Computer Languages, Systems & Structures 54 (2018) 20–38

Contents lists available at ScienceDirect

Computer Languages, Systems & Structures

journal homepage: www.elsevier.com/locate/cl

Using the local context for the definition and implementation of visual languages

Gennaro Costagliola, Mattia De Rosa*, Vittorio Fuccella

Department of Informatics, University of Salerno, Via Giovanni Paolo II, Fisciano 84084, SA, Italy

ARTICLE INFO

Article history:
Received 13 February 2018
Revised 28 March 2018
Accepted 3 April 2018
Available online 12 April 2018

Keywords:
Local context
Visual languages

ABSTRACT

In general, visual languages need to be simple in order to be easily used and understood. As a result, many of them have simple constructs that can be defined by simply describing local constraints on the constituent elements. Based on this assumption, in a previous research, we developed a local context methodology for the specification of the syntax of simple visual languages such as flowcharts, entity-relationship diagrams, use-case diagrams. In this paper, we extend the methodology by defining a new technique for a local context based semantic translation of a visual language. The technique uses XPath-like expressions, called SGPath, together with a data flow model of execution. As for the case of local syntax checks, attributes and rules to calculate them are defined for each element of the language. For a given element in the abstract sentence graph, the SGPath expressions are used to gather values from its neighbors in order to allow the rules to calculate its semantic attributes. The new methodology has been implemented in the tool LoCoMoTiVe and been tested on visual languages such as entity-relationship diagrams, flowcharts, trees.

1. Introduction

Since their introduction, visual languages have been defined as part of systems which enhance communication by the use of visual elements. Diagrams, maps, images and pictures are examples of visual sentences and they are used as representations of mental concepts that need spatial contexts in order to be described naturally. Their role is to facilitate the communication among people since, when adequate, visual communication is more direct and of immediate understanding compared to the verbal or text type of communication. Because of this, the use of visual languages can be found almost in any context ranging from art to engineering.

It must be admitted, however, that badly designed visual languages can lead to visual formulations that are very difficult to compose and interpret. In this case, they fail their main reason of being.

In general, from a syntactic point of view, this occurs when a language presents many syntactic rules binding elements which can be very far in a sentence. As an example, in the case of textual programming languages, one might consider the matching parenthesis in languages such as C or Java.

To overcome this problem, block visual languages such as Scratch have eliminated syntax dependencies between far language elements, by adding shape information to each element and reducing the composition of a program to the creation of a puzzle. The syntactic rule here is very simple: “a visual program is syntactically correct if and only if each tile well

* Corresponding author.
E-mail address: matderosa@unisa.it (M. De Rosa).

https://doi.org/10.1016/j.cl.2018.04.002
1477-8424/

interlocks with its neighbors". In this case, the validity of the local shape constraints on each tile guarantees the correctness of the whole visual program independently on how many elements it is made of.

This is also why block languages are now very popular and extensively used in education for teaching introductory programming to non-experts, [1].

In previous research, [2,3], we have shown that not only block languages but also many other very well known and used programming visual languages such as unstructured flowcharts, data flow languages and entity-relationship diagrams can be syntactically specified by mostly setting local constraints on the language elements. This has allowed us to avoid writing complex grammars for these types of languages by greatly simplifying the design of a visual language from a syntactic point of view.

In particular, our methodology, known as *local context-based visual language specification*, only requires the language designer to define the *local context* of each symbol of the language. The local context is seen as the interface that a symbol exposes to the rest of the sentence and consists of a set of attributes defining the local constraints that need to be considered for the correct use of the symbol.

In this paper, we continue our previous work by facing the semantic translation of a visual language based on the local context. We propose a method based on XPath-like expressions, called SGPath expressions, and a data flow model of execution in order to define the semantic translation rules for the language by specifying rules for each single language element as opposed to defining semantic rules for complete phrases.

In particular, for a given node of the abstract syntax graph returned by the syntactic phase, the SGPath expressions are exploited to gather values from its neighbors to be used in the translation. XPath-like languages have shown to be very well-suited for navigation through graphs when querying data held in the nodes, [4].

The new methodology has been implemented as part of the tool LoCoMoTiVe [3] and tested in two different case studies, aimed at generating code from two different visual languages: C-like code from flowcharts and SQL code from entity-relationship diagrams.

The paper is organized as follows: the next section refers to the related work; Section 3 recalls the main concepts of the local context specification of visual languages; Section 4 describes the SGPath specification; Section 5 describes the local context-based semantic definition (LCDS) and its analysis algorithm; Section 6 describes the LoCoMoTiVE tool that implements the methodology; finally Section 7 concludes the paper with final remarks and a brief discussion on future work.

2. Related work

Several strategies to model diagrams as visual language sentences have been conceived in the past. Diagrams have been represented either as sets of attributed symbols, with the "position" of the symbol in the sentence represented through typed attributes (*attribute-based approach*) [5], or as sets of relations on symbols (*relation-based approach*) [6]. Despite the fact that the two approaches appear to be different, they both consider a visual sentence as a set of symbols and relations among them. This structure can be represented as a spatial-relationship graph [7] built by adding a node for each graphical symbol and adding an edge for each spatial relationship between symbols (nodes).

In the attribute-based approach, the relations are derived by matching attribute values, while in the relation-based approach the relations are explicitly named.

Based on these representations, various formalisms have been suggested to represent the visual language syntax, each one associated to ad-hoc scanning and parsing techniques: (Extended) Positional Grammars [8], Constrained Set Grammars [9], Relational Grammars [10], Reserved Graph Grammars [11] to name some (other approaches and details can be found in [12] and [13]). These visual grammars are defined, in general, by specifying an alphabet of graphical symbols together with their "visual" appearance, a set of spatial or topological relationships, and a set of grammar rules, usually in a context-free like format even though their descriptive power is mostly context sensitive.

Various software tools for visual language prototyping have been designed and implemented based on the different types of visual grammar formalisms. These include: VLDesk that is based on positional grammars [14], GenGed that is based on Hypergrah grammars [15], Penguin that is based on constraint logic based grammars[16], DiaGen [17] that is based on Hypergrah grammars, VisPro that is based on Reserved Graph Grammars [18], AToM3 [19], VL-Eli [20] and its improvement DEViL [21]. Most of the systems include the possibility to add visual language semantics specifications but these are all specified at the level of grammar productions (see [22] as an example). DEViL also provides an object-oriented like domain specific language and a library of "visual patterns" that can be used to easily specify common concepts such as lists, sets, tables, trees, etc. However, our work goes a step further by completely removing the grammar specification.

Even though context-free like rules are well known, visual grammars are usually difficult to define and read. This may be the reason why there has been not much success for these techniques in real-world applications. Many visual languages used today are syntactically simple languages that focus on the basic graphical elements and their expressive power, and therefore they do not need complex grammar rules to be specified. Because of this, we feel that our methodology allows a simpler specification for many of them, making it less demanding to define and quickly prototype visual languages with their semantic translation.