Fast prototyping of visual languages using local context-based specifications

Gennaro Costagliola, Mattia De Rosa, Vittorio Fuccella
Dipartimento di Informatica, University of Salerno
Via Giovanni Paolo II, 84084 Fisciano (SA), Italy
{gencos, matderosa, vfuccella}@unisa.it

Abstract

In this paper we present a framework for the fast prototyping of visual languages exploiting their local context based specification.

In previous research, the local context specification has been used as a weak form of syntactic specification to define when visual sentences are well formed. In this paper we add new features to the local context specification in order to fully specify complex visual languages such as entity-relationships, use case and class diagrams. One of the advantages of this technique is its simplicity of application and, to show this, we present a tool implementing our framework. Moreover, we describe a user study aimed at evaluating the satisfaction and effectiveness of users when prototyping a visual language.

Keywords: *local context, visual languages, visual language syntax specifications.*

1 Introduction

Due to the ever growing evolution of graphical interfaces, visual languages are now a well established form of digital communication in many work and research environments. They are being used extensively by engineers, architects and others whenever there is the need to state and communicate ideas in a standardized way. This is traditionally happening, for example, with software engineering UML graphical notations but it is also catching on in other research fields, such as, for example, synthetic and system biology, [21, 16].

In the 1990s and the early 2000s, the formalization and implementation of visual languages have excited many researchers and many proposals have been defined. In particular, formalisms were defined mainly based on the extension of textual grammar concepts, such as attributed grammars [13, 17, 9, 23] and graph grammars [3, 14], and on metamodeling [4].

Lately, a new proposal for the specification and interpretation of diagrams from the only syntactic point of view has been given in [7]. This research is motivated by the need to reduce the complexity of the visual language syntax specification and originated while working on the recognition of sketch languages and on the difficulty of recognizing the symbols of the language. In order to disambiguate sketched symbols, the more knowledge is given on each symbol and on its possible interactions with the others, the better. The methodology, known as *local context-based visual language specification* requires the 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.

At first, this approach is, then, more lexical than syntactic: the idea is to provide a very detailed specification of the symbols of the language in order to reduce complexity when specifying the language at the sentence level. On the other hand, due to the easy-to-use and intuitiveness requirements, many practical visual languages have a simple syntax that can be completely captured by the local context specification as defined in [7]. To show this, the syntax of the binary trees and of a Turing complete subset of flowcharts have been completely specified through local context in [7].

When considered as a syntax specification, however, the simplicity of the approach is counterbalanced by its low expressiveness, especially with respect to the more powerful grammars formalisms.

In this paper, we define new features to be added to the original local context definition in order to push the expressiveness of the methodology and to allow the complete syntactic specification of complex visual languages such as entity-relationships, use case and class diagrams. We present the tool LoCoMoTiVE (Local Context-based Modeling of 2D Visual language Environments) implementing our framework which basically consists of a simple inter-

DOI reference number: 10.18293/VLSS2015-050

face allowing a user, in one screen, to define the symbols of the language and their local context. Moreover, we demonstrate the usability of the tool in a user study, in which participants are asked to define and test a visual language after a short introduction to the tool. Besides the participants' ability to define the visual language, we also administered a System Usability Scale (SUS) [5] questionnaire to evaluate their satisfaction with the tool.

The rest of the paper is organized as follows: the next section refers to related work; Section 3 gives the background information on the "local context-based visual language specification", sketches the three new features and presents a complete syntax specification for the use case diagrams; Section 4 is devoted to describe the tool; the experiment is presented in Section 5; some final remarks and a brief discussion on future work conclude the paper.

2 Related Work

In recent years many methods to model a diagram as a sentence of a visual language have been devised. A diagram has been represented either as a set of relations on symbols (the *relation-based approach*) [22] or as a set of attributed symbols with typed attributes representing the "position" of the symbol in the sentence (the *attribute-based approach*) [12]. Even though the two approaches appear to be different, they both consider a diagram (or visual sentence) as a set of symbols and relations among them or, in other words, a spatial-relationship graph [3] in which each node corresponds to a graphical symbol and each edge corresponds to the spatial relationship between the symbols.

Unlike the relation-based approach, where the relations are explicitly represented, in the attribute-based approach the relations must be derived by associating attribute values.

Based on these representations, several formalisms have been proposed to describe the visual language syntax, each associated to ad-hoc scanning and parsing techniques: Relational Grammars [23], Constrained Set Grammars [17], and (Extended) Positional Grammars [10]. (For a more extensive overview, see Marriott and Meyer [18] or Costagliola et al. [8].) In general, such visual grammars are specified by providing an alphabet of graphical symbols with their "physical" appearance, a set of spatial relationships generally defined on symbol position and attachment points/areas, and a set of grammar rules in context-free like format.

