Ontological Approach to Design Reasoning with the Use of Many-Sorted First-Order Logic

Wojciech Palacz(✉), Ewa Grabska, and Grażyna Ślusarczyk

The Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University, ul. Łojasiewicza 11, 30-348 Kraków, Poland
{wojciech.palacz,ewa.grabska,grazyna.slusarczyk}@uj.edu.pl

Abstract. This paper is a continuation and extension in developing the knowledge-based decision support design system (called HSSDR) which communicates with the designer via drawings. Graph-based modeling of conceptualization in the CAD process, which enables the system to automatically transform design drawings into appropriate graph-based data structures, is considered. Hierarchical graphs with bonds are proposed as a representation of designs. An ontological commitment between design conceptualization and internal representations of solutions, which enables us to capture intended design models, is described. Moreover, the first-order logic (FOL) of HSSDR is replaced by many-sorted FOL that makes it possible to define different sorts in specification of functions and predicates in semantics and design constraint verification.

1 Introduction

This paper is focused on maintaining ontological compatibility between drawings and their graph-based representations in a CAD process. Design drawings are created on the basis of the conceptualization characteristic for a given design domain. Conceptualization concerns objects, concepts, and other entities that are assumed to exist in the considered domain of discourse, and the relationships that hold among them.

The drawings by which a CAD system communicates with the designer are usually converted to some internal data structures. In many CAD programs those structures are graph-based, because graphs are well-suited to model subcomponents of the designed object and relations between those subcomponents. An example of such a system is HSSDR (Hypergraph System Supporting Design and Reasoning) described in [3]. It was developed as a tool for designing floor layouts and validating their correctness in respect to an adjustable set of rules.

Every action undertaken by the designer modifies a displayed drawing and this modification is in turn reflected in the HSSDR's internal data structure in the form of a hierarchical graph. Every modification done by the designer triggers a constraint verification round. Design constraints are specified as logic formulas. Two constraint sets, which verify compliance with fire code regulations and check if all paths leading to certain important rooms are monitored by cameras, were considered in [3].

L. Rutkowski et al. (Eds.): ICAISC 2016, Part II, LNAI 9693, pp. 364–374, 2016.
DOI: 10.1007/978-3-319-39384-1_31

Experiences gained during implementation and testing of the original version of HSSDR have suggested two potential improvements. The first one is concerned with the way of modeling parts of the layout. Areas, rooms and walls were treated as elements of the domain of discourse, while other objects, like doors and cameras, as properties (attributes) of these elements. This approach caused the lack of ontological compatibility between conceptualization and the hypergraph representation of drawings. This paper proposes a new representation in the form of hierarchical graphs with bonds, which makes it possible to include all of the objects as elements of the domain of discourse.

Another proposed improvement concerns the language in which the design constraints are specified. The FOL is replaced by many-sorted FOL. It allows for enhanced syntactic checking of the constraints.

This paper is organized as follows: Sect. 2 gives some insights in human-computer interaction in the context of computer-aided design. Section 3 presents the current modeling system, while Sect. 4 then proposes its modifications. In Sect. 5 a new type of hierarchical graphs, so called the graphs with bonds, is defined. Sections 6 and 7 consider the reasoning mechanism of HSSDR based on many-sorted FOL language, while reasoning examples are given in Sect. 8. The paper ends with a conclusion.

2 Human-Computer Interaction in CAD

During the conceptual design the designer has to describe the domain of discourse being a subset of his/her cognitive domain. There exists the need to keep in mind objects, concepts, and other entities that are assumed to exist in the considered domain of discourse, and the relationships that hold among them. Conceptualization is one of the most challenging aspects of designing because it forces designers to considers many disparate factors. According to [2] a conceptualization can be defined as follows.

Definition 1. *A conceptualization is a pair* $\mathcal{C} = (\Delta, \Theta)$*, where:* Δ *is a set called the universe of discourse, and* Θ *is a set of relations on* Δ*.*

During the computer aided design process externalization of designer's conceptualization takes the form of design drawings. They are generated by the designer with the use of a visual editor and constitute the first type of representation storing information about design solutions. Design drawings are composed of transformed basic shapes which are specified on the basis of the conceptualization. The design drawing can be formally defined as follows.

