An Architectural Design System for the Early Stages

Sviataslau Pranovich, Jarke J. van Wijk, and Huub van de Wetering
Department of Mathematics and Computer Science, Technische Universiteit Eindhoven
Eindhoven, The Netherlands
{s.pranovich, j.j.v.wijk, h.v.d.wetering}@tue.nl

Abstract

A wide variety of drawing packages is available for architectural design. However, most of these systems are oriented to the production of final technical drawings, and only few support the early phase of design. In this paper, we present a new approach for a design system to support this phase. The method is based on a framework from architecture on the meaning of drawings in architectural design during the early design stage. In such drawings, not only standard graphics elements like contours, but also other elements like grids and axes of symmetry play an important role. The interactions between such elements encode high-level knowledge related to the design process. These elements are encountered also in standard drawing systems, but often only as tools. We propose to consider such elements as graphical objects. Relations between these objects can be defined to capture higher-level information on the structure of the design. Additionally, we offer a natural user interface for the designers, which enables them to explore design space effectively and efficiently.

Keywords: *drawing system, early architectural design, geometrical transformations, user interface.*

1. INTRODUCTION

Current drawing systems have reached a high level of sophistication, and are suited for the production of the final technical drawings in the final phases of the design process. However, they do not offer support for the early phase of the design process when concept formation is important. Such drawing systems require designers to specify many details in the drawing, while the designer does not care about them in this stage of the design process. Moreover, detailed drawing restricts the design creativity, whereas a system that supports early design should support and stimulate the generation of new ideas.

In order to develop a system that supports architects in an architectural fashion, it is necessary to look at the way architects work in the early phase of design. Research of H. Achten [1] in the use of drawings by architects leads to the view that architects use well-defined forms of graphic representations (graphic units) to depict their design intentions. Examples of these graphic units are contours, circulation schemes, grids, schematic subdivisions, axes of symmetry, and zones. The design is developed by means of these graphic units and interactions between them. We use the results of Achten [1] as a basis for our approach and propose to use these graphic units as the basic building blocks.

Current drawing systems typically offer only a basic set of primitives, supplemented with tools to achieve additional effects: to mirror objects, to align objects with respect to each other, to align them to a grid, etc. These tools have a limited scope: for instance, the fact that two objects are mirrored copies of each other is not stored explicitly; after a mirroring tool has been applied, higher-level knowledge (relations between objects) disappears.

In computer graphics terms, our approach can be formulated as follows. We propose to represent tools as geometric objects (presented in a form of graphic units) themselves: for instance, to introduce a symmetry axis as an object that has a graphic representation, can be manipulated, and influences other objects. Graphic units in this case allow establishing permanent relations between them, which is not possible when tools are used instead. Therefore the relations between graphic units can be used to store high-level information related to a design process.

In the next section we overview existing approaches for the support of designers in drawing systems and briefly present our approach. In section 3 we give a description of graphic units and their features. In section 4 we describe the handling of geometric transformation of objects in more detail and give some examples. In section 5 we present a user interface for the system. In section 6 we evaluate the system and in section 7 we draw conclusions.

2. BACKGROUND

Several directions have been pursued to support designers in the early phase of design. One direction is to attempt to bring a drawing tool closer to a sketching tool, another direction is to offer more support for conceptual information, for instance on relations between objects.

Sketching tools like SILK [12], SmartSketch [20] provide *beautification*. The designer can sketch free hand, the systems attempts to recognize common graphic elements from this input. The Pegasus [11] system introduces *predictive drawing* that predicts the user's next drawing operation based on the existing drawing. But in general, systems supporting freehand sketching with beautification techniques still suffer from a lot of limitations [19].

The other direction for design support aims at enabling the user to enter and use higher-level information. As a first step, many drawing systems offer tools and aids, such as mirroring, alignment, grids, gravity [22], and snap-dragging [6]. With these tools users can establish relationships between objects, but unfortunately most of the systems forget these relationships after the positioning operation is complete [9]. Another well-known support aid is grouping. Objects are merged, possibly recursively, and can be manipulated as a group.

