Specifying Generic Depictions of Language Constructs for 3D Visual Languages

Jan Wolter
University of Paderborn
Department of Computer Science
Fürstenallee 11, 33102 Paderborn, Germany
jan.wolter@uni-paderborn.de

Abstract—Several modeling domains make use of three-dimensional representations, e.g., the “ball-and-stick” models of molecules. Our generator framework DEVil3D supports the design and implementation of visual 3D languages for such modeling purposes. The front-end of a language implementation generated by DEVil3D is a dedicated 3D graphical structure editor, which is used to construct programs in that domain. DEVil3D supports the language designer to describe the visual appearance of the constructs of the particular language in terms of generic 3D depictions. Their parameters specify where substructures are embedded, and how the graphic adapts to space requirements of nested constructs. This paper motivates 3D visual languages and illustrates the specification process for 3D languages using DEVil3D with focus on generic depictions.

Keywords—three-dimensional depictions, visual languages, visual programming, automated generation.

I. INTRODUCTION

Visual languages are particularly beneficial for domain-specific applications, since they support graphical metaphors of their domain. Examples are LabVIEW [1] and the well-known UML. Both languages use two-dimensional representations, e.g., boxes and lines connecting them, in order to visualize dataflow or dependences. But for some domains, using the third dimension is advantageous or necessary: Inherently three-dimensional languages that make use of real-world objects as architecture-like modeling domains can be represented without loss of information only in 3D. The “ball-and-stick” models of molecules visualize atoms as balls and bonds between them as sticks. The arrangement of the atoms in 3D space is the result of the electron cloud repulsion and therefore the arrangement of language constructs to one another is inherently 3D.

Another argument in favor of 3D languages is to assign a semantic meaning to each dimension. A good example is the web-based 3D editor ToneCraft [2] that lets the user build music in 3D by composing boxes into a matrix-like area: the y-axis represents the pitch of a tone, the x-axis represents the time and the z-axis makes it possible to layer sounds. Even Petri Nets can benefit from a three-dimensional representation, for example, Petri Nets modeling different aspects—such as control flow and data flow—that have connections to one another. The representation of such Petri Nets in 2D is confusingly complex due to crossing edges and confound aspects. This can be solved by laying out the Petri Net in three dimensions, where each aspect is represented on a different plane, which can be stacked together to one Petri Net [3].

Moreover, the third dimension can be used to overcome limitations in 2D arrangements. For example, in some cases the 2D representation of UML diagrams is not efficient enough and can be extended to 3D, e.g., to overcome the problem of intersecting edges in sequence diagrams. Alternatively, the third dimension can be used to focus on specific classes of interest in class diagrams [4].

The AgentCubes system [5], the successor of AgentSheets, makes use of three-dimensional objects to allow children to create interactive three-dimensional games. The key challenges of AgentCubes are intuitive mechanisms to create 3D objects incrementally, including subsequent programming and animation aspects.

Our approach relies on a tool that allows a language designer to specify complete 3D language implementations including a 3D structure editor as a front-end and a transformation into textual code as its back-end. Therefore, we are developing the generator framework DEVil3D (Development Environment for Visual Languages in 3D) that accomplishes this task. One central part of developing visual languages is the definition of the visual appearance of language constructs. The above mentioned 3D languages and programming environments consist of objects with different 3D shapes. Some languages—as molecular models, ToneCraft, or Petri Nets—use relatively simple shapes like cubes, spheres, or cylinders. But the 3D scenes composed with AgentCubes consist of more complex shapes, visualizing real world objects. DEVil3D provides so called generic depictions that accomplish the task of representing language constructs ranging from simple to more complex shapes. Such depictions have parameters, which specify where substructures are embedded, and how the graphic adapts to space requirements of nested constructs.

Fig. 1 shows an exemplary language construct consisting of a blue box and a green sphere. The sphere contains a text label and the box is able to embed further language constructs visualized as red boxes. These language constructs are laid out as a list, which grows along the x-axis. When new constructs are inserted, the box has to adapt its space requirements to the needs of the nested list and the sphere has to move right.
The structure editors generated by DEViL3D are usable with a classical mouse and comprise techniques for navigation, interaction, and layout. For navigation purposes, each editor comprises a first-person-view camera, which lets the user navigate inside the scene and explore it. The layout and interaction tasks are encapsulated in visual patterns. The language designer does not need to implement such functionality. The assignment of visual patterns to language constructs is generally sufficient. Interaction tasks are automatically tailored to the needs of the representation of a visual pattern. Inserting language constructs is triggered by so-called insertion contexts and the modification of already inserted objects is provided by dedicated widgets that perform tasks like translating, scaling, or rotating.

