Virtual Reality and Logic Programming as Assistance in Architectural Design

Dominik Strugała and Krzysztof Walczak(✉)

Poznań University of Economics and Business,
Niepodległości 10, 61-875 Poznań, Poland
{strugala,walczak}@kti.ue.poznan.pl,
http://www.kti.ue.poznan.pl/

Abstract. In this paper, we describe a new approach to designing architectural spaces, called SADE, in which virtual reality and logic programming techniques are used to support and simplify the architectural design process. In the presented system, users have the possibility to design, configure and visualize architectural spaces within an immersive virtual reality environment, which helps them get better understanding of the created spaces. The system simplifies the design process by taking into account formal design rules, which describe domain knowledge, such as the construction law, technical conditions, design patterns, as well as preferences of a designer. The use of additional knowledge represented in the design rules can significantly shorten and improve the design process.

Keywords: Virtual reality · Logic programming · Architecture · 3D content · Unity

1 Introduction

The increasing use of interactive 3D technologies, such as virtual reality (VR) and augmented reality (AR), for building immersive multimodal human-computer interfaces is possible mainly due to the significant progress in the efficiency of computing and graphics hardware, continually increasing bandwidth of computer networks, as well as increasingly sophisticated forms of presentation and interaction with 3D content available on end-user's equipment.

Technical progress is largely driven by the quickly growing market of computer games based on 3D user interfaces. However, the use of these techniques is not only limited to applications in the entertainment industry. They can be – and in fact often are – successfully used also in other areas, such as rapid prototyping of products, e-commerce, medicine, tourism, education, and training. The use of 3D/VR/AR techniques may not only lead to the implementation of improved versions of existing services, but it makes it also possible to implement a new class of applications and services, which otherwise would not be possible.

In the architectural industry, digital techniques have been widely used for a long time. 3D presentation and interaction methods largely improve the process

© Springer Nature Switzerland AG 2019

of creating architectural designs and their subsequent visualization, and enable better parameterization of individual components (BIM) [4,10,11]. The next step is virtual reality, which enables photorealistic presentation of the created designs in real time. Currently, VR techniques are used in architecture mostly for visualization of designs already created with some other design tools. The real challenge, however, is to enable the use of VR directly in the design process. A designer would be able to modify an architectural design while being immersed in its photorealistic VR visualization. However, there is a mismatch between the complexity of the design process and the availability and the precision of interaction techniques used in VR. The method of designing must be simplified to enable it to be performed directly in VR.

In this paper, a new approach to designing 3D architectural spaces is presented. In this approach, called SADE (*Smart Architectural Design Environment*), virtual reality and logic programming techniques are used to support and simplify the design process. In the presented system, users can design, configure and visualize architectural spaces within an immersive virtual reality environment, which helps them get better understanding of the created spaces. The system simplifies the design process by taking into account formal design rules, which contain domain knowledge, such as the construction law, technical conditions, design patterns, as well as preferences of a designer. The use of additional knowledge represented in the design rules has the potential to significantly shorten and improve the design process. The application architecture along with a prototype of the architectural design tool, implemented as an extension to the Unity IDE, are also presented.

The rest of the paper is organized as follows. Section 2 presents an overview of the state of the art in architectural design. In Sect. 3, the concept, the requirements and the implementation of the Smart Architectural Design Environment are discussed. Section 4 presents a step by step design example. Finally, Sect. 5 concludes the paper and discusses possible further development and research directions.

2 State of the Art

The advancement of technology has allowed the development of various approaches to the design of architectural spaces – both 2D and 3D. Many of these approaches aim at providing comprehensive design environments, while reducing the amount of information that must be supplied at the design stage and decreasing the duration and complexity of the creation process. This, however, is often associated with the increase of the complexity of the software used in the design stage. Commonly used tools use several approaches described below.

2.1 2D Drawings

2D design applications rely on the use of 2D drawings for modeling of architectural spaces. These applications enable creating 3D spaces with the use of 2D elements of various types and complexity, starting with simple lines in Autodesk AutoCAD and ending with complex multi-layer elements such as walls, e.g., in Graphisoft ArchiCAD or Autodesk Revit. The complexity of creating a space in this way depends on the degree of integration between the elements. For example, comparing two software packages from the same company – AutoCAD and Revit, one can conclude that creating complex architectural designs with AutoCAD is more time consuming than with Revit [3,18]. This is due to different ways these two applications handle the designed content. First of all, AutoCAD is a purely planar application that can be compared to a sheet of paper. Changes made in one place must be marked separately in another. In turn, Revit automatically transfers each change to the other planes (cross-section, elevation, projection). This can be a key aspect when the time for creating a design is critical. Nevertheless, each of these tools in its own environment has advantages over the other. The possibility of editing each element separately gives a developer full control over the drawings. On the other hand, automatic transfer of a single change to all related drawings facilitates and accelerates work on a project.

