, the filtered SRN includes derived relations $PE_{SE,WS}$ (*"partially encloses"*) and N_{SE} (*"is near"*) between *se*'s and whole spaces. Whole space node weights correspond to the distance to the nearest door node with degree 2. Such a door is an exit door. Whole spaces with weight 1 have an exit door. This

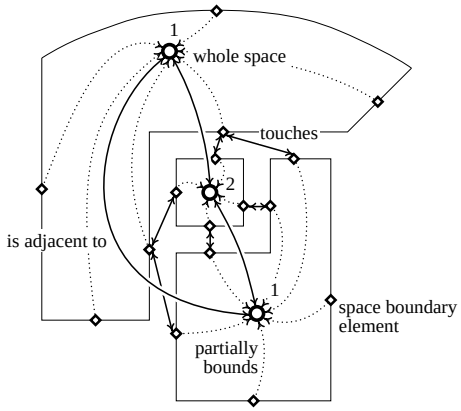

Figure 2: Example layout. The filtered SRN includes base relations $PB_{SBE,WS}$ (*"partially bounds"*), T_{SBE} (*"touches"*), and derived relation A_{WS} (*"is adjacent to"*) between whole spaces and *sbe*'s. Weights correspond to distances from whole space nodes to nearest *sbe* nodes with degree 1.

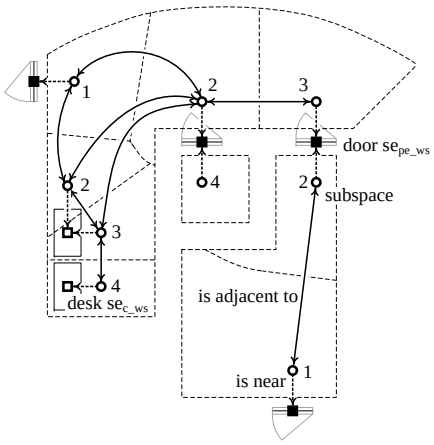

Figure 3: Example layout. The filtered SRN includes base relations A_{SS} (*"is adjacent to"*) and $N_{SS,SE}$ (*"is near"*) between subspaces and *se*'s. Weights correspond to distances from subspace nodes to nearest door nodes with degree 1.

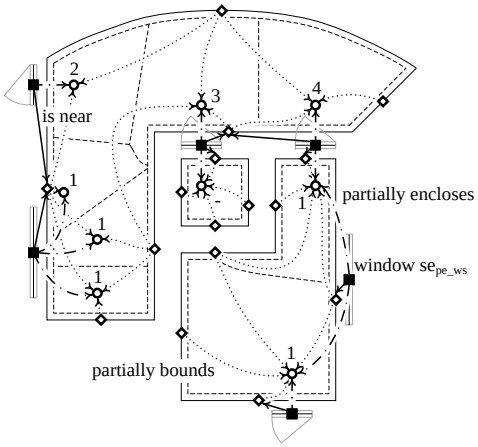

Figure 4: Example layout. The filtered SRN includes base relations $N_{SE,SBE}$ (*"is near"*), $PB_{SBE,SS}$ (*"partially bounds"*), and derived relation $PE_{SE,SS}$ (*"partially encloses"*) between *se*'s, *sbe*'s, and subspaces. Weights correspond to distances from subspace nodes to nearest window nodes. Distances are derived from paths that do not include doors.

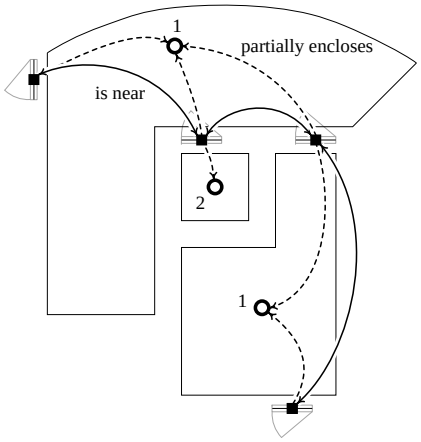

Figure 5: Example layout. The filtered SRN includes derived relations $PE_{SE,WS}$ (*"partially encloses"*) and N_{SE} (*"is near"*) between *se*'s and whole spaces. Weights correspond to distances from whole space nodes to nearest door nodes with degree 2.

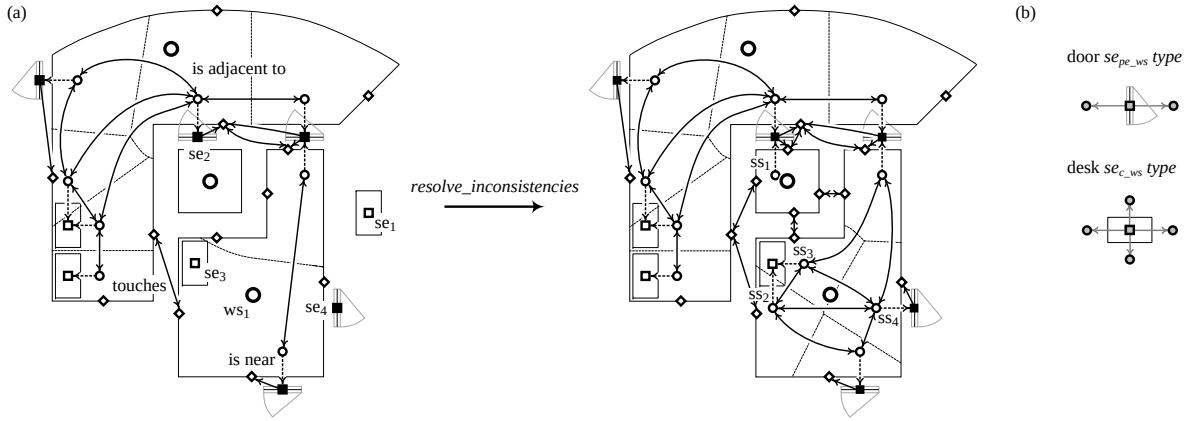

Figure 6: Inconsistency resolution example. (a) Spatial inconsistencies in the argument layout (left hand side) are identified and resolved by the *resolve_inconsistencies* operation in the result layout (right hand side). (b) Door and desk *se* types used by the operation in this example.

information is useful for evacuation design.

Similarly, metrics for the analysis of space connectivity networks [52, 53] may be applied to SRNs. For example, the filtered SRN in Fig. 3 has two cycles, the average degree of subspace nodes is 2.8, and the average distance (or depth) of subspaces with respect to exit doors is 2.4.

3.4. Resolution of spatial inconsistencies

The concept of spatial consistency of layouts is relevant for converting space layouts created in BIM authoring systems to network-based space layouts and for layout operations presented in Sections 4 and 5. In general, spatial consistency refers to the correctness and accuracy of spatial data [54]. A spatially consistent layout meets certain constraints on spatial relations between le's [8]. For example, no pair of whole spaces in a layout may overlap, or each subspace must be contained in a whole space. An inconsistency resolution operation that evaluates spatial constraints to identify and resolve spatial inconsistencies in layouts is described in [8]. Inconsistencies are resolved interactively by users or automatically based on pre-defined rules.

The inconsistency resolution operation is illustrated with an example in Fig. 6. The argument layout on the left hand side of Fig. 6 (a) features several spatial inconsistencies that are identified and resolved automatically. For example, space element se_1, which is a desk se_{c_ws} (whole space contained se), is not contained in a whole space. It is removed and not included in the consistent result layout on the right hand side of Fig. 6 (a). Most inconsistencies are resolved by removal of le's. On the other hand, four subspaces $ss_1 \ldots ss_4$ are inserted to ensure that space elements se_2, se_3, and se_4 have all subspaces that are feasible in their contexts. Missing subspaces are instantiated based on user-defined se types (Fig. 6 (b)). In case of desk se_3, only ss_2 and ss_3 are feasible because the desk is located in the corner of its whole space. Since these are new subspaces, all subspace volumes in whole space ws_1 are inconsistent and must be regenerated. The

inconsistency resolution operation also ensures the consistency of sr's in an SRN. For example, an existing A_{SS} element in ws_1 is removed and seven new ones are inserted in order to correctly reflect adjacencies between subspaces. The consistency of sr's that are not shown in the figure is maintained in a similar manner.

Space layouts created in BIM authoring systems typically do not include sbe's, sr's, and subspaces. These spatial inconsistencies may be identified and resolved with the *resolve_inconsistencies* operation. As will be shown in Section 7, it is thus feasible to model multiple space views by reusing existing, spatially inconsistent space layouts.

4. Fundamental layout operations

4.1. Overview

Fundamental layout operations are defined that support modeling of multiple space views. They are part of an emerging space modeling language. A formal definition of the language's syntax, however, is not an issue in this paper. Unary operations (Section 4.2) accept an argument layout and generate a result layout. They include *select*, *aggregate*, *decompose*, and $update_{WEIGHT}$ operations. Binary operations (Section 4.3) accept two argument layouts and generate a result layout. They include *union*, *intersect*, and *subtract* operations. Variants of operations such as the *update* operation are desirable but not described here for space reasons.

With the exception of the $update_{WEIGHT}$ operation, which does not modify a layout's structure, the inconsistency resolution operation (Section 3.4) is invoked in the processing of each operation in order to identify and resolve spatial inconsistencies that are due to prior modifications of a layout. That is, result layouts returned by fundamental operations are spatially consistent. This execution behavior has two benefits. First, users are relieved from manually maintaining the spatial consistency of layouts, which is a time-consuming and error-prone task. Second, it is feasible to safely compose (or chain) operations into expressions as there are no spatial inconsistencies in layouts that could cause operations to fail or generate misleading results.

4.2. Unary operations

4.2.1. Select

The *select* operation selects le's from an argument layout E based on a filter on le's in E. For example, a layout consisting of whole spaces that are offices may be selected with the *select* operation from an argument layout that also includes service rooms and circulation spaces. The operation has the signature

$$select_{F_{LE}}(E)$$

where

— $F_{LE} = ({}_{p_1}(LE), {}_{p_2}(LE), \ldots, {}_{p_n}(LE))$ is a filter that consists of a list of predicates on the set LE of all le's in E – le's that pass F_{LE} are selected to be included in the result layout together with certain spatially related, non-passing le's (as defined below), and

— E is a layout (a layout operation expression).