A large number of tools exist for prototyping visual languages. These are based on different types of visual grammar formalisms and include, among others, VLDesk [9], DiaGen [19], GenGed [2], Penguin [6], AToM3 [11], and VL-Eli [15].

Despite the context-free like rule format, visual grammars are not easy to define and read. This may explain why there has not been much success in transferring these techniques from research labs into real-world applications. We observe that several visual languages in use today are simple languages that focus on basic components and their expressive power. These languages do not need to be described with complex grammar rules. Our approach provides a simpler specification for many of them, making it easier to define and quickly prototype visual languages.

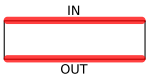
3 Local Context Specification of Visual Languages

According the to the attribute-based representation, a visual sentence is given by a set of symbols defined by a set of typed attaching points linked through connectors. In [7], the local context specification of a visual language consists in the definition of the sets of graphical elements (named *symbols*) and spatial relations (named *connectors*) composing the language and, for each of them, their attributes. These are:

- Name of the graphical symbol;
- Graphical aspect (for example, specified through an svg-like format);
- Maximum mumber of occurrences of the symbol in a sentence of the language;
- Attachment areas: an attachment area is a set of points (possibly one) through which symbols and connectors can be connected. For each area the following attributes have been identifed:
  - Name of the attachment area;
  - Position: the location of the attachment area on the symbol or connector;
  - Type: an attachment area of a symbol can be connected to an an attachment area of a connector only if they have the same type;
  - Local constraints: such as the max number of possible connections for an attachment area.

As a further and sentence level constraint, a local-context based specification may assume that the underlying spatial-relationship graph of any sentence of the specified language is connected.

As an example, Table 1 shows the attributes of the statement symbol STAT and the connector ARROW when considered as alphabetical elements of a particular set of flowcharts. The specification states that STAT has the graphical aspect of a rectangle; it has two attaching areas named IN and OUT corresponding to the upper and lower sides of the rectangle, respectively; it may occur zero or more times in a flowchart; the attachment area IN can only be connected to an attachment area of a connector with type *enter*, while the attachment area OUT can only be connected to an attachment area of a connector with type *exit*. Moreover, IN may participate in one or more connections, while OUT may participate in only one connection. As re-

Symbol name	Graphics	Symbol occurrences	Attachment areas		
			name	type	constraints
STAT		≥ 0	IN	enter	$connectNum \geq 1$
			OUT	exit	$connectNum = 1$

Connector name	Graphics	Attachment areas		
		name	type	constraints
ARROW		$HEAD$	enter	$connectNum = 1$
		$TAIL$	exit	$connectNum = 1$

Table 1: Attribute specification for the symbol STAT and the connector ARROW.

gards the connector ARROW, its graphical appearance is given by a line with a head and the attachment areas are located to the head (HEAD) and tail (TAIL) of the arrow itself. An arrow can be connected to only one symbol through its head and only one symbol through its tail. In [7], complete local context specifications for a particular set of Turing complete flowcharts and for binary trees are given to show how local context can be used to fully specify the syntax of visual languages.

3.1 New Local Context Features

In order to capture as much as possible of the syntax of complex languages other than flowcharts and binary trees and to keep simplicity, new local features need to be added to the original definition of local context. In particular, we introduce the possibility of

- defining *symbol level* constraints involving more than one attaching area of a symbol/connector as opposed to constraints on individual attaching areas;
- assigning multiple types to attaching areas;
- defining constraints to limit connector self loops.

These features allow us to give complete local context specifications of complex languages such as the entity relationship diagrams, class diagrams, and use case diagrams. In the following, we show the practical usefulness of this extension by referring to the local context specifications of the use case diagrams and the entity-relationship diagrams.

Symbol level constraints. Table 2 shows the binary version of the *relation* symbol of the well-known entity-relationship (E-R) graphical notation. Each vertex of the symbol has one attaching point (Up, Down, Left or Right) of type *enter*. In order to force its use as an E-R binary relation (as opposed to ternary) the constraints need to set the sum of all their connections to two, apart from requiring that the number of connections to each attaching point be at most one. In this case, the feature simplifies the specification by avoiding that a designer define all the possible ways a binary relation symbol can be attached to the other symbols.