Constraint techniques make a powerful addition to the interaction techniques available in graphical editors. SketchPad [22] was the first drawing system that used explicit constraints, defined by the user. It allowed lines to be constrained by relationships with other lines (perpendicular, parallel, etc.). ThingLab [13] extended that

notion by providing a general simulation environment. Constraints in Sketchpad and ThingLab are bi-directional and allow objects to be attached and updated simultaneously. Many other systems have been developed that provide constraint-based solutions for graphical applications, such as Garnet [15], Coral [23], Unidraw [25], ArtKit [24], WHIZZ'ED [8], Briar's [10] and Inventor [21].

Complex interactions of many constraints require sophisticated constraint satisfaction techniques. Constraint satisfaction problems are sometimes impossible to solve, if, for example, there are conflicting constraints. Even if the solver is able to find a solution, it must help the user understand how and why it got to the new state. In fact, the success of constraint-based approaches to drawing has been limited by difficulty in creating constraints, solving them, and presenting them to users.

In our work we do not focus on improving existing approaches (such as defining a new type of sketching tool or building a new constraint based drawing system), but we rather offer a new approach, which is based on the designer's view on drawings.

Research in the use of drawings by architects on the early design stage has led to the framework of Generic Representations: Architects use well-defined forms of graphic representations to depict their design intentions [3]. These forms have been identified and described as graphic units. A *graphic unit* is a set of graphic elements that are organized in a specific way and that have a generally agreed upon meaning for the designer. Some examples are: contour, functional symbols, circulation scheme, zone, etc. Graphic units can be considered as a medium to express the ideas in an architectural design [2]. A set of related graphic units defines a *generic representation* [1], where relations between graphic units play an important role. We propose to use graphic units as building blocks for a design system, i.e. to treat them as geometrical objects that can be edited, manipulated, and can be interrelated [17].

The user in such a system defines a design in terms of graphic units and relations between them. We use the idea of data-flow for the propagation of manipulations on graphic units. Manipulations are propagated through a graph, where the nodes are graphic units and edges are relations between them.

In summary, we aim at enabling the designer to use graphics units with an architectural meaning, such that these graphic units have intuitive behavior and respond to the user actions in meaningful and predictive way.

3. APPROACH

We begin the description of our approach by presenting graphic units and their features.

3.1 Overview

A user can create, edit, and delete graphic units: instances of predefined types of graphic units. Each graphic unit has a visual representation, and can be geometrically transformed. For this we use geometric transformation tool, the KITE manipulator [16]. Standard geometric transformations (translation, rotation, scaling), and also skewing are supported.

Besides the type of a graphic unit, it has more detailed information associated, for instance the exact shape of a contour. The user can change this information in a standard way: for instance to change the shape of a contour the user drags its vertices.

The user can define uni- and bi-directional relations between graphic units. A relation between graphic units presents the semantic and physical connection between them and is visualized as an arrow.

Transformations of a graphic unit are propagated along relations between graphic units. The spanning tree algorithm is used to define a subtree of the graph for the propagation of the transformation. Separate decisions can be made how the transformations are dealt with when they are propagated along the graph. The user can define which types of transformations have to be passed and/or applied for each graphic unit and relation.

For the manipulation of connected graphic units we use a special geometrical engine [18]. In section 4 we will elaborate this further.

3.2 Types of graphic units

Every graphic unit has a special meaning for the architect. Below we give a description of the main graphic units in our system and their features (for other graphic units see Achten [2]).

Contour: the most encountered graphic unit in the drawing of the architect. It is the basic unit to construct the design. The variety of contours is large: open contour, closed contour, self-intersecting contour, etc. In our system a contour is represented as a list of connected points. A contour is visualized as a polyline (see Figure 1).

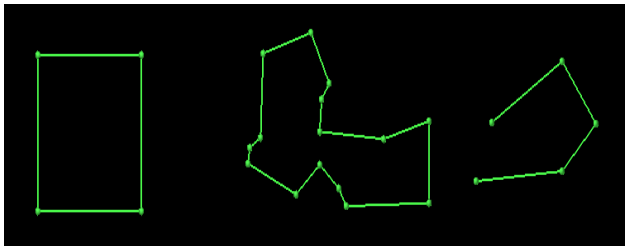

Figure 1. Contours.

It has no special features implemented, except gravity: it is sticky for other objects.