There is a possibility to build three-dimensional spaces with the use of 2D tools, such as AutoCAD, in a similar way as it is possible with Revit or ArchiCAD, which are specialized for creating 3D models, but the process is complicated, and – therefore – 2D tools are most often used in conjunction with specialized 3D modeling applications. Such an approach gives more freedom and introduces a hierarchy to the architectural space design. By focusing on individual design stages, it is possible to have greater control over the entire process, however, this is also often associated with the increase in the duration of the development due to the need for incorporating changes in both the 2D drawings and the 3D model.

2.2 Visual Content Modeling

The continuous technical progress has caused a significant increase in the availability of 3D modeling environments. There are many examples of 3D modeling tools, but the most commonly used in both the entertainment and architectural industries are: 3ds Max, 3D-Coat, Blender, Maya, Modo, Rhino, and Zbrush. These sophisticated environments intended for professional users provide a wide variety of tools and operations, which enable any object to be modeled. However, their comprehensive capabilities require that users have experience in 3D modeling. Nonetheless, creation of 3D content does not always have to be difficult. Proper orientation on specific domains and constraining available functionality permits creation of design environments that are user-friendly. Examples of such environments are: AutoCAD Civil 3D, Ghost Productions, SketchUp, and Sweet Home 3D. Designed with specific type of application domain in mind, they enable efficient creation of 3D content without the need of extensive experience in 3D modeling. However, it also results in a significant reduction of the generality of the content creation process.

2.3 Parametric Modeling

In the field of architecture, very popular is the parametric modeling approach, i.e., creating content with the help of parameterized complex content templates, instead of direct free shape modeling used in typical 3D content creation applications [22,23]. In this way, content is generated on demand using a specific set of parameter values. With the help of ready-made templates, the designer creating a given piece of content needs only to ascribe values to specific parameters, which does not require comprehensive knowledge. This limits to a large extent the flexibility of content creation, but also reduces the amount of information a designer needs to provide. These types of templates are often inspired by natural world phenomena such as the Voronoy diagram, which resembles arrangement of cells on a leaf. One of the applications that enable generation of this type of content is the Grasshopper extension to the Rhino modeling package [8].

2.4 Augmented Reality

Augmented reality is a promising technique in the field of architectural design [1,2,24]. However, currently the use of AR in this domain is low. This is due to several factors. Firstly, the available AR applications do not permit professional design of architectural spaces. Secondly, high cost and low availability of AR devices such as Magic Leap and Microsoft HoloLens, which facilitate interaction within the AR environments, discourage potential users and result in low market penetration. Devices which are widely available and popular, such as tablets and smartphones, do not provide the quality and the efficiency that would allow for effective creation of 3D architectural designs. Some companies provide simple AR design applications through application markets, such as the Google Play or the AppStore, but they are mainly limited to browsing and matching specific products (e.g., furniture), without the possibility of performing full architectural design.

2.5 Virtual Reality

Although virtual reality offers great benefits in architectural visualization [14,16], currently there is very limited access to applications permitting design of architectural spaces in VR [20]. There are software packages and extensions to modeling programs that enable generating a VR environment from an existing architectural model. In such an environment, a user can freely navigate to examine the designed space [5,7,9], but without the possibility to create, modify or even annotate the visualized space. Applications that enable creation of objects in VR are very limited in their functionality.

The currently used mainstream design process is the preparation of a ready-made 3D model in a program purely aimed at creating 3D architectural designs and exporting the model to a generally available game engine, such as Unity 3D [19] or Unreal Engine [6]. Consequently, to achieve the expected result, there is a need to posses knowledge both in the field of developing 3D designs and in operation of game engines. Often, there are issues related to compatibility between the modeling programs and particular game engines. By importing an architectural design into a game engine, a higher-quality visualization can be achieved, but still there are no means of modifying the design.

2.6 Design Patterns in Architecture

Design patterns are recognized and proven ways to manage the designed space. Patterns are based mainly on previously developed projects and current design trends [17]. Their usage is not strictly required, but non-usage of a relevant pattern is usually treated as a mistake. In cases when it is not possible to obtain an acceptable solution based on the applicable patterns, it is allowed to omit them. The use of design patterns can be seen as an attempt to introduce standardization into architectural design. One of the most well-known sets of design patterns in architecture is Feng shui [13].

2.7 Summary

High complexity of 3D content creation software and the large number of aspects to be considered during the design process make creation of architectural designs a sophisticated task on many levels. The above described approaches are suitable for solving different problems in various contexts. In a practical approach, it is necessary to combine several of these methods to ensure the expected final effect. Even this, however, does not guarantee achieving a satisfactory result due to limitations in the presentation of the final designs. There is an evident lack of an approach that would enable to combine professional architectural design process with the benefit of doing this in a virtual reality environment with the support of rules describing the architectural design process.

3 Smart Architectural Design Environment

In Sect. 2, several different approaches to creating 3D architectural designs have been presented. Each of them offers some specific advantages, but there is clearly a lack of an integrated solution that could facilitate architectural design, by enabling real-time visualization and modification of the design in an immersive VR environment, while simplifying the design process to mitigate the deficiencies of VR as a design user interface.