Definition 2. *Let* F *be a set of admissible transformations of the form* $f : R^n \to R^n$*. Let* S *be a finite set of basic shapes being bounded subsets of* R^n *such that for each two* $s, s' \in S$ *and* $f \in F$*,* $f(s) = s' \Rightarrow s = s'$*. Let* $Q \subset S \times F$ *be a space configuration.*

A design drawing with configuration Q *is specified as* $Z(Q) = \cup_{(s,f)\in Q} f(s)$*.*

In HSSDR the designer can generate an *observable world state* in the form of a design drawing (Fig. 1). In each step of the design process the designer can change his conceptualization, i.e., number of elements of the domain of discourse Δ and/or relations of Θ on Δ because both the requirements and the design become more refined as the project proceeds. The conceptualization for the designer's world can be represented by the sequence of observable world states and denoted by W. The conceptualization which includes all entities and relations defined for W is denoted by $\mathcal{C}_W$.

The designer communicates with the design system using a visual design language, i.e., a set of design drawings. Let us consider the specialized CAD editor for designing floor layout composed of polygons which are placed on an orthogonal grid. These polygons represent functional areas or rooms. According to designer's convention each line shared by polygons in the drawing is associated with one of two relations. Lines with a door symbol on them represent the accessibility relation among components, while continuous lines shared by polygons denote the adjacency relation between them. An example of a floor layout generated by the designer with the use of the editor is shown in Fig. 1.

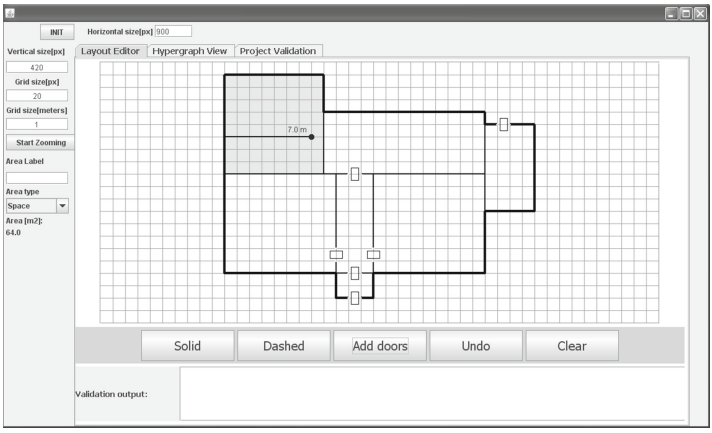

Fig. 1. The design drawing representing a floor-layout created by the designer in HSSDR

A majority of visual languages is characterized by a vocabulary being a finite set of basic shapes and a finite set of rules specifying possible configurations of these shapes. Basic shapes of visual languages and their spatial relationships correspond to concepts and relations defined by the conceptualization of the discourse domain. Each specialized design domain has its own visual language related to concepts of this domain [4,5].

3 The Current Modeling System

The Hypergraph System Supporting Design and Reasoning described in [3] is a system for designing floor layouts using the top-down methodology. HSSDR communicates with its users by means of a simple visual language of diagrams. It also checks if the current state of the layout satisfies design constraints specified as a set of FOL formulas.

A user starts by drawing an outline of an apartment. This outline is then divided by drawing a polyline. The resulting areas are further divided into subareas. The user continues dividing them, stopping when she reaches the level of rooms. Doors and cameras can be added either during, or after this process.

HSSDR internally uses a hierarchical hypergraph as its main data structure (see Fig. 2). Every user action modifying the displayed diagram is reflected in this hypergraph.

The hypergraph consists of two types of hyperedges: those representing rooms and areas, and those representing adjacency and accessibility relations. Functional areas and groups of rooms are at higher hierarchy levels, while single rooms are at the bottom. Graph nodes correspond to walls. Attributes are used to store rooms positions, as well as wall lengths, number and positions of doors and cameras, room labels, etc.