Multiple types and Connector self loop constraints. Table 3 shows the complete specification of the use case graphical notation, while Figure 1 shows some examples of correct and incorrect sentences. In the table, the language symbols and connectors are specified in the first and second part while, for sake of completeness, the last row declares, if present, any requirement at sentence level. It can be noted that the attachment area *Border* of the symbol ACTOR (first table row) has two types GenA and AssOut. By considering the Connector part of the table, this means that an ACTOR can be connected through its border to both the head and tail of the GENERALIZATION_A connector (through GenA) and also to the attaching point *P1:P2* of the connector ASSOCIATION (through AssOut). Moreover, because of the constraint $numLoop = 0$, a GENERALIZATION_A connector cannot be connected to the border of an ACTOR with its head and tail, simultaneously.

In Figure 1, case (b) shows the correct use of connector GENERALIZATION_Uc, while case (d) shows the correct use of connector GENERALIZATION_A.

The use of these new features allow the language designer more flexibility in the definition of the language. However, multiple types must be carefully used when dealing with connectors with the same graphical aspect since they may introduce ambiguities in the language.

It is not difficult to see that Table 3 completely specifies the syntax of the use case graphical notation as presented in *http://agilemodeling.com/artifacts/useCaseDiagram.htm* but without the optional "System boundary boxes".

With respect to a grammar definition, the new specification is basically distributed on the language elements instead of being centralized.

As a final note, the selection of which language elements are symbols and which are connectors is completely left to the language designer. Moreover, connectors may not have a graphical representation (such as the relationships "touching", "to the left of").

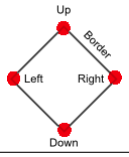
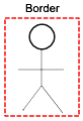
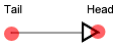
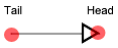
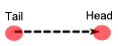
Symbol name	Graphics	Symbol occurrences	Attachment points name	type	constraints
BIN_REL		≥ 0	Up	ConA	$connectNum(X) \leq 1$ for $X = Up, Down, Left, Right \wedge (connectNum(Up) + connectNum(Down) + connectNum(Left) + connectNum(Right) = 2) \wedge (connectNum(Border) \geq 0)$
			$Left$	ConA	
			$Right$	ConA	
			$Down$	ConA	
			$Border$	ConB	

Table 2: ER binary relation specification.

Symbol name	Graphics	Symbol occurrences	Attachment points name	type	constraints
ACTOR		≥ 1	$Border$	GenA	$connectNum \geq 0 \wedge numLoop = 0$
				AssOut	$connectNum \geq 0$
USE_CASE		≥ 1	$Border$	GenUc	$connectNum \geq 0 \wedge numLoop = 0$
				AssIn	$connectNum \geq 0$
				Dep	$connectNum \geq 0 \wedge numLoop = 0$

Connector	Graphics	Attachment points name	type	constraints
ASSOCIATION		$P1{:}P2$	AssOut	$connectNum = 1$
		$P2{:}P1$	AssIn	$connectNum = 1$
GENERALIZATION_A		$Head$	GenA	$connectNum = 1$
		$Tail$	GenA	$connectNum = 1$
GENERALIZATION_UC		$Head$	GenUc	$connectNum = 1$
		$Tail$	GenUc	$connectNum = 1$
DEPENDENCY		$Head$	Dep	$connectNum = 1$
		$Tail$	Dep	$connectNum = 1$

Non local constraint
the spatial-relationship graph must be connected

Table 3: Use case diagrams language specifications.

4 The tool LoCoMoTiVE

The current implementation of the local context methodology includes the new set of constraints defined in the previous section and is composed of two different modules:

- LoCoModeler: the local context-based specification editor, and
- TiVe: a web-based visual language environment for editing and checking the correctness of the visual sentences.

4.1 LoCoModeler

The LoCoModeler module allows designers to create and edit visual language specifications based on local context, quickly and easily. Its output is the formal definition in XML format of the language that will be used during the disambiguation and the recognition of diagrams. Once the language designer has completed the specification, s/he can compile it into a web-based environment (the TiVE module) to allow users to draw sentences and verify their correctness. During language definition, this feature also allows the designer to check the correctness of the specification.

The main view of the LoCoModeller GUI is shown in Figure 2. Its main components are:

- A textbox containing the name of the language and a checkbox to enable/disable the option that diagrams must necessarily be connected;
- A table reporting the main information of symbols and connectors included in the language. It is possible to

