Journal of Computer Science and Cybernetics, V.28, N.3 (2012), 234–244

OPTIMIZING OCCUPIED MEMORY OF EMBEDDED SOFTWARE IN THE DESIGN PHASE*

PHAM VAN HUONG, NGUYEN NGOC BINH, PHAM NGOC THANH

[1]*University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam*

Tóm tắt. Trong xu thế phát triển mạnh mẽ của công nghệ phần mềm nhúng hiện nay, vấn đề tối ưu phần mềm nhúng có vai trò quan trọng. Việc đánh giá và tối ưu phần mềm nhúng trong giai đoạn thiết kế đem lại nhiều lợi ích. Trong bài báo này, chúng tôi đưa ra phương pháp mới tối ưu bộ nhớ chiếm dụng của phần mềm nhúng trong giai đoạn thiết kế dựa trên sắp xếp Tô-pô (thứ tự Tô-pô), ngôn ngữ miền xác định (DSL) và công nghệ sinh mã T4. Mỗi chương trình được đặc tả theo một chuỗi các tác vụ và quan hệ giữa các tác vụ. Theo đó, chương trình được biểu diễn bằng một đồ thị phụ thuộc của chuỗi tác vụ. Mỗi đỉnh trong đồ thị biểu diễn một tác vụ, mỗi tác vụ gồm các thông tin đặc tả như tên, đầu vào, đầu ra. Mỗi cạnh biểu diễn sự phụ thuộc giữa các tác vụ. Chương trình có thể thực hiện theo các thứ tự Tô-pô khác nhau mà không làm thay đổi kết quả nhưng mức chiếm dụng bộ nhớ của chương trình sẽ khác nhau trên các thứ tự Tô-pô. Từ đồ thị phụ thuộc, chúng tôi tìm mọi thứ tự Tô-pô và xây dựng hàm đánh giá mức chiếm dụng bộ nhớ của chương trình theo thứ tự Tô-pô để chọn chuỗi Tô-pô có mức chiếm dụng bộ nhớ thấp nhất.

Abstract. Nowadays, the optimizing embedded software plays an important role in the development of embedded software technology. The evaluation and optimization of embedded software in the design phase bring various benefits. In this paper, we propose a new method to optimize the occupied memory of embedded software in the design phase based on DSL, T4 and Topological sort. A program is specified as a chain of tasks and the relationship between the tasks. The program is expressed by the dependence graph as a directed graph. Each node in the directed graph describes a task, which consists of specification information such as name, input, output. Each edge describes the relationship between two tasks. The program working by order of tasks in the different Topological orders does not change the result, but the occupied memory and performances are different. From the dependence graph, we can find many topological orders, and each of them will have amount of occupied memory in difference. Therefore, we built a memory evaluation function to find the topological order that has the smallest amount of occupied memory.

Keywords. Embedded Software, Optimization, DSL – Domain Specific Language, T4 – Text Template Transformation Toolkit, Topological Sort, Topological Order, Dependence Graph, Chain of Tasks.

*This research is partly supported by a VNU scientific project (group A) for 2012-2013.

1. INTRODUCTION

In the development of information technology, embedded technology is a very important technology. The embedded system appears in almost fields of social life. Along to the hardware technology, embedded software engineering also has been studied and deeply developed. The embedded devices are limited on CPU, memory space, battery life [2,13]. Thus, the research on optimizing the embedded software is especially important.

Embedded software optimization is done in the different levels such as the design level, the source code level, the compiler level and the run-time level [11,14]. Although optimization in the design phase is a new approach, but it faces many challenges and it brings more benefits than the behind phase optimization method. This method is often based on the model driven software engineering. Moreover, software engineering based on DSL and T4 has been widely developed, especially, in specific domain such as embedded software, embedded systems [12,14]. The advantages of DSL and T4 are flexibility and strong code generation. In this paper, we propose an occupied memory optimization method for the embedded software based on Topological sort, DSL and T4. We aim the following objectives: modeling the embedded software and generating the parameters from the model automatically; getting all of Topological orders from a dependence graph and evaluating the amount of occupied memory, under each Topological order to select the Topological order that occupy the smallest memory. First, we define a DSL and build the meta-model for modeling the embedded software by a dependence graph. The dependence graph is a directed graph that consists of a set of tasks and the relationship among tasks. Second, from this graph, we use the T4 to get information from the model and transform this information into the mathematical expression of the dependence graph. Then, we find the different Topological orders. Each Topological order describes the execution order of tasks. The execution of a program by any Topological orders also reaches the same result but there are different sizes of occupied memory. Finally, we construct an occupied memory evaluation function for each Topological order to find the best Topological order.

