A Modeling Method by Eliminating Execution Traces for Performance Evaluation

Kouichi Ono[†], Manabu Toyota[‡], Ryo Kawahara[†], Yoshifumi Sakamoto[‡], Takeo Nakada[†] and Naoaki Fukuoka[§]
[†] IBM Research - Tokyo, 1623-14 Shimotsuruma, Yamato-shi, Kanagawa, 242-8502 Japan
[‡] Component Technology Solution, IBM Japan, Ltd., 338 Enpukuji-cho, Nakagyou-ku, Kyoto, 604-8175 Japan
[§] Tokyo R&D Center, KYOCERA MITA Corp., 2-14-9 Tamagawadai, Setagaya-ku, Tokyo, 158-8610 Japan
[onono|mtoyota|ryokawa|sakay|nakada]@jp.ibm.com, naoaki_fukuoka@kyoceramita.co.jp

***Abstract*—This paper describes a system-level modeling method in UML for performance evaluation of embedded systems. The core technology of this modeling method is reverse modeling based on dynamic analysis. A case study of real MFPs (multifunction peripherals/printers) is presented in this paper to evaluate the modeling method.**

I. Introduction

It is necessary to decide the system architecture in the early stages of product development to achieve the required performance. System-level simulations based on models are good solutions for performance estimation in the early stage of product development. Unified Modeling Language (UML) is widely used to model a large variety of application software. Nowadays, it is generally recognized that embedded and real-time systems are good target for UML [2].

This paper proposes a reverse modeling method in UML for performance evaluation of system-level design of legacy embedded systems. The modeling method is a reverse engineering for creating abstract behavioral features using dynamic analysis of the existing systems.

II. Related Work

Tonella and Potrich proposed a reverse modeling method using static flow analysis [6] which was a technique for the automatic extraction of UML interaction diagrams from C++ code. The method makes no abstractions when interaction diagrams are extracted. Therefore, it is not able to create abstract models of system-level design.

Briand et al. proposed a reverse modeling method using dynamic analysis [1] which formally defined a model transformation approach using metamodels and consistency rules. The method also makes no abstraction when a model is created by applying rules.

III. Modeling Method and Simulation

An overview of our model-based performance evaluation method is shown in Fig. 1. The method starts by capturing the execution traces of the target embedded system's behaviors [4]. The execution traces consist of the entries and exits of function invocations with their timestamps and the values of selected parameters. The execution traces are eliminated by the reverse modeling, sequence diagrams are created, and then state machine diagrams are created. After that, the model is

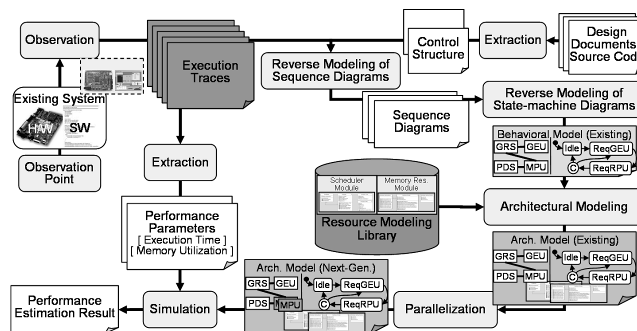

Fig. 1. Overview of Model-based Performance Evaluation

modified to represent the architectural changes that denotes parallelization to improve the system performance for the next-generation products. At the same time, performance information is extracted from the execution traces as external files, and used when the model is executed to simulate the system performance. This means that the model simulation is a sort of trace-driven simulation [5]. A simulation of the modified model with the performance parameter files will estimate the system performance at the next-generation products.

This paper is focusing on the reverse modeling method, which is the key technology of our work. The method involves the modeling using dynamic analysis, which creates a model from the execution traces captured while observing the behaviors of the embedded system. There are two reasons why the method is defined as a method using dynamic analysis. The first reason is the accuracy of the system behaviors. Static analysis can extract precise information from the source code. However, several aspects of dynamic characteristics make accurate analysis difficult. These include data dependencies, pointers to functions, and others. Dynamic analysis can elucidate the performance of software components.

