Report Number: WUCS-92-42

1992-10-01

Hyperflow: A Uniform Visual Language for Different Levels of Programming

Takayuki Dan Kimura

We propose a visual language, Hyperflow, for system programming as well as for end user shell programming. Hyperflow is designed for a multimedia pen computer system for children. It is a dataflow-based graphical language similar to Show and Tell. In order to demonstrate the capability of Hyperflow, we solve the programming problem of implementing a help command for children to telephone their instructor or parents using voice communication hardware (modem, microphone, speaker, and a clock). The resulting program includes visual programs to implement device drivers for the modem and clock hardware.
... Read complete abstract on page 2.

Hyperflow: A Uniform Visual Language for Different Levels of Programming

Takayuki Dan Kimura

Complete Abstract:

We propose a visual language, Hyperflow, for system programming as well as for end user shell programming. Hyperflow is designed for a multimedia pen computer system for children. It is a dataflow-based graphical language similar to Show and Tell. In order to demonstrate the capability of Hyperflow, we solve the programming problem of implementing a help command for children to telephone their instructor or parents using voice communication hardware (modem, microphone, speaker, and a clock). The resulting program includes visual programs to implement device drivers for the modem and clock hardware.

Hyperflow: A Uniform Visual Language for Different Levels of Programming

Takayuki Dan Kimura

WUCS-92-42

October, 1992

Department of Computer Science
Washington University
Campus Box 1045
One Brookings Drive
St. Louis MO 63130-4899

To appear in the Proceedings of 1993 ACM Computer Science Conference, Indianapolis, Indiana.

This research is partially supported by the Kumon Machine Project.

Hyperflow: A Uniform Visual Language for Different Levels of Programming[1]

Takayuki Dan Kimura
Department of Computer Science
Washington University in St. Louis
(314) 935-6122 tdk@wucs1.wustl.edu

Abstract

We propose a visual language, Hyperflow, for system programming as well as for end user shell programming. Hyperflow is designed for a multimedia pen computer system for children. It is a dataflow-based graphical language similar to Show and Tell. In order to demonstrate the capability of Hyperflow, we solve the programming problem of implementing a help command for children to telephone their instructor or parents using voice communication hardware (modem, microphone, speaker, and a clock). The resulting program includes visual programs to implement device drivers for the modem and clock hardware.

1. Introduction

In his keynote speech to the 4th UIST Symposium van Dam urged the unification of the user interface and the application [van Dam 91]. There are two aspects to be considered: First, the user interface and the application must be more tightly coupled, i.e., the user interface needs more "semantic feedback" from the application. Second, there should be a single environment for development of both.

In this paper we propose an approach to fulfill van Dam's demand through a visual programming language called Hyperflow. Even though visual programming languages are traditionally considered to be suitable only for simple programming by novice users, there is no reason to restrict the principles and applications of visual languages to the end user domain. Recent software development tools, such as Interface-Builder [NeXT 90], are becoming more visual (graphics-oriented). A visual language for system programming may not be distant. Such a visual language could be used by end users as a visual shell language and by system programmers to implement device drivers and compilers, thus offering an attractive software tool for all levels of programming.

Hyperflow is a visual programming language designed for a pen-based multimedia computer system. It is a part of the research project to develop a system for teaching mathematics to children from pre-school to the 12th grade. The project began in January 1991 and should finish in 1995. An overview of the project is given in [Kimura 92a]. The pen computer system is based on custom-made hardware platform using the MC68030 and DSP56001 chips. Software development for the system involves various levels of programming, from assembly language programming for the DSP chip to visual shell language programming by school children. Hyperflow is intended to be a uniform language tool for all types of software development, from

1 To appear in the Proceedings of 1993 ACM Computer Science Conference, Indianapolis, Indiana.
This research is partially supported by the Kumon Machine Project.

end user applications to hardware device drivers.