If there is a predicate on whole spaces in F_{LE}, then whole spaces are selected explicitly based on the values of predicates in F_{LE}. If there is no such predicate, then whole spaces are selected implicitly if they are spatially related to subspaces, se's, or sbe's that are selected explicitly. Similarly, if an se is selected explicitly but not its nearby subspaces, then these subspaces are selected implicitly (and vice versa).

More specifically, operation processing involves these steps:

1. Filter F_{LE} is evaluated to explicitly select le's from the argument layout E to be included in the result layout.
2. If there is no predicate on whole spaces in F_{LE}, then whole spaces that are spatially related to explicitly selected le's are selected implicitly. A whole space ws is related to a layout element le if le is a
   (a) subspace that is contained in ws,
   (b) whole space contained se (se_{c_ws}) that is contained in ws,
   (c) partially whole space enclosing se (se_{pe_ws}) that partially encloses ws, or a
   (d) space boundary element that partially bounds ws.
3. If there are explicitly selected se's that are near non-selected subspaces, then these subspaces are selected implicitly (and vice versa).
4. The intermediate layout generated by steps 1.–3. is passed to the *resolve_inconsistencies* operation to generate the spatially consistent result layout (Section 3.4).

A layout that is passed to the *select* operation does not need to be spatially consistent. The *resolve_inconsistencies* operation, which is invoked in step 4, ensures that the result layout is spatially consistent. It is thus feasible to retrieve le's from space layouts created in BIM authoring systems, which are typically spatially inconsistent, as mentioned in Section 3.4. This facilitates reuse and subsequent modification of existing space layouts with other layout operations.

If whole spaces are selected neither explicitly nor implicitly, then an empty layout is returned by *select*. An empty layout is considered as a valid, spatially consistent layout, however, it does not include any le's and sr's [8].

Examples of the *select* operation are shown in Fig. 7. In the first example (Fig. 7, top right), all whole spaces are selected explicitly from the argument layout A (Fig. 7, top left). A relational algebra style notation is used to make filter definitions concise and easy to read [55]. The wildcard ($*$) predicate is read as 'all'. The *resolve_inconsistencies* operation inserts missing sbe's and sr's because the intermediate layout includes

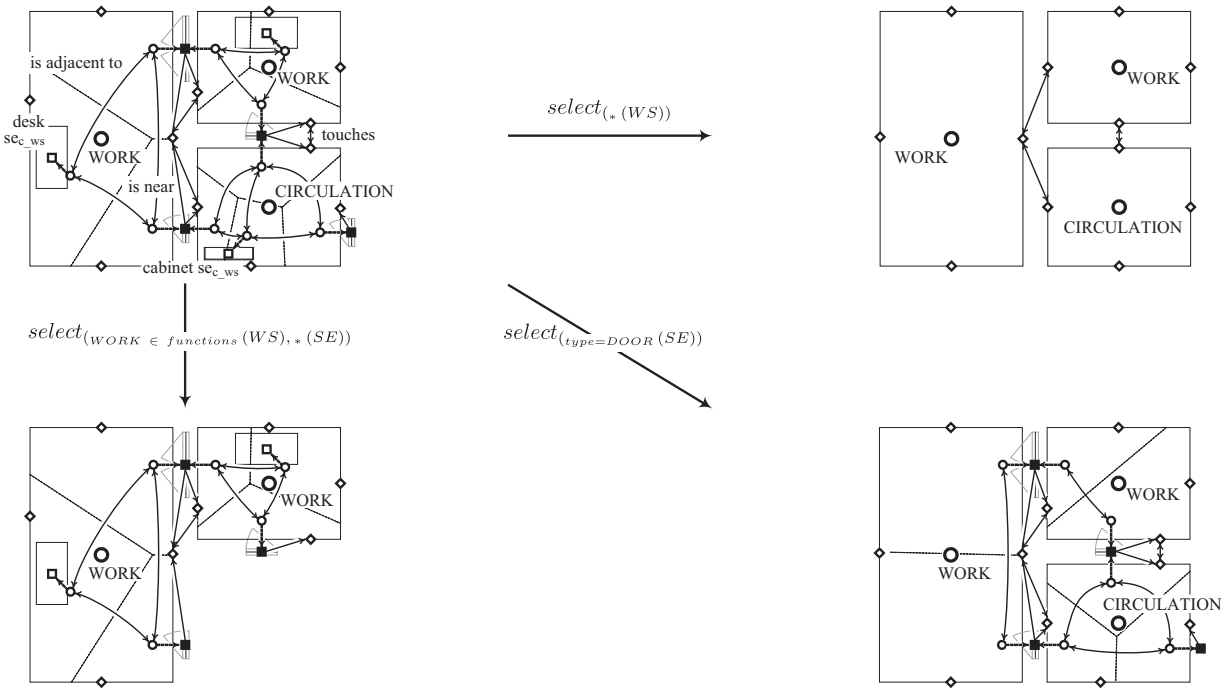

Figure 7: Examples of the *select* operation.

whole spaces only (step 4). In the second example (Fig. 7, down left), whole spaces are selected explicitly from A by a predicate on their $functions$ attribute. That is, whole spaces that are $WORK$ spaces are selected. All se's are selected by the second predicate. The cabinet se_{c_ws} in the $CIRCULATION$ whole space is selected explicitly (step 1) but not contained in a selected whole space. It is therefore removed when inconsistencies in the intermediate layout are resolved. Subspaces that are near explicitly selected door and desk se's are selected implicitly (step 3). In the third example (Fig. 7, down right), only doors are selected explicitly from A. All whole spaces are selected implicitly because they are partially enclosed by doors (step 2 (c)). On the other hand, only those subspaces that are near doors are selected implicitly (step 3). Consequently, subspace volumes are inconsistent and need to be regenerated by the *resolve_inconsistencies* operation (step 4).

4.2.2. Aggregate

The *aggregate* operation merges sets of whole spaces in an argument layout E into larger whole spaces. Whole spaces in a merge set belong to the same group, as specified by whole space attributes, and are connected in a filtered SRN of E, as specified by filters on le's and sr's. Spatially consistent se's, subspaces, and sbe's are also included in the result layout. For example, the *aggregate* operation may be used to derive a layout with whole spaces that result from merging whole spaces with the same function in an argument

layout. The operation has the signature

$$aggregate_{\,G_{WS},\,F_{LE},\,F_{SR}}\,(E)$$

where

— $G_{WS} = (g_{ws,1}, g_{ws,2}, \ldots, g_{ws,k})$ is a list of whole space attributes that are used to group whole spaces in E,

— $F_{LE} = ({}_{p_1}(LE),\ {}_{p_2}(LE), \ldots,\ {}_{p_m}(LE))$ is a filter on le's in E – le's that pass F_{LE} are selected as nodes in the filtered SRN of E,

— $F_{SR} = ({}_{p_1}(SR),\ {}_{p_2}(SR), \ldots,\ {}_{p_n}(SR))$ is a filter on sr's in E – sr's that pass F_{SR} are selected as edges in the filtered SRN of E, and

— E is a layout.

Whole space grouping is inspired by grouping in aggregation operations in relational algebra [55]. Le's and sr's that pass filters F_{LE} and F_{SR} define the filtered SRN that is used to determine whole space connectivity.

Operation processing involves these steps:

1. The set of whole spaces WS in the argument layout E is partitioned into merge sets $\{WS_1 \subset WS, WS_2 \subset WS, \ldots, WS_n \subset WS\}$ such that all whole spaces in a merge set WS_i belong to the same whole space group and are connected. Moreover, no pair of whole spaces in different merge sets belong to the same group and are connected. Whole space groups are determined based on whole space attributes, as specified by G_{WS}. Whole spaces in a group have the same values for all G_{WS} attributes. Whole space connectivity is determined based on the filtered SRN of E, as specified by F_{LE} and F_{SR}. Directions of sr's are ignored to determine whole space connectivity.
2. Whole spaces in a merge set WS_i with more than one whole space are replaced by a new whole space ws. The volume of ws corresponds to the solid union [48] of whole space volumes in WS_i.
3. The intermediate layout consisting of new and unmodified whole spaces, subspaces, sbe's, and se's is passed to the *resolve_inconsistencies* operation to generate the spatially consistent result layout.

Additional processing is required to derive values for non-spatial attributes of new whole spaces. An example is the $functions$ attribute. This aspect, which is also relevant for other operations, is not addressed in this paper.

Examples of the *aggregate* operation are shown in Fig. 8. In the first example (Fig. 8, top right), grouping of whole spaces in the argument layout A (Fig. 8, top left) is done based on the whole space functions attribute. The filtered SRN of A includes whole spaces and the A_{WS} relation. Two whole spaces that are $WORK$ spaces are adjacent and thus merged. In the second example (Fig. 8, down left), aggregation is done