The rest of paper includes following parts: Section 2 – Related work; Section 3 – Optimizing the memory of embedded software based on Topological sort; Section 4 – Develop the framework of DSL, T4 and implement the program that supports optimizing; Section 5 – Experimental; Section 6 – Conclusion and future work.

2. RELATED WORK

In recent years, there are few authors using the DSL to design a model of embedded system. For example, the research [4] defined DSL and developed the framework to specify and design real-time embedded system. In this paper, [1] the authors also study and develop the DSL for hardware-software co-design based on FPGA. There are some researches on embedded software optimization in the design phase such as the research [2,17] on "Performance optimization on mobile trade-off with the battery life time based on model driven engineering". It developed a DSL, and then used an Eclipse framework to develop mobile software structure, generate the code simulating the functions and run them to evaluate performance and balance the battery lifetime. The researchers [3,16] proposed "Mobile application optimization based on model driven engineering and code generation for production lines". According to the approach of embedded software and embedded system optimization based on DSL, we have proposed two

optimization methods such as "Hardware-software co-design to optimize embedded system in the design phase based on DSL" and "Embedded software performance optimization in the design phase based on DSL [5] and code generation" [6]. In these investigations, we defined two DSL to model and integrated T4 to generate code from models automatically. Their experimental results are prospective.

On the other hand, although DSL and T4 are applied widely to model and generate code for software, they can not be applied to optimize embedded software by evaluating the models directly before. Because it is hard to evaluate performance, memory size, power consumption in the design phase. Our approach in this paper is to optimize the occupied memory of embedded software by evaluating the model directly.

3. OPTIMIZING THE MEMORY OF EMBEDDED SOFTWARE BASED ON TOPOLOGICAL SORT

3.1. The dependence graph of embedded software

Normally, the structure of a program has only the function main as an entry point. Other functions are called from the function main. With the object-oriented programming language, in order to implement the program executed directly, the program needs to have a single class that contains the function main declared by public and static. Therefore, we can describe the program design as a kind of dependent graph, in which the node represents a task, and the edge represents a relationship between two tasks. Each task will be implemented as a function in the later phase and it includes name, return type and a parameter list. Tasks are dependent or independent. Thus, a program is expressed by the dependence graph of task and is defined by the formalism in Eq. (1).

$$G = \{T, E\} \; with \; T = \{t_i | i = 1..N\} \; and \; E\{e_{ij} = (t_i, t_j); i, j = 1..N\}, \tag{1}$$

where:

- t_i is the node of graph and e_{ij} is the edge from t_i to t_j,
- Each node t_i corresponds to a task,
- Each edge e_{ij} shows that t_j is only executed when t_i finishes,
- N is the number of tasks.

3.2. Topological sort on the dependence graph

As described in the previous section, the program includes a set of tasks and it is represented by the dependence graph. With the same set of tasks but different executing order will affect to the amount of occupied memory and performance. Also from this task set, there are many different implementations following to the different orders of tasks that satisfy the dependency graph. These implementations do not change the program execution result. This is the Topological order on a directed graph. Therefore, from the dependence graph, we can

find many execution ways of the program and each Topological order presents an execution way.