The second reason is abstraction. The complexity of model must be reduced by abstraction of behaviors. To abstract the behaviors, it is crucial to analyze the execution traces from an appropriate viewpoint. Unimportant information for the behaviors as seen from the selected viewpoint should be eliminated in the generated model. Since our method assumes the model is used for the performance estimation by its simulation, our concern is performance. For the dynamic analysis, execution

978-3-9810801-6-2/DATE10 © 2010 EDAA

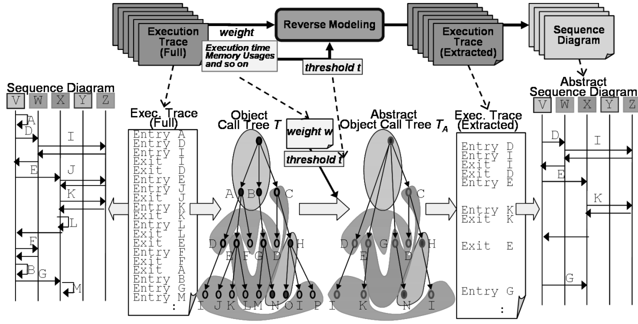

Fig. 2. Abstraction of Execution Traces

traces can be abstracted when the execution traces include performance information, such as execution times, resource utilization, and so on. However, in the static analysis, source code cannot be abstracted to reveal the performance because performance information is not included in the source code and the performance cannot be predicted without execution.

IV. Reverse Modeling of Behavior

The reverse modeling creates behavioral features from execution traces. The execution traces are abstracted based on the performance, so that unimportant invocations are merged to important invocations. The importance of each function invocations is calculated as a weight from the performance information in the execution traces. For example, the execution time of each invocation can be used as the weight. The selection based on the importance of the function invocations is called *abstraction of execution traces*. The selected invocations are regarded as representatives of the system behavior, which are dominant about the system performance on the behavior. Therefore, the total execution time of selected invocations is a large portion of the entire execution time of system behavior.

The outlook of *abstraction of execution traces* is shown in Fig. 2. An execution trace includes entries and exits for function invocations, timestamps, and other performance information, Each function is subject to the identified object.

An object call tree T is defined as

$$T = (N, s), \tag{1}$$

where N is a set of nodes that denote function invocations, and s is a mapping from the node to a sequence of nodes that denotes a list of function invocations called from inside the invocation. The mapping s is defined as a partial function:

$$s : N \longmapsto \mathbf{seq}\, N, \tag{2}$$

where $\mathbf{seq}\, N$ is a set of sequences of zero of more elements in the set N. The sequence $s(n)$ is a list of sub-nodes of a node n in the object call tree.

Also, a mapping from the function invocation to object is defined as a total function:

$$o : N \longrightarrow O, \tag{3}$$

where O is a set of identified objects. The object $o(n)$ is an object where the function invocation n is the subject.

Suppose that the weight of the function invocations w is given, which is a mapping from function invocations N to the weight W. It is defined as a total function as follows.

$$w : N \longrightarrow W \tag{4}$$

The mapping w must satisfy an invariant such that

$$\forall n \in N \left[w(n) \geq \sum_{i=1}^{\#s(n)} w(s(n)[i]) \right], \tag{5}$$

where $\#s(n)$ is the length of sequence $s(n)$ and $s(n)[i]$ is an element as position i in the sequence $s(n)$.

The execution time of function invocation satisfies the invariant because the execution time of every function invocation equals the total amount of execution time of the sub-function invocations. Therefore, the execution time can be used as w. Alternatively, the amount of the datagram received from or sent to the network or bus during the function invocation, and the amount of memory allocated during the function invocation can be used as w.

An abstract object call tree T_A is defined as

$$T_A = (N_A, s_A), \quad N_A \subseteq N, \quad s_A : N_A \longmapsto \mathbf{seq}\, N_A, \tag{6}$$

where the node sequence $s_A(n)$ denotes a sequence of node children as function invocations within the function of node n in T_A. In additions, N_A and s_A must satisfy the relationship:

$$\begin{aligned} &\forall n \in N_A,\ \forall i \in \mathbb{N} \\ &\quad [\ 1 \leq i \leq \#s_A(n) \rightarrow \\ &\qquad r(n, s_A(n)[i], s, o(n), o(s_A(n)[i]))\ ], \end{aligned} \tag{7}$$

where $\mathbb{N}$ is the set of natural numbers. And the reachability predicate $r(m, n, s, o_m, o_n)$ means that it is reachable from the node m to n through the intermediate nodes which are subject to o_m or o_n and are calculated by s. The predicate r is defined as follows.