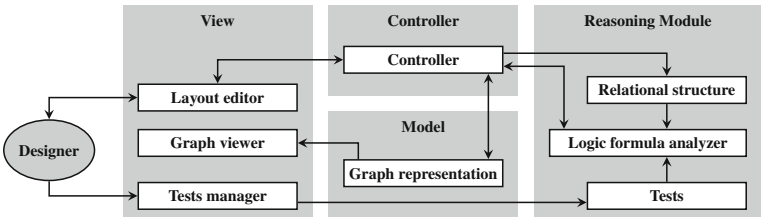

Fig. 2. The internal architecture of HSSDR

The design constraints are stored in external text files, thus they can be easily modified or replaced at any time, even in the middle of a design session. Constraints can refer to elements of the layout (rooms, walls, doors and cameras) and to a fixed set of relations and functions (adjacency between two rooms, room type, distance between two doors, etc.). Thanks to the use of widely known FOL formalisms they are readable by developers, experts and users.

The constraints are logic sentences, and as such can be evaluated as true or false in context of a specific relational structure [1]. HSSDR defines a structure which reflects the current state of the hypergraph. Results of evaluation are presented to the user; all false results are flagged as constraint violations.

An example of such results can be seen in Fig. 3. The text field near the bottom shows the name of the file and the message associated with the failed test. The layout element which failed this test is marked red on the diagram.

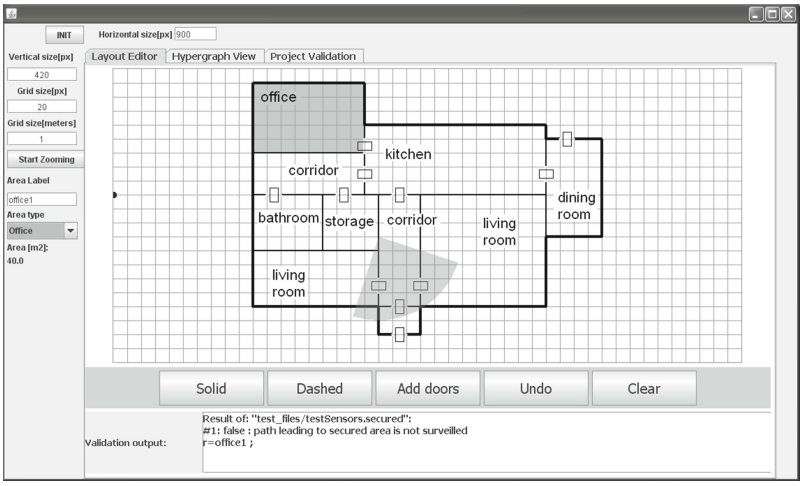

Fig. 3. A finished layout which does not fulfill one of design constraints

4 The New System

Experiences gained during implementation and testing of the original version of HSSDR have pointed to several areas of potential improvement. One of them was the type of graph used as the main data structure and the way elements of discourse were modeled in this graph. The original way of representing an area or a room by a special kind of hyperedge together with its attached nodes was convenient for representing accessibility and adjacency relations between areas/rooms, but became problematic when the representation was extended to include doors and cameras. From an ontological point of view areas and rooms are treated as elements of the domain of discourse, while other objects as doors and cameras as properties (attributes) of these elements. In other words, there does not exist ontological commitment between conceptualization and its representation in the form attributed hypergraph. Therefore, this paper proposes a new model which uses hierarchical graphs with bonds. This type of graphs makes it possible to include all of the object as elements of the domain of discourse. Bonds are distinguished graph nodes which can specify arguments of relations on different levels of detail.

Another proposed improvement concerns the language, in which the design constraints are specified. The original HSSDR used FOL; it can be replaced by many-sorted FOL. Introduction of sorts divides the domain of discourse into subsets: of rooms, of doors, of numbers, of room types, etc. Predicates and functions used in formulas are annotated with information about sorts of their arguments and results. This allows for enhanced syntactic checking of formulas.