Hyperflow is a dataflow-based visual language similar to Show and Tell [Shu 88]. A program in Hyperflow consists of boxes and arrows, a box representing a process and an arrow representing a data flow between processes. Continuous data types such as audio signals and animations are available in Hyperflow. Continuous data types are necessary not only for multimedia processing but also for processing pen-strokes. As a user interface tool, Hyperflow is an extension of various window systems such as X Windows [Scheifler 88], with more independent computation power for each window. A window communicates not only with the user, but also with other windows through visually specifiable dataflow. The user controls the activities of a window by scribing a gestic command on it. Each window has a separate set of commands to be activated by the user's gestures. Each Hyperflow component is a window which can respond to a large variety of gestic commands if necessary, in contrast with mouse-based widgets that can only respond to clicking and dragging.

In this paper we demonstrate the capability of Hyperflow by solving a specific programming problem. We introduce various Hyperflow constructs by example. The programming problem is to implement a help command for children to call by phone his/her instructor or parents using the voice communication hardware (modem, microphone, speaker, and a clock). We assume that no operating system capabilities are available other than the Hyperflow language processing system. Even though Hyperflow was designed as a multi-lingual programming language, (both graphical and textual programming is possible), we present a solution using only graphical programming constructs.

In the next section we present an overview of Hyperflow's semantic concepts. In Section 3, we define the programming problem. In Section 4, we give a solution with commentaries. In the final section we evaluate our solution and discuss future directions.

2. Overview of Hyperflow

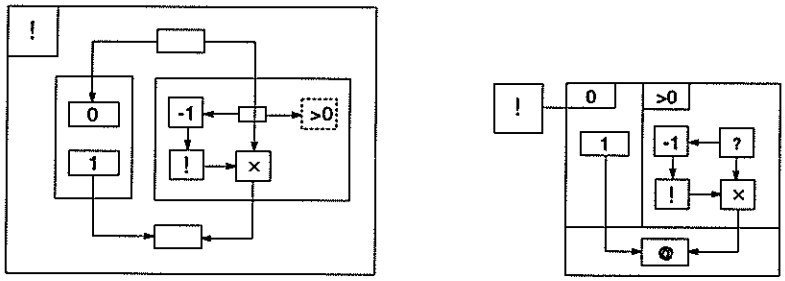
In this section we summarize the basic constructs of Hyperflow. The design philosophy and the semantic model are described in more detail in [Kimura 92b].

2.1 Show and Tell

Show and Tell [Kimura 86] was one of the first generation of visual programming languages whose semantic model was based on the concept of dataflow. It was developed on the Macintosh as a programming language for school children. We learned from the Show and Tell project that in spite of inherent concurrency, novice users find dataflow easier to understand than control flow. We also learned that the mouse is difficult to use for children.

In Show and Tell, a program consists of nested boxes and arrows. Each box represents either an operator (function) or an operand (variable). Each arrow represents the transfer of a data value from one box to another. An example Show and Tell program is given in Figure 1 (1). It recursively defines the factorial function. The iconic name of the program is given in the upper-left corner. Each case is represented by a separate box. A dotted box contains a predicate. The empty boxes represent formal parameters. When the user enters an argument on the Macintosh

screen, (e.g., the numeral 5, into the upper empty box) and selects the menu item **solve**, the result of the factorial computation, (the numeral 120,) appears in the lower box. If the user wishes to observe the dynamic inner workings of a component box, e.g., [!] inside the body before the multiplication, the user can obtain a visual representation of the function in another window by double clicking the box while the program is executing. For this particular example, the same visual representation will be displayed with a different argument, since the function is recursively defined. Thus for this Show and Tell program a visual program and its user interface are identical.



(1) In Show and Tell (2) In Hyperflow

Figure 1: Factorial Program

A factorial program in Hyperflow is given in Figure 1 (2). The line without an arrow head means that the two boxes have the same denotation, i.e., they are synonymous. The program denotes a *vip* (*visually interactive process*, to be discussed later) which receives a number by mail. The number is interpreted as a command name to the process. There are two commands for the factorial vip, each representing a different input case, either zero or non-zero. Each arrow denotes mailing a message from one box to another. The '?' box generates the incoming command (argument) and the '@' box represents the operation of sending mail from the factorial function box to a box that is connected by an outgoing arrow. The '@' box is shared by the two commands.

2.2 Design Philosophy

There are three different levels of intent for the Hyperflow design; as a pen user interface, a visual shell language, and as a visual system programming language. The pen interface requires a new paradigm where gesturing or handwriting is a major mode of command entry. With a single gesture, (instead of the two steps necessary in the menu-based interface,) the user can select both an object and an operation. In lieu of the menu, Hyperflow offers a vip that can respond to gestic commands.