Grid: the alignment frame for the elements, which structures the design. The variety of grids is large: rectangular (Cartesian) grid, tartan grid, hierarchical (generic) grid [4], polar grid, curvilinear grid, etc. The Cartesian grid covers about 75% of grids that the architect uses in a design process. The Cartesian grid is an orthogonal grid with constant spacing. We have defined grids as two sets of not necessarily orthogonal lines. Each set is called a grid component and is visualized as a set of parallel lines. The *i*-th line in the set is defined by

$$\vec{x}(t) = \vec{b} + (\sin\alpha, -\cos\alpha)\cdot t + i\cdot(\cos\alpha, \sin\alpha)\cdot p,$$

where $\vec{b}$ defines the position of the grid component, α defines the orientation of the grid component, and p defines the period of the grid component. By combination of grids more complex grids, such as the tartan grid can be defined (see Figure 2).

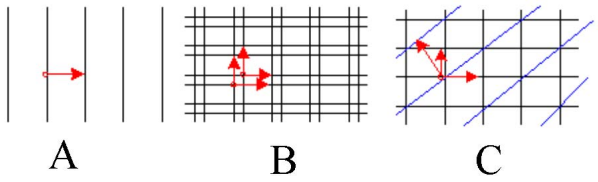

Figure 2. A – One dimensional grid; B – tartan grid; C – complex grid.

A grid has gravity: Vertices close to a grid line snap to it. In order to avoid complex images the grid is highlighted only for related (connected by relations) objects and fades away from them, according to:

$$T = \frac{1}{1+D^k}, \qquad T \in [0,1]$$

where T is the transparency of the grid, D is the distance from the related object, and k is the fading coefficient.

Axial system: presents a notion of symmetry between objects in the drawing of the architects. In our design system the user can mirror an instance of a selected graphic unit by creating an axial system. An axial system keeps the symmetry between twin graphic units by transferring mirrored geometrical transformations between twin graphic units. An axial system is visualized as a dashed line (see Figure 3).

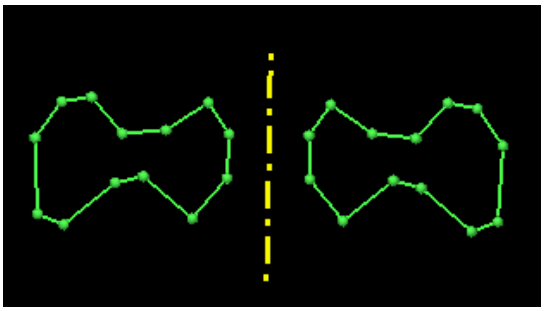

Figure 3. Axial system mirrors contours.

Zone: presents a general characteristic for the set of objects, which geometrically belong to some area. For instance, to define the space that is related to water (kitchen, toilet, etc.) architects use a wet zone. A zone is visualized as a semitransparent filled polygon (see Figure 4).

Figure 4. Zone.

Zones structure the design. Semantically it is close to standard grouping. The difference is that the decision whether an object belongs to a group or not, is based on its spatial location. All objects that are covered by a zone are related to this zone. If an object is moved outside a zone then the relation between them is deleted, if it is moved inside a zone, a new relation is created.

Image: for inspiration in the design process architects often use images. The designer can place different images on his workspace in order to trace elements and to draw on top of them, or he can place them in the drawing as illustrations.

4. GEOMETRICAL ENGINE

We have defined the basic building blocks, we now describe a geometrical engine, that handles transformations of graphic units. The engine is based on the propagation of geometrical transformations between anchor points (see details in 4.1) using the relations between them, where the anchor points define the origins for the local transformations of associated graphic units.

4.1 Anchor points

Anchor points are used by the geometrical engine as start- and end points for the propagation of geometrical transformations, and to provide for the user geometrical flexibility for the manipulation of graphic units. An anchor point can be associated to a graphic unit. If it is associated to a graphic unit, then the anchor point has properties that tell which types of transformations have to be passed to this graphic unit (e.g. is the graphic unit scalable, translatable, etc.). The user can define relations between anchor points along which transformations are propagated. Relations have also associated properties that record which types of transformations must be passed or blocked. Every graphic unit has at least one anchor point associated (one anchor point is created automatically for every graphic unit). In section 4.5 we will elaborate on the case where a graphic unit has more than one anchor point. Figure 5 shows the links between anchor points, graphic units, and relations in UML notation.

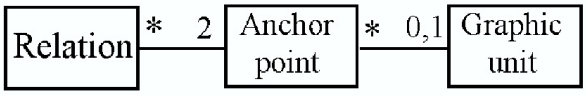

Figure 5. Relations, anchor points and graphic units.