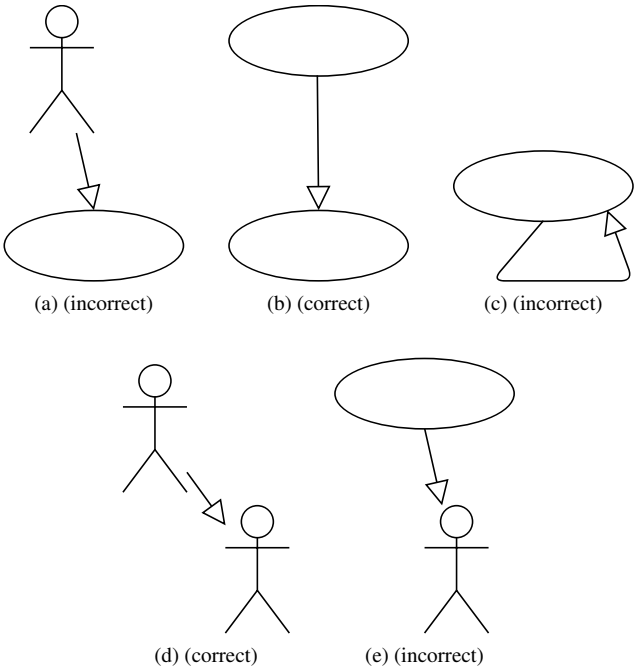


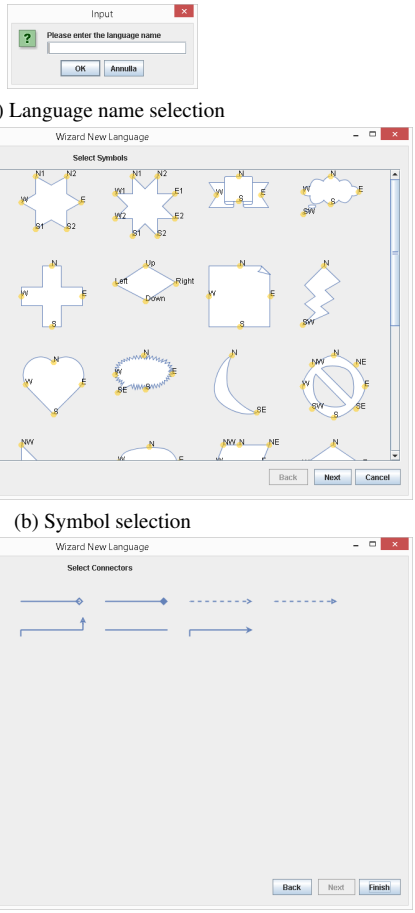
Figure 1: Simple instances of syntactically correct and incorrect use case diagrams.

interact with the widgets in the selected row to edit and/or delete it. The user can add new symbols or connectors by using the buttons below the table.

- A panel (on the right) showing a graphical preview of the symbol/connector selected in the table. It is possible to change the graphical representation of the symbol by using the button *Change*.
- A table (in the center) showing the information related to the selected symbol/connector. Each row specifies the attachment points and their constraints. It is possible to add new rows by using the buttons above the table;
- A textarea to specify the symbol/connector level constraints through C language-like expressions.

4.1.1 Wizard

A new language can be defined by using a wizard interface. Through a sequence of three different views, the user chooses the name of the language, its symbols and connectors (see Figure 3). Symbols and connectors are chosen from a hierarchical repository and their definition already includes some attachment points having default types/constraints. The user can choose whether to keep the default values or modify them.



(a) Language name selection

(b) Symbol selection

(c) Connector selection

Figure 3: Wizard.

4.2 TiVE: the visual language environment

Once the language is defined, the diagrams can be composed by using the symbols and the connectors defined in its specification. This can be done through the graphical editor TiVE, which is a web application enabling diagram composition directly in the web browser and which is created by and can also be launched from the LoCoModeler.

Figure 4 shows the environment. The central component is the working area where the diagrams are composed. The symbols and connectors used for diagram composition are displayed in the sidebar, which contains only those elements included in the definition of the language. An element can be selected and dragged in the central working area.

The upper toolbar provides shortcuts to features such as zoom manipulation, changing fonts, checking diagram correctness, etc.

The correctness of a diagram can be checked at any point of the diagram composition. The diagram in Figure 5 represents an ER diagram with two entities interconnected by

