Towards a Classification to Facilitate the Design of Domain-Specific Visual Languages

Sándor Bácsi, Gergely Mezei

Abstract: Domain-specific visual languages (DSVLs) are specialized modeling languages that allow the effective management of the behavior and the structure of software programs and systems in a specific domain. Each DSVL has its specific structural and graphical characteristics depending on the problem domain. In the recent decade, a wide range of tools and methodologies have been introduced to support the design of DSVLs for various domains, therefore it can be a challenging task to choose the most appropriate techniques for the design process. Our research aims to present a classification to guide the identification of the most relevant and appropriate methodologies in the given scenario. The classification is capable enough to provide a clear and precise understanding of the main aspects that can facilitate the design of DSVLs.

Keywords: domain-specific visual languages, modeling, classification

Introduction

The graphical interface of domains-specific visual languages (DSVLs) provides an improved abstraction of the problem that allows the rapid development for a specific domain. Expressiveness, flexibility and usability are the most important aspects in the effective development with a DSVL. Each problem domain requires a different visual representation to meet the requirements of the targeted domain, thus it is essential to choose the most appropriate representational concepts in design-time. The exactness of the choice depends on how expressively the chosen concepts describes the DSVL and on the specific needs of the targeted domain. To support the design process, various kinds of classifications have been created in the last decades. There are classifications mainly based on formalism [1], while other classifications are based on metamodeling approaches [4]. On the other hand, there are existing classification frameworks which support the design of visual languages [2]. Due to the increasing use of DSVLs, a wide range of new tools and methodologies have been introduced recently based on completely new ideas. Our research aims to analyze and compare the most relevant methodologies on a larger scope which can support the design of new domain-specific visual languages.

In this paper, we present the main results of our classification methodology. The classification is mainly based on the nature of the graphical objects that compose the visual language, the connection types among the graphical objects and the composition rules. In our research, we have analyzed a wide range of existing DSVL methodologies and also created several case studies for different domains to exemplify the most relevant graphical and structural characteristic. We have used two metamodeling frameworks (Eclipse Modelling Framework [9] and Visual Modeling and Transformation System [8]) and a visual programming editor builder (Google Blockly [7]) to examine the most applicable methodologies. In this paper, we provide a summary of our work focusing on the results rather than on the case studies applied.

Towards the Classification of Visual Languages

In this section, we present the main aspects of the classification. Each subsection of this section represents one dimension of the classification.

Methods of the abstract syntax definition

There are two key methods for the abstract syntax definition of a DSVL: metamodeling methodologies and non-metamodeling approaches.

Metamodeling methodologies provide possibilities for the definition and management of DSVLs based on the abstract notion of visual entities and of relations among them. These frameworks are capable of specifying the abstract syntax of a DSVL and expressing the additional semantics of existing information. The metamodel can expressively define the structure, semantics, and constraints for a

family of graphical models. Hence, complex structures and relations can be flexibly described by the usage of metamodeling concepts.

While metamodeling methodologies are based on various kinds of instantiation techniques, non-metamodeling approaches provide a template-based structure for creating visual entities. The main characteristic of these approaches is that they have a limited set of features which can be used on the different abstraction levels, thus complex structures cannot be visualized flexibly and expressively. One of the newest non-metamodeling approaches is Blockly [7]. It supports a large set of features for different domains. In Blockly the graphical objects are called blocks which can be customized as the basic building elements of the language. Due to the template-based and weakly typed structure of Blockly, type constraints cannot be applied.

Relation type

Based on the relation type, domain-specific visual languages can be grouped into two subclasses: containment-based and connection-based subclasses.

In containment-based languages, the graphical entities can be embedded in each other to express visual sentences of the targeted domain, therefore, no connections are needed. Beside embedding, the graphical entities can be attached to other entities (as a method and its parameters) and chained together (as in a call stack). A pre-defined set of containment rules or constraints have to be defined to restrict the way of embedding, attaching and chaining. Blockly and Scratch [6] are widely used examples of containment-based languages. Both Blockly and Scratch support building blocks that can be connected like puzzle pieces in order to create visual sentences.

Connection-based languages consist of two different kinds of building elements: entities and connections, i.e. nodes and edges. While the data is basically expressed by entities, the flow of the model and the relations among entities are defined by connections. Connections may also have properties to ensure the customization of the relations among entities. Beside the connection-based patterns, entities may have containment-based nature, for example, they can be embedded in each other.

Flow type

Based on the flow type, domain-specific visual languages can be grouped into three subclasses: data flow languages, control flow languages and languages with no flow.

Data flow visual languages visualize the flow of data focusing on the steps of data processing. Data flow concepts are based on the idea of disconnecting computational actors into stages that can execute concurrently. Data flow DSVLs visualize the processes that are undertaken, the data produced and consumed by each process, and the storing graphical objects needed to hold the data. It is possible to visualize what the system will accomplish by the flow of data.

Control flow visual languages visualize the logic of computation by describing its control flow. Control flow DSVLs graphically express the order in which instructions or statements are executed or evaluated. The graphical objects mainly represent the control structures and conditional expressions of the language, thus it is possible to visualize how the system will operate by the flow of control.