The new system keeps the name HSSDR, but the first letter now stands for a hierarchical graph, not for a hypergraph. So, it can refer to any kind of a hierarchical graph, from graphs described in [7], through hypergraphs used by the

previous version of HSSDR and B-graphs proposed for the new version, to other kinds which may be used in the future. For example, multi-hierarchical graphs seem promising as models for multi-storey buildings and other cases where single hierarchy based on spatial containment is insufficient.

5 Hierarchical B-Graphs

B-graphs are meant to represent objects and their fragments or elements (known as bonds) that can be used as arguments of relations. Relations are defined between bonds. We distinguish two kinds of bonds: engaged and free, which correspond to arguments of existing and potential relations, respectively. This distinction of bonds is essential for defining operations on graphs that reflect modifications of design drawings.

B-graphs used in this paper are hierarchical, because of the need to represent nested areas. Hierarchy also allows for sub-bonds (for example, a room has walls as its fragments and one of them has a door – this door is a fragment of the room, but also is subordinate to the wall). Instead of one set of nodes, B-graphs have two: a set of object nodes and a set of bond nodes. They also have a set of edges. Every edge connects bonds belonging to two different objects. HSSDR needs to store additional information about layout elements, therefore B-graphs allow for labels and attributes on nodes and edges.

Let Σ be a finite alphabet used to label object nodes, bond nodes and edges. Let A be a nonempty, finite set of attributes. For every attribute $a \in A$, let D_a be a fixed, nonempty set of its admissible values, known as the domain of a.

Definition 3. *A B-graph G is a tuple $G = (O, B, E, bd, s, t, ch, lab, atr)$, where:*

- *O, B, E are pairwise disjoint finite sets, whose elements are respectively called object nodes, bond nodes, and edges,*
- *$bd : O \to 2^B$ is a function assigning sets of bond nodes to object nodes in such a way that $\forall x \in B\ \exists! y \in O : x \in bd(y)$, i.e., each bond belongs to exactly one object,*
- *$s, t : E \to B$ are functions assigning to edges source and target bond nodes, respectively, in such a way that $\forall e \in E\ \exists x, y \in O : s(e) \in bd(x) \wedge t(e) \in bd(y) \wedge x \neq y$,*
- *$ch : O \cup B \cup E \to 2^{O \cup B \cup E}$ is a child nesting function such that:*
  - *$\forall x, y, z \in O \cup B \cup E : x \in ch(y) \wedge x \in ch(z) \Rightarrow y = z$, i.e., a graph element cannot be nested in two different places,*
  - *$\forall x \in O \cup B \cup E : x \notin ch^+(x)$, i.e., a graph element cannot be its own descendant,*
  - *$\forall x \in B$, $\forall y \in O$: $x \in bd(y) \Rightarrow x \in ch(y) \vee (\exists z \in B : x \in ch(z) \wedge z \in bd(y))$, i.e., a bond must be nested either in its object, or in some other bond belonging to this object,*
- *$lab : O \cup B \cup E \to \Sigma$ is a labelling function,*
- *$atr : O \cup B \cup E \to 2^A$ is an attributing function.*

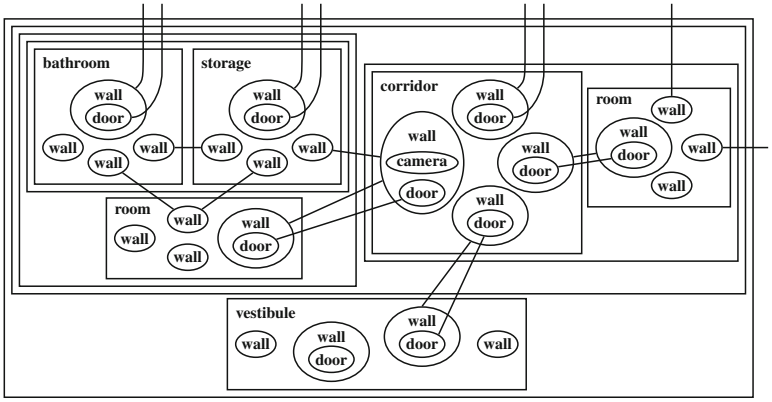

Fig. 4. A part of a graph model corresponding to the layout displayed in Fig. 3

A part of the B-graph being the internal representation of the layout shown in Fig. 3 is presented in Fig. 4. Bonds without the outgoing edges in Fig. 4 represent free bonds.

