# Automatic Decomposition of Java Program for Implementation on Mobile Terminals

Takaaki Umedu, Shigeharu Urata, Akio Nakata and Teruo Higashino

Graduate School of Info. Sci. and Tech., Osaka Univ., Japan
Yamadaoka, Suita, Osaka 565-0871, Japan
{umedu, s-urata, nakata, higashino}@ist.osaka-u.ac.jp

## Abstract

*In this paper, we propose a method for partitioning a given application program that exceeds resource limits of mobile terminals into two module sets. Only a part of modules of the given application is assigned on a mobile terminal and the rest of modules are running on its proxy server, and that the mobile terminal invokes the modules on the server using remote method invocation. It is desirable that we can minimize the total amount of communication, delay time and power consumption between the mobile terminal and its server (here, we call the total amount as the total cost).*

*In the proposed technique, first, a given Java program is repeatedly simulated on a single machine, and we collect the statistics information to estimate the total cost. Then, we give the resource limitation of the mobile terminal such as the memory size and an objective function that shows what total cost should be minimized. Under those constraints, our tool divides optimized division by using Simulated Annealing (SA). We have applied our technique to some application programs and examined its usefulness by evaluating their total costs.*

## 1 Introduction

Recently, handheld terminals such as mobile phones or PDAs have become very popular. And many additional functions have been given to such devices. Especially, since many mobile phones that equip Java Virtual Machine have appeared, the requirement for the applications running on mobile phones has been becoming larger. However, because of the limitations such as available memory size and processing power, not so many applications can be executed on such handheld devices. On the other hand, remote method invocation facilities such as Java RMI [1] and HORB [2] and distribution methods to execute Java programs in distributed environments by using such facilities have been studied.

In this paper, we propose a distribution method where given applications that exceed the limitation of mobile terminals will be executed in mobile terminals virtually by using remote method invocation. In our approach, only a part of the application that includes its user interface is executed on a mobile terminal and large sized modules that cannot be executed in the mobile terminal are executed on its proxy server. The two parts communicate each other by using remote method invocation and the whole system is executed as the same as the original application running on a single machine. In this case, the division is required to satisfy the limitation of mobile terminals and moreover it is better that the division satisfies the requirements from its user's environment. So in this paper, we propose a method for obtaining the division that is as good as possible in a sense of the given metrics under the given restrictions.

To derive such a division, the statistical data about the amount and number of communication among modules are needed. For remote method invocation in Java, many studies are done such as a study of measurement and optimization [3], studies of measurement and scheduling for improvement of real-time systems [4, 5] or a study of efficient implementation of RMI [6]. However, these studies are based on the measurement of performance of distributed applications in practical environments. So, it is difficult to apply those techniques to divide applications that are not specified as programs running on distributed environments.

For statistical performance evaluation, there are many studies based on analysis of source codes such as studies of slicing of parallel Java programs [7, 8]. However, here for simplicity of discussion, we use a simulation based performance evaluation technique. In our technique, additional codes for performance evaluation are inserted to the given source code automatically. The codes are inserted to be called before all the method invocations and record the amount and number of communication between two modules (classes). The statistical performance data can be collected by executing the modified code repeatedly with considering various situations.

Then a division is derived where it satisfies all the restrictions such as the memory size and it is optimized under the given metrics. Here, we have proven that this division problem is NP-hard. Since in general the optimized solution cannot be derived in practical time, we use a heuristic algorithm to get an approximated solution. To solve such NP-hard problems, there are many studies for approximation such as Min-Cut method [9] proposed by Kernighan and Lin, its advanced method [10] and a combination of Min-Cut based graph division algorithm and GA (generic algorithm) [11]. Moreover, SA(Simulated Annealing) [12], liner-time heuristic division method [13] and other methods are proposed. In our technique, we use SA based method because it is known that relatively good solutions can be derived in reasonable computation time.

We have developed a performance evaluation tool and division tool based on SA. Then we have applied our method to some example applications by some dividing metrics. By this evaluation, we have checked our method can reduce calculation time extremely by compared with brute force method and the derived division can be as good as the optimized answer by searching widely.

