Synchronisation methods in graph-based knowledge representation for large-scale design process

Leszek Kotulski and Adam Sędziwy
Department of Applied Computer Science,
AGH University of Science and Technology,
A1. Mickiewicza 30, Krakow, Poland
Email: kotulski@agh.edu.pl
Email: sedziwy@agh.edu.pl

Barbara Strug*
Department of Applied Computer Science,
AGH University of Science and Technology,
A1. Mickiewicza 30, Krakow, Poland
and
Department of Physics, Astronomy and
Applied Computer Science,
Jagiellonian University,
ul. Lojasiewicza 11, Krakow, Poland
Email: bstrug@agh.edu.pl
*Corresponding author

Igor Wojnicki and Sebastian Ernst
Department of Applied Computer Science,
AGH University of Science and Technology,
A1. Mickiewicza 30, Krakow, Poland
Email: wojnicki@agh.edu.pl
Email: ernst@agh.edu.pl

Abstract: Automation is an important issue in computer-aided design systems. One of the most important issues in achieving such an objective is an appropriate knowledge representation. Graphs and graph-based formalisms constitute a mean which allows for an effective representation of designs. On the basis of graphs representing the structure (static) of the design, graph grammars are a tool allowing to represent the dynamics of the design process. In this paper, we present an approach to the synchronisation of different aspects within the design process carried out by different and independent transformation systems supporting different aspects of the design. Such a synchronisation is required every time one of the systems tries to modify an area shared with the other one. The proposed mechanism is illustrated by an example of successful synchronisation in shared areas or on shared elements of the object being designed. The objective has been achieved by using different representations at different layers of the design.
Keywords: design process representation; design synchronisation; graph-based knowledge representation; CAD.


Biographical notes: Leszek Kotulski received his PhD in Computer Science at the AGH University of Science and Technology in Krakow and DSc from the Wroclaw Technical University in 1984 and 2001, respectively. He is a Professor at the Faculty of Electrical Engineering, Automatics, Computer Science and Biomedical Engineering of the AGH University of Science and Technology and is the Head of the Department of Applied Computer Science of AGH. His research interests include issues related to distributed systems, graph transformations, agents systems and software development methodology. He is an author of over 110 publications he also published two books. He is a member of ACM and IEEE.

Adam Sędziwy is an Assistant Professor in the Department of Applied Computer Science at the AGH University of Science and Technology, Cracow, Poland. He holds a Master’s in Theoretical Physics and Computer Science at the Jagiellonian University, Cracow, Poland and PhD and DSc in Computer Science from AGH. He is the author and co-author of works on syntactic pattern recognition, graph-based distributed processing, multi-agent systems, application of formal methods in the area of design and control of smart lighting systems and smart grids.

Barbara Strug received her PhD in Computer Science from the Institute of Fundamental Technological Research of the Polish Academy of Sciences (IPPT PAN) in Warsaw in 2002 and DSc from the Department of Applied Computer Science at the AGH University of Science and Technology, Cracow, Poland in 2014. She is an Associate Professor at the Faculty of Physics, Astronomy and Applied Computer Science, Jagiellonian University and Research Associate Professor at the Department of Applied Computer Science of AGH. Her research interests include computer aided design, knowledge representation, graph-based representation and graph transformations, graph mining and graph kernels. She is an author of over 60 papers and attended numerous conferences. She is a member of ACM.

Igor Wojnicki received his PhD in Computer Science from the AGH University of Science and Technology in 2004. Currently, he holds the position of an Assistant Professor at the Department of Applied Computer Science, AGH University of Science and Technology, Cracow, Poland. His research interest focuses on artificial intelligence, intelligent systems, knowledge representation, inference methods and databases with special attention given to rule-based solutions. He is an author of over 40 papers published in international journals and presented at numerous conferences. He is also an AAAI member.