### III. GENERIC DEPICTIONS

Generic depictions describe the visual appearance of constructs of a particular language in terms of generic 3D graphics. From a formal point of view generic depictions are an abstract concept that can be described by a quadruple of graphical primitives, representation properties, containers, and stretch intervals: \( D = (P, R, C, I) \). A set of graphical primitives \( P \) determine the graphical representation of a language construct: \( P = \text{Box} \cup \text{Sphere} \cup \text{Cone} \cup \text{Cylinder} \cup \text{Arrow} \cup \text{Line} \cup \text{Quad} \cup \text{Torus} \cup \text{3DModel} \cup \text{Text} \). 3D models, which can be created by modeling frameworks like Blender [10], allow to integrate objects with more complex shapes. The representation properties \( R \) describe materials, e.g., color or texture definitions, that can be mapped to graphical primitives \( P \). A set of containers \( C \) is responsible to embed nested objects of arbitrary size, when the generic depiction is instantiated. Each container needs a unique name. The specification of layout properties is managed by a set of stretch intervals \( I \). Such intervals determine, which part of a container grows, when the size of nested objects exceeds the container’s size.

The best way to build visual constructs is to do it visually. Such an approach ensures a close mapping between the domain world and the notation (according to the cognitive dimension

![Image 319x78 to 571x289](image)

**Fig. 2.** Specification process with DEViL3D.

**II. DEViL3D**

The generator system DEViL3D [6] combines approved concepts of the predecessor system DEViL [7], [8] and new aspects necessary to construct three-dimensional programs. These allow to generate 3D structure editors, supporting the direct manipulation paradigm [9]. Fig. 2 visualizes the specification process. The language designer has to specify the abstract structure, visual representations, and code generation. The abstract structure describes the language constructs and how they are connected, without defining a concrete representation. The specification of the visual representation is based on attribute grammars, which are translated into computations of the graphical representation and arrange objects in 3D space. The language designer can define a set of code generators that transform the 3D program into different textual representations. DEViL3D gets these specifications as input and generates a language processor which has a dedicated 3D graphical structure editor as its front-end. Domain experts use such editors to construct three-dimensional programs of their domain, e.g., molecular models.

For the specification of the visual representation visual patterns play an important role because they encapsulate common representation arrangements like three-dimensional sets, lists, line connections, or cone-trees. The language designer chooses visual patterns from a library and assigns them to symbols of the grammar in a declarative way. The generic depictions, which define the visual representation of a language construct, are referenced by visual patterns. Fig. 2 shows exemplary depictions necessary to specify an editor for molecular models. Such depictions—basically representing atoms and bonds—play the role of building blocks from which molecules are be constructed.

![Image 133x454 to 150x474](image)

**Fig. 3.** 3D editor for generic depictions.
closeness of mapping [11]). Fig. 3 shows a screenshot of the generic depiction editor that provides a 3D canvas in which the language designer visually composes a set of generic depictions for some language constructs. We have developed the editor by pursuing a bootstrapping approach, because it is specified with DEViL3D. The depiction shown in the figure consists of two graphical primitives in form of a box and a sphere, two containers named c1 and c2, and four stretch intervals. One container is located inside the box, the other one in the sphere. Each stretch interval is responsible for one dimension and is located on light red colored coordinate axes. The abstract visual program in Fig. 1 uses the generic depiction specified in Fig. 3.

Containers constitute the interface of generic depictions. If a language designer wants to change the visual representation of a language construct, two generic depictions can replace each other, if they coincide according to the number and the naming of their containers. A generic depiction has to fulfill requirements to be semantically correct. Each container must be covered by at least one stretch interval in each dimension. Otherwise, it is not clear how to respond to increasing size requirements of nested constructs. Any two stretch intervals must not overlap, so that in all cases a uniquely determined interval is responsible to stretch the container. The generic depictions editor is able to indicate violations of these requirements.

Language constructs of 3D visual programs—which support embedded substructures—have to adopt their size according to the requirements of these substructures. Such nested structures require the specification of containers and stretch intervals. In general, the embedded constructs can be laid out according to any visual pattern. In the example of Fig. 1 the list pattern is used for the constructs inside the blue box.

By inserting more constructs into the list, the container’s size could become insufficient. To adapt the size, an algorithm automatically stretches the container linearly. The algorithm operates on one depiction and iterates over all containers for each spatial dimension. If the preferred size, determined by the nested constructs, exceeds the actual container size, all stretch intervals that intersect the container are computed. The intervals of a container that are covered by a stretch interval will be stretched linearly. For this purpose, each container must be covered by at least one stretch interval.