## 2 Outline of proposed technique

In our proposed technique, a given application is divided into two parts where each class is assigned to either server side modules or client side modules. These two parts of modules communicate with each other so that their behavior is the same as the given application.

### 2.1 Restrictions for target applications

Here, we give some restrictions for the target applications.

- The source codes of the target application are available.
- All the interactions between classes are done by method invocation.
- All the parameter variables are simple data structure that can be passed by value.
- After each method invocation, the called class never keeps the reference to parameter variables of the invocation.

At first, since our performance evaluation method is done by modifying source codes, if a part of source codes is not available like as the standard library, these classes are not cared in division and they are placed on both the server side and client side. Secondly, if some shared variables are used for interaction, they must be replaced by access methods of private variables before applying our method. Thirdly, all the parameters must be passed as if they were passed by value to measure the amount of communication correctly. Lastly, if some classes keep the references to parameter variables, they must be combined by a wrapper class that hides such complicated interactions into it.

### 2.2 Assumption for target environment

In our technique, the divided two parts interact with each other by using Java RMI, in target environment. RMI must be available between the server side modules and client side modules in both directions.

Now in many environments of mobile phones, RMI is not supported and only pull type communication from terminals is served. However, push type communication from servers is defined in WAP2.0, which is the next generation of handheld communication device standard. So by using such services to simulate RMI, our technique can be applied. On the other hand, even in the current generation of cellular phones, push type communication from the server side can be available by using short mail services provided in some cellular phone environments such as C-mail service of ezplus provided by KDDI Japan. So by using such services as triggers, RMI can be simulated in these environments.

## 3 Estimation method of statistical information for optimal division

In our method, we collect statistical data to divide given applications by simulation. By inserting special codes to given source nodes of applications that collect statistical information, and then by executing the inserted codes, the simulation is carried out.

### 3.1 Statistical information and optimizing parameters

At first, we define the target of optimization as follows.

- amount of communication
- communication delay
- power consumption in handheld devices

Here, we assume each parameter is estimated as follows. The amount of communication is evaluated by measuring the communication between modules. The communication delay is approximated by the number of method invocation, since the delay is mainly caused by the overhead of remote method invocation. We assume the power consumption is in proportion to the amount of codes on handheld devices. So we estimate the power consumption by the duration that the modules are executed in handheld devices. Moreover, since the handheld devices often has much less power than servers, we can make the divided application faster by assigning the modules consuming much time into the server side. In our technique, the optimization process is based on an objective function that is constructed as the weighted summation of these three parameters in order to get various optimized division according as we need.

Hereafter, we show our evaluation method.

**Estimation of the amount of communication**

In this paper, we use the summation of size of data exchanged by remote method invocations between classes placed on other sides through the execution of the application as the total amount of communication. So we must estimate it by assuming that all methods are invoked through remote method invocation.

Here we define the amount of communication of a remote method invocation as *data size of parameter values + data size of the return value + overhead of remote method invocation.*

Since the actual amount of communication is slightly different in each execution of application, we must use the averaged data after executing the application many times with practical usage.

**Estimation of communication delay**

In our technique, we assume that communication delay is mainly caused by the overhead of communication and estimate it by the number of remote method invocations simply. In practice, since the communication delay depends on the data size and their transmission speed, we must coordinate the weight of the amount of communication and the number of remote method invocations.

**Estimation of time spent by each module**

In order to estimate the power consumption of a class, we measure the time spent by the methods of that class. Here we measure them simply by calculating the duration from time a method is called to the time the method returns. If we cannot assume that there is no other large load on the environment for measurement, and the target application is not multi-threaded, we must measure them by using more detailed profiling tools such as Extensible Java Profiler [14].