A graph instance is obtained by adding attribute values. Each design drawing is represented by a corresponding B-graph instance defined as follows.

Definition 4. *An instance I of a B-graph G is a pair $I = (G, val)$ where $G = (O, B, E, bd, s, t, ch, lab, atr)$ and $val : O \cup B \cup E \times A \to D$, with $D = \bigcup_{a \in A} D_a$, is a partial function assigning attribute values in such a way that $val(x, a)$ is defined if and only if $a \in atr(x)$ and $val(x, a) \in D_a$ if defined.*

The proposed B-graphs allow for maintaining ontological compatibility between drawings and their graph-based representations.

Definition 5. *Let W denote the designer's world and $\mathcal{C}_W = (\Delta_W, \Theta_W)$ be a conceptualization for W. Let $G_W = (O, B, E, bd, s, t, ch, lab, atr)$ be a family of B-graphs representing world states of W.*

An ontological commitment between $\mathcal{C}_W$ and G_W is a mapping $\Im_W : \Delta_W \cup \Theta_W \to O \cup B \cup E$, such that for each C_w, where $w \in W$, there exists B-graph $g_w \in G_W$ such that concepts of Δ_W are mapped to object and bond nodes of g_w and relations of Θ_W to edges of g_w.

6 Many-Sorted First-Order Logic

The explicit specification of design constraints related to the reasoning mechanism based on representation of drawings in the form of B-graph instances must be formal, i.e., the expressions must be defined in a formal language. We use a many-sorted FOL language which introduces representing distinct kinds of objects to reason about [6]. The concept of sort allows one to specify for functions and relations what sorts of their domains and ranges are required.

The formal description of many-sorted FOL formulas starts with the following definition of a signature.

Definition 6. *A signature σ is a tuple $\sigma = (S, X, C, F, P)$, where:*

- $S = \{s_1, s_2, \ldots, s_k\}$ *is a set of sorts,*
- X *is a set of variables, which is partitioned into subsets $X_1 \ldots X_k$ with each X_i containing variables of sort s_i,*
- C *is a set of constant symbols, which is partitioned into subsets $C_1 \ldots C_k$ with each C_i containing constants of sort s_i,*
- F *is a set of n-ary function symbols the type $s_{i_1} \times \cdots \times s_{i_n} \rightarrow s_{i_{n+1}}$, where $n \geq 0$, and each $s_{i_j} \in S$,*
- P *is a set of n-ary predicate symbols of the type $s_{i_1} \times \cdots \times s_{i_n}$, where $n \geq 0$, and each $s_{i_j} \in S$.*

Definitions of σ-terms and σ-formulas are analogous to the definitions of these concepts for the classic FOL language. A formula where all variables are bound by quantifiers is known as a sentence. Sentences are either true or false, but only after all symbols used in formulas are assigned a specific semantics. This semantic information is provided by a relational structure defined as follows.

Definition 7. *For a signature σ, a σ-relational structure R is a map with the following properties:*

- *each sort s_i is mapped to a nonempty set A_i,*
- *each constant symbol $c \in C_i$ is mapped to an element of A_i,*
- *each function symbol $f : s_{i_1} \times \cdots \times s_{i_n} \rightarrow s_{i_{n+1}}$ is mapped to a function $f^R : A_{i_1} \times \cdots \times A_{i_n} \rightarrow A_{i_{n+1}}$,*
- *each predicate symbol $p : s_{i_1} \times \cdots \times s_{i_n}$ is mapped to a subset $p^R \subset A_{i_1} \times \cdots \times A_{i_n}$.*

7 Reasoning Module of HSSDR

The HSSDR's reasoning module interprets symbols of the logic language signature on the basis of B-graphs representing design drawings. This interpretation should be compatible with the ontological commitment between conceptualization and graph structures (Definition 5). On the basis of this assumption the vocabulary of the reasoning module is defined.