The GUI users are deprived of the power of a shell language in graphical form due to the lack of abstraction capabilities in GUI frameworks. Visual languages offer the needed abstraction mechanisms. System programming is a complex task requiring various modes of communication among programmers. Visual communication is no exception. Visual languages offer a formal method of visual communication, thus forming the basis for visual system programming.

Hyperflow was designed to achieve four goals: understandability, responsiveness, universality, and extendibility. The understandability is partly enhanced by adopting dataflow rather than control flow for visualization. Most visual languages are dataflow-based because of its intuitive

understandability [Hils 92]. Another contribution to understandability comes from its parsimonious set of primitive concepts. A Hyperflow program specifies a homogenous ensemble of communicating processes. There are no functions or procedures. There are no classes or type declarations. In order to enhance its responsiveness to pen gestures, the Hyperflow kernel (run-time support system) includes real-time tasking based on preemptive scheduling. The universality is achieved in two important aspects: multimedia capability based on continuous data types and concurrent processing. The latter is essential for system programming. The extendibility of Hyperflow is achieved by two features: prototype based programming [Unger 87] and multi-lingual programming. The vip, the basic component of Hyperflow, is an encapsulation mechanism. Every vip is a potential prototype for new vips. A specification of vip activities can be given in visual form using Hyperflow's syntax, or in textual form using any traditional programming languages. This feature is needed for importing existing software.

2.3 System Decomposition

Hyperflow design presumes that the user interface is as ubiquitous as the computation in graphic computer applications. By user interface we mean any communication medium between the computation and the user. By the user we mean not only the end user but also a system programmer who designs and implements the computation. This leads to a different way of decomposing system software than the traditional partitioning of the top level software into the interface part and the application part where each part is then decomposed into modules and sub-units. Instead, we propose to decompose the system into functional modules, (i.e., vips in Hyperflow), then each module is divided into the interface part and the computation part. We also propose to apply the same decomposition strategy for system processes such as device drivers. The two approaches are compared in Figure 2 (1) and (2).

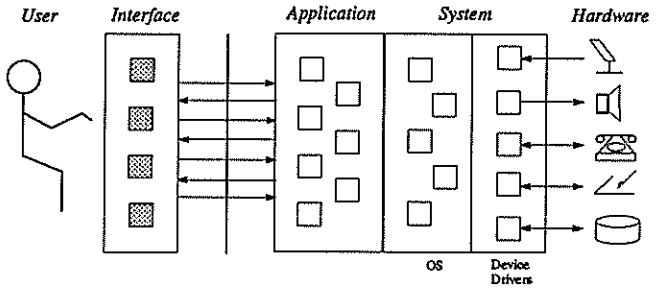


Figure 2 (1): Traditional Approach to Software Decomposition

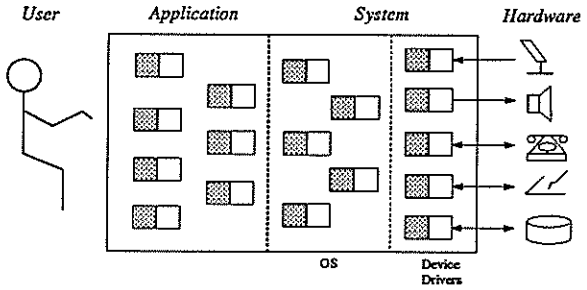


Figure 2 (2): Hyperflow Approach to Software Decomposition

2.4 VIP: Visually Interactive Process

A *vip* is a concurrent process with a user interface. It is the only unit of system decomposition. Computations are carried out by a homogeneous community of vips interacting with each other through the exchange of *signals*, either *discrete* or *continuous*. Continuous signals are timed data streams such as audio and video. Every vip is visually accessible to the user unless the user chooses to hide it. In Hyperflow's syntax each vip is represented by a box and each signal exchange is represented by an arrow.

There are four different modes of process communication in Hyperflow: *mailing*, *posting*, *channeling*, and *broadcasting*. Each mode is represented by a different type of arrow in Hyperflow's syntax. They are summarized in Figure 3.