## 3.2 Insertion of measurement codes

In our method, to measure the above three profiles, the measurement codes are inserted to the given application automatically as follows.

```
class Main {
 public static void main(String args[]) {
       :
  Result.AddData(CLASS_Main, CLASS_Sub1,
                RMI_OVERHEAD+4);        // Inserted
  sub1.method1(x);
  Result.AddData(CLASS_Main, CLASS_Sub2,
                RMI_OVERHEAD+8);        // Inserted
  sub2.method2(y);
       :
 }
}
class Result { // class for measurement
 public static final int RMI_OVERHEAD = 20;
 public static final int CLASS_Main = 0;
 public static final int CLASS_Sub1 = 1;
 public static final int CLASS_Sub2 = 2;
 public static final int NUM_CLASSES = 3;
 // amount of communication
 private static int T[][] = new int[NUM_CLASSES][NUM_CLASSES];
 // number of method invocation
 private static int C[][] = new int[NUM_CLASSES][NUM_CLASSES];
 // time spent by each class
 private static long Q[] = new long[NUM_CLASSES];
 // the class that currently executing method belongs to
 private static int CurrentClass = CLASS_Main;
 // the time when currently executing method was invoked
 private static long InvokedTime = System.currentTimeMillis();
 public static void AddData(int from, int to, int size) {
  T[from][to] += size;
  C[from][to] ++;
  long CurrentTime = System.currentTimeMillis();
  Q[from] = CurrentTime - InvokedTime;
  InvokedTime = CurrentTime;
 }
}
```

The inserted codes will call the recording function of *Result* class to record the information. In the above example program, we assume that the overhead of RMI is 20 bytes to calculate the amount of communication. A constant is defined for each class. The exchanged data size and the number of method invocations are recorded in two dimensional arrays T and C, respectively. Those two dimensional arrays are reffered by a pair of a caller class and a callee class. The time spent by each class is recorded in array Q.

## 4 Formulation of module assignment problem

We have formulated our module assignment problem as follows.

**input:**

- a set of modules $M = \{m_1, ..., m_n\}$
- the memory size of each module $S : M \mapsto \mathbf{N}$
- available memory on the client side $m_c \in \mathbf{N}$

- modules explicitly assigned to the server side or client side $M_S, M_C \subset M$
- the amount of communication between modules $T : M \times M \mapsto \mathbf{N}$
- the number of method invocations between modules $C : M \times M \mapsto \mathbf{N}$
- the time spent by each module $Q : M \mapsto \mathbf{N}$
- the weights for the optimization parameters $K_1, K_2, K_3 \in \mathbf{N}$

**output:**

- a pair of assignments $M_s$ and $M_c$ that satisfy the following restriction and minimize the value of the objective function (here, $M_s$ and $M_c$ denote the server side modules and client side modules, respectively).

**restriction:**

- $M_s \cup M_c = M, M_s \cap M_c = \emptyset$
- $\sum_{m \in M_1} S(m) \leq m_c$

**objective function:**

$$F(M_s, M_c) = K_1 \sum_{m_1 \in M_s, m_2 \in M_c} T(m_1, m_2)$$
$$+K_2 \sum_{m_1 \in M_s, m_2 \in M_c} C(m_1, m_2) + K_3 \sum_{m \in M_c} Q(m)$$

Here, we assume that $T(m_1, m_2)$ denotes the sum of the ammounts of communication from $m_1$ to $m_2$ and that for the reverse direction. If we prefer to give different weights for the both directions, we can do so. $K_1$, $K_2$ and $K_3$ denote the weights for the amount of communication, the number of method invocations and the CPU time spent in the client side, respectively. We can optimize them by adjusting the values of $K_1$, $K_2$ and $K_3$.

Here, we discuss about the computational complexity for this module assignment problem. Since a graph partitioning problem dividing a given set of vertexes $V = v_1, ..., v_{2n}$ of $G = (V, E)$ into two sets $V_1$ and $V_2(|V_1| = |V_2| = n, V_1 \cup V_2 = V, V_1 \cap V_2 = \emptyset)$ by minimizing the cut $W(V_1, V_2) = \sum_{v_1 \in V_1, v_2 \in V_2, (v_1, v_2) \in E} 1$ is known as a NP-hard problem [15].

