Multi-objective Optimization for Embedded Software at Model Level Based on DSL and T4

Pham Van Huong, Nguyen Ngoc Binh and Bui Ngoc Hai
University of Engineering and Technology, Vietnam National University, Hanoi, Vietnam

Abstract

Optimizing embedded software can be done in the different phases of the software life such as design, implementation and compilation. The optimization in the early stages of software life is very significant. In this paper, we propose a Pareto multi-objective optimization method of embedded software in the design phase based on DSL (Domain Specific Language) and T4 (Text Template Transformation Toolkit) code generation technology. From the class diagram, we define the measures that support to evaluate performance and used memory capacity. Based on these measures, we analyze and define the objective performance function, the memory objective function and the global objective function for Pareto optimization. We also developed the DSL and T4 framework to create class diagrams and get parameters automatically from the class diagrams.

1. Introduction

Today, in the development trend of information technology, Software Engineering has also developed strongly, especially in embedded software and object-oriented software. Development environment of embedded software is limited in: the CPU's processing ability, memory size, battery life time, problems of energy consumption, real-time problem [1, 2]. Therefore, optimization problems in software development are of important significance. The problems of embedded software optimization often include the following levels: design level optimization, source code level optimization, compiler level optimization, environment level (hardware-software co-design optimization, instruction set optimization) and runtime level optimization [3, 4, 5]. Optimization in the design phase is a new approach and although representing many challenges, it brings more benefits than other optimization methods at later phases.

Optimization at design phase is often based on model-driven software engineering, software performance engineering and there have been a few studies about this issue, the most well-known out of which is the research by Michalis Anastasopoulos, Thomas Forster and Dirk Muthig about optimizing Mobile application performance trade-off with battery lifetime based on Model-driven engineering [1, 2]. In this research, the author built a Domain Specific Language based on the Eclipse open source framework for constructing Mobile software architecture, generating simulation code and running tests on simulation code to evaluate performance and trade off with battery lifetime. According to this approach, these authors studied application optimization based on model-driven engineering and code generation template for the product lines [5]. Following software modeling approach, 2009, Zhihui Yang proposed the research about Domain Specific Modeling approach for component-oriented software. In this research, Yang synthesized aspects of Domain Specific Language (DSL), Component-based Software model and definition, DSL construction for component-based software in Eclipse [3]. Then he based on DSL to generate simulation code, evaluate and optimize based on model transformation. However, these methods do not analyze, evaluate directly the performance and measures from the components and structure of the model. On the other hand, evaluating the performance and measures at design phase is very difficult. Also, most measures in model level only evaluate maintainability, reusability but not directly the performance measure, efficiency of memory usage [8, 9, 10, 11]. Moreover, the measures of quality, performance, memory often conflict, such as: memory storage optimal may reduce the execution speed of the program. Therefore, in this paper we propose a new approach: Pareto optimal for embedded software from class diagrams based on DSL and T4. The research in this paper is conducted to address three problems:

- Software architecture specification and parameters extraction from the models to evaluate performance, memory usage and other quality measures.
- The direct analysis and evaluation of performance, memory usage from class diagrams

- Pareto Optimal trade-off between performance and efficiency of memory usage from class diagrams

2. Measures based on a class diagram

2.1. Parameters

Classes are central components in the class diagram. A class is the template defining a set of objects with the same attributes and behaviors. A class diagram consists of a set of packages, classes, interfaces, relationships and constraints. To construct the performance evaluation function on the class diagram, firstly we derive parameters affecting to the software performance directly from the class diagram. These parameters are described in Table 1.

Table 1. Parameters used to evaluate the software performance

Parameters	Symbol	Description
Class method	S	Is static method, that is allocated memory when loading program
Instance method	O	Is non-static method allocated memory when creating an object
Class variable	X_j	X_j is a static attribute j in a class, it is allocated memory when loading program.
Instance variable	Y_j	Y_j is non-static attribute j in an object, it is allocated memory when creating an object.
Method parameter	P_k	Is parameter k of a method
Total of classes	A	Number of classes in a class diagram
Total of class methods	B_i	Number of static methods in the class i
	B	Number of static methods in a class diagram
Total of class variables	C_i	Number of static attributes in class i
	C	Number of static attributes in a class diagram
Total of Instance methods	D_i	Number of non-static methods in class i
	D	Number of non-static methods in a class diagram
Total of Instance variables	E_i	Number of non-static attributes in class i
	E	Number of non-static attributes in a class diagram
Total of Parameters	F_j	Number of parameters in method j

2.2. Measures affecting to performance and memory capacity

Before constructing the performance evaluation function from the class diagram, in this section, we focus on the analyzing components and the structure of the class diagram to build the measures affecting to software performance [9, 11]. These measures are shown in Table 2.