Fig. 4 shows a schematic sketch of the generic depiction of Fig. 3 reduced to the x-axis to illustrate the stretch algorithm for one dimension. Initially the actual size of container c1 is a, but the embedded constructs need more space, so the preferred size is a + b. Hence, container c1 must be stretched to reach the new size. The stretch process behaves as if the containers and the primitives were printed onto elastic rubber and the start and end position of the stretch intervals were handles. To stretch the container and the primitives, the algorithm “pulls” the margins of the interval. Areas that are not covered by stretch intervals are not be stretched. In particular, the distance of two such points must remain the same after the transformation. Hence, the distance d between the box and the sphere is the same after application of the stretch algorithm and the sphere with container c2 inside has simply to move right.

This algorithm is encapsulated in DEViL3D and therefore language designers do not need to care about the dynamic adaptation of generic depictions, because DEViL3D automatically provides it. From an editor user’s point of view, the algorithm provides a natural behavior when new language elements are inserted.

IV. RANGE OF APPLICATION

To demonstrate that our approach of specifying generic depictions is feasible for a wide range of three-dimensional languages, we present further two 3D languages and focus on the specification of generic depictions for these languages.

Petri Nets can benefit from a three-dimensional representation. Rölke [3] presented the idea of 3D Petri Nets without indicating a 3D editor that allows to construct effectively in 3D. By using DEViL3D, we have specified such an editor. To realize the visual representations for Petri Nets, five generic depictions are needed. One depiction consists of a container that embeds the whole Petri Net and adjusts its size according to the stretch algorithm when new objects are inserted. Then there are generic depictions for transitions (visualized as boxes), places, tokens (both visualized as spheres), and arrows. The depiction for places consists of a container that is located inside the sphere and can accommodate tokens. Using the generated Petri Net editor (right-hand side in Fig. 5(a)), we have constructed a Petri Net, which has been taken from Rölke’s paper. The Petri Net is partitioned in two planes: On the lower plane the control flow of the Petri Net is located and the place on the upper plane is responsible for the data-flow. The first transition writes data to this place that other transitions read.

Inspired by the web-based 3D editor ToneCraft, we have specified a Music in Space editor with DEViL3D. The generated editor (see Fig. 5(b)) lets the user insert distinctively colored boxes representing music instruments in a matrix like area. From such a 3D program a music string according to the JFugue music programming API [12] can be generated and played. One depiction is needed, which provides a container that embeds the whole program, which is constructed according to the matrix pattern. For each instrument a depiction representing a colored box is needed.

V. RELATED WORK

The basic idea of containers and stretch intervals—as used in our three-dimensional generic depictions—goes back to the VPE system [13] that generates 2D visual language editors. Furthermore, the idea is successfully used in the generator system DEViL [7], which generates 2D language editors, too.

The idea for three-dimensional languages goes back to a publication of Glinert [14]. Najork [15] developed the first 3D language Cube, which is semantically similar to Prolog and makes use of the data-flow paradigm. In this language all language constructs are represented by a cube. The hierarchical
nesting of constructs is an inherent concept of the Cube language. In the context of generator systems for visual languages, Minas has supervised an exploration [16] of 3D languages in the context of his DiaGen/DiaMeta [17] frameworks.

The work of Chung et al. [18] exactly addresses the topic of our paper. They have developed a tool called 3DComposer that is related to our editor for generic depictions. 3DComposer is a tool for specifying so called 3Dvixels, which are used as building blocks for 3D applications. Such 3Dvixels can be generally used in 3D applications, which include 3D languages and also visualization tools. The usage in different applications is possible by generating reusable software components in form of JavaBeans. The construction of exemplary 3D programs is being done directly in 3DComposer by the end user. 3DComposer is not part of a generator framework, which would distinguish between language designer and language user. Hence, 3DComposer does not need concepts such as containers or stretch intervals.

The AgentCubes [5] system uses a mechanism to construct language object representations by using a so called Inflatable Icons Editor. It allows users to quickly draft 3D objects by drawing 2D images and turn them step by step into 3D models by using a diffusion-based inflation technique. In 3D games, which are specified with AgentCubes, these models are mostly located on the ground plane. Hence, the flat bottom side resulting from the inflation approach is not a concern.

VI. CONCLUSION

We have described the process of specifying generic depictions for 3D visual languages with the generator system DEVil3D. The language designer specifies the depictions by using an editor that was also generated with DEVil3D in a bootstrapping approach. A particularly important part of generic depictions are containers, because they can embed nested constructs. A language designer can add a container to a depiction, but does not need to care about its dynamic behavior, because DEVil3D automatically provides stretch mechanisms. The generic depictions editor is able to specify depictions for a wide range of 3D languages covering languages as Petri Nets or molecular models with rather simple visual representations but also languages, which consist of real-world objects that have more advanced visual representations.

ACKNOWLEDGMENTS

The author gratefully acknowledges the support of the German Research Foundation (Deutsche Forschungsgemeinschaft – DFG); contract no. KA 537/6-1.

REFERENCES