International Journal of Computer Applications (0975 – 8887)
Volume 17– No.1, March 2011

A Study on Visual Programming Extension of JavaScript

Wajid Ali
National University of Computer and Emerging Sciences
B-Block, Faisal Town
Lahore, Pakistan

Kanwal Sultana
National University of Computer and Emerging Sciences
B-Block, Faisal Town
Lahore, Pakistan

Sophia Pervez
National University of Computer and Emerging Sciences
B-Block, Faisal Town
Lahore, Pakistan

ABSTRACT
Visual programming extension of JavaScript language is presented and discussed. The aim of designing this language is to make programming simple, easy and more understandable to the audience who are fresh or new in programming, to improve the correctness with which people perform programming tasks, and/or to improve the speed with which people perform programming tasks. It provides ease to programmers, i.e. they are not required to learn or remember syntax. They will just have to enter the values using visual constructs. It is an attempt to make programming and maintenance of programs easier, and therefore cheaper. A hybrid technique is used to design VPxJavaScript, both symbols and text will be used to construct a program using this language. While deciding symbols to be used in, it was kept in mind that symbols must be as few as possible and most understandable shapes have been selected to represent corresponding constructs of JavaScript. The above aspect helped us in designing visual programming in order to keep the language more readable, understandable and writable. This paper highlights the usefulness of each symbol and discusses visual representation of normal JavaScript code. This paper also discussed the problems in JavaScript and proposed solution of problem and its equivalent visual representation.

General Terms
Visual Programing Language, Theory of Programming Languages.

Keywords
VPL, Visual Programming Language, JavaScript.

1. INTRODUCTION
Research on visual programming languages (VPL) got started in an aim at offering possibilities of solving problems by describing their properties or their behavior using graphical/iconic definition [8]. It is an attempt to make programming easy, useful for the people who are new in programming or they were not to remember syntax. VPL also makes the maintenance of programs easier, and therefore cheaper. There is no doubt that programming, especially when program size grows, is difficult; many books have been written about the subject. It is also a known fact that program maintenance is a major contributor to total software costs [3]. Based, at least in part, on the observation that programmers, when trying to figure out some difficult part of a program, often resort to drawing pictures [2], early research on visual programming languages was begun. Even in recent VPL research, terms like ease of use and intuitive are still used prominently.

There is not a consensual definition for visual languages (VL). The intuition says that almost everything that uses composition of figures, instead of words, in order to transmit a message, can be considered a VL. In this sense, there are many types of VL. Examples cover a large range from the daily used musical scores or traffic signals, and those more specific like modeling languages for definition of Entity-Relation Diagrams (ERD), Class Diagrams, and so forth [9]. The idea of visual programming is based on computer graphics, programming languages, and human-computer interaction [10]. A visual Language manipulates visual information or supports visual interaction, or allows programming with visual expressions. The latter is taken to be the definition of a visual programming language. Visual programming environments provide graphical or iconic elements which can be manipulated by the user in an interactive way according to some specific spatial grammar for program construction [4].

VPLs are used in many areas of computing. In databases, the main usage of visual languages is to help on drawing the tables and relations between them, rather than using SQL notation. In software development, they are mostly used to draw the system's structure and its behavior with modeling languages as referred before. In interface design for stand-alone or web based applications, visual programming languages that allow the drag, drop and composition of interface elements like buttons, textboxes, windows, and so on [9].

The variety of Visual Programing Languages in the history is summarized in section 2. Section 3 includes the classification of Visual Programming Languages. Visual Programming Languages issues are presented in Section 4. The proposed design of VPxJavaScript is presented in Section 5. The problem of JavaScript and proposed solutions are discussed in Section 6. The conclusion words are included in section 7.