Here, any graph partitioning problem can be reduced to the module assignment problem as follows.

- $M = V \cup \{v_c\}$
- $\forall v \in M : S(v) = 1$
- $m_c = |V|/2 + 1$
- $\{v_c\} \in M_c$
- $T(v_1, v_2) = \begin{cases} 1 & ((v_1, v_2) \in E) \\ |E| + 1 & (v_1 = v_c \text{ or } v_2 = v_c) \\ 0 & (\text{otherwise}) \end{cases}$
- $K_1 = 1, K_2 = 0, K_3 = 0$, and all the other parameters are set to 0 or $\emptyset$.

For the given cost function $W(V_1, V_2)$ of the graph partitioning problem, we define the objective function of our module assignment problem as $F(M_s, M_c) = |M_s| \times (|E| + 1) + W(V_1, V_2)$ where $M_s = V_1$ and $M_c = V_2 \cup \{v_c\}$ hold. For any division of $G = (V, E)$, the cost $W$ holds $W(V_1, V_2) \leq |E|$. Therefore, for any two divisions $\{M_{s1}, M_{c1}\}$ and $\{M_{s2}, M_{c2}\}$, $|M_{s1}| < |M_{s2}| \Rightarrow F(M_{s1}, M_{c1}) < F(M_{s2}, M_{c2})$ holds. So the optimal result $M_s$ of our module assignment problem always minimizes the size of $|M_s|$. Also, the number of elements in $|M_s|$ holds $|M_s| = |M| - m_c = |V|/2$ because at most $|V|/2$ vertexes can be assigend to $M_c$. In this case, since the value of the objective function holds $F(M_s, M_c) = (|V|/2) \times (|E| + 1) + W(M_s, M_c - \{v_c\})$, the optimized division must minimize $W(M_s, M_c - \{v_c\})$. So we can get the optimized division of $G = (V, E)$ from the optimized assignment. Since this reduction can be done in polynomial time, our module assignment problem is proved as NP-hard.

## 5 Optimizing the assignment of modules

Since this module assignment problem is NP-hard, we cannot the optimized result in practical time. So we use a heuristic algorithm to get approximate results. Here, we use a SA (Simulated Annealing) based method.

In SA based methods, candidate results are repeatedly improved in order to obtain the optimized result. A neighbor of the current candidate is selected as the new candidate randomly and if the new candidate is better than the current one, the current candidate is replaced by the new candidate. Also even if the new candidate is worse than the current one, the replacement occurs in some probability. The probability starts with a large value and is decreased mildly. It is known that by making the decrement very mild and trying enough times, relatively good results can be obtained.

## 6 Example applications

### 6.1 Ex1: randomly generated modules

At first, we have applied our technique to some randomly generated problems.

Table 1: Amount of communication between server and clients
(1) 30 modules

| # of loops | 10 | 50 | 100 | 150 |
|---|---|---|---|---|
| SA | 170KB (0.06s) | 130KB (0.29s) | 104KB (0.56s) | 104KB (0.85s) |
| brute force | 98KB(200.54s) | | | |

(2) 50 modules

| # of loops | 10 | 100 | 1000 | 2000 |
|---|---|---|---|---|
| SA | 276KB (0.14s) | 276KB (1.42s) | 253KB (14.13s) | 213KB (28.31s) |
| brute force | —(—) | | | |

- available memory on client side: 120.0KB
- number of modules : 30, 50
- size of modules : randomly generated at most 30.0KB.
- amount of communication : randomly generated

The results are shown in Table 1. Each table shows the amount of communication between the server and client where the numbers shown in the parentheses are time spent for calculation.

From the above results, we can say that by using SA based method we can obtain enough good results as an approximation for large sized problems that cannot be solved by the round robin method. And it is shown that enough large counts of loops are needed to derive good results close to the optimal ones.