3.3. The memory evaluation function of the Topological order

From the dependence graph, there are many Topological orders presenting the different execution order of Tasks in the program. Thus, to evaluate the Topological order that has the best effectiveness of memory usage, we will build the evaluation function that is used to calculate the amount of occupied memory when the is executed program under the Topological order. We analyze the process of memory allocation and layout of memory in the program execution. Figure 1 shows the layout of memory during the program execution. The static memory space is allocated when the program is loaded from the secondary memory. It contains the static data and source code. The stack memory space is to store local variables, parameters for each function. After the function ends, the memory frame allocated for the function is recovered. The heap memory space contains elements allocated and recovered dynamically during executing the program. It also contains objects created. With the program that consists of a main program and sub functions, the process of automatic allocation and automatic release of memory during the execution time is as shown in Figure 2.

Suppose that the program has N tasks and t_i denotes for task that has the order i^{th} in the Topological order. t_i has the returned data type r_i. And, we also suppose that the program is not recursive and it is executed sequentially on the single CPU system. When executing t_i, local variables and parameters of t_i are allocated in the stack memory space and it will be released when t_i returns. After the task is done, the returned result must be stored in the local variables of the function main. These variables are allocated in the stack memory when they are declared and assigned the returned value of the task. These variables are released when the program finishes. Although the execution time of a task does not depend on the location of the task, but the resident times of these local variables are different. So in the different Topological orders, the amounts of memory occupied are also different. Without loss of generality, we assume that the execution time of each task is a time unit. Then, to store returned result from t_i, we need to declare a local variable of the main function and allocate memory for this variable. And this variable will be resident in memory until the main function finishes. Therefore, the amount of occupied memory of the variable depends on the type of data returned by t_i and the order of tasks. Moreover, by the difference of the amount of occupied among the Topological orders is only caused the local variables of the function main that contains the returned result. Because the other memory area used when executing a sub function is withdrawn when the sub function finished. So, we suppose that the amount of stack memory occupied by t_i is equal to $(N - i)$ size(r_i). From analysis of occupied memory caused by each task, we build the occupied memory evaluation function in an execution under the Topological order as in Eq. (2):

$$f = \sum_{i=1}^{N} (N - i) \times size(r_i), \tag{2}$$

where:

- f is the evaluation function of the amount of memory occupied,

- N is the number of tasks of the program and it is also the number of tasks in the Topological order,
- r_i is the returned data type of t_i.

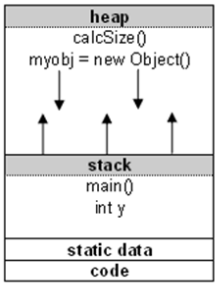

Figure 1. Layout of memory in the program execution

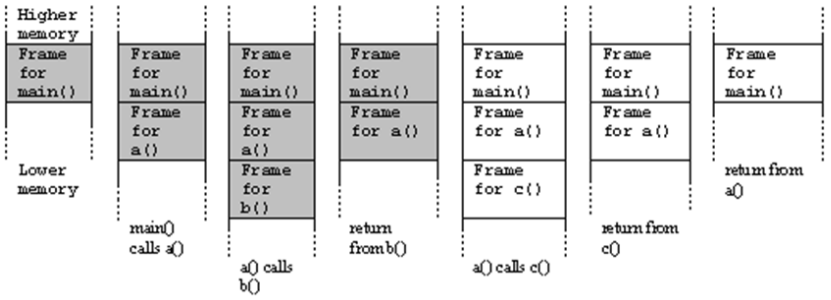

Figure 2. The Stack memory space in the program execution

3.4. Selecting the optimal Topological order

When the program is executed under any Topological order of tasks, the total size of memory used by the program is the same but the resident time in memory is different. Eq. (2) is used to evaluate the amount of memory occupied during the program execution on a Topological order. The chosen Topological order is the chain having the minimal f. To find the best Topological order, we implement a simple algorithm: get the input as a set of Topological orders in Section 2.3; browser and calculate f for each chain, and select the best Topological order on which the f is minimal.