In this section, we describe the concept of the *Smart Architectural Design Environment* – the main contribution of the paper. First, motivation for the work is provided, followed by requirements, and a description of the approach itself.

3.1 Motivation

During the architectural design, highly complex 3D models are being created. There is a lot of freedom in this process, but there are also constrains. Properly modeled constraints help to simplify the process by lowering the variability of possible results. An example of well-used constraints is parametric modeling, described in Sect. 2.3, which hugely simplifies the design task. However, parametric modeling takes into account only physical properties of objects.

Further simplification of the design process may be possible by taking into account other rules, such as construction law, technical conditions, design patterns, and preferences of a designer. These rules, however, require a specific approach. Firstly, they do not describe only individual objects, but relationships between objects and groups of objects. Secondly, they operate on a different (higher) level of abstraction. Finally, the constraints are often changing, e.g., when new construction materials and elements are introduced, new law comes into force, or a new style of design is used by an architect.

These new constraints must be translated into information processable by a computer program. Construction law and technical conditions are legal concepts, which rise the problem of their interpretation. It is necessary to describe them in the form of unambiguous rules. The design patterns, which are principles developed based on experience, are more challenging because of their subjective nature. They can suggest particular combinations of choices (e.g., colors) or relationships between objects (e.g., relative location).

The applicable laws and principles must be assigned to individual objects, but they must reflect relationships with other objects or types of objects. Therefore, a taxonomy of objects is required and rules must be programmed in a way enabling dynamic composition and linking of rules for all objects present in a given space. For this purpose, formal specification of an ontology and logic programming languages (e.g., Prolog) may be used. Logic rules may be associated with classes of architectural spaces, classes of objects or even specific objects in the ontology, thus enabling expression of constraints at different levels.

The use of a content library is needed to simplify the management of objects and rules – both at the creation stage and during their later use. To enable easy access and sharing of both particular objects and designs by design teams, a remotely accessible shared repository may be used. There is a large variety of modeling techniques and programs currently used by the designers (e.g., 3ds Max, Blender or SketchUp), and therefore it is important to enable importing models in various data formats.

An important decision is the choice of the user interface style. On the one hand, classical user interfaces (screen, mouse, keyboard) provide a convenient and familiar working environment for architects. On the other hand, VR interfaces – even if they are constrained in their interaction capabilities and precision – clearly provide an added value in permitting to immediately see the changes in a realistic visualization. It is important, therefore, to enable the use of both interface styles. A specific UI must be provided within the VR space for presentation of messages and warnings. This may be accomplished with pop-up windows, annotations, or changing the color of individual objects.

3.2 Requirements

The motivation presented in the previous section can be converted into a list of requirements, as presented below.

1. The system should enable immersive presentation of an architectural space and natural navigation of users and designers within the space.
2. The system should enable modification of the architectural space, while a designer is immersed in the space. In general, the space being designed can be a part of a more comprehensive 3D virtual environment (e.g., a kitchen being a part of a flat, a flat being a part of a building).
3. The system should allow for using a blueprint or an empty model of an architectural space, but also for using elements taken from existing content libraries. It is important that these elements could be imported in various formats, not only limited to currently popular ones such as .3DS, .FBX or .OBJ, but also formats used in other applications.
4. The system should enable parameterization of objects. Parameters should describe individual objects as well as connections between objects.
5. The system should permit categorization of objects to enable associating rules with specific categories of objects and creating dependences between interdependent elements of the designs (e.g., washing machine and bathtub).
6. The system should enable the use of semantics as a formal way of describing the role, the structure, and the interaction of particular design components to permit automatic reasoning as well as efficient search and parameterization of components.
7. The system should provide support for expressing design rules, which represent architectural design patterns, common practices, and other constraints.
8. The rules should be independently associated with spaces, objects and their categories, and then combined ad-hoc during the design process, thus forming a set of rules applicable to a particular part of the design.
9. The system should enable labeling of selected regions in order to assign them to specific semantic categories, which may be affected by different sets of design rules.
10. The design environment should enable access to a repository of already designed and described components, at the same time providing mechanisms to modify them.
11. Finally, the system should enable definition of actions to be executed as a result of evaluating rules (e.g., suggesting an element, limiting a value, informing the designer about errors).

Requirement 1 is met by existing game engines and VR environments, but the use VR is not currently within the mainstream of architectural design practices. Requirement 2, in a general case, is difficult to meet due to the complexity of the 3D design process. Therefore, in the current practice, permitted modifications are limited to predefined surfaces (e.g., painting colors on a view of a wall) and basic changes to particular elements, without the reference to other objects inside the environment [12,15]. Requirement 4 (parameterization) is partially supported by the existing Building Information Model (BIM) environments. The use of parameters, however, is limited to simple values (e.g., weight of a bathtub) and it does not support complex relationships (e.g., the distance between an electric device and a bathtub). Requirements 3, 5 and 10 are available within VR/AR design environments, such as the Unreal Engine and Unity. Requirements 7, 8, 9 and 11 are at an early conceptual stage or are not considered in current systems. Finally, requirement 6 has no reference in current VR/AR or architectural modeling environments.

