International Journal of Information, Control and Computer Sciences
ISSN: 2517-9942
Vol:1, No:10, 2007

Object-Oriented Programming Strategies in C# for Power Conscious System

Kayun Chantarasathaporn, and Chonawat Srisa-an

Abstract**—**Low power consumption is a major constraint for battery-powered system like computer notebook or PDA. In the past, specialists usually designed both specific optimized equipments and codes to relief this concern. Doing like this could work for quite a long time, however, in this era, there is another significant restraint, the time to market. To be able to serve along the power constraint while can launch products in shorter production period, object-oriented programming (OOP) has stepped in to this field.

Though everyone knows that OOP has quite much more overhead than assembly and procedural languages, development trend still heads to this new world, which contradicts with the target of low power consumption. Most of the prior power related software researches reported that OOP consumed much resource, however, as industry had to accept it due to business reasons, up to now, no papers yet had mentioned about how to choose the best OOP practice in this power limited boundary.

This article is the pioneer that tries to specify and propose the optimized strategy in writing OOP software under energy concerned environment, based on quantitative real results. The language chosen for studying is C# based on .NET Framework 2.0 which is one of the trendy OOP development environments. The recommendation gotten from this research would be a good roadmap that can help developers in coding that well balances between time to market and time of battery.

Keywords**—**Low power consumption, object oriented programming, power conscious system, software.

I. Introduction

IN this era, mobile devices gain much more popular from so many supportive reasons such as lower price and better communication infrastructure. When mentioning about mobile equipments, one of the major issues we have to concern is the battery life.

In the past, these devices were specifically created in term of both hardware and software. About the software, the languages used for development were mostly hardware specific assembly. The pro of doing like this was gaining high performance while did not consume so much battery. However, the major con of this strategy was it could not produce software fast and various enough. Not just the supportive software, the system software, sometimes, was pressured by time-to-market factor.

Right now, in desktop application market, the trend of software development has been migrated to OOP (Object-Oriented Programming) creation. Using this strategy has main benefits in reusable objects while can conceal some secret things by encapsulation. With OOP, the software house can loosen time-to-market constraint. However, the output of this style development has a significant drawback as it consumes much higher resources. This is the major contradiction to the nature of mobile equipments that having longer battery life need to have low power consumption system.

Low power consumption is related to both hardware and software. However, the scope of this article is focused in just software aspect. As the trend of applications developed for mobile devices goes in the same direction as ones on desktop, using OOP, this research tries to find the appropriated way, recommendation, of using major OOP principles while consuming as less power as possible.

The details in this article are as follows. There will be the mention about other studies related to low power consumption software. Next, the major characteristics of OOP those can substitutes to one another are raised. Then, the results of resource consumption comparisons among the comparable commands are discussed. Finally, the conclusion of this study is the recommendation of major OOP command usage in development under power conscious system.

Kayun Chantarasathaporn is a Ph.D. student in Faculty of Information Technology, Rangsit University, Muang, Pathumtani, 12000, Thailand (e-mail: kayun@kayun.com).
Chonawat Srisa-an is the assistant professor in Faculty of Information Technolgy, Rangsit University, Muang, Pathumtani, 12000, Thailand (e-mail: chonawat@rangsit.rsu.ac.th).

II. Low Power Consumption Software Related Study

Up to now, there have been quite numerous research articles pointing at software and its energy consumption. However, they can be classified to just a few scopes, such as, power analysis at low level language, compilation techniques those can create energy optimized codes, strategies for creation and implementation of software for power concerned system, boundary of usage time in embedded software, tools that help automatically find power critical points, and comparison of energy needed among different writing styles at the layer of high level language. The samples of researches just mentioned are as follows.

A well-known article [5] which is considered as the first research in the field of low power consumption in the software viewpoint is one from Tiwari and his team. They studied the power consumption of each major assembly command for specific CPU, 486DX2-S and the reasons in low level of software those affect power desire, such as, inter-instruction and cache miss effects. Though everyone had known that different commands should need different level of power, this Tiwari's work clarified how much they were.

Tiwari also recommended compilation techniques for the focus of low energy in another article [1]. He pointed out that the compiler should reorder the instructions to reduce switching since this activity required more power. Also, the code generated from compiler should choose using register instead of memory when possible since the registers use less power than memory.