In HSSDR the following sorts are defined: $areas, walls, doors, sensors, labels$ and $numbers$. There are constants representing labels ("hall", "bathroom", etc.) and numbers. There are functions: $type : areas \rightarrow labels$, $width$, $length : areas \rightarrow numbers$, $doorsDist : doors \times doors \rightarrow numbers$ (returns a distance between two pairs of doors in a single room), and standard arithmetic operators, either $numbers \times numbers \rightarrow numbers$ or $numbers \rightarrow numbers$.

Two-argument predicates: $adjacent, accessible : areas \times areas$, $doors\,InRoom : doors \times areas$ (returns true if the doors are embedded in a wall of the room), $sensorInRoom : sensors \times areas$, $isPassageWatched : doors \times doors$

(returns true if every path between two pairs of doors in a single room is watched by one or more sensors), and standard arithmetic comparison operators: $numbers \times numbers$.

One-argument predicates (i.e., sort subsets): $Areas : areas$, $Rooms : areas$, $Walls : walls$, $Doors : doors$, $ExternalDoors : doors$, and $Sensors : sensors$.

Only Rooms and ExternalDoors define proper subsets of their sorts. Other predicates always return true, and thus define sets equal to their whole sorts. From the formal point of view these predicates are not necessary, and are provided only because constraint writers sometimes prefer to write constraints with explicitly specified quantifier ranges, e.g., *forall a in Rooms: type(a) = "bank vault" => exists s in Sensors: sensorInRoom(s, a).*

For a given B-graph I representing a layout, the corresponding relational structure R is constructed as follows:

- $A_{areas} = O_I$, $A_{walls} = \{x \in B_I : lab_I(x) =$ "wall" $\}$, $A_{doors} = \{x \in B_I : lab_I(x) =$ "door"$\}$, $A_{sensors} = \{x \in B_I : lab_I(x) =$"sensor"$\}$, $A_{labels} = \Sigma$, $A_{numbers} = \mathbb{R}$,
- label and number constants are mapped to themselves,
- $type^R(x) = lab_I(x)$, $width^R(x) = val_I(x,$ "width"), $length^R(x) = val_I(x,$ "length"), $doorsDist^R(x, y)$ is implemented algorithmically,
- arithmetic operators are defined as themselves,
- $adjacent^R = \{(x, y) \subset O_I \times O_I$: exists $e \in E_I$ such that $s_I(e) \in bd_I(x), t_I(e) \in bd_I(y), lab_I(e) =$ "adjacent"$\}$,
- $acessible^R = \{(x, y) \subset O_I \times O_I$: exists $e \in E_I$ such that $s_I(e) \in bd_I(x), t_I(e) \in bd_I(y), lab_I(e) =$ "acessible"$\}$,
- $doorsInRoom^R = \{(x, y) \subset B_I \times O_I : lab_I(x) =$ "door", $x \in bd_I(y)\}$,
- $sensorInRoom^R = \{(x, y) \subset B_I \times O_I : lab_I(x) =$ "sensor", $x \in bd_I(y)\}$,
- $isPassageWatched^R \subset B_I \times B_I$ is calculated by Java code,
- arithmetic comparisons are defined as themselves,
- $Areas^R = A_{areas}$, $Rooms^R = \{x \in O_I : ch_I(x) \cap O_I = \emptyset\}$, $Walls^R = A_{walls}$, $Doors^R = A_{doors}$, $ExternalDoors^R = \{x \in A_{doors}$: there is no $e \in E_I$ such that $s_I(e) = x\}$, $Sensors^R = A_{sensors}$.

8 Reasoning Examples

This section provides several examples of tests, written in a form acceptable to HSSDR.

The following constraint requires at least one bedroom present in the layout: *exists x: type(x) ="Bedroom"*. The HSSDR's reasoning module, when loading this formula, will recognize x as a variable of an unspecified sort and "Bedroom" as a constant of the labels sort. Since x is used as an argument of the *type* function, it apparently must be of the areas sort. This function produces results of the labels sort, which means that both sides of the equality sign are of the same sort. The formula is syntactically correct. The reasoning module uses exhaustive search when evaluating formulas with quantifiers. In this case, since x is of the

areas sort, it will loop over all areas (and rooms) currently present in the layout – that is, over elements of the $Areas^R$ set.