Sebastian Ernst holds the position of an Assistant Professor at the Department of Applied Computer Science, AGH University of Science and Technology, Cracow, Poland. He received his PhD in Computer Science from the AGH University of Science and Technology in 2010. His research interests include artificial intelligence, automated planning, graph-based methods and simulation of dynamic systems. He has been involved in numerous international research projects and is the author of many papers published in international journals and presented at conferences.
1 Introduction

Design is a complex process which usually involves taking into account different aspects of an artefact to be designed. These aspects often have to deal with different levels of details of the design or different layers. Thus, the design process has to be considered as the sum of operations performed on all of the design layers. In a large-scale design process, there may be a large number of such layers and a single design task can touch upon many of them. In some cases, a design task may be done independently from other ones and only modify/touch some parts or layers of the design. Yet, in many situations, a single design task would result in requiring changes at different layers of the design, what may result in invoking other design actions. In such cases, some way of synchronisation of design actions on different layers will be essential for successful design.

Designing a building is such an example of a complex design task, in which a lot of different aspects have to be taken into account. The architectural design of buildings can be seen from different point of view: by the external design, as the structural design, as the design of the internal structure and layout, as the design of the utilities and appliances or as the interior design and furnishing.

Each of these aspects can have its own separate objectives, constraints, requirements and so on. In this paper, we focus mainly on differences between the external and internal aspects of building design. As the building is seen differently from the point of view of the external and internal design, these differences can also be reflected in the way such an object (a building) is represented, what in turn can result in slightly different formal mechanisms used.

Each design task considered in this paper can also be seen as a composition of a number of sub-processes carried out in different layers of the design, for example in the internal and external part of a building. While in many cases, a task can be successfully performed on its own layer without interfering with other layers, it is obvious that sub-processes can meet in some points. So it is necessary to provide a mechanism which ensures coherence between them, especially in the areas of contact. The consistency between different aspects (layers) of design is essential for achieving architectural goals.

A first requirement in order to be able to automate the design process is having an appropriate knowledge representation of the designed object (building).

The representation must be flexible enough to store different types of data, as well as structural and spatial properties of an object. It has also to be able to support updating/modifying the knowledge stored at any time. In this paper, hypergraphs are used as the basic representation of a building structure. For such a representation design/modification/update tasks can be represented by means of formal graph transformations (productions), applied to the underlying representation of a design. These transformations can originate from independent graph grammars, which describe operations performed on the different layers of design, for example interior and exterior. Thus, it is inevitable that applying some production of one grammar impacts the elements
of the other structure and thus triggers transformations carried out by that other party. We introduce in the paper a two-phase synchronisation mechanism which guarantees consistent execution of coupled productions, belonging to different graph grammars.

The work presented here is motivated by the need for a mechanism coordinating different and independent graph grammars, dealing with different aspects of design, but having productions operating on contacting or shared areas of design.

The proposed approach is illustrated by a case study from the domain of architectural design, where it is used to ensure consistency of the design process performed simultaneously inside (including its internal structure) and outside (including its environment) of a building.

The paper is organised as follows. The next section presents a review of the related work in the domain of computer aided design and graph-based representation and transformation. In Section 3, a brief overview of formal models used in architectural knowledge representation is presented. It also includes formal definitions of hypergraphs and hypergraph grammars, used in further considerations as the underlying design knowledge representation. In Section 4, the mechanism of synchronisation for hypergraph grammars is discussed. A case study is given in Section 5, while Section 6 presents some conclusions and future works.

2 Related work

There are a number of different types of graphs that can be used. Many of them have been successfully used as a way of representing design knowledge at different stages. In Grabska (1994), a model of composition graphs and the realisation scheme for this graph have been proposed. Composition graphs build on simple graphs by adding bonds to nodes. The bonds designate possible connections between parts of the objects represented by nodes. The realisation scheme on the other hand provides a formal model of transforming graph representation into a visual instance of the design that can be presented to the designer. Another extension of simple graphs is based on adding hierarchy to graphs. In Grabska et al. (2004), the hierarchical model has been used in the optimisation of skeletal structures. Hierarchical graphs are defined as consisting of edges and hierarchical nodes, which can in turn contain other nodes and edges. This graph can be considered as an extension to nested graphs, which allow for ‘nesting’ of graphs in graph nodes but do not allow for edges to connect nodes at different levels of nesting. Hierarchical graphs used in Grabska et al. (2004) put very low constraints on how edges can connect nodes: only edges connecting nodes with their ancestors/descendants in hierarchy are forbidden. As designs often consist of many interconnected components such an approach is more suitable for design tasks. In some design problems, the relations between components may not be binary. In such a case another extension of simple graphs can be used: a hypergraph. In Kotulski and Strug (2013), a hypergraph-based model, which uses hyperedges to represent both relations and components of the object, has been proposed. This approach is also used in this paper to represent the internal structure of the building.