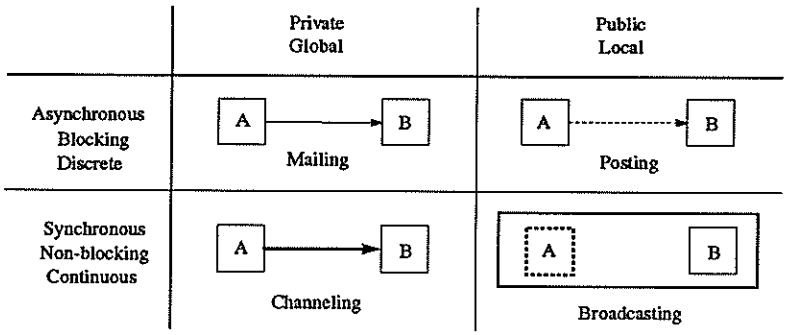


Figure 3: Communication Modes

Mailing is modeled after private and discrete human communication by **letters**. Similarly, posting is modeled after public and discrete communication through a **bulletin board**, channeling after private and continuous communication by **telephone**, and broadcasting after public and continuous communication by **TV**. However, in Hyperflow, broadcasting is also used for discrete communication such as sending an urgent command to all the neighbors in an emergency situation. Posting is local and so is broadcasting. Mailing is global and so is channeling. In traditional programming, a memory cell is a bulletin board which posts a datum received by mail.

A vip consists of at most one *mailbox* (a bounded FIFO queue of discrete signals), at most one *bulletinboard* (a discrete signal), the *body* (a vip ensemble) and the *command set* (a set of vip ensembles). The body is the container of the main activities of the vip. Each command has a textual or gestic name. A command (i.e. a named ensemble) becomes active only when its activation is requested by mail, by broadcasting, or by a user's gesture. Only one command can be active at a time. The body is active concurrently with a command. Any member of an ensemble, the body or a command, can access the mailbox and the bulletin board. The vip provides the local communication environment for its ensemble members. Posting and broadcasting are effective only among the ensemble members contained in the same vip.

There are system defined commands innate to all vips. Among them are:

start	make the vip active
stop	make the vip inactive.

Usually those two commands are activated by broadcast signals from the parent vip for scheduling purposes. When sequential processing is required among the members of an ensemble, the parent vip broadcasts start and stop signals alternately. The ordering is determined by the relative

position of each box in the parent box. (Note that each process is represented by a box in Hyperflow.) We call this convention *positional scheduling*. Different scheduling policies can be defined by the Hyperflow user. The default scheduling policy in Hyperflow is as follows:

the vip body	concurrent (data dependency only)
each command	sequential from left to right then top to bottom.

3. Problem Definition

In this section we define a programming problem that involves different levels of system programming, from implementing device drivers to constructing a user defined shell command.

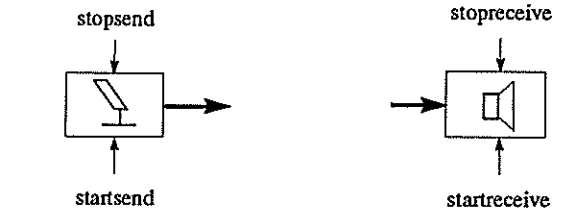
3.1 Hardware Resources

Our pen computer system for children contains the following hardware components to facilitate voice and data communication through telephone lines:

Modem (9600 bps, Hayes compatible)
Speaker with D/A converter
Microphone with A/D converter
Digital Signal Processor (DSP56001)
Line clock (1/60 second tick).

The digital signal processor is used (among other things) for real-time implementation of the CELP voice compression algorithm [Campbell 91] so that voice communication can be carried out through a 4800 bps channel. (The remaining 4800 bps capacity of the modem is used for simultaneous tablet data communication.)

We assume that the microphone vip (Figure 3(1)) contains the complete facility to encode continuous speech into a 4800 bps stream of compressed voice data using the DSP chip with CELP, and that the speaker vip (Figure 3(2)) can similarly decode a stream of compressed voice data back into continuous speech. The microphone vip is controlled by *startsend* and *stopsend* signals, and the speaker vip is controlled by *startreceive* and *stopreceive* signals.