Table 2. Measures affecting to performance

Measures	Symbol	Description
Size of class variables	S_1	The total of the allocated memory size of static attributes in the class diagram
Size of class methods	S_2	The total of the allocated memory size of static methods in the class diagram
Size of executed class methods	S_3	The total of the allocated memory size that will be used when executing the static methods in the class diagram
Size of instance variables	S_4	The total of the allocated memory size of non-static attributes in the class diagram
Size of instance methods	S_5	The total of the allocated memory size of non-static methods in the class diagram
Size of executed instance methods	S_6	The total of the allocated memory size that will be used when executing the non-static methods in the class diagram

2.2.1. Size of class variables

This measure is calculated by a total of memory amount allocated statically for all static attributes of all

classes in the diagram. The static elements are allocated memory once when loading the program into memory. With the notation in Table 1 and Table 2, we construct the formula for calculating the memory size total of the static attributes in the class diagram as the following formula:

$$S_1 = \sum_{i=1}^{A} \sum_{j=1}^{C_i} Size(X_j) \quad (1)$$

2.2.2. Size of class methods

The class methods are static methods that do not belong a specific object. They are commonly called by class name and allocated once when loading the program. According to the parameters in Table 1 and Table 2, this measure is defined as formula (2):

$$S_2 = \sum_{i=1}^{A} \sum_{j=1}^{B_i} Size(Reference\ Variable) \quad (2)$$

2.2.3. Size of executed class methods

The Methods of executed class size is the size of memory used when executing a static method. When the static method is called, firstly the parameters are allocated and when the method done, it needs to save the returned results in memory. Therefore, according to the parameters in Table 1 and Table 2, this measure is defined as formula 3:

$$S_3 = \sum_{i=1}^{A} \sum_{j=1}^{B_i} \left(Size(return\ type\ j) + \sum_{k=1}^{F_j} size(P_k) \right) \quad (3)$$

2.2.4. Size of instance variables

The Size of Instance Variables is the method of the object and it is only allocated memory when the object is created. Therefore, from the parameters in Table 1 and Table 2, this measure is defined as formula 4:

$$S_4 = \sum_{i=1}^{A} \sum_{j=1}^{E_i} Size(Y_j) \quad (4)$$

2.2.5. Size of instance methods

Instance methods are non-static method, allocated memory when creating the object and only used after creating the object. Therefore, according to the parameters in Table 1 and Table 2, this measure is defined as formula (5):

$$S_5 = \sum_{i=1}^{A} \sum_{j=1}^{D_i} Size(Reference\ Variable) \quad (5)$$

2.2.6. Size of executed instance methods

The size of executed instance methods is the total of memory size used when executing the instance methods. It includes the allocated memory for parameters and the memory containing the returned result of the instance method. According to the parameters in Table 1 and Table 2, this measure is defined as formula 6:

$$S_6 = \sum_{i=1}^{A} \sum_{j=1}^{D_i} \left(Size(return\ type\ j) + \sum_{k=1}^{F_j} size(P_k) \right) \quad (6)$$

3. Pareto optimization from a class diagram

3.1. Objective functions

Based on the metrics defined in section 2, we build performance objective function, memory objective function and global objective function for Pareto Optimal to choose the best trade-off between performance and memory usage. Each type of attribute and method will be allocated memory and is called in different ways that affect the execution of the program. To construct the objective functions, we analyze the execution of an OOP program and analyze the dependence of performance on the attributes, methods and parameters in the class diagram.

When the program is required to execute, firstly, the source code of class will be loaded into memory, static components including class variables, class method also allocated memory in the load time. Therefore, the static components only take one static allocation and one memory access for use after the load is finished. When using the instance variables and instance methods of the object, the object must be created. When creating objects, we need to perform two steps: access memory to fetch and execute the object creation statement; allocate memory dynamically for the instance variables and instance methods of the object. Therefore we need a dynamic allocation operation and two memory access operations to be able to use the components of the object. When we execute a class method, we need a memory access operation to point to the statements of method, and a memory allocation for the parameters of the method, also, because to call the static method we need through the class in memory, so a static execution process require at least one memory allocation operation and three memory access operations. At design phase, with class diagram, we do not care and can't evaluate the statement set of method. When we execute an instance method, first we must create an object, an instance methods need called through an object, so it need one dynamic allocation operation and two memory access. Execution process of an instance method is also like a class method, so it needs at least an additional allocation and three additional memory access operations. So, total execution process of an instance method needs at least three memory allocation operations and five memory access operations.

