

# An Object–Oriented Visual Dataflow Language

Da–Qian Zhang   Sute Lei   Kang Zhang  
*Department of Computing, Macquarie University,  
Sydney, NSW 2109, Australia*

## Abstract

Visual representation can help programmer to program in a direct manipulated way. This paper presents a visual language which employs object–oriented methodology and dataflow diagrams. In the dataflow network, processing stations are objects which can be multiparadigm in the low level implementation. What paradigm should be selected is determined by application domain. This supports programmers of different levels and different domains. In a high level, dataflow diagram is used to construct the relationships between objects. To simplify the complicated relationships, a hierarchical representation called dataflow unit is employed.

## 1. Introduction

The past few years have seen a rapid growth in the number of highly interactive application programs, not only on personal computers but also on professional workstations. User interfaces for these application programs have begun to take full advantage of the graphics capabilities of modern workstations [1]. However, the majority of computer users do not know how to program [6]. They buy computers with software packages and are not able to modify the packages to make small changes. Therefore, a visual system is highly desirable which enables the user to draw graphs intuitively to define a sequence of tasks, and also supports customisation and extension of the packaged software. Such systems are called *visual programming systems*. Previously designed systems include HI–VISUAL [5], GRClass [7] and Hyperflow [4]. In these systems programming is carried out through a spatial manipulation of visual elements on the screen [5] whereby the utilisation of computer is enhanced.

In spite of their popularity, visual programming systems have a number of limitations. For instance, they are usually restricted within given application domains; they are inefficient in visual representations; and they support only a limited set of data types and operators [2, 6]. To solve these problems, we identify the following criteria for designing a general–purpose visual language:

- To support more effective code generation and hierarchical design, the language should be defined on the object–oriented basis, since objects can embody states and behaviour to model real–world objects in a more efficient manner than traditional software modules.
- Visual programs should be at least two dimensional and support direct manipulation. It should make the best use of the advanced interactive graphics and window technology.
- The operational visual semantics should be easy to understand from the visual representation of programs. Dataflow approach has been commonly accepted as an easy way to visualise the execution of parallel programs and operation of an information system.
- The language should be tailored for users of all levels. In other words, it should support not only flexible system programming, but also application–oriented programming.
- Language components should be customisable. The user should be able to define the operational semantics

as well as the visual appearance of various language components.

In this paper, we propose an object-oriented visual language called POL (*Picture Object Language*) which meets the above criteria and is capable of describing real-world activities and applications in a more intuitive fashion. POL is hierarchically defined to serve the requirement of users of different levels; system programmers can use the lower-level language primitives to construct high-level and domain-oriented language components, while application programmers can use the high-level features of the language to build domain-specific applications. Such a hierarchical object-oriented design environment can greatly increase programmers' productivity on modern interactive workstations.

## 2. Multi-Level Modelling in POL

Different users may have different experiences and attitudes towards programming. Experienced programmers tend to abstract general underlying structure of a program, whereas novice programmers tend to concentrate more on the syntactic details. It is the experts who are thinking in terms of semantic categories and solving strategies of the problem; while novices in terms of syntactics. POL is a visual language and also a language development environment. For a visual language, the visual components are usually logical objects with visual representations. POL employs objects as logical entities and direct manipulatable pictures as physical entities. POL can be used to construct a basic set of domain specific picture objects. These picture objects can then be used to construct domain oriented POL which is used for programming applications. Figure 1 illustrates four levels of abstraction that POL supports. From level two to level four, POL is increasingly domain oriented. Picture classes (PCs) which support domain applications are self-contained for programming domain applications, assuming the relationships among picture objects (POs) have been defined in a lower level POL with object-oriented modelling method. The higher level POLs are suitable for novices, while the lower level POLs are usually preferred by experienced programmers. The suitability of POL to the users at different levels is listed in Table 1.