(1) Microphone Box (2) Speaker Box

Figure 3: Voice Processors

The modem has two modes: control and data (Figure 4). In control mode, the modem receives through the TDB register one of the three different AT commands: resetting (ATZ), dialing (ATD followed by the phone number), and hanging up (ATH). After dialing, the modem may respond back by posting (on the RDB register) a BUSY signal, a CONNECT signal, or no response while the other party is ringing. When the CONNECT signal is posted and the signal is read, the

modem transfers into the data mode. In data mode a continuous signal, any stream of values sent to the TDB, is transmitted to the phone line until the escape sequence "+++" is sent. Similarly, the incoming stream, another continuous signal, from the phone line is generated by the modem through the RDB register.

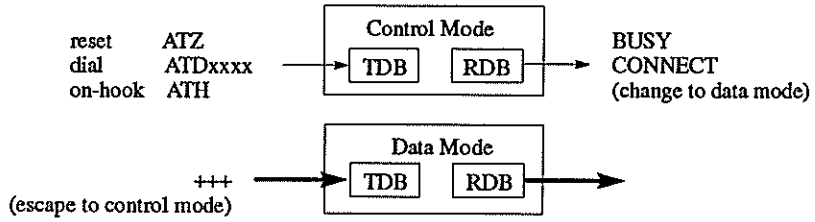


Figure 4: Modem Control

The line clock is used to detect NO ANSWER with a time-out after the dialing command is executed on the modem. The line clock causes a CPU interrupt every sixtieth of a second when the interrupt is enabled. We assume that the device's interrupt vector address (IVA) is $40. The interrupt mask is in its Control Status Register (CSR).

3.2 Help Command

A child studying math with the pen computer at home may need immediate help from the teacher or from his/her parents. Furthermore, the child may wish to design his/her own help command to perform the following operations in sequence:

> Call Teacher. If the teacher's phone does not answer or is busy, then
> Call Mother. If her phone does not answer or is busy, then
> Call Father. If his phone does not answer or is busy, then exit.

Once the help command is constructed and given an iconic name, the child can activate it by double-tapping the box containing the icon whenever the child needs help.

The programming problem for us is to construct a set of software tools in Hyperflow by which the child can construct his own help command. In other words, our problem is how to make the communication equipment accessible to children using Hyperflow. In particular, we need a device driver for the modem written in Hyperflow, and one for the line clock.

3.3 Hyperflow Primitives

In Hyperflow's syntax, receiving or sending a signal in different communication modes is denoted by a box containing special characters as illustrated in Figure 5. Note that broadcasting is separated into two forms. *Inward broadcasting* sends a signal to all vips owned by the sender including itself. If a "stop" signal is broadcast inward, it is equivalent to the sender stopping itself. Outward broadcasting sends a signal to all of the neighbors (ancestors and siblings) of the sender. Starting and stopping channel communication is accomplished by sending a start/stop command to the operation box (Figure 6).

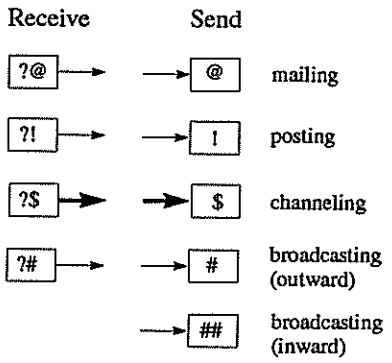


Figure 5: Communication Operations

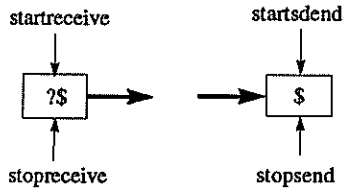


Figure 6: Control of Channeling Communication

4. Solution

We construct our solution from the bottom-up: first, a device driver for the line clock, second a modem driver, third a phone program, and finally a help command. For each program we present the vip specification first, then explain the syntax and semantics of each component.

4.1 Line Clock Driver

The vip implementing a timer (clock driver) is defined in Figure 7. Its overall structure is shown in Figure 8. The timer vip consists of three commands and the body part shared by the commands. The three commands are named *reset*, *start*, and *$40*, respectively. The last command name is system-defined, representing a hardware interrupt from the line-clock device whose IVA is $40 hexadicimal. Once the vip is created, the body part becomes active and remains so until the vip is deleted. A command is activated, one at a time, when the vip receives a signal that starts with the command name, either by mail or by broadcast.