## 6.2 Ex2: an existing application

We have chosen an existing Java application for editing pictures. This application consists of 68 classes where the average size of classes is about 5 KB. At the first, we have collected our statistical information for this application. The results are shown in Fig.1. Each graph shows (1) the amount of communication and (2) the number of method invocation between each pair of two classes (the pairs that never communicate with each other are omitted in these graphs) and (3) CPU time spent by each module.

Then, we have applied our module assignment algorithm to the application based on the above information.

To evaluate the usefulness of our technique, we have evaluated in some conditions by changing the optimization parameters $K_1, K_2, K_3$ and available memory on the client. The results are shown in Table 2. The first column shows the result of optimization for the amount of communication where only parameter $K_1$ is used while $K_2$ and $K_3$ are ignored. The second and third columns show the results of optimization for the number of method invocations and power consumption respectively in the same way.

Table 2: Result of division for the example application

| memory of client (KB) | amount of comm. (bytes) | number of method invocation | CPU time of client (ms) |
|---|---|---|---|
| | | $K_1 = 1, K_2 = 0, K_3 = 0$ | |
| 45 | 234014 | 4162 | 134199 |
| 50 | 18566 | 873 | 687890 |
| 55 | 2870 | 104 | 709080 |
| 60 | 2242 | 73 | 709080 |
| | | $K_1 = 0, K_2 = 1, K_3 = 0$ | |
| 45 | 234014 | 4162 | 134199 |
| 50 | 18958 | 908 | 687900 |
| 55 | 3594 | 136 | 755907 |
| 60 | 3462 | 108 | 691717 |
| | | $K_1 = 0, K_2 = 0, K_3 = 1$ | |
| 45 | 238198 | 4297 | 363280 |
| 50 | 236322 | 4236 | 378151 |
| 55 | 236090 | 4230 | 378151 |
| 60 | 235294 | 4218 | 378151 |

From these results, we can say our technique is useful for designing practical applications running on mobile terminals.

## 7 Conclusion

In this paper, we have developed a tool for gathering statistical information of Java programs and proposed a module assignment technique where the amount of communication between the server side and client side, elapsed time or power consumption on handheld devices are minimized. We also have applied our technique to some examples and obtained useful results in reasonable time.

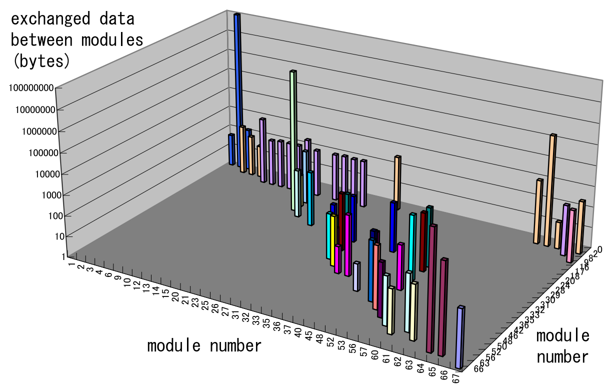
As our future work, we are planning to apply our technique to various applications. We also would like to find more effective approximation algorithms.

## References

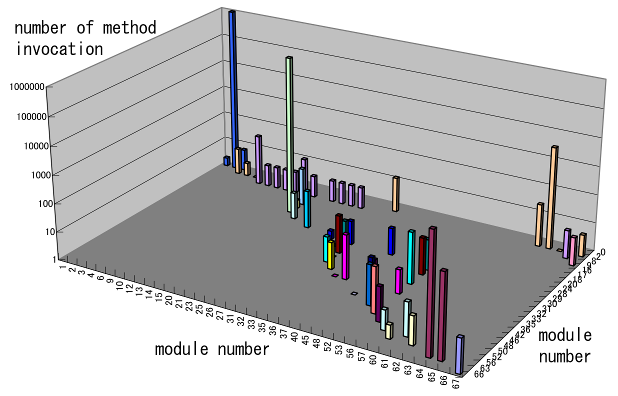
[1] Grosso W. : Java RMI, *O'Reilly & Associates, Inc.* (2002)