3.3 The SADE Approach

The use of VR for designing architectural spaces requires simplifying the process as much as possible. The SADE approach is an attempt to partially handle this problem by implementing the domain knowledge into a set of well-defined rules. The application of rules does not only facilitate the use of VR interfaces, but most importantly it supports design experts in their daily work through the simplification and safeguarding of the design process. To some extent, it also enables design or configuration of proper architectural spaces by non-experts.

In the SADE approach, an immersive VR system is used both for the design and for the visualization (requirements 1 and 2). The design is performed using semantically described components, with well-defined meaning, parameters, and interactions (requirements 3, 4, 5, 6). The components are stored in shared semantic on-line repositories (requirement 10). The design process is supported and simplified by verification of formally defined rules, which describe possible actions and therefore limit the complexity of the design process (requirements 7, 8, 11). While designing a space, a designer associates it with a specific class of spaces (e.g., kitchen, office) thus activating appropriate sets of rules to use (requirement 9). Rules may be also associated with classes or instances of architectural objects used within the spaces. All rules are merged in real time enabling proper interaction between all components in the space and the space itself.

The SADE approach uses a common ontology of concepts. A fragment of the ontology with classification of concepts is presented in Fig. 1. The ontology describes both architectural spaces and particular classes of objects, which may be present in the spaces. The starting point is an abstract class *Architectural Entity*. It can be either an *Architectural Space* or an *Architectural Object*. Subclasses of the Architectural Space represent different categories of areas with different sets of design rules. The first example subcategory is *Room*. It includes categories such as *Living Room*, *Kitchen* and *Bathroom*. Some categories can have specific variants, e.g., *Large Living Room*. In some cases, rules for several categories can be combined in a subcategory, e.g., *Living Room with Kitchen*. Another type of architectural spaces is represented by the *Open Space* class. These are fragments of larger spaces, which cannot be clearly assigned to any specific type of room.

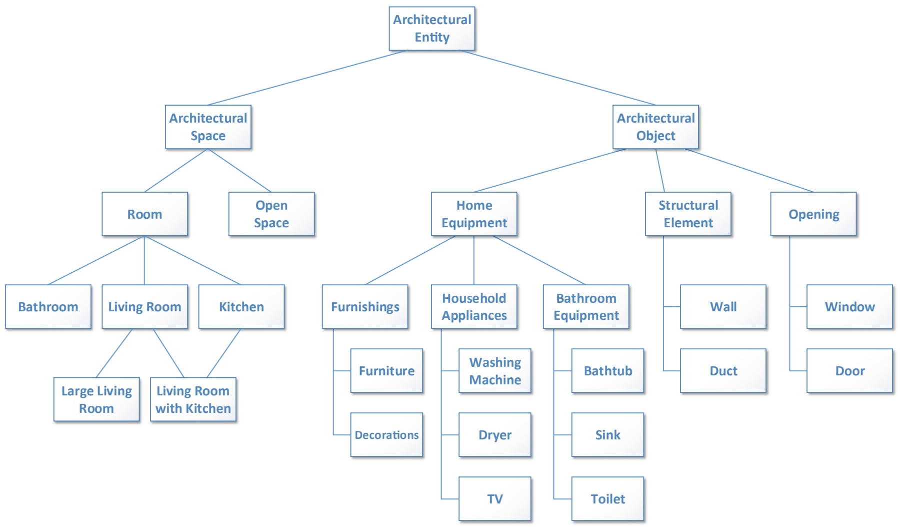


Fig. 1. Classification of concepts in the SADE Ontology.

The second type of Architectural Entity is represented by the *Architectural Object* class, which includes *Structural Elements*, *Openings* and *Home Equipment*. Structural elements are static parts of the construction, e.g., walls or ducts. Opening, as the name suggests, contains all structural openings, i.e., windows and doors. The third type of architectural objects are elements of the *Home Equipment*, encompassing such classes as *Furnishings*, *Household Appliances*, and *Bathroom Equipment*.

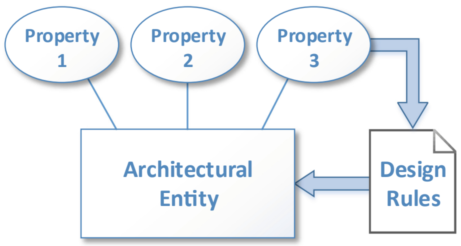


Fig. 2. Architectural entity with properties and design rules.

Each of the entities may be associated with a specific set of *Design Rules* (Fig. 2) governing the use of the entity in a design. The design rules are expressed as clauses in the Prolog language. The clauses may be parameterized with values of properties of the entities. During the design process, a property value may be set by the designer explicitly (e.g., type or weight of the object) or implicitly (e.g., position or size of the object). Properties and rules may be associated with any kind of architectural entity, including both spaces and objects.

