Visual Patterns in the VLEli System

Matthias T. Jung, Uwe Kastens, Christian Schindler, and Carsten Schmidt

University of Paderborn, Fürstenallee 11, 33102 Paderborn, Germany,
{mjung, uwe, tolkien, cschmidt}@upb.de

1 Characterization of the Toolset

Visual languages have an important role in modelling systems, in specification of software, and in specific application domains. A processor for a visual language consists of a graphical frontend attached to phases that analyse and transform the visual programs. Hence, the construction of a visual language processor requires a wide range of conceptual and technical knowledge: from issues of visual design and graphical implementation to aspects of analysis and transformation for languages in general. We present a powerful toolset that incorporates such knowledge up to a high specification level. Visual editors are generated by identifying certain patterns in the language structure and selecting a visual representation from a set of precoined solutions. Visual programs are represented by attributed abstract trees. Hence, further phases of processing the visual programs can be generated by state-of-the-art tools for language implementation. We demonstrate that ambitious visual languages can be implemented with reasonable small effort and with rather limited technical knowledge. The approach is suitable for a large variety of visual language styles.

The main concepts of our approach are described by the three layers shown in Fig. 1: The tool VLEli generates visual structure editors from specifications. It is built on top of tools for graphical support (Tcl/Tk and Parcon) and for language implementation in general (Eli). The topmost layer contains variants of visual patterns, each of which encapsulates the implementation of visual language elements in terms of specifications for VLEli.

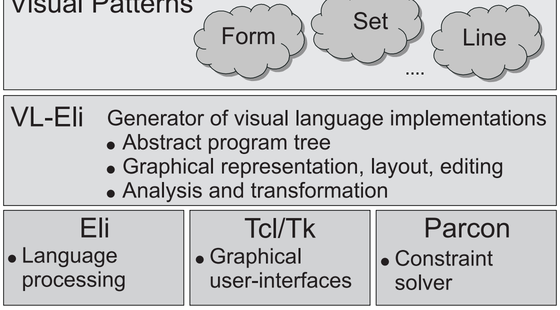

Visual Patterns
Form
Set
Line
....
VL-Eli
Generator of visual language implementations
• Abstract program tree
• Graphical representation, layout, editing
• Analysis and transformation
Eli
• Language processing
Tcl/Tk
• Graphical user-interfaces
Parcon
• Constraint solver

Fig. 1. Multi-Layered Specification

The central tool VLEli is conceptually based on attribute grammars: It generates visual editors, which operate on abstract program trees as their central data structures. They perform computations during tree walks as specified for the particular language. The computations create and modify the graphical representation of the program via the interface to the graphical support tools in the bottom layer. The abstract program tree and the attribute grammar method also enable the direct connection to many tools in the Eli system which solve a wide range of general language implementation tasks.

The topmost layer of our approach contains a set of precoined descriptions of various graphical language elements in terms of the underlying layers. They are used to compose a visual language implementation without writing lower level specifications explicitly. Each such description is considered to be a variant of an abstraction that we call a *visual pattern*. It comprises the common properties of visual language elements with respect to their abstract structure, the interaction operations needed for them, and their visual concept. For example the `List` pattern represents an ordered sequence, allows element insertion and deletion, and visualizes the structure by drawing the elements side by side in a row. A language designer instantiates such a pattern variant and associates its components with certain constructs of the tree grammar.