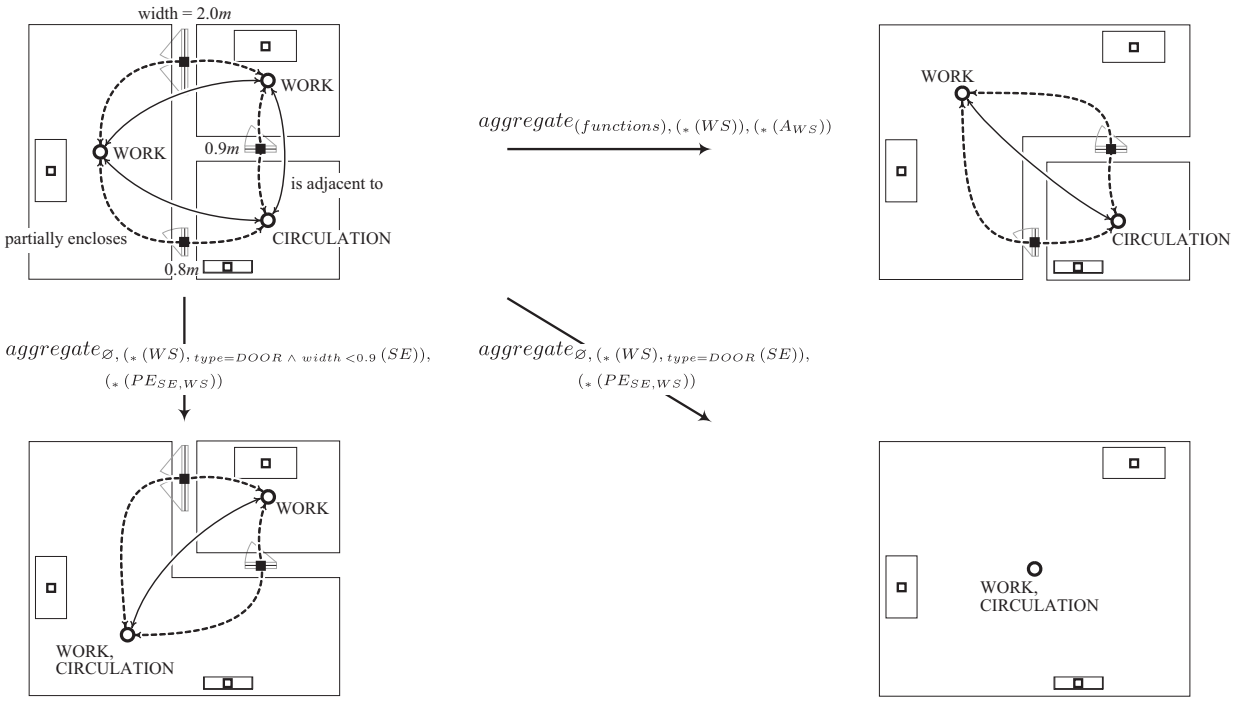

Figure 8: Examples of the *aggregate* operation.

based on whole space connectivity only as no whole space grouping attributes are specified ($G_{WS} = \varnothing$). The filtered SRN includes whole spaces, door *se*'s that are less than 0.9 m wide, and the $PE_{SE,WS}$ relation. A $WORK$ and a $CIRCULATION$ whole space are connected via a door that is 0.8 m wide and therefore merged into a new whole space in the result layout. In the third example (Fig. 8, down right), filters are similar to those in the previous example, except that all doors are included in the filtered SRN. Since all whole spaces are connected by doors, they are merged into a single whole space.

4.2.3. Decompose

The *decompose* operation breaks down whole spaces in an argument layout E into smaller whole spaces based on a filter on subspaces in E. The volumes of the smaller whole spaces are derived from the volumes of decomposed whole spaces and the positions of subspaces selected by the filter. Decomposed whole spaces are replaced by smaller whole spaces. Whole spaces that are not decomposed as well as spatially consistent *se*'s, subspaces, and *sbe*'s are also included in the result layout. For example, the *decompose* operation may be used to derive a layout from an argument layout where subspaces that are near luminaires are used to decompose whole spaces in which they are contained. The operation has the signature

$$decompose_{F_{LE}}(E)$$

where

— $F_{LE} = ({}_{p_1}(LE), {}_{p_2}(LE), \ldots, {}_{p_n}(LE))$ is a filter on attributes of le's in E – subspaces that pass F_{LE} are selected to decompose whole spaces together with certain non-passing subspaces that are spatially related to passing le's (as defined below), and

— E is a layout.

A subspace is selected explicitly if it passes F_{LE}, and implicitly if it is spatially related to an le that passes F_{LE}. Whole spaces that contain more than one explicitly or implicitly selected subspace are decomposed. The volume of each decomposed whole space and the positions of selected subspaces that are contained in it are used to derive geodesic Voronoi cells ([51], Section 3.3). These cells become the volumes of new whole spaces that replace the decomposed whole space. Any subspace in an argument layout, regardless of its volume type, may be selected to decompose whole spaces. For example, subspaces with sphere or cone volumes may be used as well as subspaces with geodesic Voronoi volumes.

Operation processing involves these steps:

1. Filter F_{LE} is evaluated to explicitly select le's from the argument layout E. A subspace is selected explicitly to decompose whole spaces if it passes F_{LE}.
2. A subspace is selected implicitly if it does not pass F_{LE} but is spatially related to an explicitly selected layout element that is not a subspace. A subspace ss is related to a layout element le if le is a
   (a) whole space that contains ss,
   (b) space boundary element that partially bounds ss, or a
   (c) space element that is near or partially encloses ss.
3. Each whole space which contains at least two selected subspaces is decomposed. Positions of these subspaces and the volume of the decomposed whole space are used as, respectively, sites and obstructions to generate geodesic Voronoi cells. Whole spaces that contain no or only one subspace are not decomposed.
4. A new whole space is created for each selected subspace that generated a geodesic Voronoi cell in step 4. The cell becomes the volume of the new whole space.
5. Decomposed whole spaces are removed.
6. The intermediate layout consisting of new and unmodified whole spaces, subspaces, *sbe*'s, and *se*'s is passed to the *resolve_inconsistencies* operation to generate the spatially consistent result layout.

Examples of the *decompose* operation are shown in Fig. 9. In the first example (Fig. 9, top right), whole spaces in the argument layout A (Fig. 9, top left) are decomposed based on subspaces $ss_1 \ldots ss_8$. These subspaces are selected explicitly based on their weight, which is 2. Each whole space in the result layout contains a luminaire because $ss_1 \ldots ss_8$ are near luminaires. In the second example (Fig. 9, down left), only door *se*'s are selected explicitly. Subspaces $ss_9 \ldots ss_{15}$ are selected implicitly to decompose whole spaces

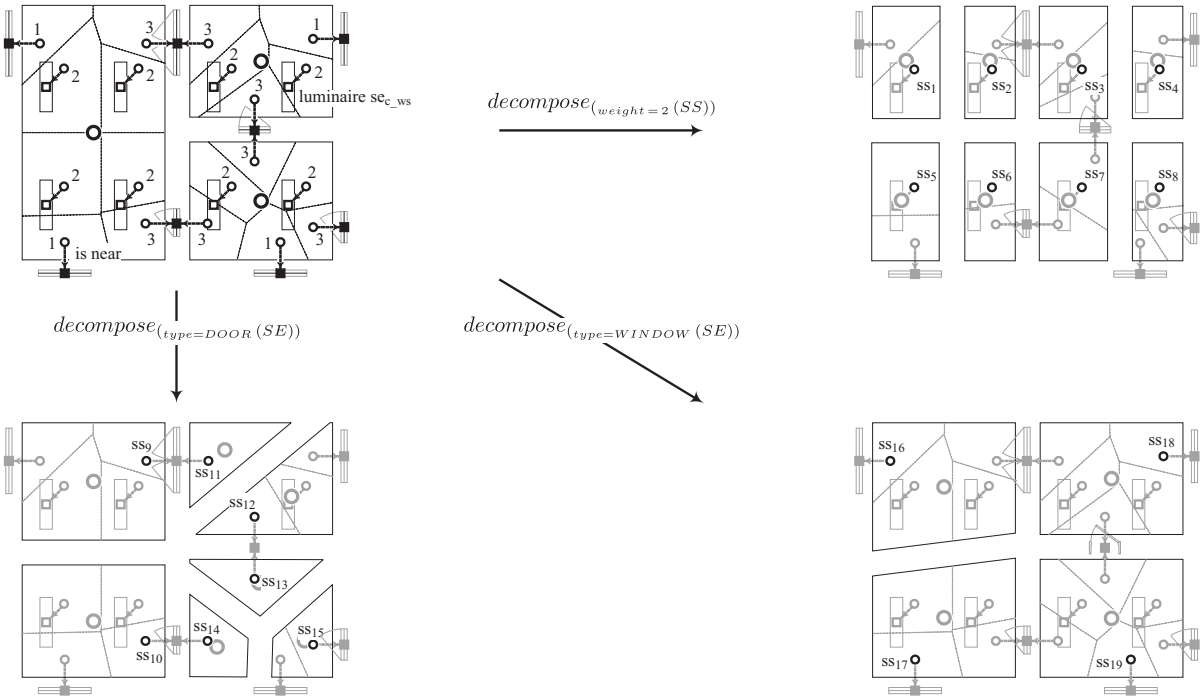

Figure 9: Examples of the *decompose* operation. Subspaces $ss_1 \ldots ss_{19}$ are selected to decompose whole spaces.

because they are near doors. Three luminaires are not properly contained in new whole spaces and therefore removed when the intermediate layout is converted to the spatially consistent result layout (step 6). In the third example (Fig. 9, down right), whole spaces are decomposed based on implicitly selected subspaces $ss_{16} \ldots ss_{19}$ that are near explicitly selected window *se*'s. Subspaces ss_{18} and ss_{19} are the only selected subspaces in their whole spaces. Therefore these whole spaces are not decomposed (step 3).

4.2.4. Update$_{WEIGHT}$

The *update*$_{WEIGHT}$ operation derives weights of shortest paths from source *le*'s to nearest destination *le*'s in an argument layout E and assigns these to weight attributes of source *le*'s. Shortest paths are restricted to a filtered SRN of E, as specified by filters on *le*'s and *sr*'s. For example, the *update*$_{WEIGHT}$ operation may be used to derive a layout from an argument layout where luminaire weights correspond to the weights of shortest paths from luminaires to nearest windows. The operation has the signature

$$update_{\,WEIGHT,\,F_{LE_s},\,F_{LE_d},\,F_{LE},\,F_{SR}}\,(E)$$

where

— $F_{LE_s} = (_{p_{s,1}}(LE),\ _{p_{s,2}}(LE), \ldots,\ _{p_{s,k}}(LE))$ is a filter on *le*'s in E – *le*'s that pass F_{LE_s} are selected as source *le*'s (le_s's),

— $F_{LE_d} = ({}_{p_{d,1}}(LE), {}_{p_{d,2}}(LE), \ldots, {}_{p_{d,l}}(LE))$ is a filter on le's in E – le's that pass F_{LE_d} are selected as destination le's (le_d's),

— $F_{LE} = ({}_{p_1}(LE), {}_{p_2}(LE), \ldots, {}_{p_m}(LE))$ is a filter on le's in E – le's that pass F_{LE} are selected as nodes in the filtered SRN of E,

— $F_{SR} = ({}_{p_1}(SR), {}_{p_2}(SR), \ldots, {}_{p_n}(SR))$ is a filter on sr's in E – sr's that pass F_{SR} are selected as edges in the filtered SRN of E, and

— E is a layout.