Every graphic unit has a local origin for transformations. The origin is the position of the associated anchor point. The propagation of a transformation starts at a point selected by the user. Next the transformation is propagated through the anchor points and relations, affecting the graphic units that are associated to the anchor points. Anchor points and relations can be added and changed by direct manipulation.

4.2 The anchor points graph

Anchor points and relations between them establish a directed graph (APG - anchor points graph), where the anchor points are the nodes and the relations are the edges of the graph. At most one relation exists between two points in each direction. The geometrical engine uses the graph of anchor points as a transformable skeleton for the propagation of manipulations between graphic units. We do not impose restrictions to the graph of anchor points: the graph does not have to be connected; multiple independent subgraphs can be used; the graphs can contain cycles. However, this can lead to ambiguities, for instance multiple paths can exist from the anchor point to which the

transformation was applied to an anchor point downstream. To prevent these ambiguities, we derive a spanning tree [5, 14] on the fly when transformations are propagated. This ensures that every connected graphic unit is affected only once.

In the next sections we will describe in more detail how linear (rotation, scaling, and skewing) and non-linear (translation) transformations are propagated over a unique path in the graph resulting in displacement of anchor points.

4.3 Propagation of linear transformations

Consider a linear transformation L applied in anchor point a, which is used as a local origin. The transformation L is propagated over the spanning tree resulting in a displacement $\vec{\Delta}(L,x)$ of anchor point x:

$$T(x) = a + L(x-a) = x + \vec{\Delta}(L, x-a) \qquad (1)$$

Given a spanning tree of the graph, a unique path in this tree exists between a and another anchor point b: $a = b_0, b_1, ..., b_n = b$. For every anchor point b we write $L(b)$ in terms of this unique path between a and b. The transformation of b, ignoring blocking relations on the path from $b_0 = a$ to $b_n = b$, can be expressed as

$$T(b) = b + \vec{\Delta}(L, b-a) = b_n + \sum_{i=1}^{n} \vec{\Delta}(L, b_i - b_{i-1}) \qquad (2)$$

where we use that $b - a = \sum_{i=1}^{n} b_i - b_{i-1}$, and that $\vec{\Delta}(L,x)$ is a linear function of x.

The effect of blocking relations is defined as

$$T(b) = b + \sum_{i=1}^{n} \phi(i) \cdot \vec{\Delta}(L, b_i - b_{i-1}) \qquad (3)$$

where the function $\phi(i)$ defines the blocking characteristics of a relation between anchor points b_{i-1} and b_i; $\phi(i) \in \{0,1\}$.

A geometrical transformation is propagated through the graph of anchor points as long as these points have outgoing relations. The system supports two modes for propagation of geometrical transformations in the graph, which are defined by different functions $\phi(i)$. We offer two possibilities:

$$\phi_1(i) = P_{Ri} \qquad (4)$$

$$\phi_2(i) = \prod_{i=1}^{i=n} P_{Ri} \qquad (5)$$

where P_{Ri} is a blocking property of relation R_i between points b_i and b_{i-1}; $P_{Ri} \in \{0,1\}$. In (4) the blocking properties of relation R_i independently define the transformation propagated from node to node in the APG. In (5) the chain of relations from the start relation R_1 to the relation R_i is taken into account: once blocked, the transformation is not propagated further.

As an example, an anticlockwise rotation is applied in point b_0 to the graph of anchor points $\{b_0, b_1, b_2, b_3\}$ (see figure 6). The dashed lines depict blocking relations. Figure A shows the initial positions of the anchor points; figure B shows the result of the transformation if $\phi = \phi_2$; figure C shows the result of a transformations if $\phi = \phi_1$.

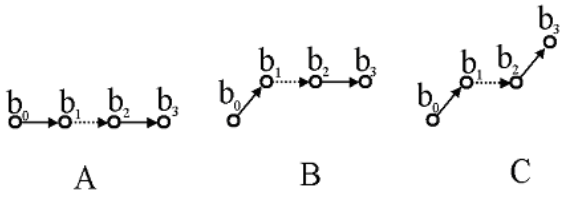

Figure 6. The propagation of rotation in the chain of anchor points. Dashed lines visualize the relations that block rotation; normal lines visualize the relations that do not block rotation.