[2] HORB Open : http://www.horbopen.org/ (2001)

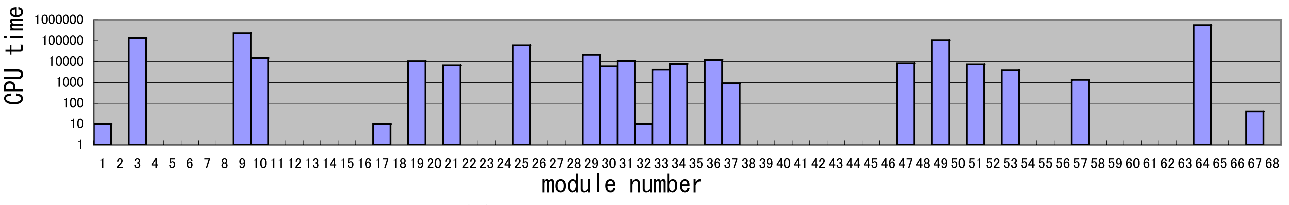
[3] Matjaz B.J., Ivan R., Marjan H., Alan P.S. and Simon N. : Java 2 Distributed Object Models Performance Analysis, Comparison and Optimization,



(1) Exchanged data between modules



(2) Number of method invocations between modules



(3) CPU time spent by modules

Figure 1: Interaction among modules and CPU time spent by modules in an example application

*Proc. of 7th Int. Conf. on Parallel and Distributed Systems (ICPADS'00)*, pp. 239–246 (2000)

[4] Kalogeraki V., Melliar-Smith P.M. and Moser L.E. : Using Multiple Feedback Loops for Object Profiling, Scheduling and Migration in Soft Real-Time Distributed Object Systems, *Proc. of 2nd IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing*, pp. 291–300 (1999)

[5] Flores A.P., Nacul A., Silva L., Netto J., Pereira C.E. and Bacellar L. : Quantitative Evaluation of Distributed Object-Oriented Programming Environments for Real-Time Applications, *Proc. of 2nd IEEE Int. Symp. on Object-Oriented Real-Time Distributed Computing*, pp. 133–138 (1999)

[6] Kono K. and Masuda T. : Efficient RMI, Dynamic Specialization of Object Serialization, *Proc. of 20th Int. Conf. on Distributed Computing Systems (ICDCS 2000)*, pp. 308–315 (2000)

[7] Zhao J. : Slicing Concurrent Java Programs, *Proc. of 7th Int. Workshop on Program Comprehension*, pp. 126–133 (1999)

[8] Zhao J. : Multithreaded Dependence Graphs for Concurrent Java Program, *Proc. of 1999 Int. Symp. on Software Engineering for Parallel and Distributed Systems*, pp. 13–23 (1999)

[9] Kernighan B.W. and Lin S. : An Efficient Heuristic Procedure for Partitioning Graphs, *Bell Syst. Tech. J.*, vol.49, no.2, pp. 291–307 (1970)

[10] Krishnamurthy B. : An Improved Min-Cut Algorithm for Partitioning VLSI Networks, *IEEE Trans. Computers*, vol.33, no.5, pp. 438–446 (1984)

[11] Bui T.N. and Moon B.R. : Generic Algorithm and Graph Partitioning, *IEEE Trans. Computers*, vol.45, no.7, pp. 841–855 (1996)

[12] Johnson D.S., Aragin C., McGeoch L., and Schevon C. : Optimization by Simulated Annealing, An Experimental Evaluation, Part1, Graph Partitioning, *Operations Research*, vol.37, pp. 865–892 (1987)

[13] Fiduccia C.M. and Mattheyses R.M. : A Linear-Time Heuristic for Improving Network Partitions, *Proc. of 19th Design Automation Conference*, pp. 175–181 (1982)

[14] Sebastien Vauclair : Extensible Java Profiler, http://ejp.sourceforge.net/

[15] Garey M.R. and Johnson D.S. : Computers and Intractability: A Guide to the Theory of NP-completeness, Freeman (1979)