Weights of le_s's selected by F_{LE_s} are updated by the operation. The result layout has the same structure as the argument layout.

Operation processing involves these steps:

1. Filter F_{LE_s} is evaluated to select le_s's in the argument layout E.
2. Filter F_{LE_d} is evaluated to select le_d's in E.
3. For each le_s the nearest le_d is determined. Search for the nearest le_d is restricted to the filtered SRN of E, as specified by F_{LE} and F_{SR}. Path weights are computed from sr weights. Weights of le's and directions of sr's are ignored.
4. For each le_s the weight of the shortest path to its nearest le_d is assigned to its weight attribute.

Examples of the $update_{WEIGHT}$ operation are shown in Fig. 10. In each example, weights of luminaire se's in argument layout A (Fig. 10, top left) are updated with respect to certain window se's. Subspace volumes are not shown in result layouts for improved visualization. For simplicity, the weight of each sr in the SRN of A is assumed to be 1. That is, luminaire weights derived by the operation correspond to lengths of shortest paths to nearest windows. Alternatively, Euclidean distances between related le's could be used as sr weights. This would result in luminaire weights that more accurately reflect distances to nearest windows. In the first example (Fig. 10, top right), all windows are selected as destination le's. Shortest paths are determined based on a filtered SRN that consists of $N_{SS,SE}$ and A_{SS} relations. In the second example (Fig. 10, down left), South oriented windows se_1 and se_2 are selected as destinations. Filter F_{LE} targets se's that are not doors and subspaces. As a consequence, the filtered SRN consists of three connected components. Weights of two luminaires in whole space ws_1 are undefined because they are disconnected from se_1 and se_2 in the filtered SRN. In the third example (Fig. 10, down right), West oriented windows are selected as destinations. The filtered SRN is similar to the third example. However, it has two rather than three connected components as it additionally includes the door that is 2 m wide. Two luminaires in whole space ws_2, which is partially enclosed by that door, have weights of 6 and 7 because they are connected to the West oriented window se_3.

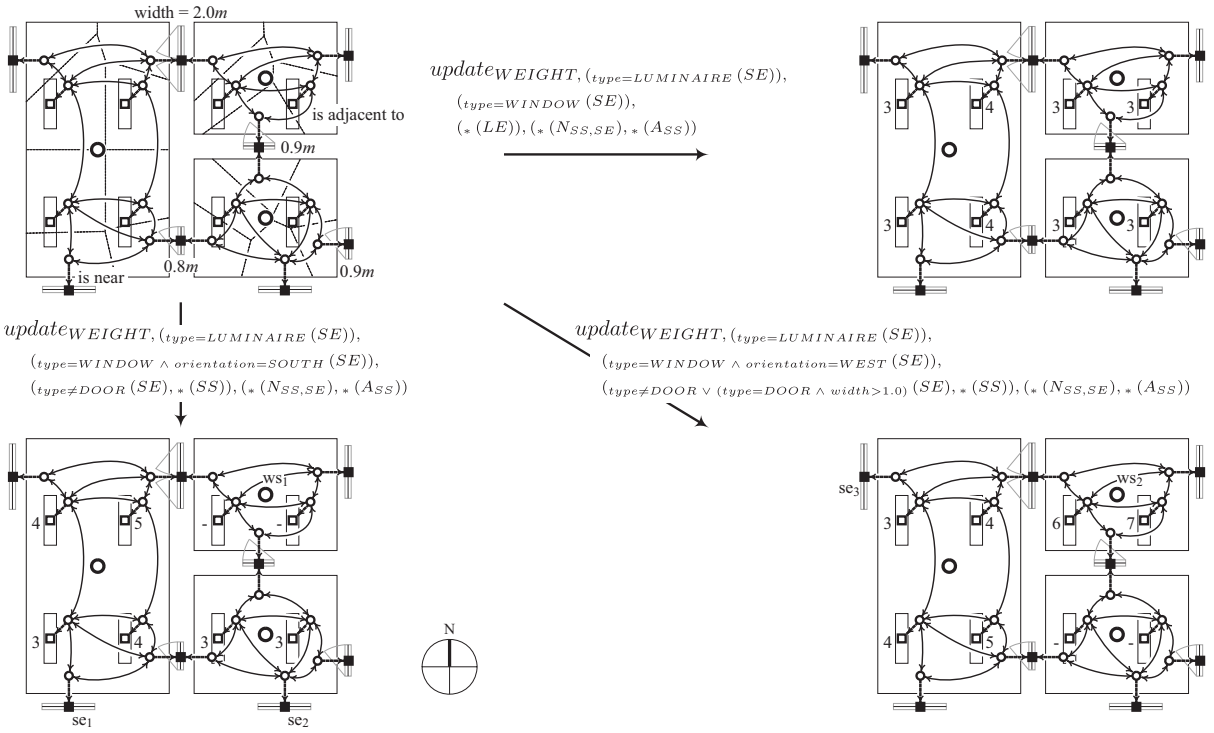

Figure 10: Examples of the *update* $_{WEIGHT}$ operation. Subspace volumes are not shown in result layouts.

4.3. Binary operations

Binary operations include *union* ($\cup$), *intersect* ($\cap$), and *subtract* ($-$) operations. As their signatures, processing, and examples are similar, each of these aspects is described for all operations.

4.3.1. Signatures

The *union* ($\cup$) operation derives a layout by merging overlapping whole spaces in argument layouts E_1 and E_2. Disjoint or touching whole spaces from E_1 and E_2 are also included in the result layout together with spatially consistent *se*'s, subspaces, and *sbe*'s. The operation has the signature

$$union\,(E_1, E_2)$$

where E_1 and E_2 are layouts.

The *intersect* ($\cap$) operation derives a layout by intersecting overlapping whole spaces in argument layouts E_1 and E_2. Spatially consistent *se*'s, subspaces, and *sbe*'s from E_1 and E_2 are also included in the result layout. The operation has the signature

$$intersect\,(E_1, E_2)$$

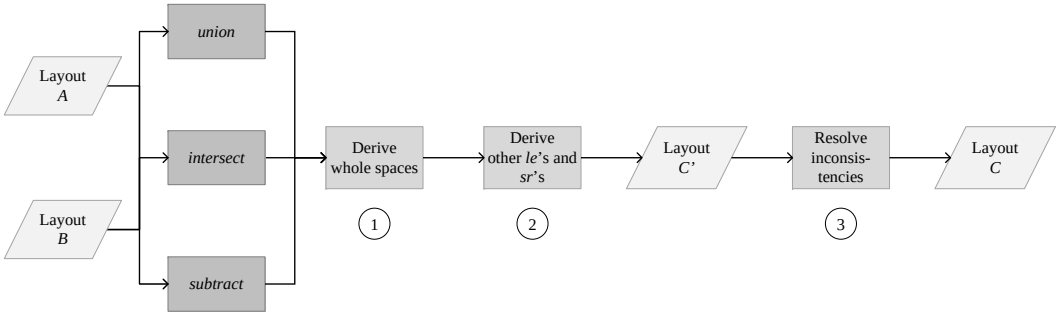

Figure 11: Processing of *union*, *intersect*, and *subtract* operations.

where E_1 and E_2 are layouts.

The *subtract* $(-)$ operation derives a layout by subtracting whole spaces in argument layout E_2 from overlapping whole spaces in argument layout E_1. Spatially consistent *se*'s, subspaces, and *sbe*'s from E_1 and E_2 are also included in the result layout. The operation has the signature

$$subtract\,(E_1, E_2)$$

where E_1 and E_2 are layouts.

4.3.2. Processing

The processing of *union*, *intersect*, and *subtract* operations involves these steps (Fig. 11):

1. Whole spaces are derived from whole spaces in argument layouts A and B.
2. Other *le*'s (that is, subspaces, *se*'s, and *sbe*'s) and *sr*'s are derived from A and B.
3. The intermediate layout C', which results from steps 1 and 2, is passed to the *resolve_inconsistencies* operation to generate the spatially consistent result layout C.

While step 1 is operation specific, steps 2 and 3 are the same for all operations.

The derivation of whole spaces (step 1) is based on the overlap relation O_{WS_A,WS_B} between whole spaces in argument layouts A and B:

$$O_{WS_A,WS_B} = \{(ws_A, ws_B) \in WS_A \times WS_B \mid clash\_bodies(ws_A.volume, ws_B.volume) \in \{CONTAINS, CONTAINS\_ABUTS, COINCIDENT, INTERLOCK\}\}$$

where

— WS_A is the set of whole spaces in argument layout A,

— WS_B is the set of whole spaces in argument layout B, and

— *clash_bodies* is a function that derives the spatial relation between a given pair of Brep solids [56].

A pair of whole space volumes are considered as overlapping if one contains the other (their boundaries may or may not abut), if they are coincident, or if their boundaries interlock (in other words, if their interiors partially overlap). Fig. 12 includes examples for O_{WS_A,WS_B} graphs.

For the *union* operation, the set $WS_{C'}$ of whole spaces in the intermediate layout C' is derived as follows (Fig. 12 (a)):

1. Each whole space in argument layouts A and B which does not participate in the O_{WS_A,WS_B} relation (that is, which does not overlap with any whole space in the other layout) is inserted in $WS_{C'}$.
2. A new whole space $ws_{C',i}$ is created and inserted in $WS_{C'}$ for each connected component $O_{WS_A,WS_B,i}$ of the O_{WS_A,WS_B} relation graph with more than one whole space node. Whole spaces that are nodes in $O_{WS_A,WS_B,i}$ are merged into $ws_{C',i}$. The volume of $ws_{C',i}$ is derived by solid union of the volumes of these merged whole spaces.

