ソフトウェア工学　52－19
（1987・2・14）

エンジン制御用リアルタイムソフトウェアの視覚化詳細テストについて

冨永 一幸　宮尾 淳一　菊野 亨　吉田 典可
広島大学 工学部

最近，自動車のエンジン制御用ソフトウェアに要求される機能の増大に伴い，ソフトウェアの開発が困難となってきている．特に，自動車のエンジンにおいてはいわゆるリアルタイム性に関する厳しい要請があり，それをより一層難しいものとしている．筆者らは既に，エンジン制御用リアルタイムソフトウェアの設計手法として，視覚化２段階設計手法を提案している．本設計手法は抽象段階と詳細段階から構成されているが，本稿では後者についてのみ述べる．詳細段階では，先ずソフトウェアを高級言語レベルで設計（詳細設計）し，次に視覚的にプログラムのテスト（詳細テスト）を実行する．これにより，ノンプログラマに対しエンジン制御用ソフトウェアの開発を可能とする．なお，詳細テストでは，実入力データに対するソフトウェアの振る舞いを視覚化した詳細シミュレーションを行う．

A Visualized Test of Real-Time Software for Engine Controller

Kazuyuki TOMINAGA, Jun'ich MIYAO, Tohru KIKUNO and Noriyoshi YOSHIDA
Faculty of Engineering, Hiroshima University
Higashi-Hiroshima, 724 JAPAN

Embedded computers have recently been used widely in the controller of automotive engines. Generally, the severe timing constraints are needed in control software. As the functions of the controllers increase, it has become difficult to develop such real-time software.

The authors have already proposed a new approach, called a two-stage and visualized approach, to develop software. This approach consists of an abstract stage and a detailed stage. This paper presents only the detailed stage extensively and shows a prototype system VD/ECS based on the proposed approach. In the detailed stage, software is developed on a high level programming language C*, and then the simulation test is performed visually for software. By using the proposed approach, even non-programmers can develop real-time software for engine controller.

1. Introduction

Recently, embedded computers have been widely used in the controller of automotive engines. These controllers must respond within a given time period [Anderson81, Quirk85]. The controllers with this property are called real-time controllers, and the software for the controllers is called real-time software.

Conventionally, most real-time software for engine controllers is directly designed at the machine code level. Then, testing of the software is performed at the instruction level [Glass80]. Thus, there mainly exist the following three problems (a)-(c).

(a) Only professional programmers can develop the real-time software.

(b) It is inefficient to modify and improve the real-time software.

(c) Refinement of one part of the software may cause side effects on the other parts.

In order to resolve these difficulties, we have proposed a top-down and visualized approach for the development of real-time software for engine controllers [Miyao86]. The proposed approach has the following novel features (1)-(3).

(1) Top-down and modular design: Top-down and modular design approach is desired to enhance productivity and understandability [Palmer82].

(2) Visualized design: To support the non-programmers who have a good working knowledge of engines, visual design capability is incorporated [Ting84].

(3) Testing at the abstract level: Testing of timing constraints is done at the abstract level of the design step [Oki85].

The proposed method consists of two stages: abstract stage and detailed stage, and each stage is composed of design phase and test phase [Miyao86].

This paper explains the detailed stage. In the detailed design phase, a task graph G, which represents structure of real-time software visually, is translated into instruction codes. Successively, in the detailed test phase, functional test and performance test are performed visually using detailed simulation on the instruction codes.

Finally, a visual design system VD/ECS based on the proposed approach is presented.

2. Design of Engine Control Software

Most newly developed engines for automobiles are controlled by embedded computers, and for each engine, control software must be developed and/or tuned. Then, the enhancement of productivity for control software is required. Furthermore, the requirements of high power and efficient engines increase the functions, such as gas injection, ignition, EGR, turbine, automatic transmission and so on, of the control software. This makes the control software complex and costly.

This section presents a top-down and visualized design method for engine control software.

2.1 Real-Time Software for Engine Controller

A real-time system consists of a controlled system and a real-time controller (see Figure 1). In the automotive engine, a controlled system and a real-time controller correspond to an engine and an engine controller, respectively. The real-time controller receives an internal state of the controlled system as input data, and then returns output data to the controlled system.