2. HISTORY OF VPL
From as early as 1963, visual programming languages research has spawned a variety of languages for the different strains of visual programming. I.B. Sutherland introduced Sketchpad, a visual programming language which is created using constraint based graphics [6]. The system allowed users to work with a lightpen to create 2D graphics by creating simple primitives, like lines and circles, and then applying operations, such as copy, and constraints on the geometry of the shapes. Its graphical interface and support for user-specifiable constraints stand out as Sketchpad's most important contributions to visual programming languages [10]. Ivan Sutherland's brother, William, also made an important early contribution to visual programming in 1965, when he used the TX-2 to develop a simple visual dataflow language. The system allowed users to create, debug, and execute dataflow diagrams in a unified visual environment [12].

The next major milestone in the genesis of VPLs came in 1975 with the publication of David Canfield Smith's PhD dissertation entitled "Pygmalion: A Creative Programming Environment" [4]. Pygmalion is an icon-based programming language in which the user created, modified, and linked together small pictorial objects, called icons, with defined properties to perform computations [10]. Mark Edel developed the graphical programming environment for the language Lisp (used in Artificial Intelligence programming), called Tinkertoy in 1988 [11]. In 1991, Takayuki Dan Kimura developed a dataflow language for pen computers called Hyperflow [13]. Margaret Burnett, Paul Carlson, et. al. developed Forms /3, a structured form-based object-oriented language which incorporates spreadsheet based ideas of cells and formulae [7] in 1992. Prograoh is an object oriented VPL, created by T. Pietryzkowski and P.T. Cox [14]. It is commercialized by by Pictorius Inc.

3. CLASSIFICATION OF VPL

The visual programming language are classifies as, purely visual languages, Hybrid text and visual system, programing by demonstration, constraint based systems and form based systems. Of course these paradigms are not mutually exclusive; one single language may belong to more than one paradigm. They include, among others, we presents a summary of the each classification scheme below:

3.1 Purely Visual Languages

Purely visual languages are characterized by their reliance on visual techniques throughout the programming process. The programmer manipulates icons or other graphical representations to create a program which is subsequently debugged and executed in the same visual environment [10]. Examples of such completely visual systems include VIPR, Prograph, and PICT.

3.2 Hybrid – Text and Visual System

These hybrid systems include both those in which programs are created visually and then translated into an underlying high-level textual language and systems which involve the use of graphical elements in an otherwise textual language [10]. Examples in this category include Rehearsal World and work by Erwig et. al.

3.3 Programming by demonstration

It is also known as programming by Example. In this system, one instructs the computer what to do by showing examples of proper behavior [8]. Example of these type systems is macro in Excel.

3.4 Constraint Oriented Systems

Constraint-oriented systems are especially popular for simulation design, in which a programmer models physical objects as objects in the visual environment which are subject to constraints designed to mimic the behavior of natural laws, like gravity. Constraint-oriented systems have also found application in the development of graphical user interfaces [10].

3.5 Form based Systems

Apparently based on spreadsheets, but more generic, since the program may consist of many forms, which need not be strictly tabular. Form-based programming uses the filling in, often with visual elements, of forms [10].

4. DESIGN STRATEGY OF VPL

A common misunderstanding is that the goal of visual programming research in general and VPLs in particular is to eliminate text. This is a fallacy in fact, most VPLs include text to at least some extent, in a multidimensional context. Rather, the overall goal of VPLs is to strive for improvements in programming language design. The opportunity to achieve this comes from the simple fact that VPLs have fewer syntactic restrictions on the way a program can be expressed (by the computer or by the human), and this affords a freedom to explore programming mechanisms that have not previously been tried because they have not been possible in the past.

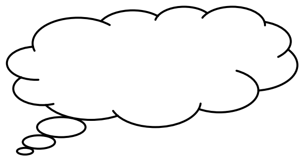
The most common specific goals sought with VPL research have been (1) to make programming more understandable to some particular audience, (2) to improve the correctness with which people perform programming tasks, and/or (3) to improve the speed with which people perform programming tasks.

5. DESIGN OF VPXJAVASCRIPT

We have used hybrid technique to design VPxJavaScript, both symbols and text will be used to code a program using this language. While deciding symbols to be used in, it was kept in mind that symbols must be as few as possible and most understandable shapes have been selected to represent corresponding constructs of JavaScript. The above aspect helped us in designing visual programming in order to keep the language more readable, understandable and writable.