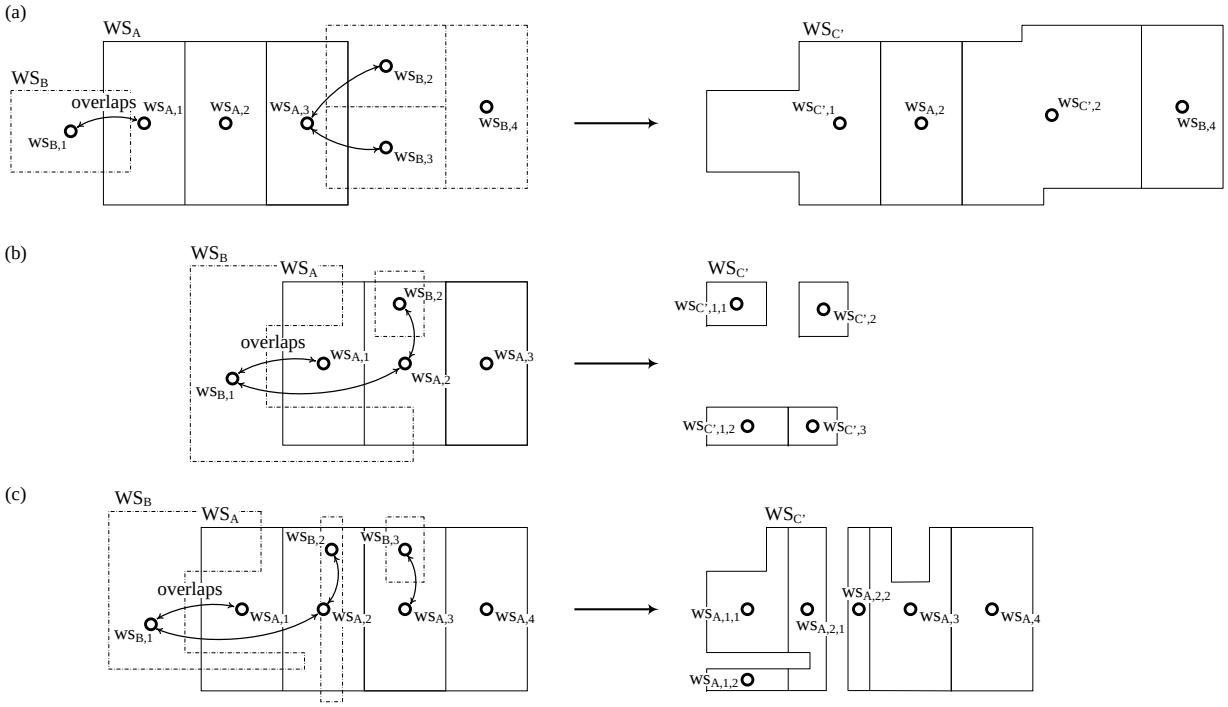

Figure 12: Examples for the derivation of whole space set $WS_{C'}$. (a) *union*. (b) *intersect*. (c) *subtract*.

For the *intersect* operation, the set $WS_{C'}$ of whole spaces in the intermediate layout C' is derived as follows (Fig. 12 (b)):

1. A new whole space $ws_{C'}$ is created for each pair of overlapping whole spaces ws_A and ws_B in A and B. The volume of $ws_{C'}$ corresponds to the solid intersection of ws_A and ws_B volumes.
2. If the $ws_{C'}$ volume has a single part (or shell), then $ws_{C'}$ is inserted in $WS_{C'}$.

3. If the $ws_{C'}$ volume has multiple, disconnected parts, then $ws_{C'}$ is split. Each part becomes the volume of a new whole space which is inserted in $WS_{C'}$.

For the *subtract* operation, the set $WS_{C'}$ of whole spaces in the intermediate layout C' is derived as follows (Fig. 12 (c)):

1. Each whole space in A (first argument layout) which does not overlap with any whole space in layout B (second argument layout) is inserted in $WS_{C'}$.
2. The volume of each whole space ws_B in B which overlaps a whole space ws_A in A is subtracted from the volume of ws_A by solid subtraction.
3. If the modified ws_A volume is non-empty and has a single part, then ws_A is inserted in $WS_{C'}$.
4. If the modified ws_A volume has multiple, disconnected parts, then ws_A is split. Each part becomes the volume of a new whole space which is inserted in $WS_{C'}$.

After the derivation of whole spaces, the next step in the processing of *union*, *intersect*, and *subtract* operations involves deriving other *le*'s and *sr*'s in the the intermediate layout C' (Fig. 11, step 2). This is done by union of subspace, *sbe*, *se*, and *sr* sets from A and B. Duplicate *le*'s, that is, *le*'s with the same unique identifier, are discarded. An *sr* element is included in C' only if related *le*'s are also included and if it is not a duplicate.

4.3.3. Examples

Union, *intersect*, and *subtract* operations are illustrated with examples in Fig. 13. Argument layouts A and B are at the top of the figure. Result layouts returned by each operation are at the bottom.

The union of argument layouts A and B returns the result layout at bottom left of Fig. 13. Whole space $ws_{A,1}$ in A overlaps with whole space $ws_{B,1}$ in B. An L-shaped, merged whole space $ws_{C,1}$ is created in the result layout which replaces $ws_{A,1}$ and $ws_{B,1}$. Its volume is derived by solid union of the volumes of $ws_{A,1}$ and $ws_{B,1}$. Whole space $ws_{A,2}$ in A touches $ws_{B,1}$ and is thus included in the result layout without modification. Door *se*'s in A and B are included because they are spatially consistent. Subspaces are included as well, however, their volumes are inconsistent and regenerated by the *resolve_inconsistencies* operation.

The intersection of A and B returns the result layout at bottom center in Fig. 13. Whole spaces $ws_{A,1}$ and $ws_{B,1}$ overlap and are replaced by whole space $ws_{C,2}$ in the result layout. Solid intersection is applied to the volumes of $ws_{A,1}$ and $ws_{B,1}$ to create the volume of $ws_{C,2}$. Whole space $ws_{A,2}$ is removed as it does not overlap with $ws_{B,1}$. Only one door and one subspace from A and B are spatially consistent and included in the result layout.

The subtraction of B from A returns the result layout at bottom right in Fig. 13. The volume of $ws_{A,1}$ is modified by solid subtraction of the volume of $ws_{B,1}$. Whole space $ws_{A,2}$ is included in the result layout

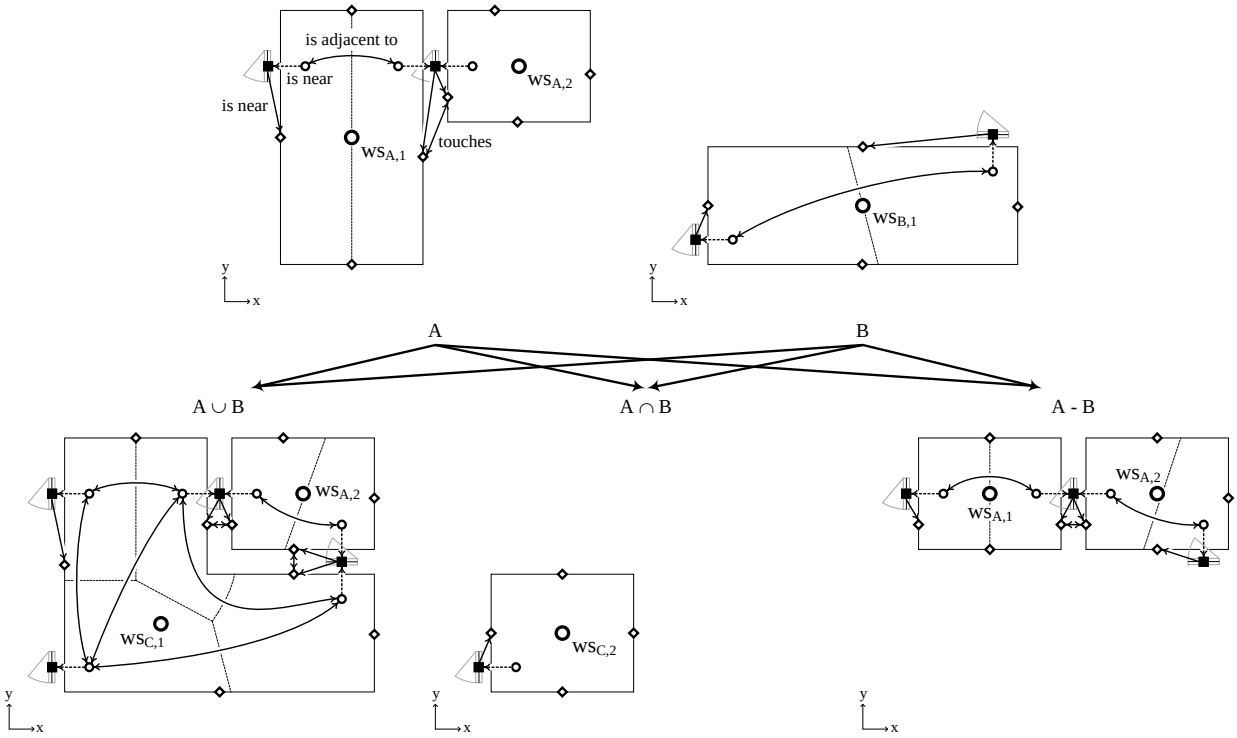

Figure 13: Examples of *union*, *intersect*, and *subtract* operations.

without modification as it does not overlap with $ws_{B,1}$. Three doors and three subspaces from A and B are spatially consistent and included in the result layout, and one missing subspace is inserted. All subspace volumes are regenerated.

5. Additional layout operations

Fundamental layout operations defined in the previous section may be composed into layout operation expressions. Common expressions may be defined as additional layout operations. With these operations, it is feasible to develop expressions that are more concise and easier to read.

An example of an additional layout operation is the symmetric difference ($\triangle$) operation, which is a composition of *subtract* and *union* operations:

$$A \triangle B = (A - B) \cup (B - A)$$

The symmetric difference operation in turn may be used to define an *overlay* operation [57]:

$$A \; overlay \; B = (A \cap B) \cup (A \triangle B)$$

Fig. 14 shows an example of the *overlay* operation. Argument layouts A and B are at the top of Fig. 14 and the result layout at the bottom.