The special features of engine controllers are summarized as follows.

(a) Most data input and data output are triggered by either irregular interrupts from

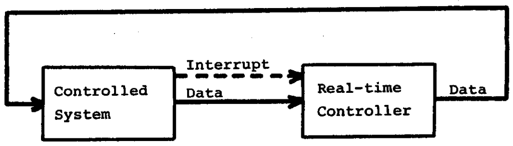


Figure 1 Real-time system.

the controlled system or regular interrupts from an interval timer. Thus, the control software should provide capability of flexible interrupt control.

(b) The controller must respond within a given time period to the interrupt. If the timing constraint is not satisfied, the engine may be stopped. Then, the engine controller is a hard real-time system in this sense. In addition, the time periods are not always the same for each task. Thus, the controller should process tasks in the order of the time period.

2.2 Two-stage and Visualized Approach

The proposed new method pays attention to the top-down design and visualization (see Figure 2). The idea of 'visualization' is introduced into the whole design phases.

The abstract design phase constructs a task graph, which represents the structure of real-time software. The task graph enables us to design abstract structure of controller visually (see Figure 3).

Then, the abstract test phase checks whether the task graph and specifications satisfy the followings or not.

(1) For each task, necessary input and output data are specified correctly.

(2) Performance constraints, especially timing constraints, are satisfied.

In the test (2), the execution time for each task must be estimated. Since the application of the real-time software is limited to the automotive engine controller, the execution time can be estimated with sufficient accuracy.

In the detailed design phase, the task graph is translated into executable object code. Then, in the detailed test phase, functional test and performance test are performed visually using detailed simulation on the instruction code.

In this paper, we concentrate our consideration only on the detailed design and the detailed test with visual capability.

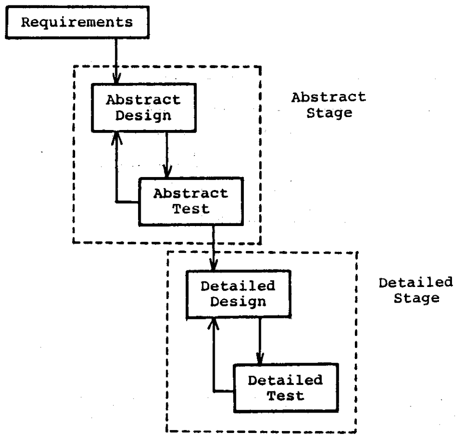


Figure 2 Two-stage and visualized approach.

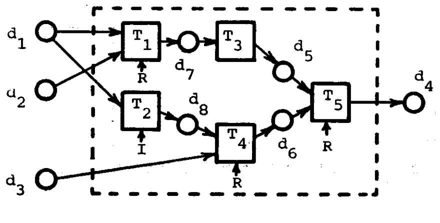


Figure 3 Task graph.

2.3 Hierarchical Structure of Controller

In the following, a real-time software consists of a set of tasks $T^*=\{T_1, T_2, \ldots, T_m\}$ and a set of data $D^*=\{d_1, d_2, \ldots, d_n\}$. Each task T_i has one or more input data and an output datum. Then, tasks are classified into the following three categories.

(1) T*(INTX): a set of tasks, for which the execution is requested by an irregular interrupt from the controlled system.

(2) T*(INTR): a set of tasks, for which the execution is requested by a regular interrupt from the interval timer in the controller.

(3) T*(UPD): a set of tasks, for which the execution is requested by the occurrence of the update of input data.

The proposed real-time controller is hierarchically constructed as shown in Figure 4. It consists of the following components.

(1) Tasks: User defined tasks, which are specified by a task graph G=(V,E).

(2) Real-time monitor: A so-called

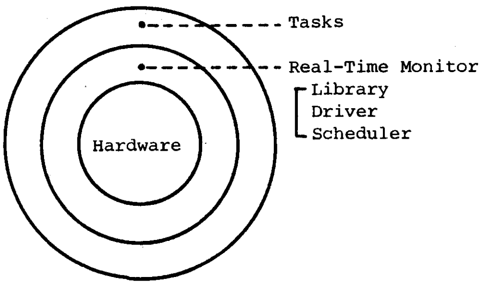


Figure 4 Real-time software for engine controller.

mini-operating system, which includes the following routines.