From analyzing an execution process of an OOP program, we give some conclusions and formulate the objective function as follows.

Performance Objective Function:

- Using static components (class members) will perform faster than using non-static components due to static memory allocation and loaded into memory as soon as the program loaded, so performance would be proportional to $(S_1 + S_2) / (S_4 + S_5)$
- In a class, when accessing data, the program using class variables is faster than that using parameters of the class methods, so performance would be proportional to S_1/S_3
- With an object, when accessing data, the program that uses instance variables are faster than the program that uses parameters of the instance methods, so performance would be proportional to S_4/S_6.

Therefore, the performance objective function is calculated by the following formula:

$$F_p = \frac{S_1+S_2}{S_4+S_5} + \frac{S_1}{S_3} + \frac{S_4}{S_6} \quad (7)$$

Memory Objective Function:

- Using the static components (class members) will take more memory capacity than the using non-static components due to the static component is allocated memory static, and only recovered when the program ends, so used memory capacity would be proportional to $(S_4 + S_5) / (S_1 + S_2)$
- In one class, with the same data object, if we use class variables, it will take more memory than using parameters transmission to class methods, because memory allocated for class variables only released when the program ends. Therefore used memory capacity is proportional to S_3/S_1
- With an object, with each element of data, if we use instance variables, it will take more memory than using parameters transmission to class methods, because the memory of instance variables is recovered only when the object is destroyed and memory parameters is freed when the method done. Therefore the amount of memory used is proportional to S_6/S_4.

So, the memory objective function is calculated by the following formula:

$$F_m = \frac{S_4+S_5}{S_1+S_2} + \frac{S_3}{S_1} + \frac{S_6}{S_4} \quad (8)$$

Global Objective Function:

$$F = w_p \times F_p + w_m \times F_m \quad (9)$$

Where: w_p is weight of the performance objective, w_m is weight of the memory objective and $w_p + w_m = 1$.

3.2. Apply Pareto Optimization to class diagram

Unlike other traditional optimization methods, Pareto optimization address to the trade-off between the optimal objectives. Each class design has a tuple of values of the objective function, called an objective vector. We apply the Pareto optimal by constructing the global objective function shown in formula 9 from the objective functions shown in formula 7 and formula 8:

Here we may consider the measures could have the same role, then we can choose the weights for the objective function are equal and $w_p = w_m = 0.5$.

From the set of class diagrams, our task is finding the best diagram according to the criteria that the global objective function is maximal.

4. Develop a DSL framework supporting Pareto optimization from class diagrams

Domain Specific Language is a programming language or specification language dedicated to a particular problem domain, a particular problem representation technique or a particular solution technique. DSL is not a new idea. There are many DSL in the specific application domain such as SQL, html or VHDL - hardware description language. However, today with the diverse development of technologies, software engineering in a different application domain also has its own characteristics so that UML can not specify details for a specific domain. Therefore, using DSL replacing UML in the specific application domain is a new and promising trend in software engineering [5, 6, 7].

In this section, we will develop a framework DSL to support the Pareto optimization at model level and allow designing class diagram, evaluating software performance of a class diagram and selecting the best trade-off between performance and used memory capacity. The first, we define DSL and build meta-model that allows creating class. The second, we create the templates used to analyze and extract the parameters from the class diagram based on T4 code generation technology. Finally, we implement the Pareto optimization algorithm to select the best trade-off class diagram between performance and used memory capacity.

4.1. Build DSL and meta-model

DSL Tools allow constructing meta-model using to specify a DSL. Meta-model is the model used to define and create the models. Process of DSL definition and meta-model construction is shown by the following steps:

- Define the logical components: Domain classes, Components, Tasks, Flows, Comment classes, Rules, Constrain and Relationship
- Create shape symbols corresponding to each logical component above. These symbols will be used to design in the graphic interface after the DSL was compiled and deployed
- Define XML files used to store definition and mapping between logical components and shape symbols.