The constraint of the form *exists r: type(r) > 42* is syntactically invalid, because results of the *type* function belong to the labels sort, and cannot be compared with constants of the numbers sort.

The last example checks if all path leading to secured areas are watched by sensors. Originally presented in [8], here it was adapted to many-sorted logic. Results produced by this test can be seen on screenshots in Figs. 3 and 5. At first, the test fails because a thief can enter through the doors in the dining room and walk unobserved through kitchen to the office. Adding a second sensor in the kitchen secures this path. In the similar way the satisfaction of standard architectural norms can be checked.

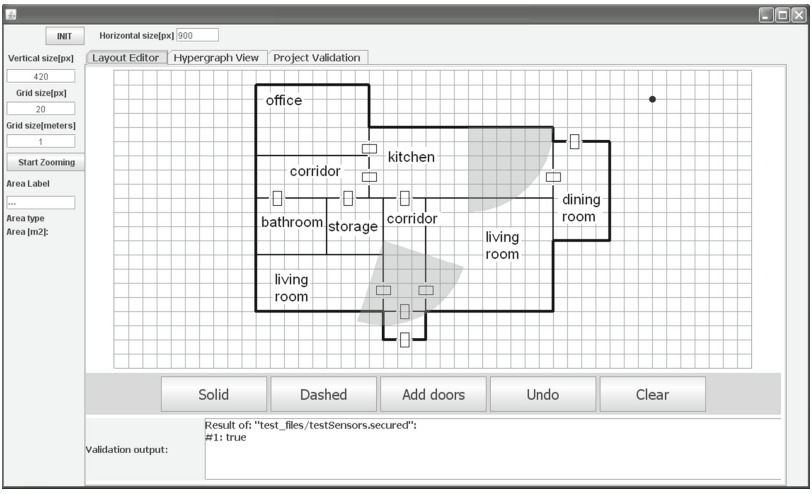

Fig. 5. Corrected version of the layout from Fig. 3.

9 Conclusions

This paper presents how graphs can be combined with logic-based knowledge representation techniques, where knowledge is represented explicitly by symbolic terms and reasoning is the manipulation of these terms. In the proposed approach the semantics of logical formulas build over many-sorted signature uses relational structures based on B-graph instances.

In our future research we shall focus attention on modelling multi-storey buildings with the use of multi-hierarchical graph representations with bonds.

References

1. Fagin, R., Halpern, J.Y., Moses, Y., Vardi, M.Y.: Reasoning About Knowledge. The MIT Press, Cambridge (2003)
2. Genesereth, M.R., Nilsson, N.J.: Logical Foundations of Artifical Intelligence. Morgan Kaufmann, Los Altos (1987)
3. Grabska, E., Borkowski, A., Palacz, W., Gajek, Sz.: Hypergraph system supporting design and reasoning. In: Huhnt, W. (ed.) Computing in Engineering EG-ICE Conference 2009, pp. 134–141. Shaker Verlag (2009)
4. Grabska, E., Łachwa, A., Ślusarczyk, G.: New visual languages supporting design of multi-storey buildings. Adv. Eng. Inform. **26**, 681–690 (2012)
5. Kraft, B., Nagl, M.: Visual knowledge specification for conceptual design: definition and tool support. Adv. Eng. Inform. **21**, 67–83 (2007)
6. Manzano, M.: Introduction to many-sorted logic. In: Meinke, K., Tucker, J.V. (eds.) Many-Sorted Logic and Its Applications, pp. 3–86. Wiley, New York (1993)
7. Palacz, W.: Algebraic hierarchical graph transformation. JCSS **68**, 497–520 (2004)
8. Palacz, W., Grabska, G., Gajek, Sz.: Conceptual designing supported by automated checking of design requirements and constraints. In: Frey, D.D., et al. (eds.) Improving Complex Systems Today, pp. 257–265. Springer-Verlag, London (2011)