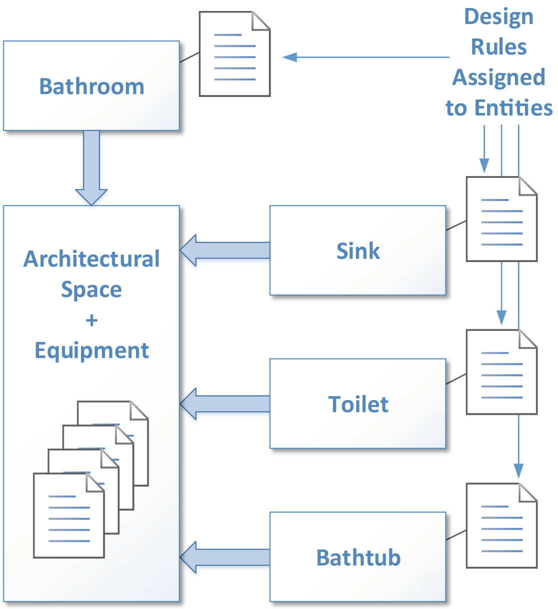

Bathroom
Design Rules Assigned to Entities
Architectural Space + Equipment
Sink
Toilet
Bathtub

Fig. 3. Merging design rules in SADE.

When an architectural space is being designed, all rules (implemented as Prolog clauses) associated with the space and particular objects that are used in the space are merged. Since all clauses use a common ontology, they can be combined automatically into a consistent set of rules, describing the space being designed. The process is illustrated in Fig. 3. An architectural space of class *Bathroom* is being designed. The class has its own set of rules, which are automatically associated with the space. The designer adds tree items of equipment – a *Sink*, a *Toilet*, and a *Bathtub*. All these elements have their own sets of rules, which are combined with the set of rules specific for bathroom. The resulting joint set of rules is used during the design process. Whenever the designer adds another element, the set of rules is updated. An example of a merged set of rules is presented in Listing 1.1

Listing 1.1. Example of merged design rules in Prolog

```
 1  %Generic utility clauses
 2  minDist(A, B, X) :- (point(A, X1, Y1), point(B, X2, Y2),
 3                      sqrt(((X1-X2)*(X1-X2))+((Y1-Y2)*(Y1-Y2)))>X).
 4  maxDist(A, B, X) :- (point(A, X1, Y1), point(B, X2, Y2),
 5                      sqrt(((X1-X2)*(X1-X2))+((Y1-Y2)*(Y1-Y2)))<X).
 6
 7  %Main bathroom correctness clause
 8  bathroom() :- verifiedItems(), verifiedDistances().
 9
10  %Items required in a bathroom
11  verifiedItems() :- (sinkExists(), toiletExists(),
12                         ( bathtubExists(); showerExists()) ).
13
14  %Checking existence of items
15  sinkExists() :- (type(_,'Sink')).
16  toiletExists() :- (type(_,'Toilet')).
17  bathtubExists() :- (type(_,'Bathtub')).
18  showerExists() :- (type(_,'Shower')).
19
20  %Distance rules for bathroom
21  toVerify(minDist('Toilet','Duct',20),'Too␣close␣to␣the␣duct!').
22  toVerify(maxDist('Toilet','Duct',100),'Too␣far␣from␣the␣duct!').
23  toVerify(minDist('Washer','Bathtub',100),'Too␣close␣to␣bathtub!').
24  toVerify(minDist('Sink','Bathtub',50),'Too␣close␣to␣bathtub!').
25
26  %Created when items are added
27  type('ID1', 'Duct').
28  type('ID2', 'Sink').
29  type('ID3', 'Bathtub').
30  type('ID5', 'Toilet').
31  type('ID4', 'Washer').
32
33  %Set when items are placed
34  point('Duct', 0, 0).
35  point('Sink', 0, 50).
36  point('Bathtub', 0, 120).
37  point('Toilet', 60, 0).
38  point('Washer', 100, 0).
39
40  %Attemt to falsify any of the rules and write a message
41  failedVerify() :- toVerify(Rule,_), \+call(Rule), !, true.
42  :- (toVerify(Rule, Msg), \+call(Rule) -> writeln(Msg); true).
43
44  %Verified if not possible to falsify any of the distance rules
45  verifiedDistances() :- \+ failedVerify().
46
47  %Check the main rule and write a message
48  :- (bathroom() -> write('All␣correct'); write('Not␣correct')).
```

In the example, the first two clauses (lines 2–5) are used to verify the minimum and the maximum distances (X) between two items (A and B). The main bathroom correctness clause (line 8) indicates that a bathroom design is correct when there are all required items and there are proper distances between the items. In the example, items required in a bathroom are a sink, a toilet, and either a bathtub or a shower (lines 11–12). Next four clauses (lines 15–18) verify whether there are objects of these classes in the design. The names of classes are taken from the SADE Ontology (Fig. 1). Lines 21–24 contain distance rules that must hold in a correctly designed bathroom. Clauses 2–24 are taken from the design rules assigned to the *Bathroom* class (Fig. 3). Clauses 27–38 indicate that items of particular classes are assigned to the bathroom design and set their positions. The clauses are taken from rules assigned to the particular classes of items (e.g., *Sink*, *Toilet*, *Bathtub*) with specific values taken from properties. In the listing, they are grouped for readability. The clause 41 is an attempt to falsify any of the distance rules (lines 21–24). If any of the distance rules fails, appropriate message is displayed (line 42). If none of the clauses fails, it means that distances are correct (line 45). The last clause (line 48) is used for testing if the main bathroom rule holds.