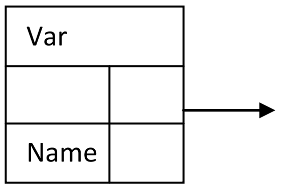
5.1 Comments

In JavaScript, there are two methods of writing comments, one is single-line comments and other is multi-line comments. The single-line comments begin with flash slash "//" and must end on same line while multi-line comments may exceed the length of a line. The opening part is "/*" and the closing part is "*/". The equivalent proposed visual representation of comment is



The above symbol is used for commenting the program where the user can give his/her desired description regarding the program. To comment program area that includes constructs, but that are not part of the program, color of the constructs will be changed. And light green color is decided for this purpose, because this color can be used to distinguish between the included and excluded constructs.

5.2 Variable Declaration

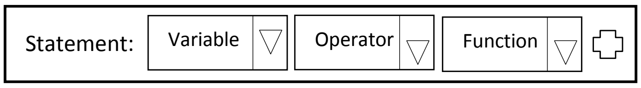


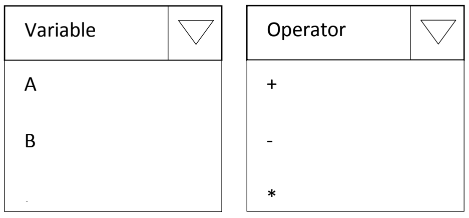
The above construct is used to declare a variable. The declared variable must have a valid name filed, whereas the value field is optional. If this value is given in inverted commas, variable type will be of string and if the variable is assigned a value of numeric type, the variable will have the type accordingly. The equivalent JavaScript examples are

Var a;

Var a = 2;

5.3 Statement



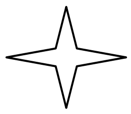


The above construct is used to include assignment statements in the program. As shown above, there are two drop down menus, one is for declared variables and the other is for mathematical operators including library functions. The rectangular bar inside the box is used to write mathematical expressions and for assignment purposes. By clicking on the plus button along the rectangular bar, another bar will appear where next statement can be written. Only one statement will be allowed to be written in one rectangular bar. The JavaScript equivalent code examples are

```
A = 5 + b – d;

B = 5.5 + “String”
```

5.4 Connector

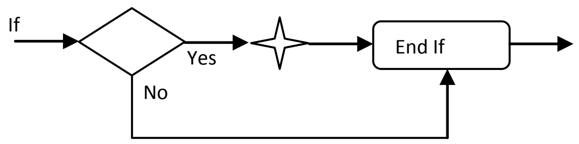


By right clicking the above symbol following popup appears.

Statement
If-else
Switch
For
While
Do While
End

The above symbol is used to add new constructs in the program. It performs the following two functions; one is work as a connector and other works as a terminator. It provides control to the program, i.e. help programmer, what to do next.

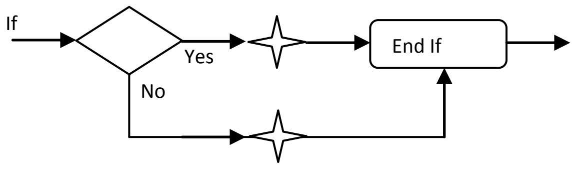
5.5 If-statement



The above construct is used for the if statement, if the condition in the diamond shaped box will be satisfied ,star will allow you to add other constructs for further processing, otherwise in case of unsatisfactory conditions, control will be given to end if part. The end if part works as parenthesis of the if-statement. The Java Script equivalent code is

```
if ( condition )
{
Connector to add new construct
}
```