4.4 Propagation of translation

Consider a translation $\overrightarrow{\Delta t}$ applied in anchor point a. Therefore the transformation T of the point x without blocking is given by

$$T(x) = x + \overrightarrow{\Delta t} \qquad (6)$$

In a chain of anchor points from a to b: $a = b_0, b_1, ..., b_n = b$ the $\overrightarrow{\Delta t}$ transformation of an anchor point b with blocking relations is simply

$$T(b) = b + \phi(n) \cdot \overrightarrow{\Delta t} \qquad (7)$$

4.5 Transformation of graphic units

We have described how geometric transformations are propagated through the graph of anchor points, next we consider how these transformations affect graphic units.

A graphic unit can have more than one associated anchor point (see Figure 7). This can be used to define auxiliary local origins for the transformations of a graphic unit. An anchor point has at most one associated graphic unit: in some special cases an anchor point can exist without a graphic unit.

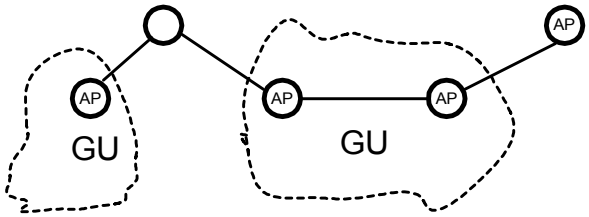

Figure 7. Relations, anchor points and graphic units.

Anchor points serve as local origins for transformations, the graphic unit itself is defined by a different set of points, which describe e.g. the shape of a contour or the spacing and direction of a grid. These points are transformed similarly as anchor points themselves; implicitly relations from anchor points to these other points are used.

Furthermore, an anchor point has a set of attributes that define which transformations are passed to the points p_k of the graphics unit. If the geometrical transformation now reaches an anchor

point on its propagation path and the attribute of an anchor point allows the transformation of the associated graphic unit, then a graphic unit is transformed, where the anchor point serves as origin for the transformation. We can present the new position of a point p_k as a sum of this point's position displacement after the transformation T, and the original position of point p_k:

$$p_k' = p_k + \vec{\Delta}_b + \phi(i) \cdot \vec{\Delta}(T, p_k - b) \cdot P_{GU} \qquad (8)$$

where $\vec{\Delta}(T, p_k - b)$ is the displacement of point p_k as a result of the transformation T, applied with respect to the anchor point b; function $\phi(i)$ is defined in equation (4) and (5); P_{GU} is the blocking property of an anchor point for the transformation of graphic unit, $P_{GU} \in \{0,1\}$; $\vec{\Delta}_b$ is a transformation of the anchor point b (which serves as a local origin), see (3). We define $P_{GU} = 0$ for translation, because the transformation is already handled in $\vec{\Delta}_b$.

The possibility to define the attributes in relations and anchor points for each type of transformation individually gives a high flexibility in drawing, editing, and therefore in the exploration of design space.

4.6 Example

Figure 8 shows a graph of anchor points, as it is presented to the user. Colours are used to visualize the spanning tree. Red (gray on picture) is used for the visualisation of relations and graphic units that will be transformed; blue (black on picture) is used for relations and graphic units that are in the tree, but blocked for this transformation. The remaining elements are coloured white.

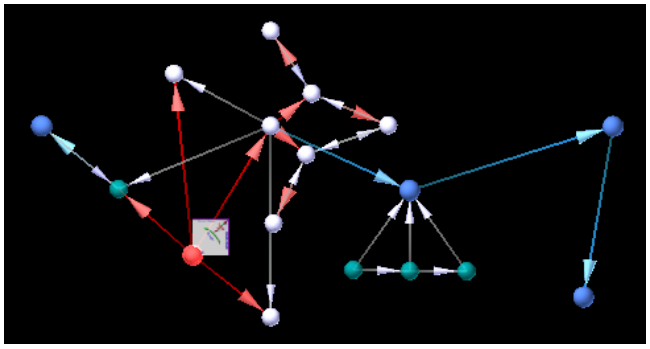

Figure 8. The propagation of a transformation

We give three examples in order to illustrate how the graph of graphic units can be transformed (see Figure 9).

On each of them three connected anchor points are used and a rotation is applied to the first anchor point (AP1). Figures A in all pictures show the placement of objects before the transformation begins, figures B show the objects after the transformation.