The graphs are capable to represent the structure of the object, but design is a dynamic process in which changes can be introduced, both during the design process and later, during the maintenance stages. Thus, such change to a model representing a design, made during the object lifecycle, has to be modelled in some way. There are a number of
methods to define such modifications. These methods include approach based on application of the theory of formal languages to computer aided design (Rozenberg, 1997; Ehrg et al., 1999a, 1999b), in particular, a graph-based representation jointly with graph grammars (Borkowski et al., 2003; Grabska, 1994; Grabska et al., 2006), as well as grammar systems (Dassow et al., 1997; Grabska and Strug, 2006). Graph grammars are rule-based formalisms which are composed of an axiom (an initial graph) and a set of rules (called productions) which describe the way in which a graph can be modified. They allow to generate a complex design representation by subsequent application of productions. Other methods allowing us to obtain graph representations of designs include evolutionary computations that were used in different domains of design. In Nikodem and Strug (2004), an evolutionary approach has been used in the optimisation of trusses. Due to a large size of many graphs in real-life tasks, an approach based on distributing graphs and hypergraphs and to using multi-agent systems, was developed in Kotulski (2008a, 2008b), Kotulski and Strug (2007, 2011) and Kotulski and Sędziwy (2010a, 2010b, 2012).

Other aspect related to the work presented in this paper deals with maintaining data for building purposes, which has also been an object of wide research and resulted in developing building information modelling (BIM) systems. This systems start to gain popularity in many countries (BuildingSmart, 2005). In many European countries, using BIM-based data exchange formats and description is mandatory for building permission/approvals. The data contained within hypergraph structures used in this paper can be accessed by different applications. Information stored in attributes of atoms can also be exported in different formats to be used by other applications which accept BIM formats, through the Industry Fundation Classes (IFC) standard (BuildingSMART, 2005). However, standard BIM formats provide only partial support for structural relationships and they do not support multi-argument relations among element, which are often very important and can be well represented in the hypergraph-based representations. Nevertheless, as the IFC is based on the XML type format it provides flexible tools to extend the formalism and to include other aspects of design.

As for the external information’s about building and its position in wider environment the CityGML semantic meta-model based on geography markup language (GML) is a highly expressive language which allows modelling urban spaces with granularity varying from a geographical region to a building interior (Gröger et al., 2008). One of its purposes is augmenting geometric-type data with the semantics of underlying objects. CityGML is also used as a data exchange format, enabling interoperability among different platforms, in particular BIM systems mentioned above. The paper deals with problems of synchronisation, which is discussed in depth in multiple resources (e.g., Salleh et al., 1995; Leopold, 2000; Tanenbaum, 2015).

3 Graph-based design knowledge representation and modification

One of the most important aspects of design is having an appropriate knowledge representation. Moreover, such a representation should allow not only for the denotation of the static state of the designed objects, but also should constitute formalism for the description of the dynamics of the design. One of such formalism is a graph, which can
be used to both represent a state of the design at a given moment and an underlying formal model for the definition of the modification process.

There are a number of different graph types, mentioned in the previous section, but in this paper, a hypergraph representation and transformation model is used as a basis for creating a formal layer for the system supporting computer aided design (Habel and Kreowski, 1987). Objects to be constructed are represented by hypergraphs, which are labelled and attributed. While simple graphs consist of nodes and edges connecting these nodes and thus each edge in such a graph connects exactly two nodes (thus, such a graph allows us to represent only binary relations between objects), a hypergraph is an extension of a simple graph, which allows for representation of multi-argument relations. A hypergraph consists of hyperedges and nodes. Hyperedges of a hypergraph are labelled, in case of design problems – by names of the corresponding relations and can be directed for asymmetrical relations. Hypergraph nodes represent components of the design object. To represent features of components and relations between them, attributing of hyperedges and nodes is used. Values assigned to attributes specify the properties of objects represented by particular nodes and relations represented by hyperedges, thus allowing us to generate an instance of a hypergraph. On the basis of the standard definition of a hypergraph, two representations, being the most suitable for describing the internal and the external structure of a building, are derived and presented below.