The algorithm used to find a Topological order from directed graph has a polynomial complexity $O(|T| + |E|)$ [15], with T is the set of tasks, E is the set of edges in the definition (1). However, to find all Topological orders in the set T, the complexity is $O(|T|!)$. Here, we propose an algorithm based on the main idea such as an order chain satisfies the set of edge E is a Topological order. The algorithm is as follows:

For each c_i order chain in factorial of N chains {
    Set the variable *isTopo* to true
    For each egde $e_{j,k}$ in the set E {
        If t_j is follow t_k in c_i then {
            Set variable *isTopo* to false
            break
        }
    }
    If *isTopo* is equal true {
        Add c_i to the set of Topological orders

```
    }
}
```

4. DEVELOPING THE DSL AND T4 FRAMEWORK FOR OPTIMIZATION

To implement the methodology of memory optimization based on Topological sort mentioned in the section 2, we build a framework that enables to specify and construct a model of the dependence graph of tasks based on DSL. We build the T4 text template to transfer to the mathematical expression of the dependence graph from the model of the dependence graph. Then, we implement the program to generate all Topological orders of tasks and select the optimal Topological order that has the smallest occupied memory during the execution time of the program.

4.1. Defining the DSL and building the meta-model file

To specify a model of the dependence graph visually, we first define a Domain Specific Language for optimizing the memory based on Topological sort named by OMTS. Then, based on the Visual Studio.NET 2010 SDK, we build the meta-model of OMTS. With the meta-model built, designers can create a model of a dependence graph in the graphical interface. The process of DSL definition and meta-model construction includes the following steps:

- Define the logical components: Process class, Task class, Comments class, Rules, Constrain, etc.
- Define the shape symbols corresponding to each logical component above. The symbols are to design models in the graphical interface after the DSL.
- Create the XML file of the meta-model that is to store the logical class definitions, the shape class definitions and the mapping between the logical classes and the shape classes.

4.2. Transforming the model into a mathematical expression of the dependence graph

To optimize memory automatically after designing the model of a dependence graph, we define the T4 Text template to transform automatically into a mathematical expression from the model of the dependence graph of the tasks based T4 technology. T4 is a powerful code generation technology, which allows automatic code generation based on the XML file of the meta-model and the XML file of the actual model. The idea of the T4 code generation technology is the reading the XML file of the actual model and the XML file of meta-model that defines the logical components, the shape components and the mapping of them to analyze and generate the accordance code in with the text template.

4.3. Implementing the program to optimize memory usage based on Topological sort

In this section, we implement a program to find the optimal amount of memory occupied during the execution time. This program get the input that is the dependence graph of the tasks extracted from the model as described in the section 4.2. Then, we implement the Johnson Trotter algorithm [10,15] to retrieve all permutations from the set of tasks (nodes of the graph). Next, we implement the algorithm finding all Topological orders of tasks in the set of all permutations. We also build the evaluation function of the memory occupied of each task that depends on the order of the task in the chain and build the evaluation function of the amount of memory occupied for each Topological order of tasks. Browse and calculate the value of the evaluation function from the Topological orders to select Topological order having the smallest memory occupied. The result of optimization and the chart of amount of memory occupied from all Topological orders are shown in Figure 3.