To build the DSL framework and meta-model supporting Pareto from class diagram, we use the class diagram example of Visual Studio.NET 2008 SDK tools. We modify and integrate specification information and templates used to generate code to this example to create framework of DSL and meta-model.

4.2. Build T4 templates used to generate code

T4 is a powerful code generation technology, allows constructing templates supporting automatic code generation based on the XML file defining DSL (meta-model file) and the XML file of the actual design model [7, 8]. T4 is flexible and it allows generating output in the different forms such as programming language, formal language or even a destination model. The idea of T4 is shown by the following steps: (1) read the XML file of the actual design model and the meta-model file; (2) transform the shape symbol to the logical symbol; (3) analyses and generate code based on T4 templates. In this section, we define the T4 templates and use them to extract the parameters from class diagrams. Figure 1 show an example of a T4 template built.

```
<#@ template inherits="Microsoft.VisualStudio.TextTemplating.
<#@ output extension=".txt" #>
<#@ P03_Pareto_Class processor="P03_Pareto_ClassDirectiveProc
<#
  // Cac bien luu tru, thong ke tham so tu bieu do
  string[] arClass;
  arClass = new string[this.ModelRoot.Types.Count];
  //duyet cac class trong model:
  foreach (ModelType type in this.ModelRoot.Types)
  { //Hien thi ten Class
#><#= "@Class:" #><#= type.Name #>
<#  //Lay thong tin lop hien tai
    ModelClass modelClass = type as ModelClass;
    if (modelClass != null)
    {
        //get methods:
        if(modelClass.Operations.Count >0)
        {#>
<#          foreach(ClassOperation op in modelClass.Operation
            {
#>          @@Method:<#=op.Name#>
<#          }
        }
```

Figure 1. A template for generating parameters from class diagram

5. Experiment

In this experiment, we use DSL framework in the section 4 to design and solve the Pareto optimization from class diagram. Here, we used well the famous 8-queen problem to deploy the experiment by following steps: (1) design 5 different class diagram of the 8-queen problem based on DSL framework as shown in Figure 2, Figure 3 , Figure 4, Figure 5 and Figure 6; (2) generate 5 parameter files based on T4 templates as illustrated in Figure 7; (3) analyze parametric, calculate the measures and execute the Pareto optimization program to choose the best trade-of class diagram between performance and used memory as shown in Figure 8.