(a) Library: Standard arithmetic operations, and special purpose routines to be defined by user.

(b) Driver: Task management routines, which include task state management, interrupt and internal timer handling.

(c) Scheduler: Deadline scheduler, which assigns a processor to each task under the policy of deadline scheduling.

(3) Hardware: Processors, which are available in the real-time controller.

This structure enhances physical independence between hardware and task. Moreover, the functions induced by multi-tasking and deadline scheduling enhance logical independence among tasks. Then, each task may be developed independently.

3. Detailed Stage

As shown in Figure 2, the detailed stage consists of two phases: the detailed design phase and the detailed test phase. In this section, each phase in the detailed stage is described.

3.1 Detailed Design

The flow of the detailed design phase is shown in Figure 5. The detailed design phase consists of three processes: task graph translation, editing of C* source code and C* compilation. At first, a task graph is translated into a source program in C*. C* language is a programming language which is specially developed for the automotive engine control software, and the syntax of C* is similar to C language [Tominaga87]. The essential difference is adopting, in C*, the static assignments of variables and functions. This saves the time needed for function calls.

Then, the source program in C* is compiled by the C* compiler into the instruction code which is executable on a hardware. The instruction code is passed to the detailed test phase.

If any wrong part is detected in the detailed test phase, the designer can modify the C* program using C* editor. The display screen of C* editor is shown in Figure 6. The modification and improvement of the program is carried out by using windows.

In this system, even the detailed design is carried out on a high level programming language. Then, it is not necessary for users to have knowledge of the assembly language.

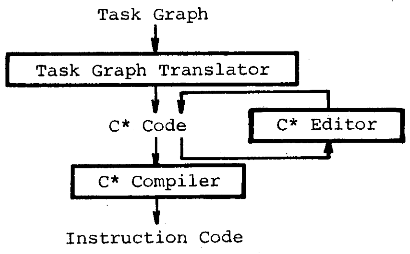
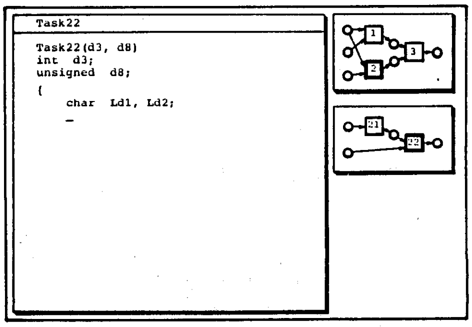


Figure 5 Flow of detailed design.


```
Task22

Task22(d3, d8)
int  d3;
unsigned  d8;
{
    char  Ld1, Ld2;
    _
```

Figure 6 Display screen of detailed design.

3.2 Detailed Test

Detailed test phase performs the functional test and performance test. These tests are carried out by using the actual input data, and executing the instruction code produced in the detailed design.

The flow of the detailed test is shown in Figure 7. The test is performed by two modules: detailed simulator and detailed tester. At first, the detailed simulation is carried out for the instruction code of the real-time software and actual input data. The behavior of the software is simulated visually by executing the instruction code under the real-time monitor. Details of the simulation are discussed in Section 4.2.

Detailed simulator outputs the history of executed tasks and their output data. The result of detailed simulator and the expected value data are inputted to detailed tester. Then, the tester compares the results with the expected value. The expected value is obtained based on the specification of the software in the abstract design phase. If the result agrees with the expected value, the software is recognized to have satisfied the requirements.

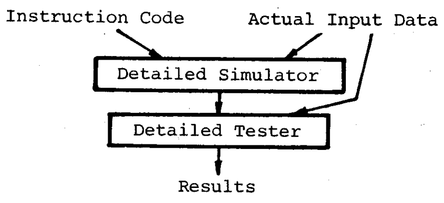


Figure 7 Flow of detailed test.

4. Visualized Detailed-Test

Detailed test is performed by visualized simulation. In this section, objectives of detailed test are described. Next, the novel features of the detailed simulation are explained using an example. Then, the detailed simulator is presented.