5.6 If-else-statement



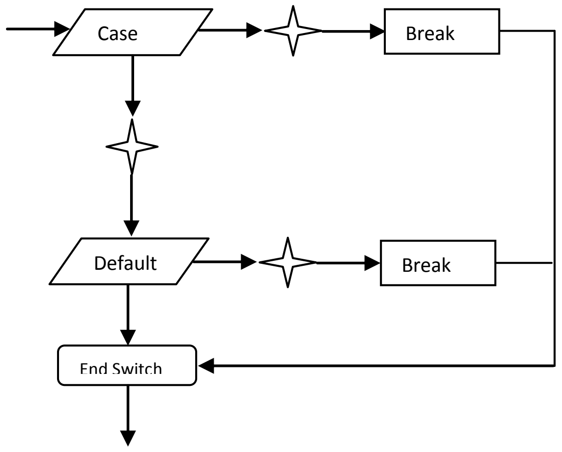
The above construct is designed for if then else condition. In case of satisfactory conditions, control will be given to the execution of if part whereas in case of unsatisfactory condition, control will be given to else part for further processing. The equivalent Java Script code for if-else statement is

```
if ( condition )
{
   Connector to add new construct
}
else
{
   Connector to add new construct
}
```

Nested If-else statement code is

```
if (condition )
{
   Connector to add new construct
}
else if (condition )
{
   Connector to add new construct
}
else
{
   Connector to add new construct
}
```

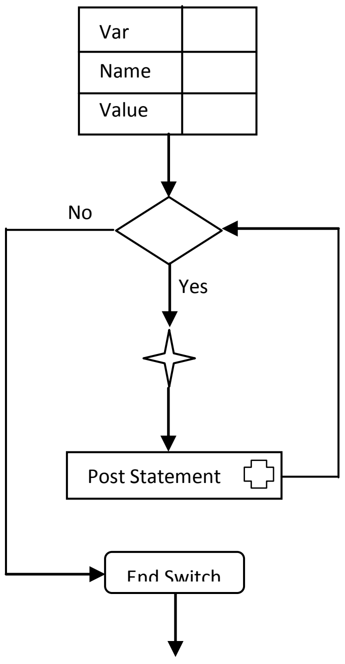
5.7 Switch Statements



The above construct is designed for switch statement, if the user wants to check more than one cases, it is given the flexibility of adding another case by clicking on the star in the switch menu. While including another case, another parallelogram will be created and further constructs can be added according to the requirement of the program. After executing a certain case, break will bring the control to the end of switch construct. The equivalent Java Script code is

```
switch ( expression )
{
   case label1:
      connector to add new construct
      break;
   case label2:
      connector to add new construct
      break;
   default:
      connector to add new construct
}
```

5.8 For Loop



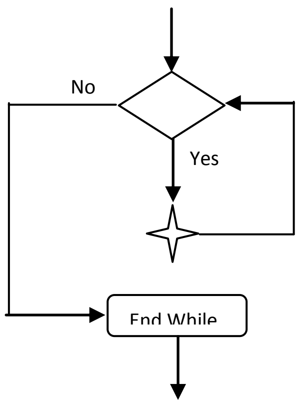
The proposed construct above is used to design for loop. In case of satisfactory condition, further processing can be done through star. After processing the body of the loop post assignment is done in order to preserve the structure of for loop. In case of unsatisfactory condition, loop will be terminated and the control is given to the end of the loop.

The Java Script equivalent code is

```
for ( [Initial Expression;] [condition;] [operation] )
{
    connector to add new construct
}
```

5.9 While loop

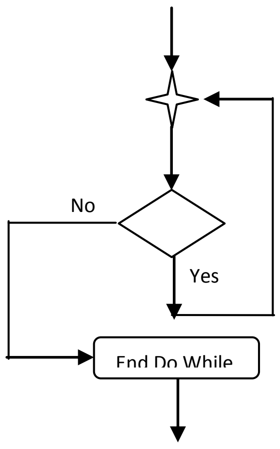
In this construct, while loop is designed by first checking the condition. In case of true result, control is given to the star for adding desired constructs. The other possibility, that is, getting false result, will lead the program to the termination of while loop.



The Java Script equivalent code is

```
while ( condition )
{
   Connector
}
```

5.10 Do-While loop



In the below construct, do while loop is designed in a way that it performs the required operation without checking any condition. In the end, a condition is checked, if the program meets the given condition, it will perform the same sequence until the given condition gives the false value as a result of execution.