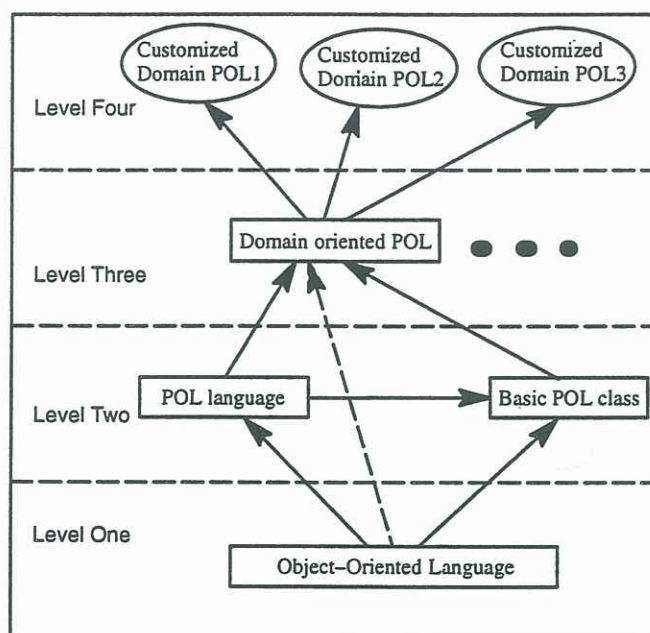


Figure 1. The hierarchical architecture of POL



Level one is an object-oriented language, which can be used by professional programmers to construct a general-purpose POL. The object-oriented language can also be a part of the implemented POL.

Level	Developed by	Used by
Level 1		Professional Programmer
Level 2	Professional Programmer	Professional Programmer + Domain Programmer
Level 3	Professional Programmer + Domain Programmer	Domain Programmer
Level 4	Domain Programmer	Domain User

Table 1. The suitability of POL to users of different levels

Level two is a general-purpose POL developed from level one. As a general visual language, POL is constructed to support object-oriented modelling and design. The basic parts of POL are picture classes (PCs). A PC is used to represent a real-world or abstract object class. The behaviors and relationships of PCs can be constructed directly using a visual editor provided by the POL environment.

At level two, computer programmers cooperate with domain specialists to develop domain-oriented POLs. This is achieved by using the lower level POL to model the domain PCs and their relationships through multiple levels of encapsulation. When the modelling process is completed, detailed definitions can be provided by textual description. By selecting a self-contained domain oriented set of PCs and RPCs, a domain-oriented POL can be developed.

Level three is a domain-oriented POL ready to be used for constructing domain applications. The PCs at level three include those defined in level two to be domain specific and those primitive PCs of level two. Therefore, a domain specialist can program, through direct manipulation on the domain PCs and general PCs of level two, to solve domain-oriented problems. Programming in domain POL can be much easier than programming in a textual language or a general purpose POL.

Level four is the application programs which are developed from the lower level POLs. At this level, domain PCs has self-contained functions which can be used to perform some complex tasks. The user can link these PCs in various structures to obtain different application programs.

### 3. High Level Visual Specification

#### 3.1 Class Visual Representation

Pictures are employed to visualise objects. A picture of a PC should satisfy the following two requirements.

- It should be able to represent all aspects of the PC;
- Only necessary information is displayed during its manipulation.

Usually, a class has three parts: class name, state, and method. We use a rounded rectangle to represent a class, as shown in figure 2(a). A rounded rectangle with a class name in it can show what the object is. It is called class name area, as shown in figure 2(b). But it may take a place larger than a hollow rounded rectangle. Similarly, There are the state area and method area you can open within the rounded rectangle. Figure 2(c) is a class representation in which the name area and the method area are opened. Any areas can open and close separately. A method has a name and input and output ports. The input ports are places of receiving data as parameters for the method and the output ports are places from which data are dispatched to other places. A method is represented by a rectangle. Like a class, you could open or close the name area and port area separately. The input port and the output port are represented by the hollow ball and the solid ball separately. If a port is both input port and output port, it will be represented by a hollow ball with a solid ball in it. Figure 2(d) is an example of a picture class. It shows there are two method in its method area. One method has an input port and an output port. The other has an input port, an output port and an input-output port.