3.1 Formal definitions

To represent structural and geometrical knowledge about a building there is a need to represent spaces (like rooms, halls), their relationships (like access, adjacency), but also other elements being parts of a building construction (like walls, windows, doors) and of the internal arrangements (like furniture, appliances). The following notion of an attributed hypergraph supports such requirements.

Let be $\Sigma$ a fixed alphabet of hyperedge and node labels and $A$ be a set of hyperedge and node attributes.

**Definition 1:** An attributed hypergraph over $\Sigma$ and $A$ is a system $G = (V, E, lb, att, val)$ where:

1. $V$ is a finite set of nodes
2. $E \subseteq 2^V$ is a finite set of hyperedges.
3. $lb: E \cup V \rightarrow \Sigma$ is a hyperedge and node labelling function.
4. $att: E \cup V \rightarrow 2^A$ is a hyperedge and node attributing function.
5. $val: (E \cup V) \times A \rightarrow Da$ is a partial mapping specifying the assignment of values to nodes and hyperedges in such a way that:

$$\forall x \in (E \cup V), \forall a \in att(x); val(x, a) \in Da$$

where $x$ is a node or an edge and $a$ is its attribute.

A hypergraph in which all attributes have a value assigned (by the function $val$) is called an instance and can be interpreted by assigning geometrical components to nodes and relations to hyperedges. In Figure 1(b), an example of a hypergraph representing a floor
layout of a flat is depicted. The floor diagram of this layout is shown in Figure 1(a). As it can be seen, each node represents an element of the layout, nodes labelled by $B$ represent bedroom, $Bt$ – bathroom, $K$ – kitchen, $ER$ – eating room, $H$ – hall, $LR$ – living room, $W$ – wall, $Wd$ – window and $D$ – external door. Each section of the wall is represented by a single node labelled $W$. Hyperedges are drawn as rectangles with labels inside and line segments linking them to nodes belonging to a given hyperedge, the label \( \text{inL} \) represents the fact that all elements represented by nodes connected by it are positioned along straight line. To improve the readability of the figure, hyperedges which connect only two nodes are drawn as continuous line segments for those which represent adjacency and dashed-line segments for those which represent being attached to another element (for example, the hyperedge connecting a wall and a window in Figure 1(b).

**Figure 1** (a) an example of a floor layout diagram, (b) a hypergraph representing this layout

A hypergraph defined above forms a good general structure and can very well represent the internal structure of the building and its layout. We also propose a specialisation of this hypergraph which allows for better representation of building structural characteristics from the point of view of the external design. It allows us to treat a building as a combination of solids.

**Definition 2:** External hypergraph representation of a convex polyhedron $S$ (referred to as a *simple solid*) is a hypergraph $G_s(V, A \cup H, \text{lab, att})$ such that:

1. $V$ is a set of nodes representing faces of $S$.
2. $A$ is a set of hyperedges representing edges of $S$.
3. $H$ is a set of hyperedges representing vertices of $S$.
4. \( \text{lab}: V \cup A \cup H \rightarrow \Sigma \) is a node and hyperedge labelling function with a corresponding set of labels, $\Sigma$.
5. \( \text{att}: V \cup A \cup H \rightarrow \Gamma \) is a node and hyperedge attributing function with a corresponding set of attributes, $\Gamma$. 
A hypergraph model of a solid facilitates determining geometric relations among its faces, edges and vertices and thus is more efficient than the intuitive graph representation consisting of a graph being a polyhedral mesh of a solid.

The next step towards the increased expressive strength of the above formalisation is modelling any solids, including the concave ones, solids with non-plane faces and solids with curved edges. The second and third case may be easily handled by means of attributing functions providing geometric data of particular entities.