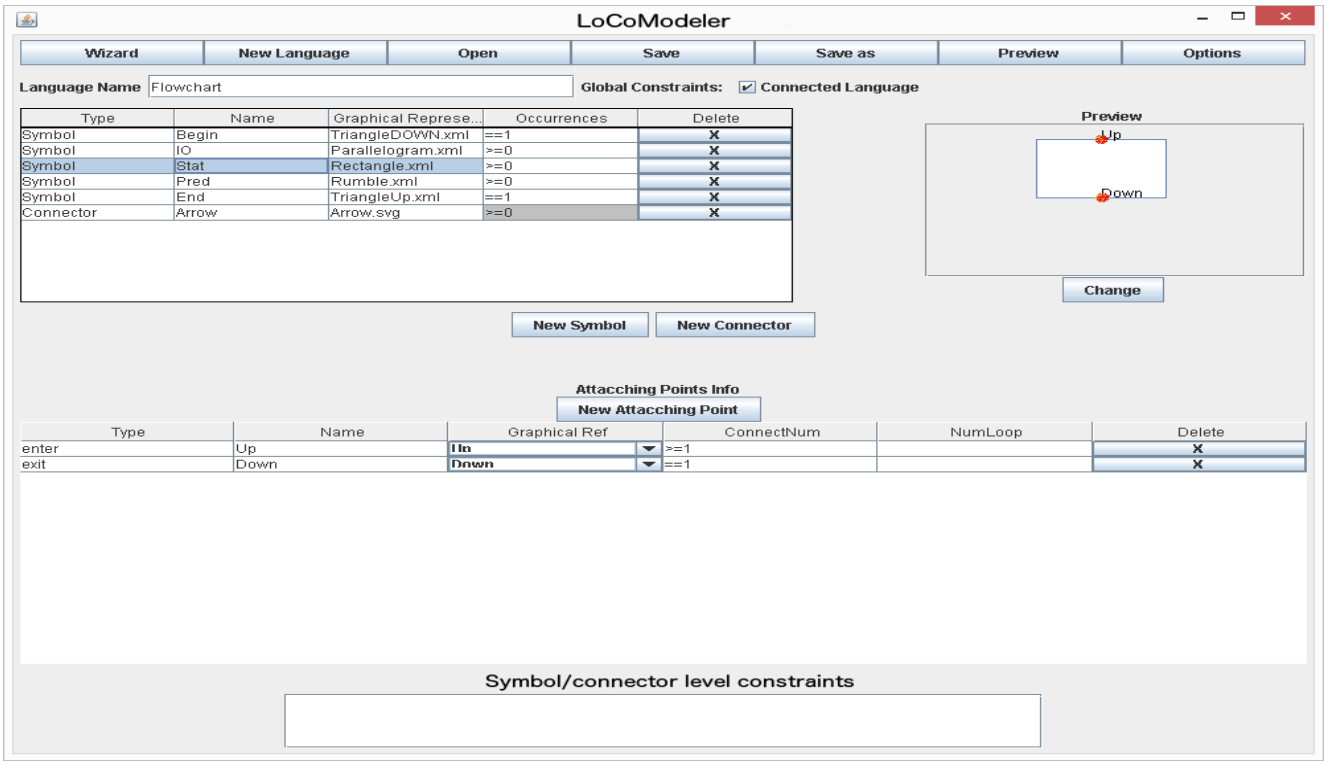


Figure 2: The Local Context-based Modeler.

a binary relation. The diagram is correct and, by launching the check, an alert reports the positive result of the verification. The diagram in Figure 6, instead, represents an incorrect ER diagram, as the relation (the diamond symbol) is connected to a single entity (the rectangle). This violates the local constraints defined for the relation symbol in the language. In fact, in this language, relations must be connected to two or three entities through the diamond's vertices.

4.3 Implementation

The LoCoModeler allows the user to produce the language specification in XML format. The specification is used during the removal of ambiguities and the recognition of symbols and connectors.

TiVE is based on Draw.io (https://www.jgraph.com), which is a free web application that allows users to create charts directly from the browser and integrates with Google Drive and Dropbox to store data. Draw.io is in turn based on the mxGraph library, which renders the diagram and stores its structure in the form of a graph, where symbols and connectors are its vertices. We modified the library to handle attaching points of symbols and connectors.

In a typical thin-client environment, mxGraph is divided into a client-side JavaScript library and a server side library in one of the two supported languages: .NET and Java. The

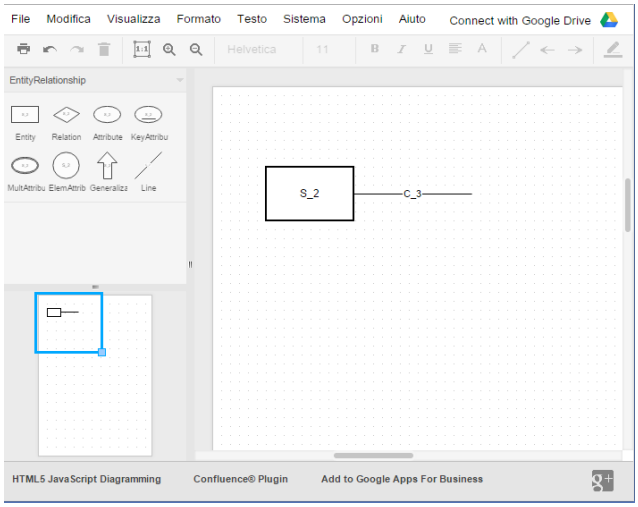


Figure 4: The TiVE home page.