The variety in properties of relations and anchor points assists to achieve a large range of geometrical functionality in the manipulations of graphic units. In other words, the designer can specify the structure of his design explicitly, which enables him to explore different realizations efficiently.

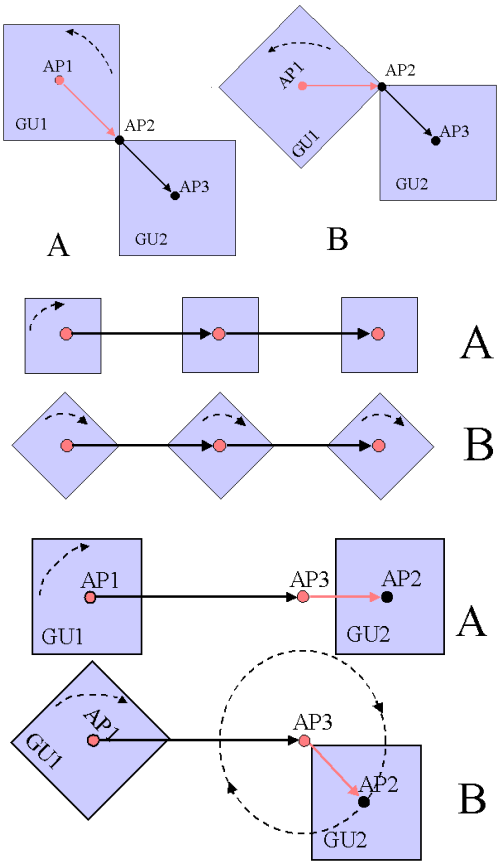

Figure 9. Examples of transformation. We use colour for the visualization of properties: black is used for the transforming relations and anchor points; red (gray) is used for relations and anchor points that block rotation.

5. USER INTERFACE

The system offers many options to define the relations between graphic units. But having too many options sometimes is not productive since the user has to manage a complicated user interface (relations and anchor points sometimes become intricate). In order to solve this problem we provide an extra user interface (skin) on top of the interface of the geometrical engine (see Figure 10). This extra user interface uses a natural interaction technique for architects and hides relations, anchor points, and their properties from the user.

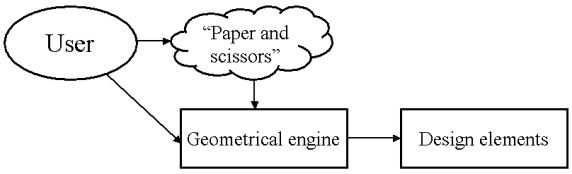

Figure 10. An extra user interface is provided on top of the user interface of the geometrical engine.

The system creates and deletes relations and anchor points without the need for explicit actions of the user. As a starting point for this user interface we use a well-known design

metaphor, which is called "paper and scissors": the designer is experimenting by means of constructing and assembling different objects from paper, placing them on top of each other and manipulating them.

In order to apply this concept in our system we make one assumption: Every graphic unit has a depth that can be changed by the user. Using the depth and layout geometry the system extracts information about relations between graphic units. We look at an example (see Figure 11): we put one sheet of paper (B) on top of the other (A). If we manipulate sheet A, then sheet B will be affected also, because the user implicitly implies a relation A→B between two sheets of paper. The system is capable to reconstruct (establish and delete) such relations between graphic units using the following criterion: If graphic units have an overlapping area, the relation between them is created (from the bottom one to the top one).

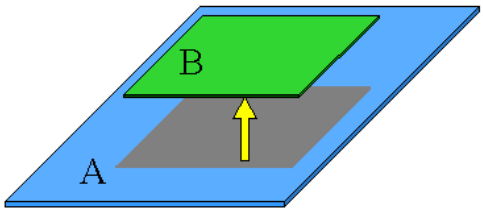

Figure 11. Example with two sheets of paper.

Moreover, the idea of auxiliary local origins (or anchor points) is implemented in the new interface as a pin (the equivalent of the paper-pin). The pin between graphic units A and B (see Figure 12) represents an anchor point that belongs to A, and has an outgoing relation to B.

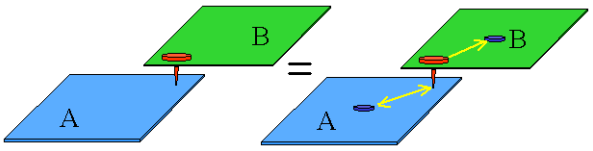

Figure 12. The Pin.

