Towards Programming for the Non-Technical

SuTe Lei
Department of Computer Science
University of Texas at Dallas
sxl015120@utdallas.edu

The tasks of software development have traditionally been done by highly-skilled and well-trained computer programmers. A solid educational background is often required to understand programming languages used in the development. For people who are educationally disadvantaged, the task of computer programming seems remote. It might appear that there is no strong reason for those people to partake programming activities. However, in today's volatile business environment, giving programming capabilities to the educationally disadvantaged might be an economically viable way to upgrade the competitiveness of a business entity. To achieve this, there is a need for an easy-to-use programming environment that has the following assumptions for its users:

- no formal trainings in computer programming.
- minimum educational backgrounds
- of different ethnicity and are unable to communicate using the commonly spoken language

Essentially the desired programming environment should enable non-technical people to perform programming tasks. The main challenge for developing such a system is making the system assessable to anyone regardless his/her technical background as well as educational background. One natural way to fulfill this need is providing an easy-to-understand user interface and capability of illustrating programming logic in a plain and simple way. A computer software program for young children will be a perfect example. This type of program offers children at very young age interesting ways of learning how to match shapes, mix colours, build and rebuild blocks. Through such a software system, children learn the concept of cause and effect relationships. For example, erasing words entered is done by clicking on an 'electric fan' which blows all letters away with sound once it is clicked.

A highly intuitive visual approach offers an effective way for educating novice users of a system. The visual language, Vedo-Vedi [3], is one example of visual approach being used for children's multilingual communication in the Internet. The focus of our work, however, is to develop a visual programming environment for commercial applications. The work is motivated by our experience in developing software systems for the lumber industry. Most of the front line lumber workers have minimum education in computer technology. However, they are sometimes given programming tasks on rearranging label printing or sorting sequence. Customized reports might also need to be generated. Our idea is developing a GUI-based programming system that allows users to build a software program simply by creating user interfaces and defining navigation flows. Figure 1 shows the organization of the proposed programming environment.

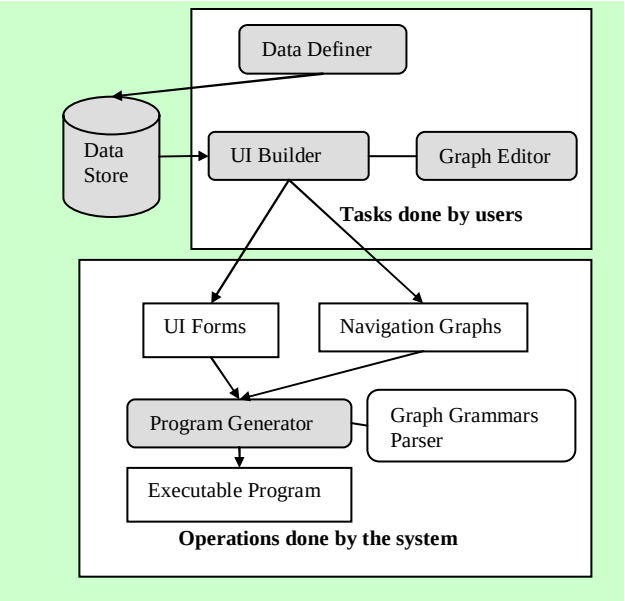

Figure 1 Organization of Visual Programming Environment.

The process of building a program involves four main steps:

1. Defining data in data definer.
2. Building UI forms and defining navigation graphs, using UI builder.
3. Parsing navigation graphs via program generator.
4. Generating an executable program.

The user participates only in Steps 1 and 2 while the system takes care of all the later steps. Data definer serves as a data entry program for inputting all possible data to be collected into a data store. The process typically starts with checking through user's existing system either electronically or on paper and inputting them into the data definer. Once the required data are defined, they will be stored into the *Data Store* which is a repository of all objects and fields for the system. Basic data accessing routines such as *create*, *delete*, and *open* are also automatically generated.

UI builder is a tool for building user interfaces. Its usages are:

- to define UI forms and set properties for each visual component.
- to define navigation graphs through visual programming and syntax-directed editing.

Figure 2 shows a prototype of the UI builder. Users can design an UI form by clicking and dropping visual components onto the form. The properties of a visual component can also be defined in the property editor. We make the interface of UI builder as close to the resulting program as possible in order to facilitate the concept of "what you see is what you get (WYSIWYG)" [2]. We believe that WYSIWYG is an appropriate paradigm for designing a programming system for novice users. Users can define exactly what they want through direct manipulation in the UI builder.