4.1 Objectives of Detailed-Test

In order to guarantee the properties of the real-time software, testing of the real-time software is performed both at the abstract level and the detailed level.

In the abstract test phase, software is checked on the task graph whether the timing constraints are satisfied or not. However, the execution time of each task cannot be known exactly in advance, since it is varied depending on input data. Thus an approximate test, which is based on the expected execution time, is carried out.

In the detailed test phase, the simulation is executed using the actual input data. Thus, the followings can be performed precisely.

(1) Functional test: The real-time software correctly realizes the functions.

(2) Performance test: The real-time software satisfies the timing constraints.

4.2 Visualized Simulation

The detailed simulation provides the following three novel features (A)-(C).

(A) Precise simulation using actual input data.

(B) Performance simulation with checking of timing constraints.

(C) Visual and interactive simulation enhancing usability of the system.

In order to achieve the above three features, the detailed simulator provides the functions described below.

(1) Actual input data: Simulation is carried out on the instruction code using the actual input data (feature (A)).

(2) Display of task graph: To confirm the proper behavior of the software, the state of each task is displayed on the task graph (features (B) and (C)). Each task has one of three states: Dormant, Ready and Running [Miyao86].

(3) Interactive input and output: Data on the task graph, deadline and interrupt data are interactively entered and modified on the

display screen (features (B) and (C)).

(4) Display of C* code: C* code for each task can be displayed on the display screen in order to edit the code (feature (C)).

(5) Task simulation: The behavior of each task is simulated to check whether the task correctly computes or not (feature (A)).

(6) Log of data: Logs of prescribed data are recorded whenever the data are updated. These are used later to analyze the behavior of the software (feature (A)).

(7) Simulation from arbitrary logical time: The simulator can go back to the previous state by using the record which stores the all data periodically (feature (C)).

(8) Trigger: The system allows users to specify conditions and actions, which confirm correctness of the software. When one of the conditions is satisfied, the prescribed actions will be invoked (feature (B)).

(9) Display of task execution: All executions of tasks that are executed at current time are displayed (features (B) and (C)).

(10) Estimation of execution time: Maximum and minimum execution times for the given sequence of tasks on the task graph are estimated (feature (B)).

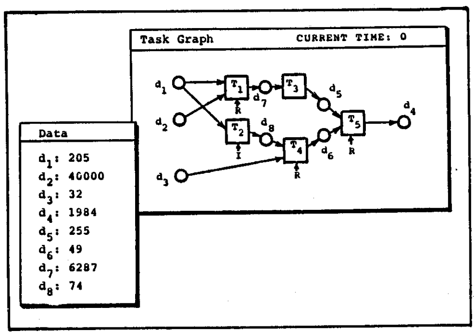
These functions are implemented by using the multiwindows as shown in Figure 8.

[Example 1] Let us consider the engine control software with the task graph in Figure 3. Figure 8 shows the display screens of the detailed simulation.

In Figure 8 (a), the state of each task is displayed using a color on the task graph. The colors green, violet and red correspond to the states dormant, ready and running, respectively. Furthermore, the values of data d_1, d_2, ..., d_8 on the task graph are displayed in another window.

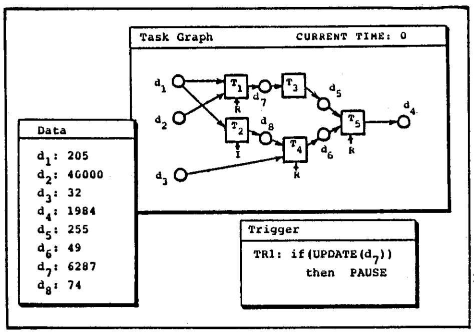
Assume that T_5 exceeds the deadline and d_5 has an illegal value. On the task graph, tasks, which exceed the deadline, are displayed using a color yellow.

The faults analysis and the improvement are performed as follows. At first, to



Data
d_1: 205
d_2: 40000
d_3: 32
d_4: 1984
d_5: 255
d_6: 49
d_7: 6287
d_8: 74

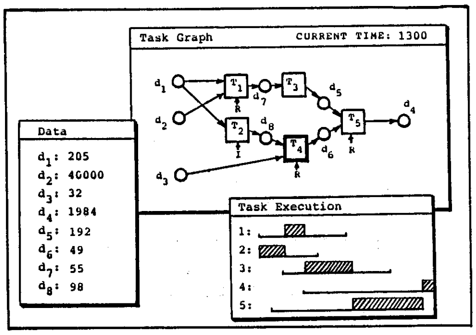
(a) Display of task graph and data.