The user is provided with pins that can connect graphic units and block propagation of particular transformations. The user can modify the pin by switching on/off the blocking of transformations. The pin, which blocks all transformations, visualized as a nail-head pyramid, where each nail-head has its own colour and marks the blocking of a particular transformation (see Figure 13).

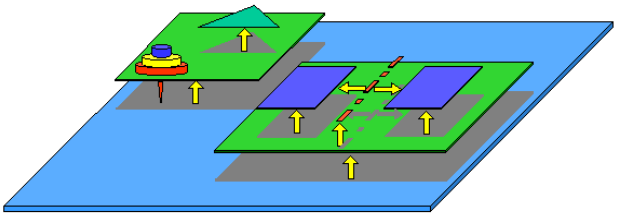

Figure 13. The arrows visualize the relations between graphic units that are extracted by the system from a design example.

The user also can use a clip (the equivalent of a paper-clip), which simply is an equivalent of a bi-directional relation between graphic units.

6. RESULTS

We have implemented a prototype of the system and tested it with a few architects (see Figure 14). Despite that the current implementation of the prototype is not perfect yet (the drawing part is rather primitive) architects find it useful and inspiring for the creation of new ideas during the design process. It is suitable especially for the preliminary design phase; the phase that follows a sketch design phase in the early design stage.

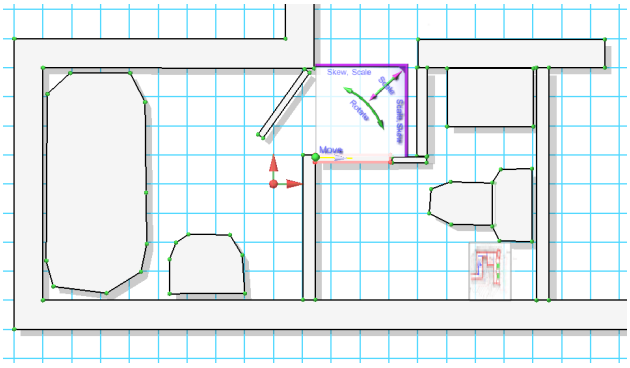

Figure 14. A design example made by an architect in the preliminary design phase.

Furthermore, architects like that objects can be manipulated effortlessly (what is provided by the functionality of the geometrical engine, see Figure 15), and the familiarity of the interaction style. Moreover they were pleased to have a user interface, that provides all functionality in visual way avoiding the need to use textual languages to program parameterized design [7] such as in AutoCAD, ArchiCAD, MicroStation, Pro/ENGINEER, CATIA, etc.

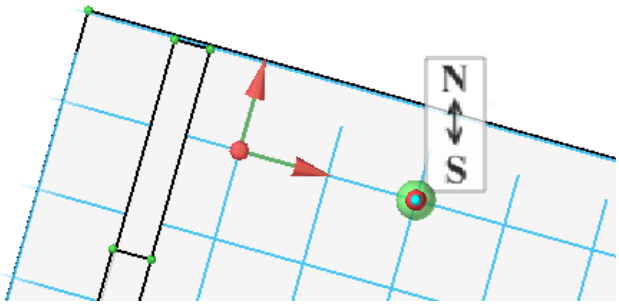

Figure 15. Here the Pin is used to stick the "North-South" sign to a grid and to freeze its orientation.

Summarising, the designer in our system can specify the structure of his design implicitly, focusing more on the design process rather than on the interaction with the system.

7. CONCLUSION

In this paper we introduced a new approach for a design system for architects, based on a framework of architect drawings.

Despite that a constrained based geometrical engine is used as a basis for this system, the paper is focusing on a technique how to capture and how to use high-level architectural information, which is defined in relations between design elements, rather than focusing on constraints problems.

We have implemented the prototype of the system and discussed it with architects. The system supports and stimulates the generation of new ideas, which is very important on the early stages of design.

At the moment we are exploring directions for improving our system. New functionalities can be added, such as new types of graphic units. It is not so difficult to offer many options and much flexibility, the main challenge however is to tune the interaction techniques and settle the visual metaphor on top of the system in such a way that the user can define what he wants in an intuitive way.

Acknowledgements

We thank Kees van Overveld for his advice and contribution to this work.