The first issue may be resolved when we assume that each concave solid (CS) S is a sum of simple solids, $S_1$, $S_2$, ..., $S_k$, such that there exists a permutation $i(1)$, $i(2)$, ..., $i(k)$ for which solids $S_{i(j)}$ and $S_{i(j+1)}$ are adherent, for $0 < j < k$. In other words, we demand connectivity of $S$. Figure 2 presents the sample set of adhering buildings, $S_1$, $S_2$, ..., $S_5$, which form the CS satisfying the above assumption.

**Figure 2** CS consisting of adhering buildings

A representation of a CS is obtained by adding edges (formally hyperedges) representing adherence of solids to a set of hyperedges of $G_S$.

**Definition 3:** Composite hypergraph representation of a connected solid $S$ is a hypergraph $G_S(V, A \cup H \cup R, lab, att)$ such that:

1. $V, A, H$ are as in Definition 2.
2. $R \subseteq P_2(V)$ is a set of hyperedges such that $e = \{v_1, v_2\} \in R \iff v_1, v_2$ represent adhering faces.
3. $lab: V \cup A \cup H \cup R \rightarrow \Sigma$ is a node and hyperedge labelling function with a corresponding set of labels, $\Sigma$.
4. $att: V \cup A \cup H \cup R \rightarrow \Gamma$ is a node and hyperedge attributing function with a corresponding set of attributes, $\Gamma$. 
3.2 Design modification

Hypergraphs can represent subsequent stages of an object at different steps of the design process. They can be obtained in the design process by the application of the transformation system (or so called hypergraph grammars described below). Moreover, a hypergraph transformation (called a grammar production) can be also used to represent the modifications of an existing hypergraph, in order to introduce a required change in an object represented by it.

The below definition introduces the notion of a hypergraph grammar.

**Definition 4:** Let $H$ be the family of attributed hypergraphs and let $H \subset H$ be its subset such that $lb(V) = \Sigma_T \cup \Sigma_N$ where $\Sigma_T, \Sigma_N$ are nonempty and disjoint sets of terminal and nonterminal node labels, respectively. Hypergraph grammar $G$ over sets of terminal and nonterminal node labels $\Sigma_T, \Sigma_N$ is a tuple $G = (G_0, \Sigma_T, \Sigma_N, P)$, where $G_0 \in H$ is an initial hypergraph (so called axiom) and $P$ denotes a set of grammar productions defined below.

**Definition 5:** A hypergraph grammar production is a tuple of the form $P = (L, R, E, C)$, where: $L \in H$ is the left side of a production, $R \in H$ is the right side of a production, $E$ stands for an embedding transformation and $C$ is an applicability predicate, responsible for checking if $P$ can be applied in a given context. Production $P$ is also denoted in a simplified form as $P: L \rightarrow R$. Application of a production $P: L \rightarrow R$ to a given hypergraph $G$ is accomplished in following steps:

a. A subgraph $H \subseteq G$, isomorphic with the left side of a production, i.e., with $L$, is taken. If a predicate $C$ forbids applying $P$ then the process is interrupted and terminated.

b. $H$ is removed from $G$ and replaced by the right side of the production, i.e., by $R$. All hyperedges incident with $H$ get ‘orphaned’, i.e., they are no longer connecting nodes.
Finally, one has to fix all ‘orphaned’ hyperedges connecting previously $G - H$ with $H$, according to the specification provided by the embedding transformation $E$. As a result hyperedges are:

1. removed
2. unchanged, being reattached to $R$
3. replaced by new hyperedges.

In order to represent the design process such a hypergraph transformation system can be defined having rules (productions) corresponding to all actions that can be applied in a given design task. While this approach is well defined and has been successfully used in many design problems (Kotulski and Sędziwy, 2012; Kotulski and Strug, 2011, 2013), its main problem is a need for a huge number of productions in a more complex, real-life design problems. The solution to this problem that we proposed is the use of a system of graph transformations with a single graph transformation system defined for each part of the design. Such systems can operate largely independently working on a part of the design.