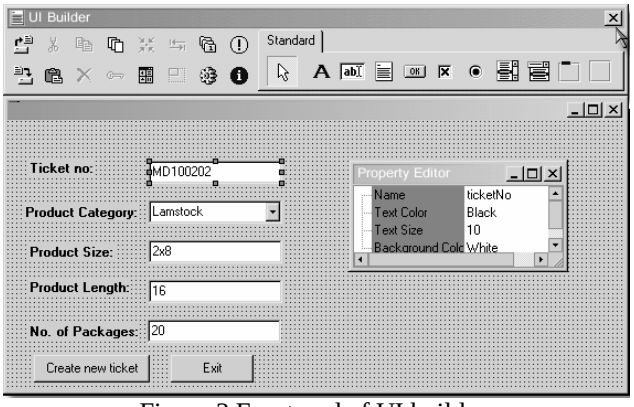

Figure 2 Front-end of UI builder.

In addition to building UI forms, users can define the execution flows of a program by drawing navigation flows in the graph editor. Figure 3 shows a prototype of the graph editor. All visual components used in the UI forms are displayed in a tree list on the top left side of the editor. System actions for data are listed in the bottom left. The system actions are the data accessing routines automatically generated during data definition process. To specify an action for a visual component, such as 'Create new ticket' button, users simply drag and drop the selected visual component onto the drawing area located on the right side of the editor, then select an action and drop it into the drawing area. The final step is making a link between these two boxes by pointing/clicking and dragging. The navigation graph in Figure 3 indicates that a ticket will be crated when CreateNewTicket button is clicked. Although we simplify the drawing process of navigation graphs, the syntax of navigation graphs is actually defined in a graph grammar formalism [4] and the drawing process is syntax-directed editing as in visual programming [1]. The advantages of using graph grammars to define the syntax of navigation graphs include the following.

- Any user-entered navigation graph can be formally verified.
- The parsing process may perform semantic execution so that the activities in a navigation graph may be simulated and/or animated as desired.

It means that the front-end tool is supported by a theoretically sound graph grammar formalism [4] and at the same time the users of the tool do not need to know how programs are generated.

Once all forms and navigation graphs are defined, basic program codes can be automatically generated by parsing navigation graphs and combining the textual format of UI forms. Basic program codes can be written in C++, Java or Delphi program languages. They are complied by a 3rd party compiler via the program generator. The entire program generating process is transparent to users. The only thing users need to do is clicking on a button to generate an executable program.

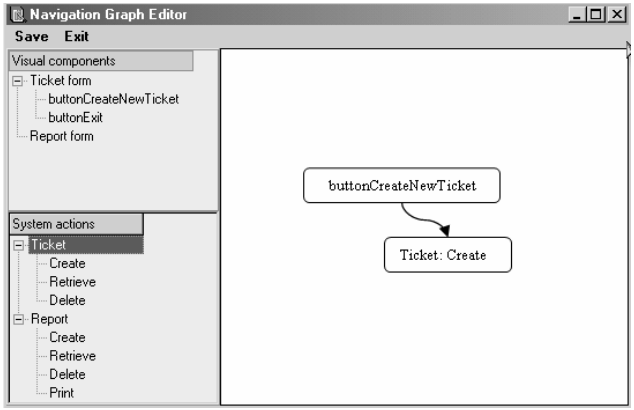

Figure 3 Graph editor for visually programming navigation graphs

The overall approach we propose in this research will provide the non-technical people to produce a structurally sound program. We believe that the proposed programming environment will benefit the educationally disadvantaged people not only in understanding the programming logics but also in creating real commercial software programs.

Reference

1. M. M. Burnett, Visual Language Research Bibliography, http://www.cs.orst.edu/~burnett/vpl.html, 2003.
2. M.M. Burnett and S.K. Chekka, FAR: An End-User WYSIWYG Programming Language for E-speak: Interim Report, TR 00-60-10, October 2, 2000.
3. S.L. Tanimoto and C.E. Bernardelli, "Introducing New Nouns in a Children's Visual Language", IEEE Symposium on Visual Languages, September 1998, Nova Scotia, Canada, pp 74-75.
4. D. Q. Zhang, K. Zhang, and J. Cao, A Context-sensitive Graph Grammar Formalism for the Specification of Visual Languages, *The Computer Journal*, Vol.44, No.3, 2001, pp. 186-200