6. Implementation

We have extended an existing layout modeling system (LMS) prototype with fundamental and additional layout operations as described in previous sections. LMS is written in C++ [58]. It relies on geometric modeling, solid modeling, and graph libraries to represent network-based space layouts and process layout operations.

Layout element geometries are modeled in LMS with boundary representation (Brep) solid data structures of the Acis geometric and solid modeling engine [8, 50]. Whole space volumes in layouts returned by *aggregate*, *union*, *intersect*, and *subtract* operations are generated by Boolean operations on Brep solids in Acis [59]. Clash detection functions in Acis are used to derive the overlap relation O_{WS_A,WS_B} between whole spaces in the processing of *union*, *intersect*, and *subtract* operations [56]. Connected components and shortest paths in an SRN are determined with graph algorithm functions in the Boost graph library [60]. These functions are accessed to process *aggregate* and $update_{WEIGHT}$ operations. Functions in the computational geometry algorithms library (CGAL) are used to generate Euclidean Voronoi diagrams from which volumes of new whole spaces are derived in the *decompose* operation [51, 61]. We use Euclidean rather than geodesic Voronoi diagrams because implementations of the former are currently not available [62].

7. Layout operation expression example

7.1. Zoning for security lighting

The feasibility of modeling of multiple space views with layout operation expressions is shown with an example in which a security lighting layout of a floor of an existing office building is automatically generated from an architectural layout. The example was implemented in the LMS prototype. It further demonstrates reuse of space layouts created in BIM authoring systems.

An objective in security lighting control is to achieve low, uniformly distributed illuminance levels in offices during unoccupied hours. A security lighting layout should include security lighting zones that contain luminaires. This facilitates the assignment of luminaires to lighting controllers. Zones should reflect proximity of luminaires to windows. This allows a controller to perform daylight harvesting e.g. on weekends by dimming luminaires that are near naturally lit windows more than distant ones. Zones should further reflect organizational factors. That is, security lighting may be controlled separately for spaces used by different organizational units.

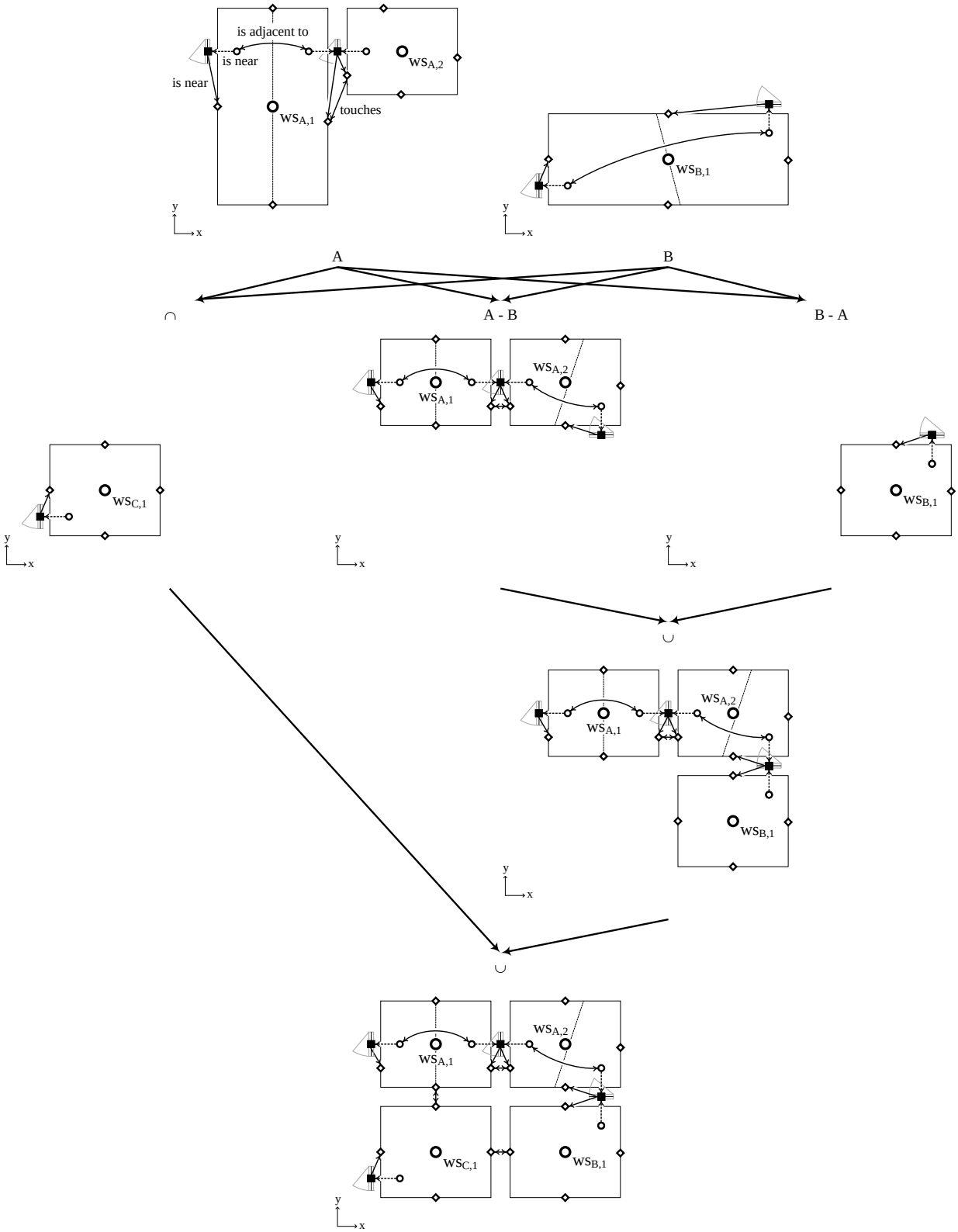

Figure 14: Example of the *overlay* operation.

7.2. Expression

We defined and implemented a layout operation expression in the LMS prototype according to these zoning requirements for security lighting in offices. The expression accepts an architectural layout A (Section 7.3) as input and returns a security lighting layout:

```
// security lighting layout
overlay (
  // intermediate security lighting layout
  aggregate_(weight), (* (WS)), (* (A_WS)) (
    update_WEIGHT, (* (WS)), (type=WINDOW (SE)), (* (LE)), (* (A_WS), * (PE_SE,WS)) (
      decompose_(type=LUMINAIRE (SE)) (
        select_(WORK ∈ functions (WS), type=LUMINAIRE(SE), type=WINDOW(SE)) ( A )
  ) ) ),
  // fire safety layout
  aggregate_∅, (* (WS), type=DOOR ∧ fire_rating<180 (SE)), (* (PE_SE,WS)) (
    select_(type=DOOR(SE)) ( A )
  )
)
```

The expression consists of two sub-expressions. The first derives an intermediate security lighting zone layout (Section 7.4) from A. Zones in this layout do not reflect the requirement of separate control by organizational units. For the sake of simplicity, we assume that zones occupied by organizational units correspond to fire safety zones. The latter are derived based on fire-resistance properties of doors. Thus the second sub-expression derives a fire safety layout (Section 7.5) from A. The result security lighting zone layout (Section 7.6) is obtained by *overlay* of the intermediate security lighting layout and the fire safety layout.

7.3. Architectural layout

We evaluated the layout operation expression for an architectural layout A of the second floor of an existing office building on the campus of Vienna University of Technology. Spaces on this floor are used by two departments and accessed by two stairs and elevators. Layout A includes 37 whole spaces and 49 window, 43 door, 42 desk, and 67 luminaire *se*'s (Fig. 15 (a)). Window, luminaire, and door *se* types were defined and used to evaluate the expression (Fig. 15 (b)). Layout A is spatially inconsistent as it does not include *sr* elements and subspaces that are feasible based on these *se* types.

We used Autodesk Architecture (AA) to create layout A [63]. AA is a BIM authoring system for architectural design with rich space modeling capabilities. The layout was subsequently loaded into LMS. At present, data is exchanged between AA and LMS based on the native data exchange format of the Acis

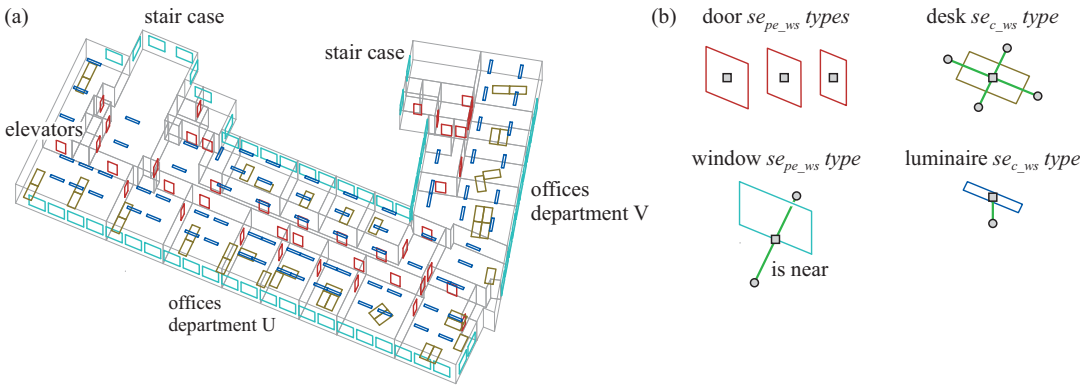

Figure 15: Architectural layout A. (a) Layout created in Autodesk Architecture BIM authoring system. (b) Door, desk, window, and luminaire se types used to evaluate the layout operation expression.

modeler. We plan to implement functions that import data into LMS via IFC in order to improve reuse of non-geometric layout data.

7.4. Intermediate security lighting layout

In the first operation of the sub-expression that derives the intermediate security lighting layout, whole spaces with $WORK$ function, including offices, meeting, and seminar rooms, are selected from the architectural layout A (Fig. 16 (a)). Luminaire and window se's that are contained in these whole spaces are selected as well. As mentioned in Section 4.2.1, a layout that is used as an argument for the *select* operation does not need to be spatially consistent. Hence it is feasible to pass the spatially inconsistent A, which was originally created in the AA BIM authoring system and loaded into LMS, to the *select* operation.