3.4 SADE Implementation

The design process is performed in an open distributed environment that enables the use and sharing of content. Design is understood here as building a space or a part of a space from scratch (which may involve importing libraries of objects) or re-using elements previously created – also by other users – and finishing with ready-made design templates.

The method of categorization and parameterization of components in SADE builds upon the previous work on SemFlex [21]. The main distinguishing novelty is the use of formally defined *design rules*, as described in Sect. 3.3. The design rules are expressed using constraint logic programming in Prolog. Prolog is a declarative programming language – the program is expressed in terms of clauses, representing facts and rules. It is the most popular logic programming language, with several free and commercial implementations available. Most importantly, Prolog enables to freely combine sets of rules given that the common ontology of concepts is provided. This feature enables implementation of separate sets of design rules for specific entities in the SADE Ontology – both spaces and objects (cf. Fig. 1), as well as combining different sources of rules, e.g., regulations, guidelines, design patterns, preferences, etc. Programming in Prolog is declarative and does not require advanced IT expertise or effort.

The SADE environment has been implemented as an extension to the Unity IDE (Fig. 4) communicating with shared repositories (SCRs) implemented as REST services operating on web servers. The Unity IDE was chosen as the starting point because of its popularity and the multitude of features it has as a cross-platform game engine and IDE. Unity enables to freely expand the design environment with new menus, custom component editors and editor windows using C# and JavaScript.

SADE can be operated in two ways. First, it can be used as a typical IDE with classic user interface providing all features of the Unity editor together with SADE extensions. Second, it can be used as a fully immersive VR environment, in which a user can directly interact with the designed space using an HMD

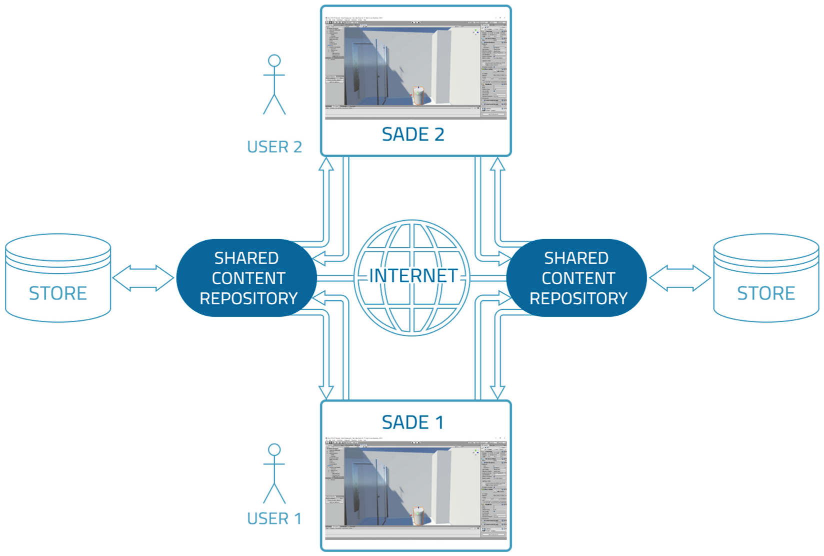


Fig. 4. Architecture of the Smart Architectural Design Environment.

and controllers. With the help of the implemented design rules, all changes are monitored, which enables the system to limit possible actions and to inform the user about potential design errors or non-conformances to applicable regulations.

4 Design Example

Creating an architectural space in SADE starts with importing a 2D projection or an empty model. On the left side of the application window there is a hierarchical list of elements within a given scene. At the beginning, it contains only basic objects, such as: Camera, Source of Light, Room Projection. The right side of the screen contains a preview of the 3D scene being designed. In the bottom left corner of the screen located is the SADE tool window. A button available in the window enables selection of the class of the architectural space being designed. When the button is pressed, a pop-up menu appears with a list of classes taken from a shared repository (Fig. 5). After selecting the class, a class window is displayed, which enables either to edit or to apply the class to the architectural space (Fig. 6a). When the class is applied, a scalable surface is added to the scene, enabling the designer to set the boundaries of the architectural space of the selected class (Fig. 6b).