Naik and Wei proposed strategies for software implementation in energy concerned environment [6]. They said in their 3 ECs implementation techniques those were, EC1, coding by using energy saving techniques, EC2, choosing appropriated algorithms and, EC3, deploying with selected strategies could help in lessen power utilization. By the way, in some cases, three mentioned things could not be all applied to the project. They proposed 3Ps[1] which agreed with Tiwari's recommendation that the program should avoid using memories and use registers since the latter required less energy. In their study, they matched blocks of high level code, written in C, with the assembly outputs and could show that registers were less power greedy. Similar to EC, in some situations, using all 3 Ps is not possible, such as, it is hardly possible to calculate matrix while get along with P2.

Studying about the time boundary used in software was done by Li and Malik [4]. They tried to find the time scope and critical points of software implementation with the help of linear programming techniques those applied to the high level language source code written in C.

Seeking automatic tool that can help code optimization was studied by Peymandoust et al. Usually, in embedded system, software should be optimized as much as possible to consume less power. However, in the past, this process was done manually. Peymandoust used Profiler to help in finding critical points in term of basic blocks and proposed Symsoft which aimed automatically find some way that could produce acceptable outputs from the same input while using less power.

There was also a study of comparisons in term of power consumption and performance between Object-Oriented and Procedural coding style [2]. The result was as expected that OOP consumed more resources than procedural one. However, the study demonstrated that this should be acceptable when compared with the benefits gaining from development in OOP style, such as, reusability, member private management, etc.

III. Major Characteristic of Object-Oriented Programming

Object Oriented Programming has been more popular because it is appropriated for this time-to-market-oriented production era. Not just OOP is easy to reuse, the encapsulation capability makes it appropriated for security concerned development.

In OOP, all kinds of member, data and function, should reside in class, as seen in Fig. 1.

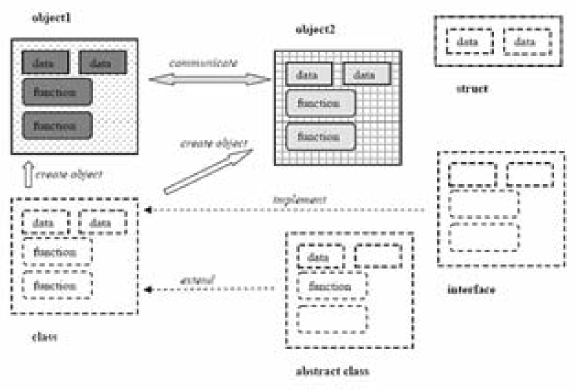

Fig. 1 General class structure and other components

Class is like a frame for its members, data and function. When the users want to use a class, they usually create an instance from the class. This class instance is often called as object. Object gets a frame from its class, however, detail of the object's members can be different object by object.

Another important characteristic of OOP is inheritance. Class can inherit from class, abstract class or interface. Abstract class is a class that has at least one abstract method. Abstract method can be thought as a frame of method, so it has no method detail inside. Interface is considered as a blueprint of class since it has no method body as well. Often, class inherited action is called "extend" while interface inherited action is called "implement".

Both kinds of class member, data member (attribute) and function member (method), can be either static or dynamic. Users can imagine static members as members of class while dynamic members are members of object. So, to use dynamic members, first, users need to create object from class. After object is created, users can use the non-static members by asking the object to refer or call them.

An additional characteristic of OOP is accessibility control. Usually, there are at least 3 keywords for this task, public, private and protected. Public means members from any classes can access while private just allows only members from the same class. Protected is in between, it is where everyone is prohibited except members from the same class and ones from inherited classes. Accessibility can be applied to both data and function members.

As mentioned above, it is clear that class is a foundation of OOP. In some case, if developers want to create a frame that do not have function members, only data ones, they may choose structure as an alternative choice.

Not just the above programming components are different in structure, to focus in this article scope, their dissimilar complexities make them diverse in power consumption. Therefore, to write OOP for power conscious system,

[1] P1. Assign live variables to registers. P2. Avoid repetitive computation of addresses. P3. Minimize memory access.

developers need to choose the right and light one if there are options.

IV. C# OOP Style & Power Consumption Analysis

In this research, we had tried to measure and compare the power consumption of some significant usages in OOP. What we raised for comparing had details as follows.

Classes
- Class (data member only)
- Struct

Prototypes
- Abstract Class
- Interface

Attributes
- Static Attribute
- Dynamic Attribute

Methods
- Static Method
- Dynamic Method
- Dynamic Anonymous Method

Dynamic Variables Call
- Bare Usage
- Using "this" Keyword

Variable Accessibility
- Private
- Protected
- Public

Method Accessibility
- Private
- Protected
- Public

About the tool in this research, we developed the software, TOM - Time Operation Measurement, which circular checked (every 10 Milliseconds) the timespan the specified process used. TOM will terminate checking itself when the watched process ends. The software can snapshot User Processor Time (UPT), Privileged Processor Time (PPT), Total Processor Time (TPT) and Memory used by the specific software process. UPT is the timespan processor uses just for that process, PPT is the time processor spends for the operating system to support that process and the TPT is the summation of UPT and PPT.