In the second operation, whole spaces are decomposed based on subspaces that are near luminaires. As a result, each new whole space in the layout returned by the *decompose* operation contains a luminaire.

In the third operation, weights of whole spaces are updated with respect to nearest windows (Fig. 16 (c)). Weights are derived based on a filtered SRN that consists of A_{WS} and $PE_{SE,WS}$ relations. Edges have weights of 1. As a result of this operation, whole spaces that are partially enclosed by a window have a weight of 1. Whole spaces that are more distant from windows have weights of 2 or 3.

In the fourth operation, whole spaces are aggregated into intermediate security lighting zones (Fig. 16 (e)). Grouping is done based on whole space weights. The filtered SRN that is used to determine whole space connectivity consists of whole spaces and the A_{WS} relation. All luminaires in a zone have the same weight or distance to the nearest window.

7.5. Fire safety layout

In the first operation of the sub-expression that derives the fire zone layout, all door se's are selected from A together with whole spaces which they partially enclose. Again, the spatially inconsistent A is passed to

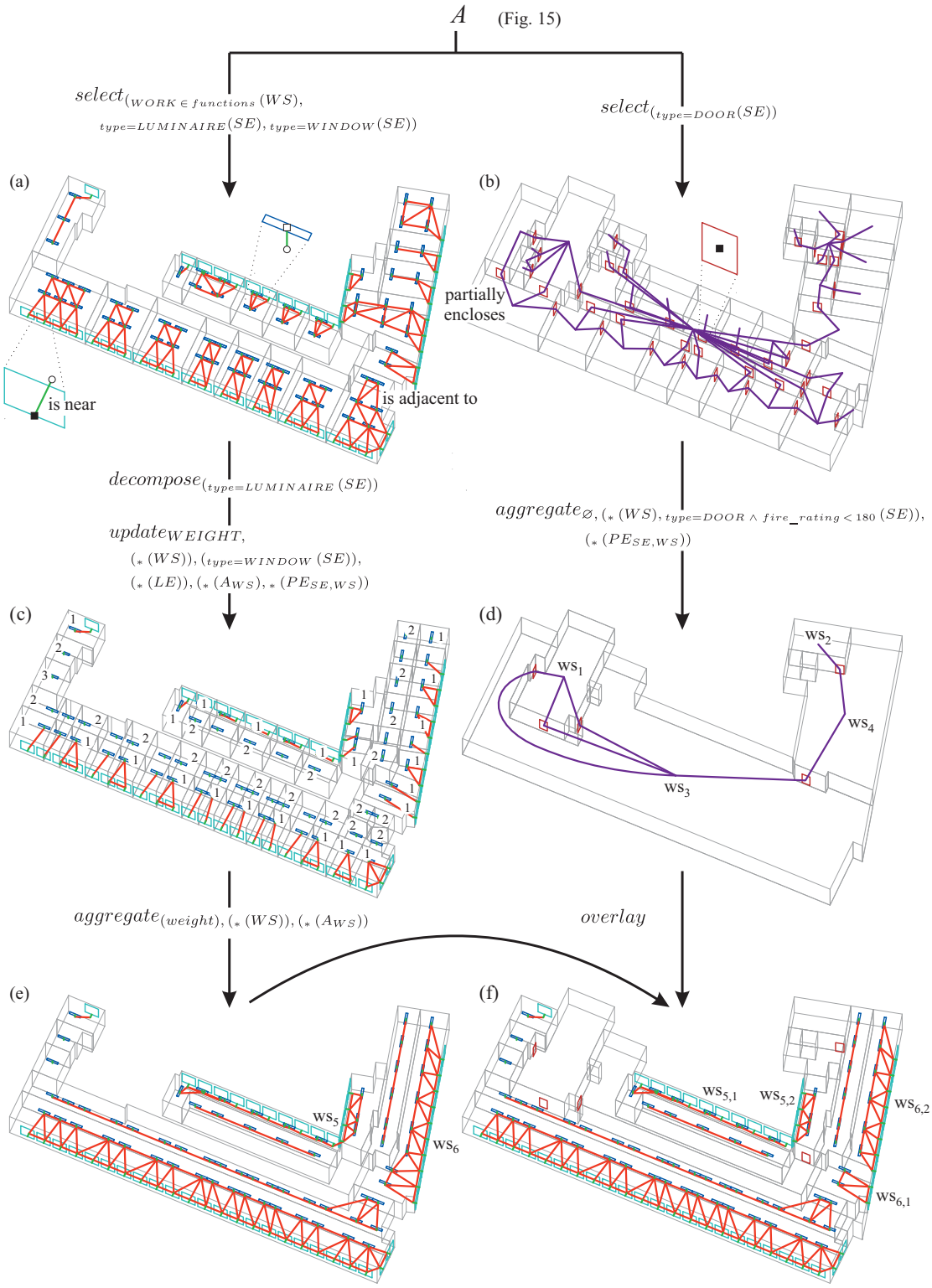

Figure 16: Layout operation expression example. (a) - (c) intermediate layouts, (d) fire safety layout, (e) intermediate security lighting layout, (f) security lighting layout.

the *select* operation. Except for inaccessible vertical duct spaces, all whole spaces in A are selected implicitly (Fig. 16 (b)). There are no subspaces in the layout resulting from this operation because it is generated with a door *se* type that does not have subspaces (Fig. 15 (b)).

In the second operation, whole spaces are aggregated into fire safety zones (Fig. 16 (d)). As no grouping attribute is specified ($G_{WS} = \varnothing$), aggregation is done based on whole space connectivity only. The filtered SRN consists of whole spaces, doors with a fire-rating of less than 180 minutes, and the $PE_{SE,WS}$ relation. That is, fire safety zones are separated by doors with a fire-rating of at least 180 minutes. Resulting fire safety zones correspond to two vertical circulation zones (ws_1 and ws_2) and two zones (ws_3 and ws_4) that are used by the two departments.

7.6. Security lighting layout

The security lighting layout returned by the layout operation expression is generated by *overlay* of intermediate security lighting and fire safety layouts (Fig. 16 (f)). Two whole spaces (ws_5 and ws_6) in the intermediate security lighting layout are split into two new whole spaces each (respectively, $ws_{5,1}$, $ws_{5,2}$ and $ws_{6,1}$, $ws_{6,2}$) because they span across fire safety zones ws_3 and ws_4. Altogether, the security lighting layout generated by the expression meets office security lighting zone requirements for daylight harvesting and separate control of spaces used by different organizational units (Section 7.1).

8. Conclusions

We have defined a set of operations on network-based space layouts that support modeling of multiple, domain-specific views of buildings. Operations may be safely composed into expressions as each operation returns a spatially consistent layout. We have implemented these operations as extensions of an existing layout modeling system prototype. With a layout operation expression example we have shown the feasibility of automatically generating a security lighting layout from an architectural layout created in a BIM authoring system.

Further layout operations are desirable to improve the expressiveness of an emerging space modeling language. The selection of layouts that include layout elements on a shortest path could be useful for indoor route planning applications. Current work aims to enhance modeling of multiple spatial abstractions of buildings with operations that derive spatial relations between layouts [64]. For example, whole spaces in different layouts may be related by containment or overlap relations. This would facilitate queries that involve the traversal of SRN's of related layouts with graph algorithms. An example is the retrieval of detailed symbolic location data, such as "the Library is located on the Ground Floor of the North Wing of the Main Building", from a collection of layouts that model a pedestrian circulation space view.

Maintaining the consistency of multiple space views is a relevant issue for future work. For example, when an architectural designer merges whole spaces in an architectural layout, dependent layouts that

model other space views become inconsistent. This could be resolved by re-evaluating expressions that generate dependent layouts. However, this approach is inadequate for addressing roundtrip or iterative design scenarios [1]. For example, modification of luminaire locations by a lighting designer would need to be reflected in the architectural layout. Additional, 'inverse' layout operation expressions are desirable to reflect such changes in the architectural layout. Similarly, the definition of reusable layout operation expressions that model space views of specific domains appears promising. Expressions could be defined to generate layouts for artificial lighting, natural ventilation, access control, and so on. On the other hand, the generality of these expressions needs to be explored, for example, their potential to model space views for classes of buildings, such as office buildings or health facilities.

Acknowledgements

The authors gratefully acknowledge support by Grant Austrian Science Fund (FWF): P22208-N22 for the work presented in this paper.

References

[1] C. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM handbook: a guide to building information modeling for owners, managers, designers, engineers and contractors, 2nd Edition, John Wiley & Sons, 2011.

[2] buildingSMART International, Industry Foundation Classes IFC2x4, [Accessed 28 January 2014] (2010). URL http://www.buildingsmart.com

[3] G. van Nederveen, F. Tolman, Modelling multiple views on buildings, Automation in Construction 1 (3) (1992) 215–224. doi:10.1016/0926-5805(92)90014-B.

[4] M. Rosenman, J. Gero, Modelling multiple views of design objects in a collaborative environment, Computer-Aided Design 28 (3) (1996) 193–205.

[5] H. Bernstein, S. Jones, J. Gudgel, B. Buckley, E. Fitch, D. Laquidara-Carr, The business value of BIM in Europe, Smart Markets Report, McGraw-Hill Construction (2010).

[6] G. Suter, Operations on network-based space layouts, in: H.-J. Bargstädt, K. Ailland (Eds.), 11th Conference on Construction Applications of Virtual Reality, Bauhaus University, Weimar, Germany, 2011, pp. 567–578.

[7] G. Suter, F. Petrushevski, M. Šipetić, Boolean operations on network-based space layouts, in: A. Borrmann, P. Geyer (Eds.), 19th Workshop of the European Group for Intelligent Computing in Engineering, EG ICE, Herrsching (Munich), Germany, 2012.

[8] G. Suter, Structure and spatial consistency of network-based space layouts for building and product design, Computer-Aided Design 45 (8-9) (2013) 1108–1127. doi:10.1016/j.cad.2013.04.004.