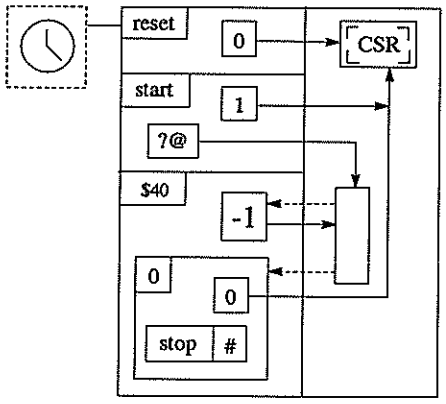


Figure 7: Timer VIP (Line Clock Driver)

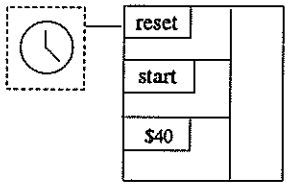


Figure 8: Overall Structure

The *reset* command disables the clock interrupts by resetting the interrupt mask bit in the clock CSR register, 

, where the four L-shapes in the icon indicate uniqueness, i.e. that there is only one process of this name. The *start* command enables the clock interrupts first, then gets a

number from the mailbox (?@ ⟶), and stores it into a **counter**, the empty box in the body part. After receiving a number by mail, the **counter** vip makes it available to any other vips by posting it on the bulletin board. The *$40* command decrements the counter first by executing the vip, -1 ⇠⇢, which reads the counter value from the bulletin board, decrements it by one, and mails the value back to the counter. If the counter has the value 0, (tested by 0), then the $40 command resets the clock by mailing 0 to the CSR, and broadcasts the "stop" signal (stop # ≡ stop ⟶ #) to all of the vips sharing the same parent vip as the timer vip. Otherwise, it executes no further operation.

One way of using the timer is to set a time-out for a function evaluation. The scheme is illustrated in Figure 9. The vip represented by the enclosing box outputs the result of evaluating the function, **F**, for the argument, **x**, within t > 0 ticks of the timer; if the evaluation takes longer than t, it outputs the message "time-out".

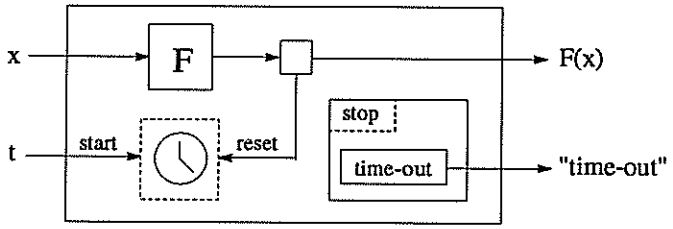


Figure 9: Function Evaluation with Time-out

If the time-out occurs before **F** completes its computation, the timer vip broadcasts the stop signal to all of the processes in the box including itself. Since all vips have *stop* as an innate command, every vip terminates itself upon receiving the broadcast signal from the timer. However, the stop command of the vip, stop , is redefined to mail the "time-out" signal before stopping. The dotted box surrounding the command name indicates that this command can be activated only by a broadcast stop signal, and not by a stop signal received by mail.

The program in Figure 9 also illustrates a syntactic convention in Hyperflow. The activation of a command can be designated either by dynamically sending the command name as the first component of a signal or by statically labelling the arrow line from the activator vip by the command name. In Figure 9, the reset command is activated when the evaluation value, F(x), is mailed to the timer. However, the timer does not use the value for resetting the clock, because the reset command of the timer does not read mail.

4.2 Modem Driver

Using the timer vip defined in the previous section, a modem driver vip, Modem , is defined in Figure 10. The vip has two commands: *dial* and *on-hook*. We assume the modem is used in the originating mode only. When the dial command is activated with a phone number (xxxxxxx) as its input parameter, the timer is set to time-out in 20 seconds and the modem is reset to control mode (ATZ), then the dialing command (ATD xxxxxxx) is sent to the modem. If the modem responds back within 20 seconds by signalling CONNECT or BUSY, the Modem vip mails the

identical signal to the activation vip. If the modem's response is CONNECT, the vip starts receiving a continuous signal (from the microphone) and transferring it to the modem. At the same time, the vip starts sending the continuous signal generated by the modem (to the speaker). Note that the body of the Modem vip remains active after the activation of the dial command is terminated. If the modem does not respond within the 20 seconds of dialling, the vip mails out NO ANSWER signal to the activator before the entire Modem vip is terminated.