Vivek Tiwari mentioned in his paper that time the processor used was directly related to the power it needed [5]. Therefore, to get the same output from similar essential working steps while controlling other kinds of element, the shorter the processor time uses the better performance of the chosen component is - in term of the power optimization.

The results from the measurement shown in this paper were done on the system that used AMD Athlon™ XP 1800+ CPU with 1 GB RAM. The software in the system were regular Microsoft Windows XP SP2, Microsoft .NET Framework Redistributable Package 2.0, the codes to be measured and TOM.

Usually, the primary concern of developers is successfully runnable program. However in this article, the focus is beyond that, we want to seek some ways that can work similar while consume less power. There are some proves showing that comparable commands require different energy levels.

A. Class and Struct

As the topic just raised, first, we compare the data-member-only class and struct. Both of them can contain group of variables or data members, but, from graph in Fig. 2 and results from TOM in Table I, it is easy to distinguish the difference of time spent. The more timespan the process takes the more power the process spends. This rule applies to this and all further comparisons. Therefore, static variable consumes more power than the dynamic because it takes around 40% longer time than dynamic variable.

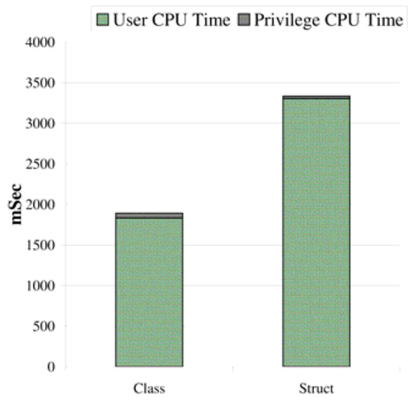

Fig. 2 CPU time usage comparison between Class and Struct

B. Abstract Class and Interface

Fig. 3 is the result of comparison between Abstract Class and Interface. Both of them can be used as class prototype, however, Interface is more restrictive since the methods inside must not have method body while Abstract Class can have some attributes or method bodies, just at least only one class is abstract. There is no significant different between using Abstract Class and Interface in similar situation.

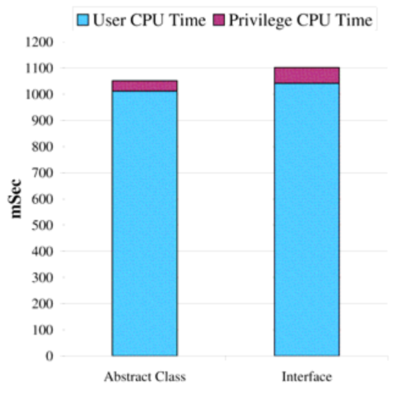

Fig. 3 CPU time usage comparison between Abstract Class and Interface

TABLE I
CPU TIME USAGE IN DIFFERENT CONDITIONS OF C# OOP CODING

ISSUE	SUB-ISSUE	MAX PROCESSOR TIME (MSEC)			
		User Time	Privilege Time	Total Time	Difference (%)
Classes	Class (data members only)	1832.63	60.08	1892.71	43.24 % less
	Struct	3304.75	30.04	3334.79	*used as base*
Prototypes	Abstract Class	1011.45	40.06	1051.51	4.55 % less
	Interface	1041.50	60.09	1101.59	*used as base*
Attributes	Static Attribute	1932.78	40.06	1972.84	41.72 % less
	Dynamic Attribute	3344.81	40.06	3384.87	*used as base*
Methods	Static Method	3625.21	60.09	3685.30	56.71 % less
	Dynamic Method	1772.55	70.10	1842.65	78.35 % less
	Dynamic Anonymous Method	8331.98	180.26	8512.24	*used as base*
Dynamic Variables Call	Bare Usage	3414.91	40.06	3454.97	0.29 % less
	Using "this" Keyword	3414.91	50.07	3464.98	*used as base*
Variable Accessibility	Private	1842.65	60.09	1902.74	44.28 % less
	Public	1842.65	50.07	1892.72	44.57 % less
	Protected	3354.82	60.09	3414.91	*used as base*
Method Accessibility	Private	951.37	60.09	1011.46	*used as base*
	Public	961.38	40.06	1001.44	0.99 % less
	Protected	961.38	40.06	1001.44	0.99 % less

C. Dynamic and Static Variable

Fig. 4 and Table I show that dynamic variable works slower than the static around 40%.