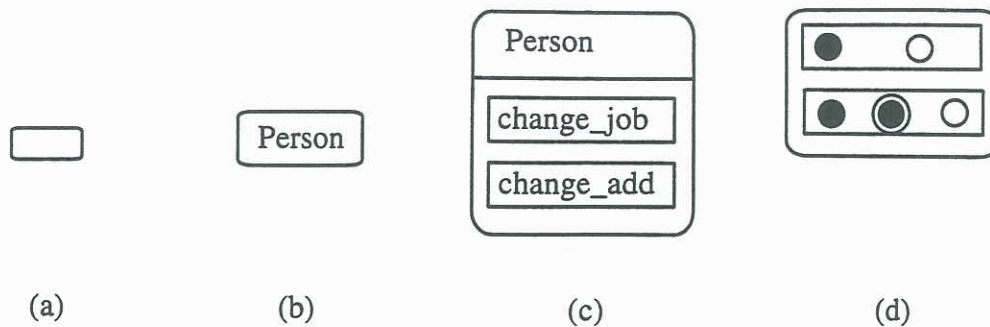


Figure 2. Class Representation

### 3.2 Processing Station

A picture object is like a processing station. It can receive data (object), process it and dispatch produced data. Processing is through one of the methods of an object. Methods of an object may or may not work parallelly, according to the nature of the object. A processing station can be fired by triggering its method(s). When all of input ports of a method receive the data, the method is triggered. However, the programmer can define trigger sets. A trigger set consists of several input ports. When the data reach these input ports, the method is triggered.

The method dispatches data through output ports when it finishes. It also can dispatch a special data which indicate some information about the work of the method. It is called processing information, as shown in figure 3(a).

A function or procedure can be seen as a particular object which only has a method, e.g. copy, merge and select. The copy is a processing station which can be used to duplicate a datum to flow to more than one places, as shown in figure 3(b). When data from different paths try to flow to the same input port, merge can be used, as shown in figure 3(c). A merge has two input ports and one output port. With merge, data coming from different input ports will go out in the same output port. Time stamps are append to the data when the data arriving. Thus, merge can output the data in the order in which they arrive. Select is another processing station used to select a path where the data will flow. In figure 3(d), the datum 'd' from input port A may flow from C or D according the condition 'c' which arrives in input port B).

### 3.3 Dataflow Activation

The execution of a PO obeys the dataflow firing rule, used in data driven computation. A dataflow computation is based on the principle of processing the data while it is in motion, 'flowing' through a dataflow network [1]. A dataflow network is a system of nodes and processing stations connected by a number of communication channels or arcs.

In POL, the processing stations are POs. Every method of a PO has a set of input ports and a set of output ports. The network is formed by linking the input port to the output port. Then the data can travel in it. The data which the processing station produces are also POs. When a PO is exported from an output port of a processing station, it will reach the input port (ports) of other processing station. A PO from a output port can create several copies to send to different ports which connect the output port separately. This can be done by 'copy' processing stations. Two output ports can also be linked to an input port where a 'merge' processing is used.

### 3.4 Trigger Event

In some cases, we need to control the flow of data. According to different condition, A datum flown from an output port may go to different input port. In dataflow diagram, it is an important event that data leave from or go into a port. Similarly, method and object can have event which indicate the begin and end of their processing. These events are called TriggerBegin and TriggerEnd separately.