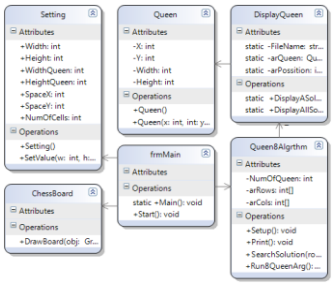

Figure 2: Class diagram 1

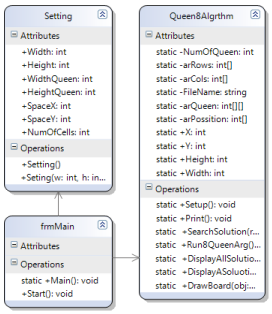

Figure 3: Class diagram 2

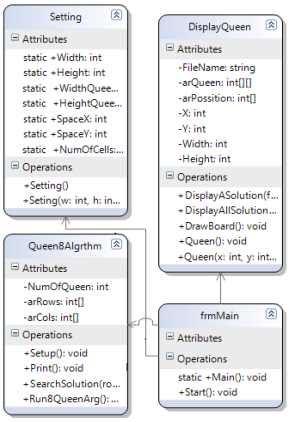

Figure 4: Class diagram 3

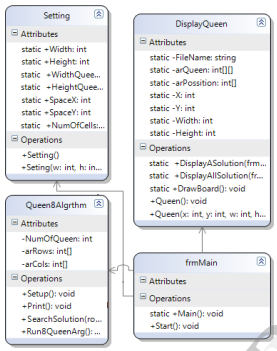

Figure 5: Class diagram 4

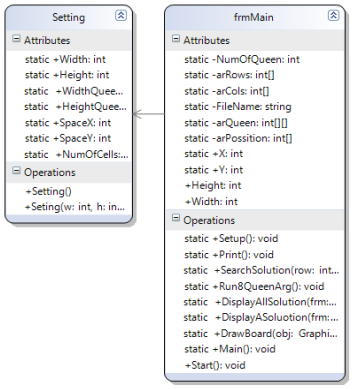

Figure 6: Class diagram 5

Result of executing the Pareto optimization program is shown in the chart contained in the Figure 12. In this chart, class diagram 3 is the selected diagram because it is balanced between the performance objective function and the memory objective function. In this class diagram, value of the global objective function is maximal.

```
@Class:Queen
        @@Method:+Queen()
        @@Method:+Queen(x: int, int: y, w: int, h: int)

        @@@Attribute:-X: int
        @@@Attribute:-Y: int
        @@@Attribute:-Width: int
        @@@Attribute:-Height: int
@Class:ChessBoard
        @@Method:+DrawBoard(obj: Graphic): void
@Class:Setting
        @@Method:+Setting()
        @@Method:+SetValue(w: int, h: int, x: int, y: int, cells: int): void

        @@@Attribute:+Width: int
        @@@Attribute:+Height: int
        @@@Attribute:+WidthQueen: int
        @@@Attribute:+HeightQueen: int
        @@@Attribute:+SpaceX: int
        @@@Attribute:+SpaceY: int
        @@@Attribute:+NumOfCells: int
@Class:DisplayQueen
        @@Method:static +DisplayASolution(frm: frmMain): void
        @@Method:static +DisplayAllSolution(frm: frmMain): void

        @@@Attribute:static -FileName: string
        @@@Attribute:static -arQueen: Queen[]
        @@@Attribute:static -arPossition: int[]
@Class:frmMain
        @@Method:static +Main(): void
        @@Method:+Start(): void
```

Figure 7: Result of generating code from class diagram

Table 3. Actual performance of 5 programs

Times	Program 1	Program 2	Program 3	Program 4	Program 5
1	1469	1305	1103	1001	998
2	1483	1302	1005	1007	995
3	1585	1288	1009	1005	984
4	1481	1295	1011	995	992
5	1519	1320	1100	980	1002
6	1487	1312	1205	1002	996
7	1512	1299	1202	998	983
8	1568	1308	1102	995	1000
9	1490	1323	1015	1005	983
10	1470	1298	1025	1011	997
Average time (ms)	1506.4	1305	1077.7	999.8	993