JavaScript library is a part of a larger web application. The JavaScript code uses vector graphics to render the chart. The languages used are SVG (Scalable Vector Graphics) for standard browsers and VML (Vector Markup Language) for Internet Explorer.

An implementation of the tool can be downloaded at the address http://weblab.di.unisa.it/locomotive.

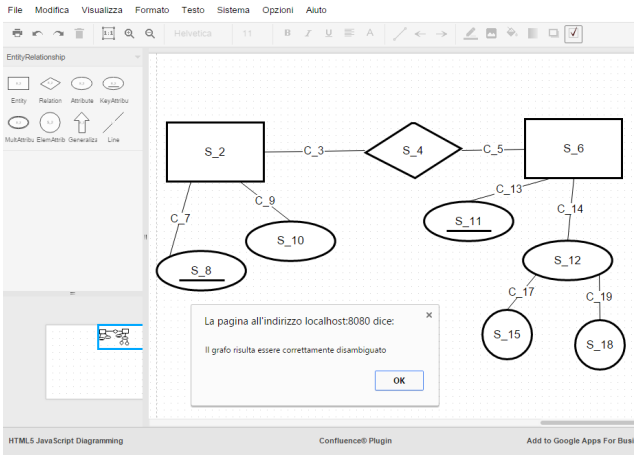


Figure 5: Successful diagram verification (no error found).

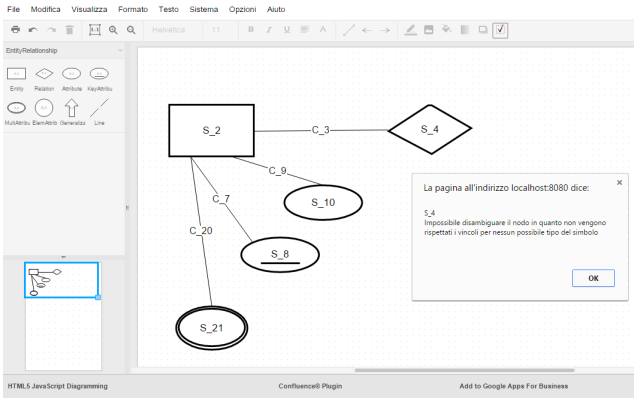


Figure 6: Failed diagram verification (one error found).

5 Evaluation

We ran a user study aimed at measuring users' capacity to define visual languages using our tool. Furthermore, we recorded the perceived usability of the system through a questionnaire.

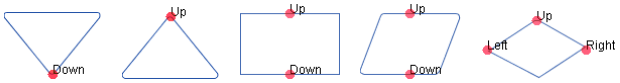
5.1 Participants

Our participants were six male and four female Italian university students in computer science (six master students and four phd students), aged between 22 and 45 ($M = 26.8$, $SD = 6.9$), with no previous experience with the system.

Participants were asked to evaluate, with a 5-point Likert scale, their prior knowledge in programming, diagrams, compilers, formal languages, flowchart and other visual languages. The average and standard deviations of the responses are reported in Table 4.

Knowledge	Avg.	St. dev.
Programming	4.40	0.70
Diagrams	3.40	0.84
Compilers	2.90	1.10
Formal languages	3.40	1.07
Flowchart	3.40	0.97
Other visual languages	2.80	1.14

Table 4: Participants prior knowledge evaluation with a 5-point Likert scale.



(a) BEGIN (b) END (c) STAT (d) IO (e) PRED

Figure 7: Flowchart symbols.

5.2 Apparatus

The experiment was executed on a *Dell Precision T5400* workstation equipped with an *Intel Xeon* CPU at 2.50 GHz running *Microsoft Windows 8.1* operating system, the *Java Run-Time Environment 7*, and the Firefox browser.

5.3 Procedure

Participants were asked, as a single task of the session, to define a visual language. In particular, they were asked to create a simplified version of flowcharts, as defined in [7], with the following features:

- The language only includes a small set of blocks (start, end, I/O, decision, processing - shown in Figure 7) and an arrow as a connector;
- The handling of text within blocks or arrows is not required;
- An arrow is always directed at the top of a block and comes out from its bottom;

Participants were asked to specify all the constraints necessary to ensure that only well formed flowcharts would pass the correctness check. Furthermore, they were required to carefully check they had defined the language correctly before submitting the task. The time limit for task completion was half an hour.

Before the experimental session, each participant had a brief tutorial phase where an operator (one of the authors) explained him/her the purpose and operation of the system and instructed him/her about the experimental procedure and the task. While showing the operation of the system, the operator also showed participants how to use our tool to define a simple visual language, in this case a simplified version of ER diagrams.