The equivalent Java Script code is

```
do
{
  Statement
}
while (expression);
```

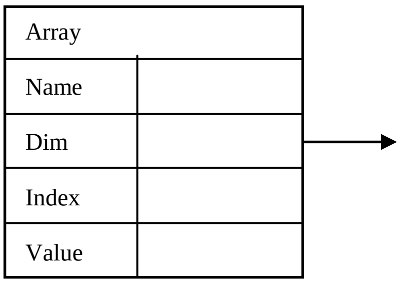
5.11 Array

The below construct is designed for array declaration and initialization. In case of declaration value will not be mandatory, while when initializing array it will be mandatory.

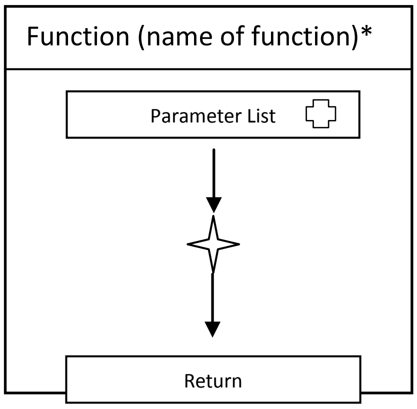
The Java Script equivalent code

```
var n_array = new Array(21, 22, 23, 24, 25);
or
var days_of_week = new Array(7);
days_of_week[0] = "Sunday";
days_of_week[1] = "Monday";
days_of_week[2] = "Tuesday";
```

```
days_of_week[3] = "Wednesday";
days_of_week[4] = "Thursday";
days_of_week[5] = "Friday";
days_of_week[6] = "Saturday";
```



5.12 Function



This symbol represents the FUNCTION in the Visual JavaScript. It has four parts. First part is reserved word function. The optional second part is the function's name. The third part is the set of parameters of the function (zero or more parameter names). The fourth part is a set of statements which presented here as connecter. These statements are the body of the function.

The Java Script equivalent code

```
function calcDiv(val1, val2)
{
        var myval
        myval = val1/val2
        return myval
}
```

6. SOLUTIONS TO THE PROBLEMS IN JAVASCRIPT

There are few problems in the design of JavaScript. We have identified them and propose solution to them to handle in VPxJavaScript.

6.1 Case Sensitivity

The JavaScript is a case-sensitive language whereas HTML is case insensitive. Because of its close association with JavaScript, it is very easy for the programmer to get confused. Therefore the program may not be writable. For example, The statement document.write() is legal, but document.Write() is not valid. A variable name strname is not the same as a variable named STRNAME. Case sensitivity in variable names causes readability problem. For example: if a user declares sum as integer and Sum as float, later he mistakenly assigns float value to sum which leads to incorrect results.

The solution to this problem is to make variable names to case insensitive i.e. strname is the same as STRNAME. To improve the readability of a program and maintain the consistency in the program, we chose VPxJavaScript case-insensitive.

6.2 Data Types

The JavaScript is a "loosely typed" language, means you don't need to specify the type of data being declared in a variable. The type of data can be changed later in the script without causing an error message. The JavaScript interpreter will determine the data type when it's processed within the script.

There are four basic data types used in JavaScript: Strings, Numbers, Booleans, and Nulls. A string is enclosed in quotes (" ..... ") to set it apart from the actual code. Either single or float quotes can be used but you must be consistent. JavaScript recognizes two types of numbers: integers and floating point numbers. Integers are whole numbers (e.g., 1, 20, 546) and floating point numbers are numbers that have a decimal point (e.g., 23.8, 23.890). Unless otherwise specified, JavaScript treats all numbers as floating point numbers, but if a calculation is performed using integers, the answer will be an integer unless the calculation itself changes it (e.g., the number is divided unevenly).