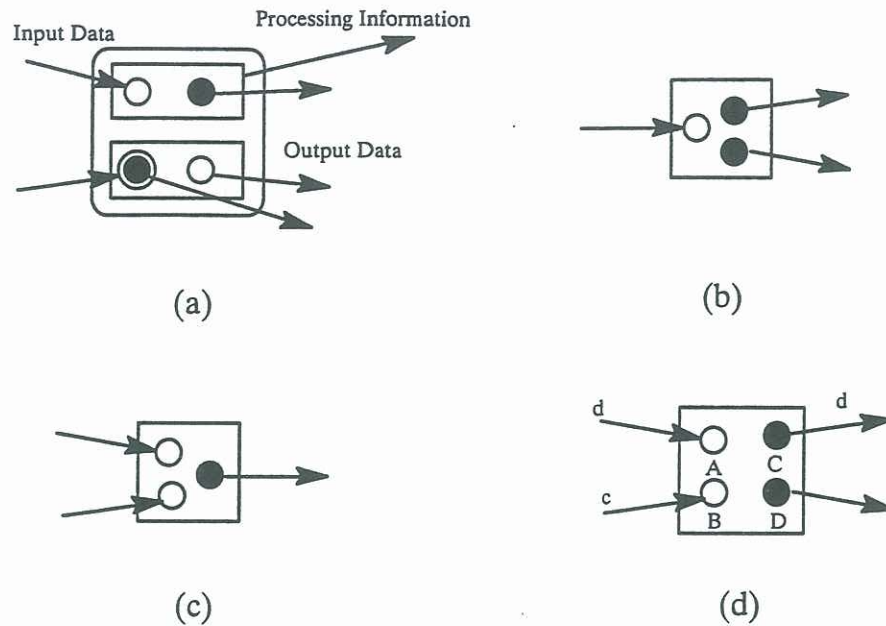


Figure 3. Processing Station

	TriggerBegin	TriggerEnd
Output Port		When the datum leaves
Input Port	When the datum arrives	
Method	When it is triggered.	When all its output ports dispatched their data.
Object	When one of its method is triggered.	When all of its method finished.

List 2. Trigger Event

### 3.5 Guard

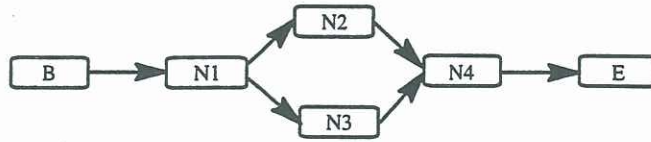
Programmers sometimes need to determine what happens when a program run. This is at least for two reasons: to debugging the program; to control the program. A guard is a special processing station used to inspect whether some events take place. It is fired by trigger events and processing information which come from other processing stations. When a guard finds that a certain event takes place, it will output a message to the dataflow network which can let the message flow to the destination. Initially, the guard has a value of false or true. Whenever one of its input port receives a datum, the value of the guard will be evaluated. If the value is changed, it will be dispatched to the dataflow network.

### 3.6 Simplify the dataflow representation

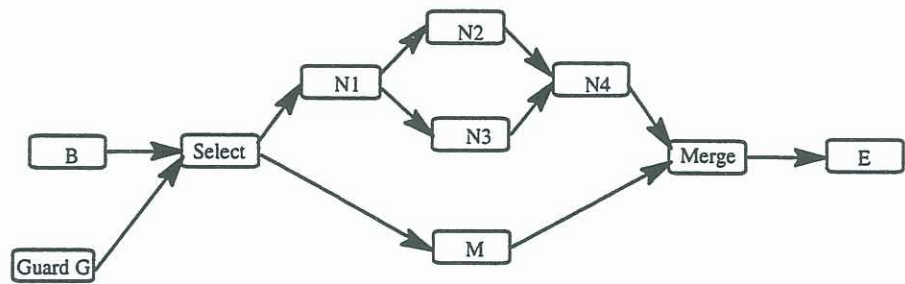
For a complicated problem, it is likely that the dataflow diagrams with complicated control are difficult to program and read in a visual representation. To simplify the representation of the dataflow diagrams, we use the dataflow unit to hide some information. A dataflow unit is a visual place where dataflow sub-diagrams are stored. These sub-diagrams have a certain logical meaning and when some events take place, they will be trig-