In Figure 4, an example of a graph transformation for the transformation system representing the design of the internal structure of the building is depicted. This transformation is responsible for moving a window from one wall to another. When $P$ is applied to a graph $G$ representing an actual state of a building being designed, then a subgraph of $G$, isomorphic with the left-hand side of the production $P$ is found and replaced by the right-hand side of $P$ in such a way that the node matched to $v_1$ is replaced by $w_1$ on the right side, $v_2$ by $w_2$ and the one labelled with $B$ remains unchanged. When node $v_1$ is replaced by $w_1$ all hyperedges that were attached to $v_1$ in the original graph are reattached to $w_1$ (as specified by the embedding rule $E$ associated with this transformation). The same is done to reattach other hyperedges. The applicability predicate $C$ for this production determines whether a wall represented by a node matched to $v_2$ has other objects attached to it and whether it is an external wall.

Figure 4 The sample graph transformation (production)$^P$

4 Synchronisation mechanism

As mentioned above the transformations systems working on different layers of the design can operate largely independently, but when the operations/changes introduced by subsequent graph productions involve shared elements, a way of synchronisation is needed. In Figure 5, an example of a building, represented by an external hypergraph is shown. Inside the solid representing a building from the external point of view a smaller solid – a single flat – represented by the internal hypergraph can be seen. The layout of
this flat is shown in Figure 1, while the building itself is a part of composite solid depicted in Figure 2.

The flat shown as a solid in Figure 5 and as a diagram in Figure 1(a), can also be seen in even more detail: when designing its internal furnishing and other arrangements. A view of one of the rooms with the furnishing is shown in Figure 6.

As the representations of different layers of the design a building can use different types of entities and thus different labelling and attributing schemes, a set of shared elements must be agreed upon in order to be able to support the communication and synchronisation processes. One of the most important of them is a common coordinate system and geometric coordinate format which have to be established. Secondly, names of attributes used by all transformation systems have to be agreed. Moreover, a matching between labels used in different transformation systems has to be defined. This matching enables an unambiguous mapping between elements that denote entities of the same type (obviously, it does not require the same names to be used in both systems). For example, a window seen by the external transformation system can be represented by a node labelled Win, by the internal layout designing system Wd and by the interior furnishing system W. To achieve such a matching a basic ontology supporting a communication process is built.

Figure 5  Two environments for graph transformations: building exterior and internal space (flat)

Figure 6  The internal furnishing of a room and its 3D depiction (see online version for colours)
The synchronisation process proposed in this paper can be described in three steps. When one of grammars (referred to as an initiator) wants to apply a transformation affecting a shared element of a building, it has to initiate the synchronisation process. Then, the second grammar (called a responder) checks data passed by an initiator and selects an appropriate transformation from its set and checks whether it can be applied. There exist three possible scenarios:

1. Production can be applied: all elements it affects are locked until the final decision from the initiator is received. The response ACCEPT is sent to the initiator.

2. The responder cannot immediately decide if a production can be applied (e.g., due to some additional operations required to be performed). The response (ACCEPT or REFUSE) is sent to the initiator with some delay. When ACCEPT is sent to the Initiator then all required resources remain locked until either EXECUTE P is received and performing P is completed or CANCEL is received. Otherwise (i.e., for REFUSE) no resources remain locked.

3. Production cannot be applied. The response REFUSE is sent to the initiator. No resources remain locked.

In response to a positive answer from the responder, the initiator decides whether to go on with the transformation. Even if the response from the responder is positive, the initiator can still decide to abandon the transformation (for example, when the responder’s answer arrived too late and the initiator has decided to perform another action).

Figure 7 Production synchronisation scheme

Notes: Grey shaded areas denote that a message may be sent with a delay (not immediately).

1We assume that a communication system underlying this synchronisation scheme assures that no message is lost. Otherwise, some more complex protocols are necessary.

When the initiator decides to perform a transformation, the responder has to perform relevant operations, otherwise he releases all locked elements. Let us note that the responder’s action, triggered by the initiator’s request, may be a single production or a sequence of productions as well (see Figure 8).
Case study

For a building design case study, we have three transformation systems. The first one is an external design system responsible for design tasks related to the design of the building solid and its external parts. The second one is responsible for the internal layout design, i.e., division of the floors into spaces, flats, halls, as well as internal doors and walls. Finally, the third transformation system is used for the interior design of the spaces that is for the furnishing of rooms.