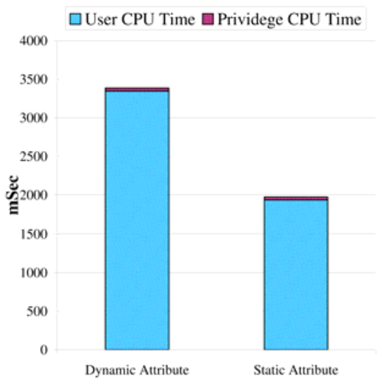

Fig. 4 CPU time usage comparison between Dynamic and Static variable (attribute)

D. Dynamic, Static and Dynamic Anonymous Method

In contrast with class attribute, dynamic method runs faster than the static around 50%. The comparison in this case has another candidate which is dynamic anonymous method. Usually, to use dynamic method, users need to declare and define object from the class, first. Yet, there is another way to be able to use dynamic method without explicit creation of object. That way is by using anonymous method. Though it is another alternative, anonymous dynamic method is very CPU intensive as the result from both Fig. 5 and Table I show that it takes around 80% longer time than regular dynamic method.

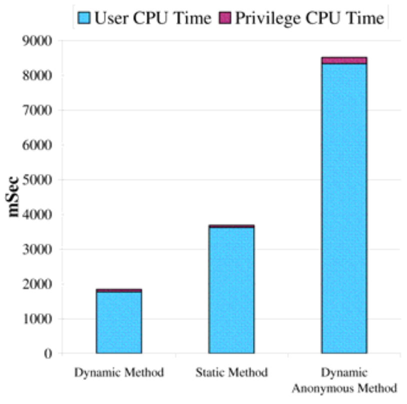

Fig. 5 CPU time usage comparison between Dynamic, Static and Dynamic Anonymous Method

E. Using "this" keyword and not using

When using dynamic class attribute locally, users may just use it barely or use with "this" keyword. There is no significant difference in term of CPU usage of this pair. The results are shown in Table I and Fig. 6.

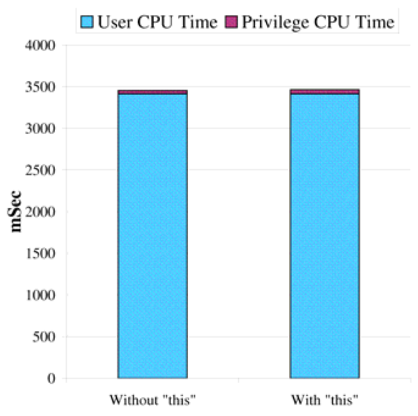

Fig. 6 CPU time usage comparison between using and not using "this" keyword

F. Private, Protected and Public Attribute

Another important feature in OOP is variable accessibility control. The most CPU consuming field is protected variable while private and public ones spend time quite close to each other. Protected attribute is slower than the other two around 40% as shown in Fig. 7 and Table I.

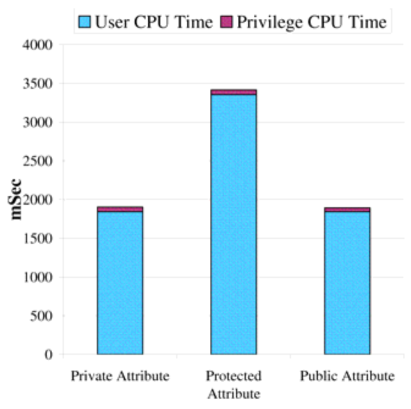

Fig. 7 CPU time usage comparison among Private, Protected and Public attribute

G. Private, Protected and Public Method

Accessibility control also applies to method but the result in term of CPU usage from private, protected and public methods are different from what they were in attribute. Result in Table I and Fig. 8 demonstrates that, in CPU consumption aspect, all of them are not significant different.

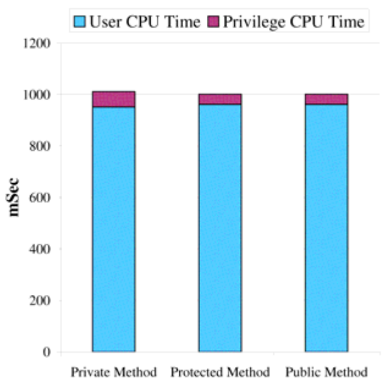

Fig. 8 CPU time usage comparison among Private, Protected and Public Method

About the memory cost in each section of the experiments, the usages of memory were almost equal in the same section, so, the difference of memory usages is not a major factor that affects power consumption in these cases.