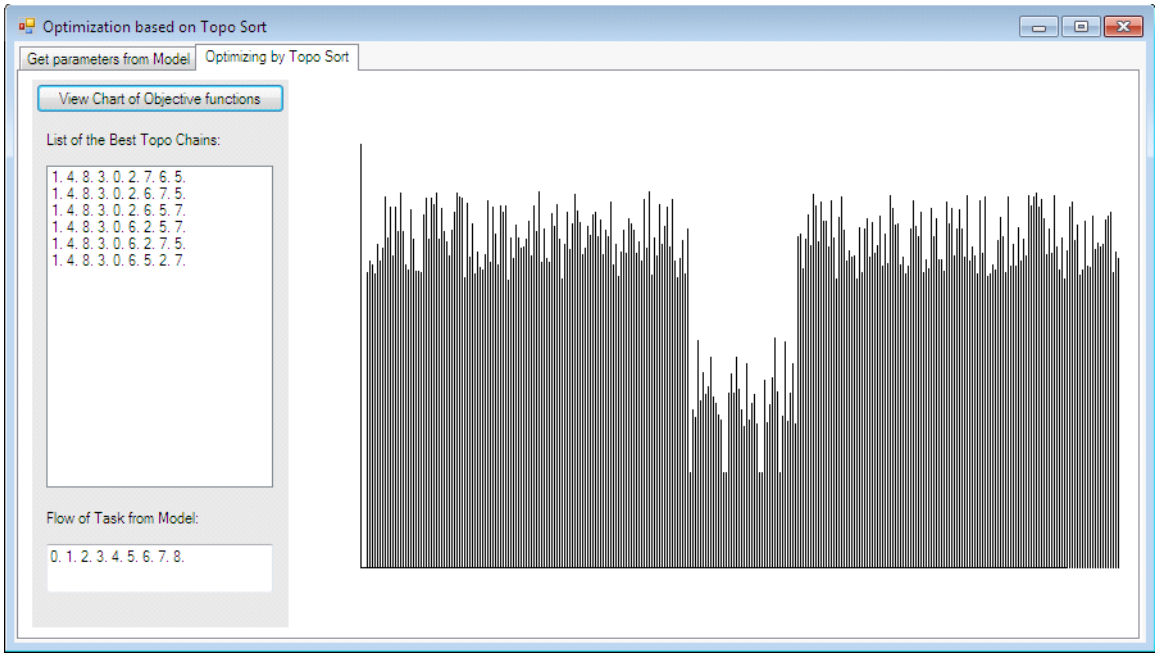

Figure 3. The best Topological order and the graph of memory usage of all task chain

5. EXPERIMENT

In this experiment, we carry out in two phases. First, we use the framework of DSL, T4 and Topological sort to model the dependence graph of task set in an application. Our framework transforms the model into the mathematical expression of the dependence graph. Then, we execute the optimal program to find the Topological order that has the minimal occupied memory size during the execution time of the application. Second, we implement

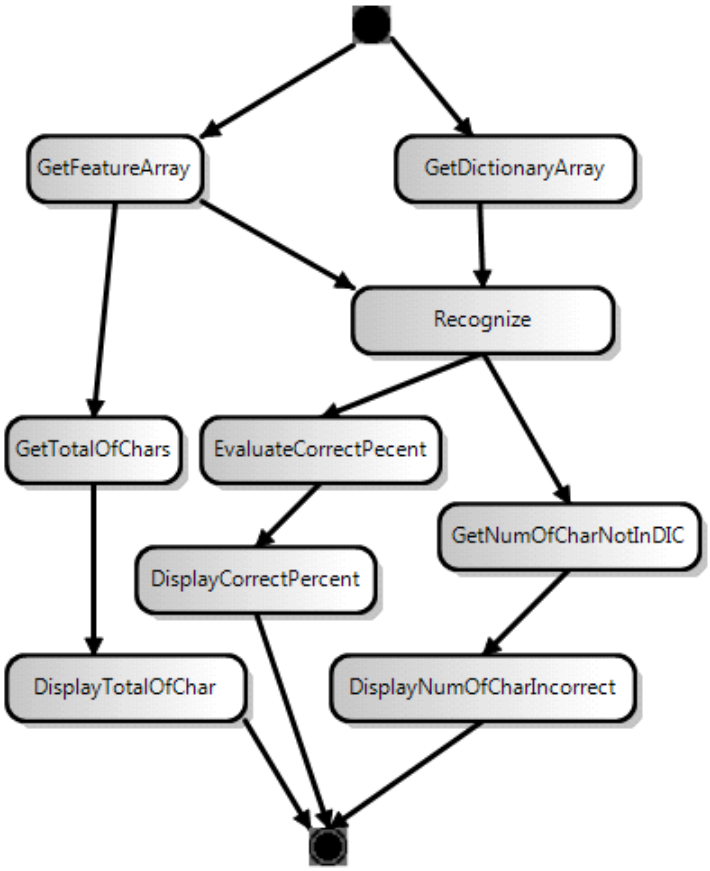

Figure 4. Task flow of the Nôm character recognition module

this application under the best Topological order of tasks. Each task is implemented as a function. The functions are called in the main function. We change order of functions under the different Topological orders but unchange content of functions to reach to the different programs. Second, we run these programs to do statistic of occupied memory amount of each program. In order to do the experiment, we use the Nôm character recognition module that is the part of the Nôm character process system developed by our research group [7,8].