A post-test questionnaire in the form of System Usability Scale (SUS) [5] was administered to participants at the end of the experiment. SUS is composed of 10 statements to which participants assign a score indicating their strength of agreement in a 5-point scale. The final SUS score ranges from 0 to 100. Higher scores indicate better perceived usability. We also gathered some participants' freeform comments.

5.4 Results

Two participants out of ten completed the experiment defining the language perfectly, seven completed the experiment with minor inaccuracies in the language definition, while only one of them completed the experiment with major inaccuracies. Here, for minor inaccuracies, we mean small errors that allow user to compose at least one invalid diagram which however satisfied the user's language specification. Typical errors are inaccuracies in defining attachment points cardinality. The participant who committed major errors was unable to compose and correctly compile any diagram. The average task completion time was 25.5 minutes.

The responses given by participants to the statements in the SUS questionnaire are reported in Table 5. In particular the responses to statements 1, 3, 5, 7 and 9 show that participants appreciate the system, that they considered it simple to use and easy to learn even for non-experts of visual languages. Moreover the responses to questions 2, 4, 6, 8 and 10 show that participants did not feel they need support to use the system and did not found the system complex, intricate or inconsistent.

The scores of the questionnaire calculated on the responses of the participants range from 37.5 to 95, with an average value of 80.0, which value indicates a good level of satisfaction [1]. As it can be seen from the data in the table, only a single participant (the one who committed major errors) expressed a negative judgment on the tool.

In addition, participants provided some freeform suggestions for improving the system: most of the criticism was expressed on the editor for diagram composition tool, which was not felt to be very user-friendly. In particular, participants noticed that some basic operations for diagram composition, such as the insertion of connectors, are surprisingly uncomfortable. Furthermore, one participant pointed out that the editor is not well integrated with the VLDE.

6 Conclusions

In this paper we have presented a framework for the fast prototyping of visual languages exploiting their local context based specification. We have shown how to define a visual language by extending the local context with three new features and have presented a simple interface for its implementation LoCoMoTiVE. Moreover, we have described a user study for evaluating the satisfaction and effectiveness of users when prototyping a visual language. At the moment, the user study has been limited to the simpler version of the local context methodology in order to provide us with a first feedback. Given the encouraging results, we are now planning to test the usability of LoCoMoTive with more complex applications.

The local context approach may then greatly help visual language designers to prototype their languages very easily. However, more studies are needed to investigate the computational borders of the approach. Our intention is not too push local context features more than needed, keeping simplicity as a priority. More complex language constructs should then be left to the following phases of the recognition process as it is the case for programming language compiler construction.

As a final goal, we are working on the integration of the local context approach in frameworks for the recognition of hand drawn sketches, as shown in [7].

References

[1] A. Bangor, P. T. Kortum, and J. T. Miller. An Empirical Evaluation of the System Usability Scale. *International Journal of Human-Computer Interaction*, 24(6):574–594, 2008.

[2] R. Bardohl. Genged: a generic graphical editor for visual languages based on algebraic graph grammars. In *Visual Languages, 1998. Proceedings. 1998 IEEE Symposium on*, pages 48–55, Sep 1998.

[3] R. Bardohl, M. Minas, G. Taentzer, and A. Schürr. Handbook of graph grammars and computing by graph transformation. chapter Application of Graph Transformation to Visual Languages, pages 105–180. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 1999.

[4] P. Bottoni and G. Costagliola. On the definition of visual languages and their editors. In M. Hegarty, B. Meyer, and N. Narayanan, editors, *Diagrammatic Representation and Inference*, volume 2317 of *Lecture Notes in Computer Science*, pages 305–319. Springer Berlin Heidelberg, 2002.

[5] J. Brooke. Sus: A quick and dirty usability scale. In P. W. Jordan, B. Weerdmeester, A. Thomas, and I. L. Mclelland, editors, *Usability evaluation in industry*. Taylor and Francis, London, 1996.

[6] S. S. Chok and K. Marriott. Automatic construction of intelligent diagram editors. In *Proceedings of the 11th Annual ACM Symposium on User Interface Software and Technology*, UIST '98, pages 185–194, New York, NY, USA, 1998. ACM.

[7] G. Costagliola, M. De Rosa, and V. Fuccella. Local context-based recognition of sketched diagrams. *Journal of Visual Languages & Computing*, 25(6):955–962, 2014. Distributed Multimedia Systems {DMS2014} Part I.