There are certain occasions while programming when we need to change a character into another while programming and also, assign a certain numeric value to a character type. In JavaScript this is not possible because of its rule of dealing all characters as strings. A new “character” data type should be introduced. A character is enclosed in single quotes and a string is enclosed in float quotes only. Introducing “character” data type adds more flexibility in VPxJavaScript for a lot of programming purposes. Now we have five data types in VPxJavaScript, String, Character, Integer, Float and Boolean.

6.3 Type Conversions

Data types can be converted automatically as needed during the course of script execution. A variable may hold a numeric value at one point of the script and a string at another one. The following statements constitute a valid JavaScript script:

```
var myVar = 12
myVar = “university”
```

JavaScript is a loosely typed language. Therefore, such conversions are allowed but are not recommended. The Mixing of strings and numbers is tricky and can generate unexpected results. When an expression includes both numbers and strings, the result evaluates to a string. Converting it to number is usually impossible.

JavaScript interpreter evaluates expressions from left to right, and only parentheses can change the order of evaluation. Take a look at the following expressions,

```
8 + 8          // 16
“8” + 8        // “88”
8 + “8”        // “88”
“8” + “8”      // “88”
8 + 8 + “8”    // “168”
8 + “8” + 8    // “888”
```

All these expressions use the string concatenation operator which is also the numeric plus operator.

In VPxJavaScript, we define two rules for data type conversion. The rules are:

1. Strings and numbers cannot concatenate. That is if the expression contains both strings and numbers, the result is not evaluated to string instead an error message will be generated of data type mismatch.
2. Implicit float conversion is also not allowed. It should throw an exception.

Strings cannot be easily converted to numbers. Also there would be loss of data, if floats are assigned to integers. In our proposed language, strings can only be operated with numbers if numbers are explicitly cast into strings or strings have been converted into numbers. If this restriction is not imposed, unintended problems can arise. Floats have to be cast into integer explicitly, so that the user knows what s/he is doing. Otherwise, a warning would be generated.

6.4 Octal literal

The Octal digits in JavaScript are written with a leading 0 digit. For example

```
var oct

oct = 098
```

Most of the time decimal constants are written with leading zeros. For example, social security numbers, license plates and area codes can be written with leading zeros. For example:

01234 -077 0312

In all such cases, JavaScript interprets them as octal numbers and this creates a great deal of problem when such numbers are compared or searched for. To solve this problem, It is suggested to write octal constants by appending octal

```
var a = oct312
```

6.5 Equality and Assignment

In JavaScript if assignment operator “=” is used in place of equality operator “==”, no error is reported and program executes normally. The symbol set is overused and highly error prone. A very common mistake is to use the equal “=” sign for equality check. For example mistakenly writing

```
if ( a = b )
```

In the place of an equality test, this is legal and meaningful statement in JavaScript. It creates a bug that can be hard to detect. Consider the code given below:

```
a = 2; b= 3;
if ( a = b )
{
        document.write( “Executing IF clause: ” );
        document.write( “a=” + a + “ and b=” + b );
}
else
{
document.write( “Executing ELSE clause: ” );
         document.write( “a=” + a + “ and b=” + b );
}
```

The output of executing IF clause is: a=3 and b=3. Thus, nothing is flagged, if you try to evaluate (a = b). b is assigned to a and, if the b is non-zero it evaluates to true.

To solve this problem, in VPxJS, there will be no operator for assignment statements, all will be done by using visual constructs. Only equality operator will be used.

6.6 Control Structure

The Braces are not necessary for control structures in JavaScript which may create readability and writability problems e.g.

```
if( condition )
    stmt1
else
    stmt2
```

This problem has been solved by giving equivalent visual representation. Now consider the dangling else problem in JavaScript.

```
if(condition)
if(condition)
stmt
else
stmt
```

This code is ambiguous because the else in this code cannot be properly connected with any of the if-statements. There are the following possible paths for this code

Path 1	**Path 2**

If(condition)
{
if(condition)
stmt
}
else
stmt
	If(condition)
	{
	if(condition)
	stmt
	else
	stmt
	}

Introducing brackets would solve ambiguity in such control structures.