In the first phase, we create a model of the dependence graph of tasks in the Nôm character recognition module. This model is shown in Figure 4. We transform this model into the mathematical expression of the dependence graphs and do optimization. The results of optimization at the model level are shown in Figure 3. When executing the optimization program, the dependency graph of the nine of tasks in the Figure 4, there are factorial of 9 permutations, 294 Topological orders obtained and 6 Topological orders that have the smallest occupied memory amount. The right part of Figure 3 illustrates the chart of the amount of occupied memory of the Topological order.

In the second phase, we implement Nôm character recognition module as in Figure 5. Then, we permute the order of performing the functions under one of the best Topological order shown in Figure 3 and under 10 Topological orders that are not the best Topological order and are selected randomly from the 294 Topological orders to run the tests on the same

configuration. Here, we do tests on the simulation device of Pocket PC on the configuration in Table I. To do statistic of the amount of occupied memory shown in Table 2, we use "Best TaskMan" program [9] to show all of the Nôm character recognition processes on Windows Mobile as in Figure 6. Then, we draw the chart of actual occupied memory amount shown in Figure 7. The experimental result shown in Figure 7 matches the evaluation of occupied memory at model level shown in Figure 3.

Table 1. Experiment Environment

Processor	Intel(R) Core(TM) i5-2430M CPU-2.40GHz
RAM	2.00GB
Operating System	Windows 7 Starter SP1
Software	.NET framework 3.5

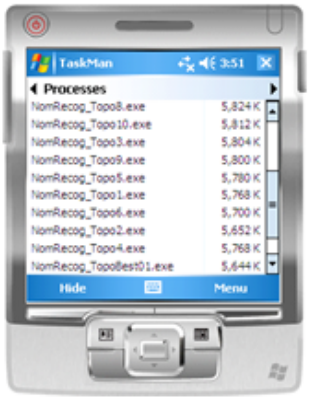

Figure 6. Show Nôm character recognition processes on Mobile by TaskMan

Figure 5. Nôm character recognition module on Mobile

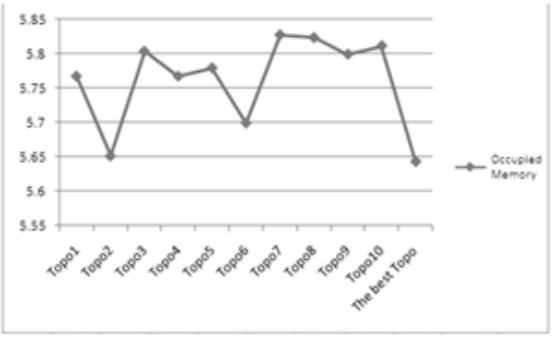

Figure 7. Chart of actual occupied memory

Table 2. Amount of actual Occupied Memory of the program corresponding to each Topological order

Topo chains	Topo1	Topo2	Topo3	Topo4	Topo5	Topo6	Topo7	Topo8	Topo9	Topo10	The best Topo
Memory usage	5.768	5.652	5.804	5.768	5.78	5.7	5.828	5.824	5.8	5.812	5.644

6. CONCLUSION AND FUTURE WORK

We have proposed the new method to optimize the occupied memory of the embedded software in the design phase based on DSL, T4 and Topological sort. Due to the limitation of embedded systems about resources and CPU, this optimization method has an important role and able to apply widely. We can extend this research for applying to other fields such as optimizing the scheduler of processes, optimizing the compiler, etc. The paper has the following main contributions. First, we built the occupied memory evaluation function of the task chain under the Topological order and proposed the new approach to optimize occupied of embedded system based on Topological sort. Second, we developed the code generation method from the model based on DSL and T4 automatically. Third, we built the framework of DSL, T4 and Topological sort to design, generate code and optimize at model level based on Topological sort.

However, there are still some challenges in the paper such as file permutation space of tasks and not supporting the cycle graphs of Topological sort. Topological sort, model transformation, DSL and T4 are the prospective investigations and able to apply widely. Based on the result of this paper, we continue to make improvements and further research such as Pareto multi-objective optimization of embedded software based Topological sort, compiler optimization, the process scheduler optimization based on Topological sort and optimization based on model transformation based on DSL and T4.

