2R-2

Smalltalk−80を用いた対話型自動配線システムの試作−II

P.ハワース　井元康宏　平山幸一

ソニー・テクトロニクス株式会社

1. Smalltalk-80

Smalltalk-80 is an object oriented programming language, and development environment. All of the software used in the development environment is available for use in applications, which substantially reduces the amount of new code that needs to be written. This includes a large amount of graphics and window management software. Smalltalk-80 is said to facilitate high productivity and reusability of code. We wished to confirm these qualities as we used them in the implementation of a experimental user interface for an autorouter.

We used the implementation of Smalltalk-80 which runs on the Tektronix 4400 series A. I. workstations.

2. Features

The user interface allows the display and modification of a printed circuit board in a standard Smalltalk-80 view (I.e. a window). This automatically makes all the usual features of a Smalltalk-80 view (Such as moving, framing and collapsing) available to the user.

All user input is via the mouse using a variety of techniques, such as menus, pop-up menus, selection of points, selection of board objects and selection of areas.

The board can contain connections, routes, pads, vias, copper areas and a board outline. These can all be input, and in most cases edited, by the user.

Routes are displayed in gray with a width that varies in proportion with the scale. Occupied areas are indicated by filled areas of gray. Gray is used because the display is monochrome, and different gray tone masks are used to show which layer a route or occupied area is on. The number of gray tones that the eye can be easily distinguished between is small, so the number of layers is limited to two. The user can also request a grid to be displayed.

The display can be zoomed in and out. Also an area of the board may be selected and displayed in a separate view. The scale in the new view is automatically set according to the size that the user selected for the new view. By repeating this, several views of different parts of the board, at different scales, can be open simultaneously. Any of these views can be used for user input.

The autorouter is run by a pop-up menu selection. As connections are routed, all views are updated. At any time, the autorouter can be stopped (also using a pop-up menu), edits carried out, and then the autorouter restarted. The user can thereby watch the progress of the autorouting, and intervene it problems seem to be developing. This capability is what makes the system interactive.

While the user is editing the board, the board data may be inconsistent (E.g. two routes on the same layer crossing). However, when a request is made to run the autorouter, all newly input data is checked for consistency, and if errors are found, they are displayed, and the autorouter will not run.

3. Program

The program structure has the usual structure for a Smalltalk-80 application that uses the standard window software (ref. 1) with slight variations. The top level decomposition has three parts: a model, which contains the application's data, a view, which carries out display functions, and a controller, which controls interaction with the user. In addition we also have the autorouter, which is in a subprogram written in C, and communicates with the model using a pipe and signals (see fig. 1). Thus the autorouter appears to the rest of the program as if it is part of the model. Although it is usual to have the view contain the display functions, in our program, the model contains much of the software required for displaying the board. This variation from the usual division of responsibilities proved to be a good decision.

4. Software

The design of the user interface commenced after a detailed specification of the interface between the user interface and the autorouter subprocess had been produced. From this the minimum requirements of the user interface could be inferred.

The first level of decomposition within the user interface was constrained to follow the standard model-view-controller pattern We just needed to decide the precise division of responsibilities between the three parts. Up to this point, design had been either conventional, or mostly predecided because we wished to use the standard Smalltalk view software.

Our lack of experience made more detailed design work difficult. We, therefore, briefly described the functions we wanted to implement, and only investigated how to implement those parts that we thought would be difficult. Further decomposition was limited to naming some classes which we thought would be needed and trying to describe their class hierarchy. Many board objects (E.g. routes, route segments, connections) could easily be assigned

Development of Interactive Autorouter Using Smalltalk -80 - II

Paul HOWARTH　Yasuhiro IMOTO　Koichi HIRAYAMA

Sony/Tektronix Corporation

classes; describing the class hierarchy was more difficult, but mostly successful.

Since design was becoming difficult, and since Smalltalk-80 is said to facilitate development by gradual improvement, we then proceeded with the implementation. We intended to decide many detailed design issues immediately prior to the implementation of the associated code. During implementation, we found that classes additional to those identified during design were required, and that the class hierarchy needed changing. In total, twenty two new classes were developed, of which five had not been identified prior to implementation.

This unconventional approach did not impede progress. Design, implementation and testing of version 1, which included nearly all the required features of the user interface, took approximately four months. (Since then, we have made numerous changes as new ideas occurred to us.) This involved one person for most of the time, and another for part of the time; both people had some knowledge of Smalltalk-80, but were still relative beginners.

A Smalltalk programmer makes constant use of parts of the Smalltalk-80 standard system software. The reusability of these parts is, therefore, proven. Reusability of new code written for an application does not, however, always occur automatically. It may require some reworking of code and class hierarchies, or design effort before implementation.

The skillful use of this large amount of standard system software requires the programmer to know its capabilities and how to use it. Thus, programming in Smalltalk-80 requires more knowledge of the development environment than a conventional high level language, but with this knowledge, productivity can be very high. Being beginners, we spent a lot of time finding out how to do things before we were able to implement them. On one occasion, several days were spent on a problem that, in the end only required twenty lines of code. The next time a similar problem occurs, we will be able to solve it very quickly.

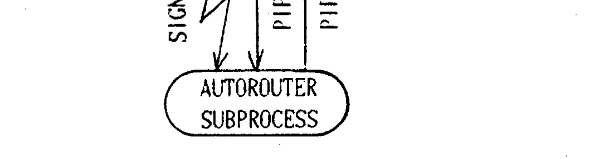

Fig1. Top Level Software Structure

Reuse of code is the most important, but not the only, reason for high productivity when using Smalltalk-80. The excellent debugging facilities also contribute greatly. The ability to compile small parts of the code, and then run the whole system immediately, was very convenient. This is especially important when, as we were, implementing by gradual improvement.

Code was tested in small pieces in order to catch bugs early. By using workspace views for the preparation of test data and execution of tests, suitable tests could be created readily. In verifying the result of a test, it was often necessary to use halt statements and the debugger. Using Smalltalk-80, complex data structures tend to be created freely and with little thought, and checking that these structure are correct could be difficult. However, by using a combination of a debugger view and chains of inspector views, it is possible to trace through these structures, usually with little difficulty. Thus, a combination of features in the development environment made incremental testing relatively easy.

Subdivision of the task between two programmers proved to be difficult. Integrating separately written code always involved changing some related software in the main body of the application, but on the other hand, Smalltalk-80's development environment made these changes easy to carry out. We expect the ability to subdivide work will improve with experience.

5. Appraisal

Maximum capacities for the number of routes, connections and the like have not been ascertained. However, the capacities and the speed of display are more than adequate for exercising the autorouter, and experimenting with different methods of user interaction.

The main problem in performance is the pipe which is very slow, and somewhat spoils the interactive capabilities. At time of writing no solution to this problem has been found. It is probably caused by the large size of the Smalltalk image and the way the data is distributed within the image; this means that the autorouter subprocess is be driven out of main memory whenever the user interface runs.

Smalltalk-80 lived up to its reputed virtues of reusability, high productivity and implementation by gradual improvement, with the slight qualification that achieving reusability of application code often required a degree of effort. A problem with dividing the task amongst team members was identified.

Smalltalk is harder to learn than a conventional high level language, but its merits soon became apparent. We confidently expect to continue increasing our productivity with greater experience.

[参考文献]

1. Ward Cunningham「Smalltalk-80によるアプリケーション・プログラムの作り方」
b i t、vol.18、No.4 1986