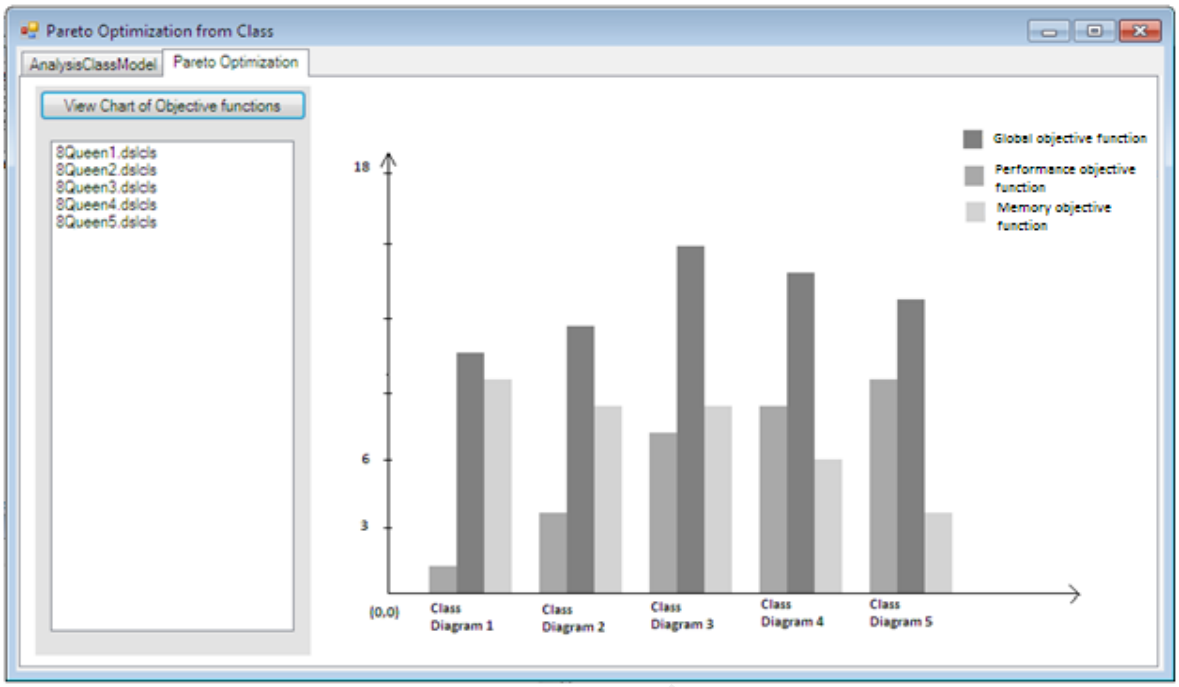

Figure 8: The chart of Pareto optimization from Class diagrams

After executing Pareto optimization from class diagram, we implement 5 different programs that are corresponding to 5 class diagrams to test the actual performance. Statements and algorithms in these 5 programs are similar. The programs are only different from structure and their components. This is to avoid the influence of the implementation on the evaluation of the model. Finally, we execute the programs, statistics and compare actual performance with the result of evaluation of performance by Pareto optimization. For each program, we execute 10 times and calculate the average executing time. Statistical results are displayed in the Table 3.

6. Conclusion and future work

The content studied in this paper presented the methodology of evaluating software performance directly from class diagram based on analyzing components and architecture of the class diagram. This is a new method and it is different from SPE method that must add more performance information to diagram and transform to performance models from UML models. And the paper also proposed a new approach to optimize object-oriented embedded software in the design phase based on multi-objective Pareto optimization. The specific contribution of the paper is as follows: firstly, we have analyze and constructed measures and objective functions directly from class diagram; secondly, we have developed the framework that allows designing class diagram and generating parameters automatically from class diagram based on DSL and T4; thirdly, we have implemented the program to solve the Pareto optimization from class diagram. Last but not least, we also have implemented experiment programs to verify the actual performance and Pareto optimal results.

Based on this research, we will study multi-objective optimization such as the reuse ability, architecture complexity, maintenance ability; optimization based on model transformation, optimization based on generating simulation code.

7. References

[1] Chris Thompson, Jules White, Brian Dougherty and Douglas C. Schmidt Bowman. *Optimizing Mobil Application Performance with Model-Driven Engineering*. P09 Proceedings of the 7th IFIP WG 10.2, 2008, pp.1-8.

[2] Michalis Anastasopoulos, Thomas Forster, and Dirk Muthig. *Optimizing Model-driven Development by deriving Code Generation Patterns from Product line architectures*. STJA, JIT. 6th Annual, Kaiserslautern, Germany, 2005, pp.425-437.

[3] Sanna Sivonen. *Domain-specific modelling language and code generator for developing repository-based Eclipse plug-ins.* VTT PUBLICATIONS 680, ESPOO 2008, pp.16-62.

[4] Armita Peymandoust, Tajana Simunic. *Low Power Embedded Software Optimization using Symbolic Algebra*. Computer Systems Laboratory, Stanford University, Stanford, CA,Proceedings of the conference on Design, automation and test in Europe, 2002, pp.1-6.

[5] Michalis Anastasopoulos, Thomas Forster, and Dirk Muthig, "*Optimizing Model-driven Development by deriving Code Generation Patterns from Product line architectures*", Fraunhofer Institute for Experimental Software Engineering (IESE) Sauerwiesen 6, D-67661 Kaiserslautern, Germany, 2007, pp.3-10.

[6] Sadagopan Srinivasan, Zhen Fang, Ravi Iyer, Steven Zhang, Mike Espig, Don Newell, Daniel Cermak, Yi Wu, Igor Kozintsev, Horst Haussecker. *Performance Characterization and Optimization of Mobile Augmented Reality on Handheld Platforms*. Intel Corporation 2008

[7] Yong-Yoon Cho, Jong-Bae Moon, and Young-Chul Kim. *A System for Performance Evaluation of Embedded Software*. Engineering and Technology 2005

[8] Pospiech F, Olsen S. *Embedded software in the SoC world. How HdS helps to face the HW and SW design challenge*. IEEE– 658, 21-24 Sept. 2003, pp:653

[9] Connie U. Smith, Catalina M. Lladó, Vittorio Cortellessa and Lloyd G. Williams. *From UML models to software performance results: An SPE process based on XML interchange formats*. 2005, pp.1-13.

[10] Dorin B. Petriu, Murray Woodside. *A Metamodel for Generating Performance Models from UML Designs*. Dept. of Systems and Computer Engineering Carleton University, Ottawa K1S 5B6, Canada, pp.1-11.

[11] Simonetta Balsamo Moreno Marzoll. *Efficient Performance Models in Component-Based Software Engineering*. Torino 155, 30172 Mestre, Italy, pp.1-10.