REFERENCES

[1] Jason Agron, Domain-specific language for HW/SW co-design for FPGAs, *Proceedings of the IFIP TC 2 Working Conference on Domain-Specific Languages, Berlin* (2009) 262–284.

[2] Michalis Anastasopoulos, Thomas Forster, Dirk Muthig, Optimizing model-driven development by deriving code generation patterns from product line architectures, *Proceedings of STJA, Kaiserslautern* (2005) 425–427.

[3] Michalis Anastasopoulos, Thomas Forster, Dirk Muthig, Model driven development for rapid prototyping and optimization of wireless sensor network applications, *Proceedings of the 2nd Workshop on Software Engineering for Sensor Network Applications* (2007) 31–36.

[4] Kevin Hammond, A domain specific language for real-time embedded systems, *Proceedings of GPCE* (2003) 37–56.

[5] Pham Van Huong, Nguyen Ngoc Binh, Bui Ngoc Hai, Vu Van Phuc, Hardware-Software Co-Design to optimize embedded system by Pareto principle and DSL, *Proceedings of IEICE ICDV, Hanoi* (8/2012) (ISBN: 978-4-88522-264-2 C3055).

[6] Pham Van Huong, Nguyen Ngoc Binh, Nguyen Thu Huyen, Nguyen Thuy Duong, Tran Nghi Phu, Embedded Software Performance Optimization Based on Generating the Simulation Code of Functions, *Proceedings of IEICE ICDV, Hanoi* (8/2012) (ISBN: 978-4-88522-264-2 C3055).

[7] Pham Van Huong, Tran Minh Tuan, Do Quoc Huy, Le Hong Trang, Nguyen Ngoc Binh, Truong Anh Hoang and Vu Thanh Nhan, Some Methodologies of Nôm Optical Character Recognition, *Proceedings of ICT.rda'08, Hanoi* (8/2008) 309–318.

[8] Pham Van Huong, Tran Minh Tuan, Do Quoc Huy, Le Hong Trang, Nguyen Ngoc Binh and Truong Anh Hoang, Some approaches to Nôm optical character recognition, *VNU, Hanoi. J. of Science, Natural Sciences and Technology* **24** 3 (2008) (ISSN: 0866-8612).

[9] http://best-taskman-s60-3rd.en.softonic.com/symbian

[10] Arthur Kahn, Topological sorting of large networks, *Journal of Communications of the ACM 5* **11** 5 (Nov. 1962) 558–562.

[11] Sangyoon Oh, Mehmet Aktas, Marlon Pierce, Geoffrey Fox, Optimizing Web service messaging performance using a context store for static data, *Proceedings of the 5th WSEAS international conference on Telecommunications and informatics* (2006) 50–58.

[12] Dorin Petriu and Murray Woodside, A Meta model for generating performance models from UML designs, *Proceedings of the 7th International Conference, Lisbon, Portugal* (October 11-15, 2004) 41–53.

[13] Armita Peymandoust, Tajana simunic and giovanni De Micheli, Low power embedded software optimization using symbolic algebra, *Proceedings of the conference on Design, automation and test, Europe* (2002) 1052–1060.

[14] Sanna Sivonen, Domain-specific modeling language and code generator for developing repository-based Eclipse plug-ins, *Proceedings of the 12th International Software Product Line Conference, HongKong* (2008) 356–364.

[15] Lambert Surhone, Mariam Tennoe, Susan Henssonow, Steinhaus-Johnson-Trotter algorithm, *Mauritius* (2010) 25–40 (ISBN: 6133260440).

[16] Chris Thompson, Jules White, Brian Dougherty, Douglas Schmidt, Optimizing mobile application performance with Model-Driven Engineering, *Proceedings of the 7th IFIP, Berlin* (2009) 36–46.

[17] Lloyd Williams, Performance evaluation of software architectures, *Proceedings of the 1st international workshop on Software and performance* (1998) 66–77.

Received on May 25, 2012
Revised on December 18, 2012