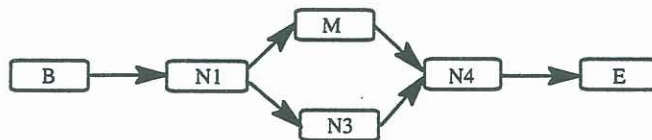
(a)



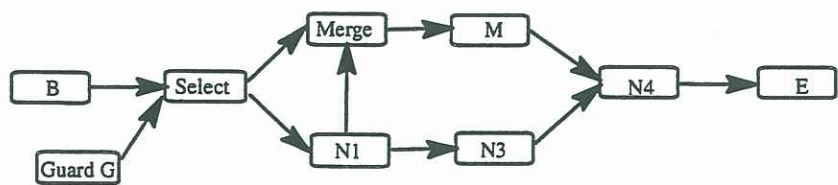
(b)



(c)



(d)



(e)

Figure 4. dataflow units and the transformation

gered. An example is shown in figure 4. Figure 4(a), (b) are two dataflow diagrams and if the Guard G is true, sub-dataflow diagram (a) will be selected as a part of dataflow net. Otherwise (b) will be select for flowing of data. These two sub-diagrams can not work at the same time. Therefore, it is convenient and efficient to select one of the sub-diagrams to show and the other to hide.

A dataflow unit links to a guard by which it may be fired. The user can manipulate direct on a dataflow unit and the dataflow will be changed by replace the corresponding sub-dataflow diagram. The sub-dataflow which is replaced will exist as another dataflow unit.

A dataflow diagram with dataflow units can be transfer to the dataflow diagram without dataflow units automatically. Figure 4(c) shows the transformation from (a) and (b) to (c). If N2 is M, as shown in (d), the result is (e).

A sub-dataflow paradigm can contain its own sub-dataflow paradigm. In this way, an dataflow can be represented hierarchically. This can simplify the representation of a complicated dataflow paradigm.

## 4. Conclusion

Dataflow diagram itself is visual in nature. The programmer can use the presented visual language to construct dataflow diagram in a direct manipulated way. This can greatly enhance the efficiency of programming. Furthermore, Object-oriented methodology is employed to broaden the application scope of the language. For representation of complication problem, dataflow unit is used for hiding information and simplifying representation. This dataflow unit representation and its nature will be further exploited in the further work.

## References

- [1] Stuart C. Schaffner and Martha Borkan, "Segue: Support for Distributed Graphical Interfaces", IEEE Computer, Dec. 1988, pp.42-55
- [2] Jose A. Borges and Ralph E. Johnson, "Multiparadigm Visual Programming Language", Proc. of 1990 IEEE Workshop on Visual language, Skokie, illinois, USA, Oct.4-6, 1990, pp.233-240
- [3] William W. Wadge and Edward A. Ashcroft, *Lucid, the Dataflow Programming Language*, Academic Press, Inc. (London), 1985.
- [4] Takayuki Dan Kimura, "Hyperflow: A Visual Programming Language for Pen Computer", Proc. of 1992 IEEE Workshop on Visual Language, Seattle, Washington, USA, Sept. 15-18, 1992, pp.125-132
- [5] M. Kado, M.Hirakawa, and T. Ichikawa, "HI-VISUAL for Hierarchical Development of Large Programs", Proc. of 1992 IEEE Workshop on Visual Language, Seattle, Washington, USA, Sept. 15-18, 1992
- [6] Brad Myers, "The State of the Art in Visual Programming and Programming Visualization", Graphics Tools for Software Engineering, Kilgour and Earnshaw (editors), Cambridge University Press, 1989. pp.3-26
- [7] Greg Rogers, "The GRClass Visual Programming System", Proc. of 1990 IEEE Workshop on Visual language, Skokie, illinois, USA, Oct.4-6, 1990, pp.48-53