Path 1	**Path 2**

if(condition)
{
if(condition)
{
stmt
}
}
else
{
stmt
}
	if(condition)
	{
	if(condition)
	{
	stmt
	}
	else
	{
	stmt
	}
	}

6.7 Function Parameters

The JavaScript is forgiving when it comes to number of parameters. For any function declared in JavaScript, if the number of parameters are too few compared to the definition of the function, all the rest would be given NULL value. In the opposite case, the parameters that have been added as extra would be included in the defined parameters. It could result in faulty calculation and malfunctioning. e.g consider the following function

```
function calcDiv(val1, val2)
{
        var myval //][ integer
        myval = val1/val2
        return myval
}
```

In this case if function is called as

calcDiv(5)

Then in this case no error would be reported and the second parameter would be assigned a NULL value, resulting in undefined value in the return value, which might cause the program to crash.

In VPxJavaScript, we suggest that the grammar should enforce that the number of parameters in the function should be same as number of parameters in the called routine.

7. CONCLUSION

The aim of this research is to design Visual Programming extension of JavaScript language to make programming in JavaScript more understandable to fresh or new programmer, to improve the correctness with which people perform programming tasks, and/or to improve the speed with which people perform programming tasks. We proposed the VPxJavaScript by considering the aspect of understandability, easy to program, readability, writability and correctness. We also identify the problem in JavaScript design and also try to solve them in proposed VPxJavaScript.

8. REFERENCES

[1] Eason, G., Noble B., and Sneddon, I. N. , “On certain integrals of Lipschitz-Hankel type involving products of Bessel functions,” Phil. Trans. Roy. Soc. London, vol. A247, pp. 529–551, April 1955. (references)

[2] Christensen, C. “An example of the manipulation of directed graphs in the ambit/g programming language”. In M. Klerer and J. Reinfelds, editors, Interactive Systems for Experimental Applied Mathematics, 1968.

[3] Ghezzi, C., Jazayeri, M., and Mandrioli, D., Fundamentals of Software Engineering. Prentice-Hall, 1991.

[4] Smith, D. C., PYGMALION: A Creative Programming Environment. PhD dissertation, Stanford University, 1975.

[5] Golin, E. J. and Reiss, S. P. “The specification of visual language syntax” Journal of Visual Languages and Computing, 2(1):141–157, 1990.

[6] Sutherland, I. B., SKETCHPAD, a man-machine graphical communication system. In Proceedings of the Spring Joint Computer Conference, pp. 329–346, 1963.

[7] Burnett, I., Baker, M. J., Bohus, C., Carlson, P., Yang, S., and Van Zee, P., "Scaling up visual programming languages.", Computer, Volume: 28, Issue: 3, March 1995, pp. 45 - 54

[8] Shetty, P. , Visual Programming Languages – Efficiency of the Visual driving Technology, 2004.

[9] Oliveira, N., Henriques, P. R., Cruz, D. da, and Pereira, M. J. V., "Visuallisa: Visual programming environment for attribute grammars specification" In Proceedings of the International Multiconference on Computer Science and Information Technology - 2nd Workshop on Advances in Programming Languages (WAPL’2009), pages 689 – 696,Mragowo, Poland, October 2009.

[10] Boshernitsan, M. and Downes, M., “Visual programming languages: A survey,” University of California, Berkeley, California 94720, Tech. Rep., December 2004.

[11] Edel, M., "The Tinkertoy graphical programming environment." IEEE Transactions on Software Engineering,1988, pp.1110 -1115.

[12] Najork, M., "Visual programming in 3-d". Dr. Dobb’s Journal, 20(12):18–31, December 1995.

[13] Kimura, T. D., "Hyperflow - a visual programming language for pen computers.". IEEE Workshop on Visual Languages, 1992, pp. 125-132

[14] Cox, P. T., Giles, F. R., Pietrzykowski, T., "Prograph: a step towards liberating programming from textual conditioning.", IEEE Workshop on Visual Languages, 1989, pp.150 - 156