2 Tool Demonstration

The demonstration of our tool set addresses three topics

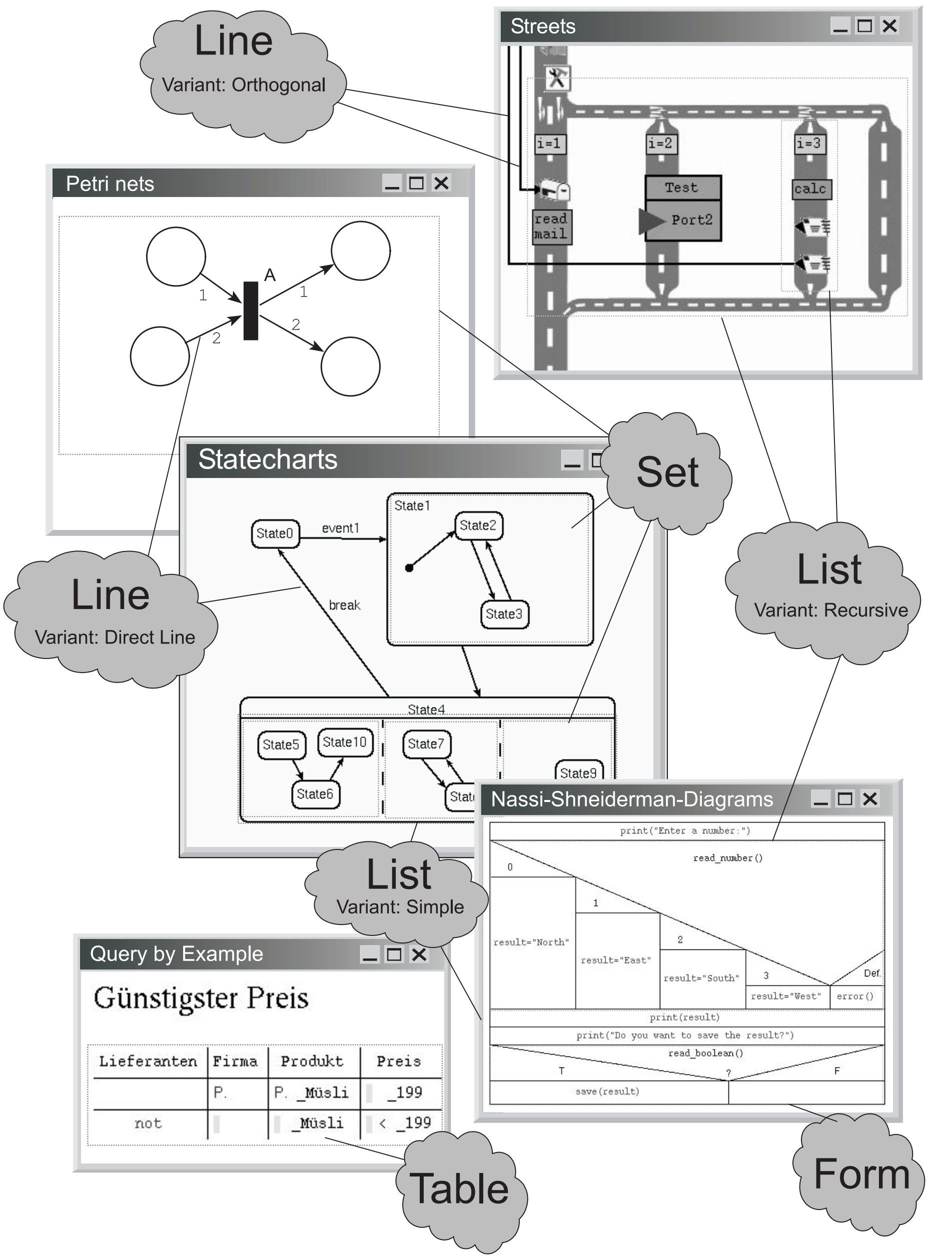
1. Method: Visual patterns used in attribute grammar specifications.
2. Generated Products: Properties of the structure editors.
3. Variety of visual language styles.

2.1 Method

A visual pattern represents an abstract concept, like the visualization of a `set` and its elements. The notion of visual patterns provides also a means of reusable implementation specification: For each visual pattern concrete variants are described in terms of composable specifications for VLEli: Such a pattern variant encapsulates operations that are needed for a visual structure editor to implement a certain graphical representation of the structural abstraction. That are operations, which

– draw graphical components, e.g. the oval of a set,
– layout components of the structure, e.g. the elements of a set within its oval,
– provide facilities for user interactions, e.g. insert and delete elements.

These operations are expressed in terms of attribute grammar computations and composed to computational roles. The language designer selects such roles and associates them to the abstract syntax symbols. Fig. 2 gives an idea of various applications of patterns in five visual languages.



Lieferanten	Firma	Produkt	Preis
	P.	P. _Müsli	_199
not		_Müsli	< _199

Fig. 2. Visual Pattern Applications in Visual Languages

2.2 Generated Products

The generated language processor consists of a graphical structure editor which builds an abstract program tree, and of analysis and translation phases which operate on that tree. The demonstration shows that the generated editors provide interactive graphical operations of high quality. The editors provide direct manipulation: The user can drag new language elements from toolbars and drop them where appropiate. The elements can be selected, and moved or deleted as required. Editing is assisted by highlighting the nearest possible location for moved language elements. Furthermore, all pattern variants provide specialized layout concepts. Thus a compact layout for the Nassi-Shneiderman-Diagrams is achieved as easily as the free non-overlapping layout of the statechart diagrams. For the latter, a constraint-solver is applied which adjusts the sizes and positions of language elements as necessary.

The implementation of these facilities comes almost for free by using variants of visual patterns.

2.3 Variety of Visual Language Styles

We checked the usability of our approach by implementing five visual languages: UML statechart diagrams and Petri-nets are examples of languages, that mainly apply the set-pattern and require a constraint-based layout to implement non-overlapping elements. Nassi-Shneiderman diagrams and Query-by-example are visual languages that use hierarchical structures visualized using automatic layout techniques. Query-by-example needs complex editing operations, e.g. to instantiate a database table. Streets is a more complex visual language for modeling parallel programs. Its implementation comprises a visual structure editor and phases for analysis and code-generation, which are generated from specifications for the Eli system.

The languages impose diverse requirements for editing and visualization features. They are implemented completely by visual patterns, in particular by variants of `Form`, `List`, `FormList`, `Table`, `Set` and `Line`. In the tool demonstration a selection of these generated products is shown to demonstrate the variety of visual language style that is addressed by our tool approach.