Data
d_1: 205
d_2: 40000
d_3: 32
d_4: 1984
d_5: 255
d_6: 49
d_7: 6287
d_8: 74

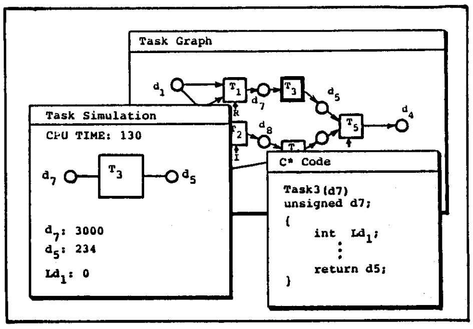
Trigger
TR1: if(UPDATE(d_7)) then PAUSE

(b) Trigger function



Data
d_1: 205
d_2: 40000
d_3: 32
d_4: 1984
d_5: 192
d_6: 49
d_7: 55
d_8: 98

(c) Task execution



d_7: 3000
d_5: 234
Ld_1: 0

```
Task3(d7)
unsigned d7;
{
    int Ld1;
      :
    return d5;
}
```

(d) Task simulation and C* code

Figure 8 Display screen of detailed simulation.

analyze T_3 and T_5 in detail, let us observe the task execution from the logical time when d_7 is updated. To do so, trigger function is available (see Figure 8 (b)). In the trigger expression, the condition clause UPDATE(d_7) becomes true when d_7 is updated. And the action of this trigger is PAUSE, which implies the pause of the simulation.

If the simulation is paused, then the users can invoke the function (9) (see Figure 8 (c)). In this case, it is found that the execution T_5, which is invoked immediately after T_3, exceeds the deadline. These two tasks have too long execution time compared with the estimation time.

Therefore, the detail behaviors of T_3 and T_5 must be further analyzed by the task simulation and the display of C* code (see Figure 8 (d)). In the task simulation, the values of local data in T_3 can be displayed. By using these, the wrong part of task T_3 can be detected. Then, the modification of the program for task T_3 is performed by using the C* editor. The same operations are applied to T_5.

4.3 Detailed Simulator

Detailed simulator is composed of the following six modules (see Figure 9).

(1) Screen Manager: This module is responsible for the visual and interactive facilities. Windows and a mouse are managed by Screen Manager.

(2) Detailed Simulation Controller: This module interprets the command from Screen Manager, and controls all modules in the simulator.

(3) Visualization Module: According to the command from the Detailed Simulation Controller, this module performs the visualized functions such as display of task graph, C* code and task execution.

(4) Simulation Data Module: This module maintains output data of the simulation to record the log of data.

(5) Trigger Module: This module manages the trigger conditions specified by the user, and

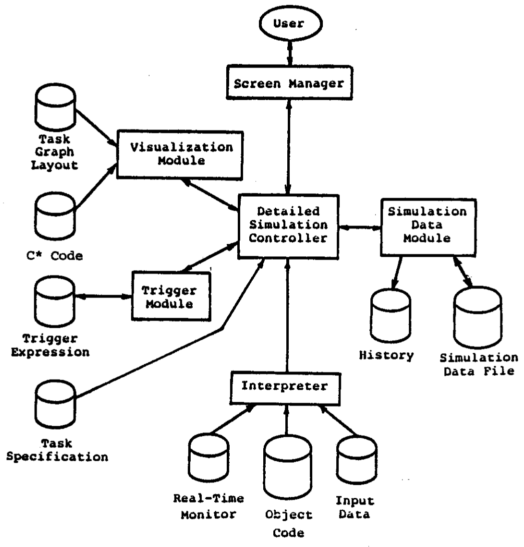


Figure 9 System configuration of Detailed Simulator.

requests the controller to perform the prescribed actions if necessary.