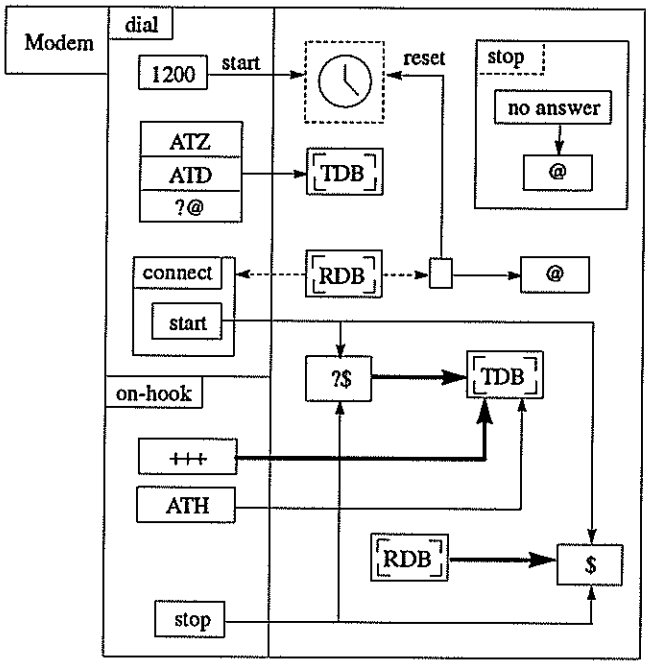


Figure 10: Modem Driver

When the voice conversation is finished, the Modem vip receives the *on-hook* signal. The command *on-hook* sends the escape code "+++" as a channel signal to the modem to switch it from data to control mode. Then, it sends the hang-up command (ATH) to the modem, followed by stopping the reception (from the microphone) and the sending (to the speaker) of a continuous signal. (See Figure 6)

4.3 Voice Phone Control

The Modem driver of Section 4.2 can be used to transfer any continuous signal through the modem and a phone line. In order to facilitate voice conversations through the modem, we need to integrate the voice processors, the microphone and the speaker (Figure 3), with the Modem driver.

The Phone vip,  , is defined in Figure 11. It has two commands, *dial* and *hang-up*. The hang-up command is a gestic command, i.e., the command can be activated only by a pen gesture scribed on the iconic name of the Phone vip.

The gesture has two line strokes crossed,  , and is entered as  .

The *dial* command activates the dial command of the Modem vip with the number sent to the Phone vip. If the modem responds back with a CONNECT signal, it starts the microphone vip to generate a continuous voice data and starts the speaker vip to play a continuous voice data received by the modem. If the modem's response is BUSY or NO ANSWER, it broadcasts a "stop" signal inward (stop | ## ≡ stop → ##) to all of the vips contained by the Phone vip terminating all of them. Note that the "stop signal" does not propagate to the vips outside of the Phone vip.

When the *hang-up* command is entered by the pen, the *on-hook* command of the Modem vip is activated, followed by stopping the channel communication by the microphone and the speaker, then finally broadcasting a "stop" signal to all of the neighbors of the Phone vip.

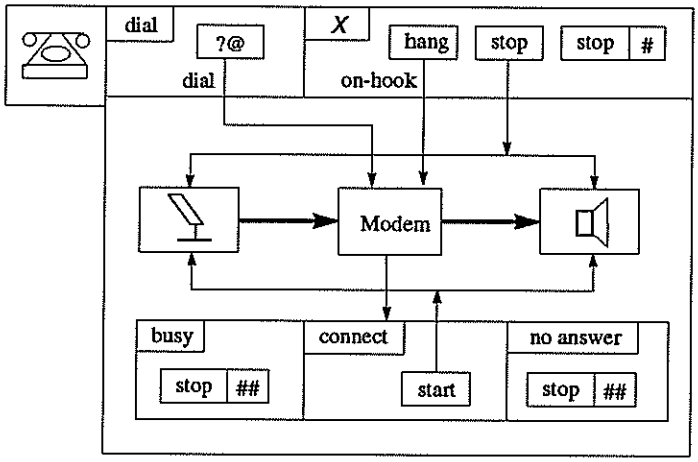


Figure 11: Voice Phone Program

4.4 Help Command

Assuming that the Phone vip is made available to a child, the child can construct a Help vip, ??? (Figure 12), for asking help by voice communication from the teacher or a parent.