After a designer defines the boundaries of the space, the actual design process starts (Fig. 7). The process consists of several phases. First, the designer needs to choose objects that are to be located in the designed space. The objects are downloaded from a repository in the form of packages (.unitypackage) along with the design rules assigned to them. After importing packages with objects, it is

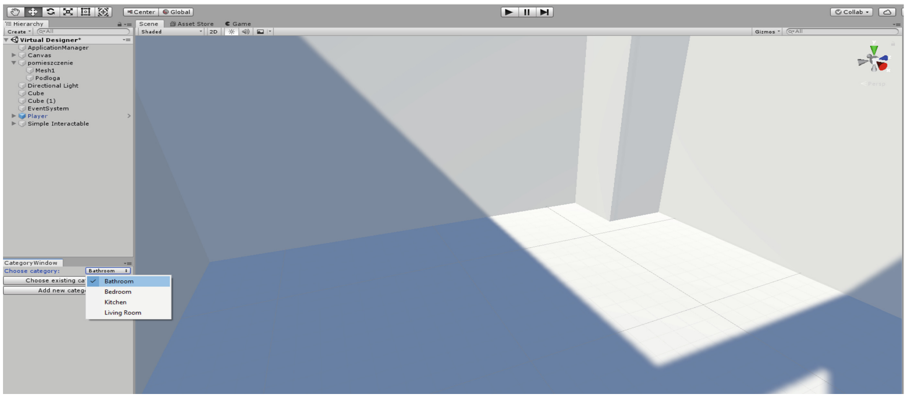
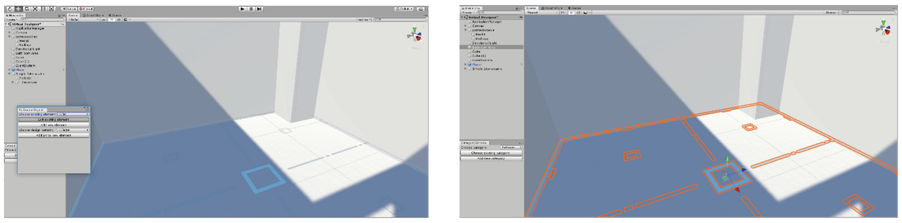


Fig. 5. SADE IDE – selecting the class of the architectural space.



(a) Applying the selected class (b) Indicating the area of applicability

Fig. 6. Indicating the area of applicability of a class of architectural spaces.

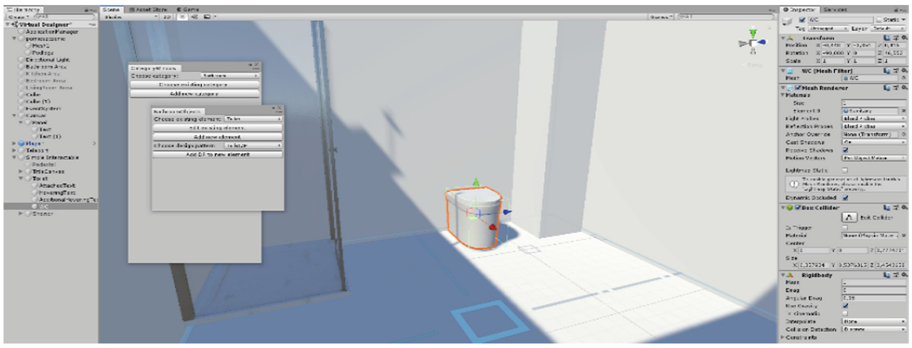
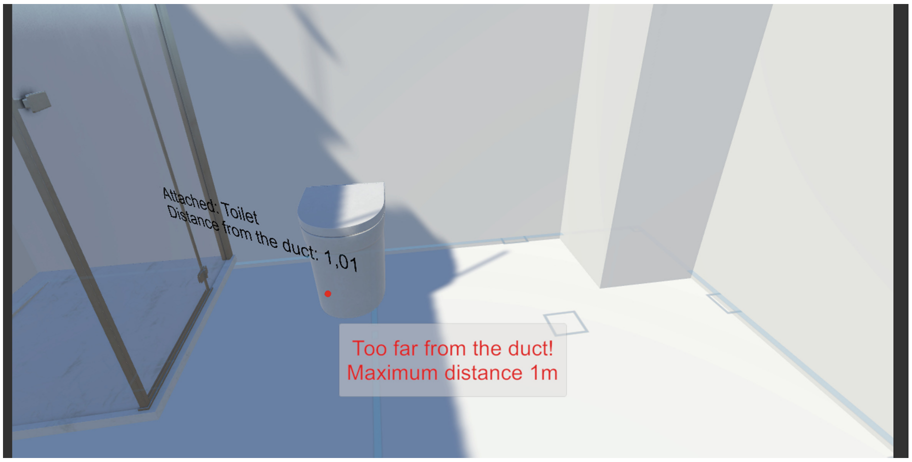


Fig. 7. Editing an item of equipment in the SADE IDE.

possible to add individual elements to the room (washing machine, toilet, etc.) by using a drop-down menu in the application.

During the design process, the SADE system verifies design rules in a given space. When the system identifies that the design is not consistent with the rules, the designer receives a warning. In the presented example, the room has been associated with the *Bathroom* architectural space class. In architecture, bathrooms have numerous restrictions regarding the mutual relations between individual elements and the required distances (Fig. 8).

At any time, a designer can switch to the immersive mode (e.g., using the HTC Vive HMD) and – with the help of controllers – can adjust the orientation and the position of objects in the space. In this process, the designer repeatedly receives feedback informing about verification of the design rules (e.g., if the distances from other elements in the space are correct).



Too far from the duct!
Maximum distance 1m

Fig. 8. Warning message resulting from the evaluation of design rules.