(6) Interpreter: Interpreter performs the software simulation by interpreting the machine instruction code.

5. Visual Design System

Based on the proposed design method, a prototype system VD/ECS (Visualized Design System for Engine Control Software) is now under development. The system configuration is shown in Figure 10.

This system consists of nine components. These components are classified into four groups corresponding to each phase.

(1) Abstract design: Abstract design phase is carried out by using Specification Manager and Task Graph Manager. These managers specify the requirements visually and construct the task graph, respectively.

(2) Abstract test: Abstract Simulator performs the simulation of the software on the task graph. Then, Abstract Tester checks whether the software satisfies the timing constraints.

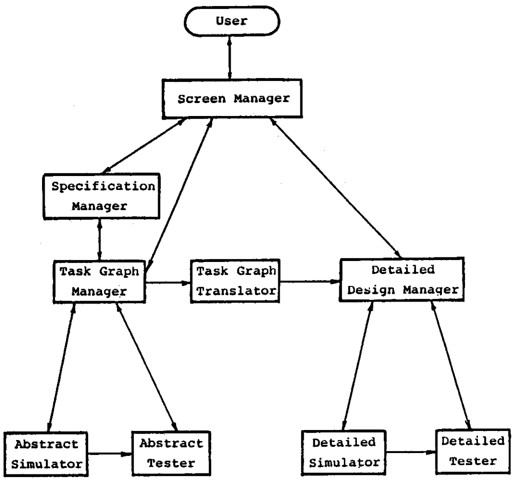


Figure 10 System configuration of VD/ECS.

(3) Detailed design: Task Graph Translator is responsible for translating the task graph into C* code. Then, the editing and compilation of C* code are performed by Detailed Design Manager.
(4) Detailed test: As mentioned in Section 3.2, detailed test is performed by Detailed Simulator and Detailed Tester.

The prototype system VD/ECS is implemented on the personal computer NEC PC-9801 by using C language of PC-UX.

6. Conclusion

This paper has discussed a new approach to develop engine control software. Especially, the details of visualized test are explained. Then, the visualized design system VD/ECS based on this approach is presented.

We are now planning to implement the following extensions.
(a) Support for generating input data: In the system VD/ECS, input data for the detailed simulation are numeric data. Then, users must produce numeric data based on specification charts of an engine. Therefore, tools, which generate numeric data from charts, should be developed.
(b) Visualization of simulation results: The detailed simulator produces histories of data in the software. The histories can be displayed visually by plotting value along time axes. This makes the analysis and the improvement of software more easy.

Acknowledgements

The authors wish to express heartfelt thanks to Mr. S. Shibahashi, Manager, Mr. K. Kobayashi, Assoc. Manager, and Mr. K. Iida, Electrical and Electronic Division, Mazda Motor Corporation. The authors would also thank Mr. S. Amako, Hiroshima University for his help on the implementation of the system VD/ECS.

References

[Anderson81] Anderson, D.A.: 'Operating systems,' Computer, 14, 6, pp.69-82 (1981).
[Glass80] Glass, R.L.: 'Real-time: The 'Lost World' of software debugging and testing,' Commun. ACM, 23, 5, pp.264-271 (1980).
[Miyao86] Miyao, J., et al.: 'Visualized testing of software for a real-time controller,' Proc. Second IEEE Workshop on Visual Languages, pp.117-124 (1986).
[Oki85] Oki, Y.: 'An approach to testing for timing constraints on a real-time controller,' Master Thesis, Systems Eng. Course, Hiroshima Univ. (1986).
[Palmer82] Palmer, D.F., et al.: 'Real-time system design, sizing, and simulation using DSIGNER,' Real-time Systems Symposium, pp.205-210 (1982).
[Quirk85] Quirk, W.J.: 'Verification and Validation of Real-Time Software,' Springer-Verlag (1985).
[Ting84] Ting, T.C., et al.: 'A multi-sensory and multi-media laboratory for human-computer interaction,' 1984 IEEE Workshop on Visual Languages, pp.149-155 (1984).
[Tominaga87] Tominaga, K.: 'A high level programming language C* for engine control software,' ECS Lab. Hiroshima Univ. Tech. Rep. No.87-02 (1987), in Japanese.