Let us assume that the transformation system responsible for external design, working on the façade of the building, modifies it by applying subsequent productions belonging to its set of transformation rules. Suppose furthermore that one of the productions being applied, say $p$, contains a node representing a window being the shared element. Thus, the external transformation system (acting as the initiator in this case) cannot apply this production directly, but has to call the transformation system responsible for design and modification of the internal structure of the building, by calling an appropriate transformation, $t$ (shown in Figure 4), of the internal system (a responder in this example). Then, as described above, there are four possible scenarios.

1. The left side of the production $t$ is matched to the current graph and the applicability predicate $C$ returns the value TRUE, thus the internal system returns ACCEPT to the initiator.

2. The predicate $C$ returns FALSE, meaning that there is another object attached to the wall represented by the node matched to $v_2$. In this case, however, the responder system may trigger a sequence of productions, $s$ checking whether it would be possible to move the object attached to the wall to another location within the graph managed by the internal system. If the triggered productions return TRUE, the internal system blocks all elements related to all of these productions and returns ACCEPT to the initiating system.

3. The predicate $C$ returns FALSE, as in the case 2, but the responder system cannot trigger any productions to modify the internal structure (for example, a place to which the window was to be moved is on the border of two walls and would be separated by the internal wall), thus the responder returns immediately REFUSE.
The responding system cannot apply the production directly as it shares the window with the interior design transformation system, so it has to call an appropriate production of this system, thus becoming an initiator and the synchronisation process becomes a recursive one.

When the initiating system receives REFUSE from the responder, it cannot apply the production \( p \). If it receives ACCEPT, there are two possible scenarios:

a. Either the initiator sends the EXECUTE \( p \) request to the responder, which applies all required productions, or the initiator decides not to go on with applying these productions and sends the CANCEL message to the responder, which unlocks all relevant elements (i.e., locked previously in the result of the first initiator).

b. The second scenario may occur in cases when the responding system delays its response for a very long time due to, for example, a large number of internal checks. In such a situation, when a timeout has been reached, the initiator may decide to apply another transformation. Thus, when it finally receives an ‘outdated’ ACCEPT message it may no longer be interested in applying \( p \).

There are two important observations that have to be made here: Firstly, as in the case of many transformation systems operating on shared elements the recursion resulting from scenario four has to be assured to stop. Secondly, the synchronisation process is always defined between two transformation systems, so the first initiator does not have to be aware of the second system being the initiator of another synchronisation process.

### 6 Conclusions and future work

The mechanism proposed in this paper allows two separate transformation systems to communicate when they have to perform such potentially conflicting operations (performed on shared elements) and to operate independently when no cooperation is needed. At this stage, we consider a synchronisation which either is carried out successfully (i.e., both transformation systems return ACCEPT) or fails altogether. In the future we plan to take into account the possibility of one system having higher priority and thus forcing, under certain conditions, changes it requires, as such a situation often happens in real-life design tasks.

The formal mechanism of synchronisation of a number of different hypergraph grammars operating on different graph representations was tested on the case study from the domain of architectural design process. The process was viewed from the perspectives of interior furnishing design, internal layout design and exterior design. In the presented example, the operation of shifting a window was considered. For the external design system, it is just an object assigned to the façade of a building. From the interior perspective, a window is an entity assigned to the internal space of a building (room, hall and so on). From the interior design system point of view, it is the constraint that has to be taken into account in the placing of some elements. Performing any action on objects which are shared by several environments may potentially cause conflicts related to a violation of design constraints and/or standards. It is possible, for example, that moving a window to a new location (on a building façade) may cause it to coincide with a partition wall inside a building from the internal layout perspective or be covered by a storage unit from the perspective of the furnishing system.
It should also be strongly emphasised that the problem of synchronisation described above (i.e., related to avoiding conflicts between different transformation systems) is not limited to this context only and can be used in other domains. It is also worth remarking that the introduced synchronisation model may be also applied to hypergraph representations derived from other building models (e.g., BIM-based descriptions). Such cases will be presented in our further works.

References