Question		U1	U2	U3	U4	U5	U6	U7	U8	U9	U10	Avg. resp.	St. dev.
S1	I think that if I needed to define a visual language I would use this system	4	5	4	4	4	3	5	3	5	5	4.2	0.79
S2	I found the system unnecessary complex	1	1	1	2	4	2	2	4	1	2	2.0	1.15
S3	I found the system very easy to use	5	5	5	4	4	4	4	2	5	4	4.2	0.92
S4	I think I would need the support of a person who is already able to use the system	2	1	1	1	1	2	2	4	1	1	1.6	0.97
S5	I found the various system functions well integrated	4	5	4	3	3	4	4	3	5	3	3.8	0.79
S6	I found inconsistencies between the various system functions	2	1	1	2	1	1	1	2	1	2	1.4	0.52
S7	I think most people can easily learn to use the system	5	4	5	4	5	4	5	3	5	4	4.4	0.70
S8	I found the system very intricate to use	2	1	1	1	2	2	2	4	1	2	1.8	0.92
S9	I have gained much confidence about the system during use	4	4	4	5	5	4	3	2	4	5	4.0	0.94
S10	I needed to perform many tasks before being able to make the best use of the system	1	1	2	1	1	1	3	4	2	2	1.8	1.03
Score		**85**	**95**	**90**	**82.5**	**80**	**77.5**	**77.5**	**37.5**	**95**	**80**	**80**	**16.33**

Table 5: SUS questionnaire results (5-point Likert scale).

[8] G. Costagliola, V. Deufemia, and G. Polese. A framework for modeling and implementing visual notations with applications to software engineering. *ACM Trans. Softw. Eng. Methodol.*, 13(4):431–487, Oct. 2004.

[9] G. Costagliola, V. Deufemia, and G. Polese. Visual language implementation through standard compiler–compiler techniques. *Journal of Visual Languages & Computing*, 18(2):165 – 226, 2007.

[10] G. Costagliola and G. Polese. Extended positional grammars. In *Proc. of VL '00*, pages 103–110, 2000.

[11] J. de Lara and H. Vangheluwe. Atom3: A tool for multi-formalism and meta-modelling. In R.-D. Kutsche and H. Weber, editors, *Fundamental Approaches to Software Engineering*, volume 2306 of *Lecture Notes in Computer Science*, pages 174–188. Springer Berlin Heidelberg, 2002.

[12] E. J. Golin. Parsing visual languages with picture layout grammars. *J. Vis. Lang. Comput.*, 2(4):371–393, Dec. 1991.

[13] E. J. Golin and S. P. Reiss. The specification of visual language syntax. *J. Vis. Lang. Comput.*, 1(2):141–157, June 1990.

[14] R. J. and A. Schurr. Defining and parsing visual languages with layered graph grammars. *Journal of Visual Languages & Computing*, 8.1:27–55, 1997.

[15] U. Kastens and C. Schmidt. Vl-eli: A generator for visual languages - system demonstration. *Electr. Notes Theor. Comput. Sci.*, 65(3):139–143, 2002.

[16] N. Le Novere et al. The systems biology graphical notation. *Nature Biotechnology*, 27:735–741, 2009.

[17] K. Marriott. Parsing visual languages with constraint multiset grammars. In M. Hermenegildo and S. Swierstra, editors, *Programming Languages: Implementations, Logics and Programs*, volume 982 of *Lecture Notes in Computer Science*, pages 24–25. Springer Berlin Heidelberg, 1995.

[18] K. Marriott and B. Meyer. On the classification of visual languages by grammar hierarchies. *Journal of Visual Languages & Computing*, 8(4):375 – 402, 1997.

[19] M. Minas and G. Viehstaedt. Diagen: A generator for diagram editors providing direct manipulation and execution of diagrams. In *Proceedings of the 11th International IEEE Symposium on Visual Languages*, VL '95, pages 203–, Washington, DC, USA, 1995. IEEE Computer Society.

[20] I. Plauska and R. Damaševičius. Design of visual language syntax for robot programming domain. *Information and Software Technologies Communications in Computer and Information Science*, 403:297–309, 2013.

[21] J. Quinn et al. Synthetic biology open language visual (sbol visual), version 1.0.0, 2013. http://sbolstandard.org/downloads/specification-sbol-visual/ [Online; accessed 4-June-2015].

[22] J. Rekers and A. Schurr. A graph based framework for the implementation of visual environments. In *Visual Languages, 1996. Proceedings., IEEE Symposium on*, pages 148–155, Sep 1996.

[23] L. Weitzman and K. Wittenburg. Relational grammars for interactive design. In *Visual Languages, 1993., Proceedings 1993 IEEE Symposium on*, pages 4–11, Aug 1993.