V. CONCLUSION

Trend of power conscious system development has shifted from proprietary software and hardware design to more generic standard platform. Programming tools used in application development has changed from low level assembly to high level procedural language and tended to be object oriented approach. OOP goes to power conscious system just because it can accelerate development lifecycle which is the crucial issue in time-to-market oriented era.

When compared with older systems, OOP is the technique that has most overhead. However, the industry has more accepted it since the limitation of resources nowadays is relaxed and also business reasons. By the way, as battery system can not long last, power usage is still a major concern. So, the research tries to find approach within OOP development that uses lower energy while provides similar output.

As shown in above research result, it is clear that though the outputs from the codes may be indifferent, each CPU timespan might not be the same significantly. The time CPU spends is direct variant to energy consumption. This is a factor we should consider when coding under power conscious system.

Table II illustrates the strategies of writing code in OOP style for lessen energy expense.

TABLE II
SUMMARY OF C# CODING STRATEGIES FOR POWER CONSCIOUS SYSTEM

What to work with	Choice 1	Choice 2	Choice 3	Recommendation
Group of attributes creation	class	struct		**class**
Class prototype	abstract class	interface		**any**
Class attribute	static	dynamic		**static**
Class method	static	dynamic	dynamic anonymous	**dynamic**
Dynamic local variable call	bare use	with "this" keyword		**any**
Attribute accessibility	private	public	protected	**private or public**
Method accessibility	private	public	protected	**any**

VI. Future Work

This study is targeted in the OOP codes those are going to be written for the low power consumption purpose. However, in real world, there are so many useful existing OOP codes those run well in regular system. If these codes can be automatically converted from regular codes to lower power consumption ones, they will be very useful and can reduce time and resources spending for rewriting codes manually. With this reason, our future research would be developing algorithms those can convert regular OOP codes to the lower power consumption ones.

Appendix A

The following are sample of basic codes used in comparing Dynamic and Static attribute. Other testing codes in this research were in similar complexity.

1) Dynamic Attribute

```
using System;
class TestDynamicVariable
{
    double i;
    public TestDynamicVariable()
    {
        for(i = 0; i < 2000000; i+=0.01)
        {
        }
    }
    public static void Main()
    {
        TestDynamicVariable tdv = new TestDynamicVariable();
    }
}
```

2) Static Attribute

```
using System;
class TestStaticVariable
{
    static double i;
    public TestStaticVariable()
    {
        for(i = 0; i < 2000000; i+=0.01)
        {
        }
    }
    public static void Main()
    {
        TestStaticVariable tdv = new TestStaticVariable();
    }
}
```

Appendix B

Fig. 9 is the sample detail graph rendered from result of TOM in testing the CPU usage of Dynamic and Static Attributes. The thick line is the static attribute that works faster and needs less CPU time than dynamic attribute.

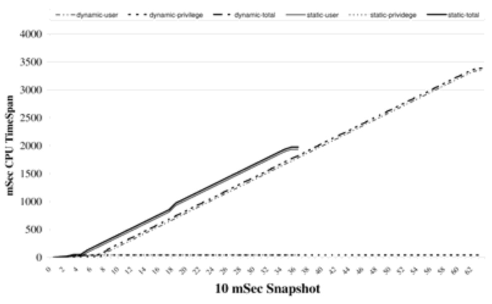

Fig. 9 Detail compared graph rendered from result of TOM in testing dynamic and static variable

References

[1] V. Tiwari, S. Malik and A. Wolfe. Compilation Techniques for Low Energy: An Overview. In 1994 Symposium on Low-Power Electronics, San Diego, CA, October 1994.

[2] Chatzigeorgiou and G. Stephanides. Evaluating Performance and Power of Object-Oriented Vs. Procedural Programming in Embedded Processors. In Ada-Europe 2002, 2002.

[3] Peymandoust, T. Simunic and G.D. Micheli. Low Power Embedded Software Optimization using Symbolic Algebra. In IEEE Proceeding of the 2002 Design, Automation and Test in Europe Conference and Exhibition, 2002.

[4] Y-T S. Li and S. Malik. Performance Analysis of Embedded Software Using Implicit Path Enumeration. In IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, December 1997.

[5] V. Tiwari, S. Malik, and A.Wolfe. Power analysis of embedded software: A first step towards software power minimization. In IEEE Transaction VLSI Systems, December 1994.

[6] K. Naik and D.S.L. Wei. Software Implementation Strategies for Power-Conscious Systems. In Mobile Networks and Applications, June 2001.