There are DSVLs which are neither data flow nor control flow because they target a static domain problem. These languages can visually represent the structure of a system or a program, therefore no flow has to be described. A widely used example of no-flow graphical modeling languages is the UML class diagram, in which the structure of the system is described by the classes and the connections among them.

The way of the problem description

Based on the way of the problem description, domain-specific visual languages can be grouped into two subclasses: imperative and declarative languages.

Declarative visual languages describe the logic of computation. For example, SparqlBlocks [5] is a declarative DSVL developed in Blockly. Declarative languages visualize sets of declarations or declarative statements. Each of these visual declarations has a meaning depending on the targeted domain and may be understood independently. A declarative style of visualization helps to understand the problems of the targeted domain and the approach that the system takes towards the solution of the problem, but is less expressive on the matter of mechanics which describe the flow of the system.

Imperative visual languages consist of visual statements that change the state of a program or a system. For example, Scratch is an imperative visual programming language. The visualized statements express the way of execution of which results in a decision being made as to which of two or more visualized paths to follow. In imperative languages, the visual sentences can be created by sequences of commands, each of which performs some action. These actions may or may not have a dedicated meaning in the targeted problem domain.

Visual representation

DSVLs have a visual concrete syntax used for the representation of graphical elements and connections. Based on the visual representation, there are two key design aspects: iconic and diagrammatic visual representation.

Entities are visualized by icons in iconic languages. For example, Lego Wedo 2.0 Software [13] provides an iconic visual language for educational purposes. The iconic language is a structured set of the related icons. An icon can be composed of other icons or can be attached to another one, thus expressing a more complex visual concept. Some icons can be immanently deceptive, some can only be interpreted within a certain domain context, thus iconic visual languages may have their disadvantages.

Diagrammatic languages mainly composed of elements with a pre-defined symbolic representation of information. The building blocks of diagrammatic languages such as geometric shapes are often connected by lines, arrows, or other visual links. Chart-like, schematic-like and graph-based visual languages are the most widely used examples.

Conclusions

In this paper we have presented several aspects of the classification for domain-specific visual languages. We believe that this classification can be used as a guide while designing DSVLs. Hence, with the help of these guidelines it is now possible to analyze the characteristics of the language, and to associate it to an appropriate class or dimension.

On the other hand, we have analyzed the features of EMF, VMTS and Blockly based on different case studies that we have created for our classification methodology. We realized that due to the limitations of Blockly, many complex problems cannot be described expressively because aggregations, references and composition rules are missing from its developer framework. Despite the limitations of Blockly, it provides a flexible and easy way to learn to design imperative and control flow DSVLs based on containment-based aspects. Unlike Blockly, both EMF and VMTS provide a large feature set for the abstract syntax definition, but they are not as effective as Blockly in definition of containment-based languages. Based on EMF, Sirius [10] provides useful features for the customization of concrete syntax.

In the future, we aim to create a framework to support the design of visual-domain specific languages based on a questionnaire built upon the classification presented. We plan to create a framework to support an intuitively usable way of designing DSVLs even for complex language constructs. We are also working on new case studies and analyzing other existing approaches to create a more detailed classification.

Acknowledgements

The research has been supported by the European Union, co-financed by the European Social Fund (EFOP-3.6.2-16-2017-00013).

References

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

[2] Costagliola, G., A. Delucia, S. Orefice, and G. Polese, A Classification Framework to Support the Design of Visual Languages. Journal of Visual Languages and Computing, 2002. 13: p. 573-600.

[3] J. Sprinkle and G. Karsai, "A Domain-Specific Visual Language For Domain Model Evolution," Journal of Visual Languages & Computing, vol. 15, no. 3, pp. 291-307, 2004

[4] P. Bottoni and A. Grau, "A suite of metamodels as a basis for a classification of visual languages," in Visual Languages and Human Centric Computing, 2004 IEEE Symposium on, sept. 2004, pp. 83-90.

[5] Bottoni, P., Ceriani, M.: SPARQL playground: A block programming tool to experiment with SPARQL. In: Proceedings of the International Workshop on Visualizations and User Interfaces for Ontologies and Linked Data (VOILA@ISWC 2015).

[6] Scratch. Retrieved March 4, 2018, from https://scratch.mit.edu/

[7] Blockly Website. Retrieved March 8, 2018, from https://developers.google.com/blockly/

[8] VMTS Website. Retrieved March 10, 2018, from
https://www.aut.bme.hu/Pages/Research/VMTS/Introduction

[9] EMF. Retrieved March 16, 2018, from https://www.eclipse.org/modeling/emf/

[10] Sirius. Retrieved March 18, 2018, from https://www.eclipse.org/sirius/

[11] Ulrik Pagh Schultz. Graphical/Visual DSL. Retrieved March 23, 2018, from
http://websrv0a.sdu.dk/ups/SSE02/slides/lecture-10.pdf

[12] Iconic Visual Languages. Retrieved March 23, 2018, from
https://people.cs.pitt.edu/ chang/365/sk1.html

[13] Lego Wedo 2.0 Software. Retrieved March 30, 2018, from
https://education.lego.com/en-us/downloads/wedo-2/software