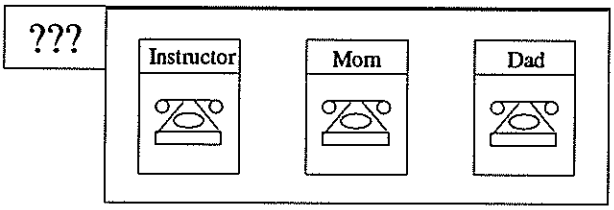


Figure 12: Help Command

The vip has no user defined commands, and the body contains three Phone vips to be executed sequentially from left to right. The left-to-right sequential execution is designated by the fat horizontal line on the top. We also assume that "Instructor", "Mom" and "Dad" represent the corresponding phone numbers. The juxtaposition of the Instructor box on top of the Phone box is a short-hand notation for mailing arrow line between the two.

The Help vip is a visual shell command for the child. A double-tapping of any copy of the Help icon by pen activates the automatic dialing sequence. When the phone is successfully connected, the corresponding Phone vip is highlighted to show its activation, and after the voice conversation is completed, the child enters the hang-up gesture onto the Phone vip to disconnect the phone. This terminates both the Phone vip and the Help vip.

5. Conclusions

We have demonstrated that the Hyperflow visual language is potentially capable of providing a uniform software development tool for different levels of programming. The Timer vip and the Modem vip are on the level of programming device drivers. The Phone vip is on the level of toolbox development. The Help vip is an example of end user shell programming.

In order to make Hyperflow a serious system programming language, development of an efficient compiler is crucial. Also important is the capability of multi-lingual vip specification. Until a Hyperflow compiler is constructed, it may be more practical to use a traditional text-based language such as C for the Timer and Modem vips, even though we used a visual specification (in Hyperflow's syntax) for the purpose of demonstration.

References

[Campbell 91] Campbell, J. P. Jr., Tremain, T.E., and Welch, J.C. "The Federal Standard 1016: 4800 bps CELP Voice Coder," *Digital Signal Processing*, vol 1 (1991), pp 145-155.

[Hils 92] Hils, D. D. "Visual Languages and Computing Survey: Data Flow Visual Programming Languages," *Journal of Visual Languages and Computing* 3:1 (1992), pp 69-101.

[Kimura 86] Kimura, T.D., Choi, J.W. and Mack, J.M. "A Visual Language for Keyboardless Programming," Technical Report WUCS-86-6, Department of Computer Science, Washington University, St. Louis, June 1986.

[Kimura 92a] Kimura, T.D. "Learning Math with Silicon Paper," Technical Report WUCS-92-12, Department of Computer Science, Washington University, St. Louis, February 1992.

[Kimura 92b] Kimura, T.D. "Hyperflow: A Visual Programming Language for Pen Computers," submitted to 1992 IEEE Workshop on Visual Languages, Seattle, Washington, September 1992.

[NeXT 90] "Interface Builder", NeXTStep Concepts Manual , NeXT Computer, Inc., Redwood City, CA 94063 (1990), pp 8-1 to 8-108.

[Scheifler 88] Scheifler, R.W., Gettys, J. and Newman, R. *X Window System.* Digital Press, 1988.

[Shu 88] Shu, N.C. *Visual Programming.* Van Nostrand Reinhold Company, 1988.

[Unger 87] Ungar D., Smith, R.B. "SELF: The Power of Simplicity," *OOPSLA'87 Proceedings.* Published as *SIGPLAN Notices*, 22:12 (1987), pp 227-241.

[Van Dam 91] van Dam, A. Keynote Speech, at the Forth Annual ACM Symposium on User Interface Software and Technology, Hilton Head, SC, November 1991.