5 Conclusions and Future Works

In this paper, a new approach to the design of architectural spaces has been presented. The approach, called SADE, is based on the concept of design rules, which are formal descriptions of commonly used architectural regulations, practices, design patterns and preferences. By employing constraint logic programming the design rules can be encoded into precise clauses that can be verified at the design time. The use of immersive mode helps the designers to see a realistic presentation of the architectural space during the design, at the same time verifying correctness of the space.

The use of the presented approach enhances the process of designing architectural spaces by helping to avoid design errors and reducing the designer's effort through automatic verification of rules. Thanks to the use of shared repositories, it is possible to use a wide range of products available on the market as library components, and also to share objects and designs with other designers.

Currently, the SADE IDE prototype – implemented as an extension to the Unity IDE – enables importing content and rules from repositories, assigning classes of architectural spaces to selected areas of designs, and creating instances of architectural objects. Programming of design rules is performed manually by editing Prolog scripts associated with the classes or objects in the repository.

As the next development step, we plan to extend the prototype by enabling definition of rules at different levels of abstraction, thus enabling implementation of the construction law, design patterns, design styles, designers preferences and technical properties of objects. The second direction of development is to expand the system of rules to automatically propose layouts of elements within a given space, using typical solutions and specific constraints.

In the future, we plan to extend the approach to support machine learning techniques, through the analysis of designers' choices and existing designs. This may further simplify the design process.

References

1. Abboud, R.: Architecture in an age of augmented reality: opportunities and obstacles for mobile AR in design, construction, and post-completion. NAWIC International Women's Day Scholarship Recipient (2013)
2. Anders, P., Lonsing, W.: AmbiViewer: a tool for creating architectural mixed reality. Association for Computer Aided Design in Architecture, pp. 104–113 (2005)
3. Dib, C.: 5 key differences between AutoCAD and Revit! (2017). LinkedIn https://www.linkedin.com/pulse/5-key-differences-between-autocad-revit-carole-dib
4. Engineering.com: BIM 101: what is building information modeling? (2016). https://www.engineering.com/BIM/ArticleID/11436/BIM-101-What-is-Building-Information-Modeling.aspx
5. Enscape: homepage (2018). https://enscape3d.com/architectural-virtual-reality/
6. Epic Games: unreal engine (2019). https://www.unrealengine.com/en-US/what-is-unreal-engine-4
7. Eyecad VR: homepage (2018). https://eyecadvr.com/pl/
8. Grasshopper homepage. https://www.grasshopper3d.com/
9. IrisVR: homepage (2018). https://irisvr.com/
10. ISO: BIM - Part 1: concepts and principles, ISO 19650–1:2018 (2018). https://www.iso.org/standard/68078.html
11. ISO: BIM - Part 2: delivery phase of the assets, ISO 19650–2:2018 (2018). https://www.iso.org/standard/68080.html
12. Kaleja, P., Kozlovská, M.: Virtual reality as innovative approach to the interior designing. Sel. Sci. Papers - J. Civil Eng. **12**(1), 109–116 (2017)
13. Mak, M.Y., Ng, S.T.: The art and science of Feng Shui - a study on architects' perception. Build. Environ. **40**(3), 427–434 (2005)
14. Paes, D., Arantes, E., Irizarry, J.: Immersive environment for improving the understanding of architectural 3D models: comparing user spatial perception between immersive and traditional virtual reality systems. Autom. Constr. **84**, 292–303 (2017)
15. Pixel Legend: virtualist (2019). https://www.pixellegend.com/virtualist/
16. Portman, M.E., Natapov, A., Fisher-Gewirtzman, D.: To go where no man has gone before: virtual reality in architecture, landscape architecture and environmental planning. Comput. Environ. Urban Syst. **54**, 376–384 (2015)

17. Salingaros, N.A.: Design Patterns and Living Architecture. Sustasis Press, Portland (2017)
18. Tobias, M.: How do AutoCad and Revit compare? (2017). https://www.ny-engineers.com/blog/how-do-autocad-and-revit-compare
19. Unity Technologies: unity game engine v. 2018.3.3f1 (2018). https://unity3d.com/unity
20. VRender: top 5 virtual reality applications for architects (2018). https://vrender.com/top-5-virtual-reality-applications-for-architects/
21. Walczak, K.: Semantics-supported collaborative creation of interactive 3D content. In: De Paolis, L.T., Bourdot, P., Mongelli, A. (eds.) AVR 2017. LNCS, vol. 10325, pp. 385–401. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-60928-7_33
22. Walczak, K., Cellary, W.: X-VRML for advanced virtual reality applications. Computer **36**(3), 89–92 (2003)
23. Walczak, K., Wojciechowski, R., Wójtowicz, A.: Interactive production of dynamic 3D sceneries for virtual television studio. In: The 7th Virtual Reality IC VRIC - Laval Virtual, pp. 167–177, Laval (2005)
24. Wang, X.: Augmented reality in architecture and design: potentials and challenges for application. Int. J. Architectural Comput. **02**(7), 309–326 (2009)