[9] buildingSMART International, MVD process, [Accessed 28 January 2014] (2013). URL http://www.buildingsmart.org/standards/mvd/mvd-process

[10] buildingSMART International, Coordination View Version 2.0, [Accessed 28 January 2014] (2013). URL http://www.buildingsmart-tech.org/specifications/ifc-view-definition/coordination-view-v2.0

[11] M. Weise, P. Katranuschkov, R. Scherer, Generalised model subset definition schema, in: Proceedings CIB-W78 Conference 2003–Information Technology for Construction, 2003.

[12] C. Eastman, M. Henrion, GLIDE: a language for design information systems, in: Proceedings of the 4th annual conference on Computer graphics and interactive techniques, SIGGRAPH '77, ACM, New York, NY, USA, 1977, pp. 24–33. doi:10.1145/563858.563863.

[13] C. M. Eastman, S. C. Chase, H. H. Assal, System architecture for computer integration of design and construction knowledge, Automation in Construction 2 (2) (1993) 95–107.

[14] C. Eastman, A. Siabiris, A generic building product model incorporating building type information, Automation in Construction 3 (1) (1995) 283–304.

[15] M. K. Zamanian, S. J. Fenves, E. L. Gursoz, Representing spatial abstractions of constructed facilities, Building and Environment 27 (2) (1992) 221–230. doi:10.1016/0360-1323(92)90024-J.

[16] R. Stouffs, R. Krishnamurti, K. Park, Sortal structures: Supporting representational flexibility for building domain processes, Computer-Aided Civil and Infrastructure Engineering 22 (2) (2007) 98–116. doi:10.1111/j.1467-8667.2006.00473.x.

[17] R. Stouffs, Constructing design representations using a sortal approach, Advanced Engineering Informatics 22 (1) (2008) 71–89. doi:10.1016/j.aei.2007.08.007.

[18] E. Grabska, G. Ślusarczyk, M. Glogaza, Design description hypergraph language, in: M. Kurzyński, E. Puchala, M. Wozniak, A. Zolnierek (Eds.), Computer Recognition Systems 2, Advances in Soft Computing, Vol. 45, Springer, 2007, pp. 763–770.

[19] E. Grabska, G. Ślusarczyk, T. Lan Le, Visual design and reasoning with the use of hypergraph transformations, Electronic Communications of the EASST 10.

[20] E. Grabska, A. Łachwa, G. Ślusarczyk, New visual languages supporting design of multi-storey buildings, Advanced Engineering Informatics 26 (4) (2012) 681–690. doi:10.1016/j.aei.2012.03.009.

[21] J. Szuba, New object-oriented PROGRES for specifying the conceptual design tool GraCAD, Electron. Notes Theor. Comput. Sci. 127 (1) (2005) 141–156. doi:10.1016/j.entcs.2004.12.032.

[22] J. Lee, Building Environment Rule and Analysis (BERA) language and its application for evaluating building circulation and spatial program, Ph.D. thesis, Georgia Institute of Technology (2011).

[23] J. Lee, C. Eastman, J. Lee, M. Kannala, Y. Jeong, Computing walking distances within buildings using the universal circulation network, Environment and Planning B 37 (4) (2010) 628–645.

[24] Y. Adachi, Overview of partial model query language, Tech. rep., SECOM Co. (2002).

[25] S.-J. You, D. Yang, C. Eastman, Relational DB implementation of STEP based product model, in: CIB T6S7 Information technology in construction, CIB International Congress, Toronto, 2004.

[26] Eurostep, Product model query language, [Accessed 28 January 2014] (2013).
URL http://www.eurostep.com/products/share-a-space-features.aspx

[27] W. Mazairac, J. Beetz, BIMQL an open query language for building information models, Advanced Engineering Informatics 27 (4) (2013) 444–456. doi:10.1016/j.aei.2013.06.001.

[28] A. Borrmann, E. Rank, Topological analysis of 3d building models using a spatial query language, Adv. Eng. Inform. 23 (4) (2009) 370–385. doi:10.1016/j.aei.2009.06.001.

[29] A. Borrmann, E. Rank, Specification and implementation of directional operators in a 3d spatial query language for building information models, Advanced Engineering Informatics 23 (1) (2009) 32–44. doi:10.1016/j.aei.2008.06.005.

[30] A. Borrmann, S. Schraufstetter, E. Rank, Implementing metric operators of a spatial query language for 3d building models: Octree and B-Rep approaches, Journal of Computing in Civil Engineering 23 (1) (2009) 34–46. doi:10.1061/(ASCE)0887-3801(2009)23:1(34).

[31] M. Egenhofer, R. Franzosa, Point-set topological spatial relations, International Journal of Geographical Information Systems 5 (2) (1991) 161–174.

[32] M. P. Nepal, S. Staub-French, R. Pottinger, A. Webster, Querying a building information model for construction-specific spatial information, Advanced Engineering Informatics 26 (4) (2012) 904–923. doi:10.1016/j.aei.2012.08.003.

[33] P. Katranuschkov, A. Gehre, R. Scherer, An ontology framework to access IFC model data, ITcon 8 (2003) 413–437.

[34] M. Bhatt, J. Hois, O. Kutz, Ontological modelling of form and function for architectural design., Applied Ontology 7 (3) (2012) 233–267.

[35] D. A. Randell, Z. Cui, A. G. Cohn, A spatial logic based on regions and connection, in: Proceedings 3rd International Conference on Knowledge Representation and Reasoning, 1992.

[36] P. Pauwels, D. V. Deursen, R. Verstraeten, J. D. Roo, R. D. Meyer, R. V. de Walle, J. V. Campenhout, A semantic rule checking environment for building performance checking, Automation in Construction 20 (5) (2011) 506–518. doi:10.1016/j.autcon.2010.11.017.

[37] P. Steadman, The automatic generation of minimum-standard house plans, Tech. Rep. Working Paper 23, University of Cambridge, Land Use and Built Form Studies (1970).

[38] J. Grason, An approach to computerized space planning using graph theory, in: Proceedings of the 8th Design Automation Workshop, DAC '71, ACM, New York, NY, USA, 1971, pp. 170–178.

[39] R. S. Liggett, Automated facilities layout: past, present and future, Automation in Construction 9 (2) (2000) 197–215. doi:10.1016/S0926-5805(99)00005-9.

[40] H. E. Feustel, COMIS - an international multizone air-flow and contaminant transport model, Energy and Buildings 30 (1) (1999) 3–18.

[41] S. Taneja, B. Akinci, J. Garrett, L. Soibelman, B. East, Transforming IFC-based building layout information into a geometric topology network for indoor navigation assistance, in: Computing in Civil Engineering (2011), 2011, pp. 315–322. doi:10.1061/41182(416)39.

[42] 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, Advanced Engineering Informatics 27 (2) (2013) 189–205. doi:10.1016/j.aei.2012.10.001.

[43] A. Khalili, D. Chua, An IFC-based graph data model (GDM) for topological queries on building elements, Journal of Computing in Civil Engineering 27 (3). doi:10.1061/(ASCE)CP.1943-5487.0000331.

[44] C. Langenhan, M. Weber, M. Liwicki, F. Petzold, A. Dengel, Graph-based retrieval of building information models for supporting the early design stages, Advanced Engineering Informatics 27 (4) (2013) 413–426. doi:10.1016/j.aei.2013.04.005.

[45] B.-C. Björk, A conceptual model of spaces, space boundaries and enclosing structures, Automation in Construction 1 (1) (1992) 193–214.

[46] A. Ekholm, S. Fridqvist, A concept of space for building classification, product modelling and design, Automation in Construction 9 (3) (2000) 315–328.

[47] Object Management Group, Unified Modeling Language, [Accessed 28 January 2014] (2011).
URL http://www.omg.org

[48] M. Mortenson, Geometric modeling, John Wiley & Sons, 1997.

[49] International Organization for Standardization, Geometric and topological representation (ISO 10303-42), [Accessed 28 January 2014] (2003).
URL http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=60762

[50] Spatial Technologies, Inc., 3d ACIS modeler, [Accessed 28 January 2014] (2013).
URL http://doc.spatial.com

[51] F. Aurenhammer, R. Klein, Voronoi diagrams, in: J. Sack, G. Urrutia (Eds.), Handbook of Computational Geometry, Chapter V, Elsevier Science Publishing, 2000, pp. 201–290.

[52] B. Hillier, J. Hanson, The Social Logic of Space, Cambridge University Press, 1989.

[53] P. Steadman, Architectural morphology, Pion, 1983.

[54] S. Cockcroft, A taxonomy of spatial data integrity constraints, Geoinformatica 1 (4) (1997) 327–343. doi:10.1023/A:1009754327059.

[55] A. Silberschatz, H. Korth, S. Sudarshan, Database system concepts, McGraw-Hill, 2006.

[56] Spatial Technologies, Inc., 3d ACIS modeler: Clash classifications, [Accessed 28 January 2014] (2013).
URL http://doc.spatial.com/index.php/Clash_Classifications

[57] D. White, A new method of polygon overlay, in: An Advanced Study Symposium on Topological Data Structures for Geographic Information Systems, Laboratory for Computer Graphics and Spatial Analysis, Harvard University, Cambridge, MA, USA, 1977.

[58] B. Stroustrup, The C++ programming language - special edition, 3rd Edition, Addison-Wesley, 2007.

[59] Spatial Technologies, Inc., 3d ACIS modeler: Basic operations - Booleans, [Accessed 28 January 2014] (2013).
URL http://doc.spatial.com/qref/ACIS/html/group__BOOLBASICAPI.html

[60] Boost, Boost Graph Library, [Accessed 28 January 2014] (2011).
URL www.boost.org

[61] CGAL Open Source Project, Computational Geometry Algorithms Library, [Accessed 28 January 2014] (2009).
URL http://www.cgal.org/

[62] E. Papadopoulou, D. T. Lee, A new approach for the geodesic Voronoi diagram of points in a simple polygon and other restricted polygonal domains, Algorithmica 20 (4) (1998) 319–352.

[63] Autodesk, Inc., AutoCAD Architecture 2012 user's guide, [Accessed 28 January 2014] (2011).
URL http://www.autodesk.com

[64] G. Suter, Modeling spatial compositions with network-based spaced layouts, in: R. Issa, I. Flood (Eds.), International Conference for Computing in Civil and Building Engineering 2014, ISCCBE, Orlando, Florida, USA, 2014.