References

[1] H.H. Achten, Generic Representations: Typical Design without the use of Types, *Proceedings of Computer Aided Architectural Design'97*, p. 117-133, 1997.

[2] H.H. Achten, Generic Representations - An Approach for Modelling Procedural and Declarative Knowledge of Building Types in Architectural Design, *PhD thesis, Eindhoven University of Technology*, 1997.

[3] H.H. Achten, Generic representations - Knowledge representation for architectural design, *Journal of Architectural Management (13)*, 1997.

[4] M.F.Th. Bax, The Design of a Generic Grid, *Proc. of International Design Participation Conference*, 1985.

[5] R.E. Bellman, On a routing problem, *Quarterly of Applied Mathematics 16(1)*, p. 87-90, 1957.

[6] E.A. Bier, M.C. Stone, Snap-dragging, *Proc. of SIGGRAPH'86*, 1986.

[7] Philip T. Cox and Trevor J. Smedley, LSD:A Logic-Based Visual Language for Designing Structured Objects, *Journal of Visual Languages and Computing*, v.9(5) , p. 509-534, 1998.

[8] O. Esteban, S. Chatty, P. Palanque, Whizz'Ed: a visual environment for building highly interactive software, *Proc. of Interact'95*, p. 121-126, 1995.

[9] M. Gleicher, Integrating Constraints and Direct Manipulation, *Symposium on Interactive 3D Graphics*, p. 171-174, 1992.

[10] M. Gleicher, A Graphics Toolkit Based on Differential Constraints, *ACM: Symposium on User Interface Software and Technology*, p. 109-120, 1993.

[11] T. Igarashi, S. Kawachiya, H. Tanaka, S. Matsuoka, Pegasus: A Drawing System for Rapid Geometric Design, *ACM Symposium on User Interface Software and Technology*, p. 105-114, 1997.

[12] J. A. Landay, B. A. Myers, Interactive sketching for the early stages of user interface design, *Proc. of ACM CHI '95 Conference on Human Factors in Computing Systems*, p. 43-50, 1995.

[13] J.M. Maloney, A. Borning, B.N. Freeman-Benson, Constraint Technology for User-Interface Construction in ThingLab II, *Proc. of OOPSLA'89*, p. 381-388}, 1989.

[14] E.M. Moore, The shortest path through a maze, *International Symposium on the Theory of Switching*, p. 285-292, 1959.

[15] B.A. Myers, Garnet: Comprehensive Support for Graphical, Highly-Interactive User Interfaces, *IEEE Computer*, v. 23 (11), p. 71-85, 1990.

[16] S.S. Pranovich, J.J. van Wijk, C.W.A.M. van Overveld, The Kite geometry manipulator, *Extened abstracts CHI2002*, p. 764-765, 2002.

[17] S. Pranovich, H. Achten, J.J. van Wijk, Towards an architectural design system based on generic representations, *Proc. of Artificial Intelligence in design'02*, p. 153-164, 2002.

[18] S. Pranovich and J.J. van Wijk, A design system based on Architectural representations, *To appear in:Interact'03*, 2003.

[19] T. Sezgin, T. Stahovich, R. Davis, Sketch based interfaces: Early processing for sketch understanding, *Proc. of Perceptive User Interfaces Workshop'01*, 2001

[20] SmartSketch, 2003, *http://www.intergraph.com/smartsketch*

[21] P.S. Strauss, R. Carey, An object-oriented 3d graphics toolkit, *Proc. of SIGGRAPH '92*, p. 341-349, 1992.

[22] I.E. Sutherland, Sketchpad: a manmachine graphical communication system, *AFIPS Spring Joint Computer Conference*, p. 329-346, 1963.

[23] P.A. Szekely, B.A. Myers, A user interface toolkit based on graphical objects and constraints, *Proc. of OOPSLA'88*, p. 36-45, 1988.

[24] R.H. Tyson, S.E. Hudson, G.L. Newell, Inegrating gesture and snapping into a user interface toolkit, Proc. of *SIGGRAPH'90: Symposium on User Interface Software and Technologies*, p. 112-121, 1990.

[25] J.M. Vlissides, M.A. Linton, Unidraw: A framework for building domain-specific graphical editors, *Proc. of SIGGRAPH'89: Symposium on User Interface Software and Technologies*, 1989.