$$\begin{aligned} &r(m, n, s, o_m, o_n) \equiv \\ &\quad o(m) = o_m \land \\ &\quad [\ m = n \land o(n) = o_n \lor \\ &\qquad m \neq n \land rs(m, n, s, o_m, o_n)\ ] \end{aligned} \tag{8}$$

$$\begin{aligned} &rs(m, n, s, o_m, o_n) \equiv \\ &\quad \exists i \in \mathbb{N}, k \in N\ [\quad 1 \leq i \leq \#s(m) \land k = s(m)[i] \\ &\qquad \land\ (o(k) = o_m \lor o(k) = o_n) \\ &\qquad \land\ r(k, n, s, o(k), o_n)\quad] \end{aligned} \tag{9}$$

Suppose that the threshold t is given as a real number in the closed interval $[0, 1]$. And N_A and s_A must satisfy the following relationship to t:

$$\forall n \in N_A \left[\sum_{i=1}^{\#s_A(n)} w(s_A(n)[i]) \geq t * w(n) \right] \tag{10}$$

The object call tree T and the abstract object call tree T_A is shown in Fig. 2. The difference between T and T_A is the elimination of some nodes whose weights are not dominant to the total amount. The node which is not dominant is eliminated in two ways. The first one is simplification of nodes in an object, which is related to Equation 7. In Fig. 2, the root node invokes the node A and A invokes D. Both the root node and A are subject to the same object V, and D is subject to W. In short, A is an intermediate node on the interaction between V and W, i.e. V interacts with W via the invocation of A. The reverse modeling method suppresses the intermediate nodes by merging them into the ancestor node which is the representative of the object interactions.

The second one is condensation of leaf nodes, which is related to Equation 10. In Fig. 2, the node E invokes J, K and L. The node E and L are subject to the same object X, and L is simplified and merged to E by the first elimination way described above. The remaining node J and K are leaf nodes and they are subject to the object Z which differs from X. It represents that there are object interaction between X and Z. It is supposed that the weight of K invoked from E is high but the weight of J invoked from E is very low. The reverse modeling method condenses the leaf nodes with low weight where nodes with high weight exist in the same object of the leaf nodes, by merging them into the ancestor node which is the representative of the object interactions. Threshold is used for deciding whether the leaf node is condensed.

In both node elimination ways, the ancestor node acts over the eliminated nodes. It means that the *abstraction* substitutes the ancestor nodes for the eliminated nodes as the representatives in the system behavior. Note that the node elimination does not remove the eliminated nodes, but merges them to the ancestor nodes for simplifying the behavioral representation.

After the abstract object call tree is created, the execution trace is extracted by filtering with the nodes N_A, then the extracted execution trace is transformed into the abstract sequence diagram shown at the right hand side in Fig. 2. State machine diagrams of the identified objects can be created from the abstract sequence diagram by existing model transformation technologies [3].

V. Case Study - MFP Print Job Processing

A case study about applying this method to a real MFP is presented in this section. An execution trace sample about the print job processing component is shown at the left side in Fig. 3. The print job processing is significant component of MFP products. The execution trace is converted to the function call tree T_f, and the performance parameters are extracted from the trace shown at the right side in Fig. 3. The nodes in T_f with digits ("14", "15" and "31") and the mark "$\star$" denotes iterative invocations, the digits are the numbers of iterations, and the mark "$\star$" means lots of iterations.

An object call tree T is created from the function call tree T_f by object identification. For the reverse modeling, it is necessary to identify objects by grouping functions or classes of the source code. Since the system-level behavior should be considered as interactions of coarse-grained objects, "modules" specified in the design documents are suitable as the objects at this case. The tree T is shown at the left side in Fig. 4. The tree T has 15,340 nodes. Since the tree T is very large, its sequence diagram must be too much complicated. Fig. 4 shows the elimination of the tree T by *abstraction* argued in Section IV, in order to create an abstract object call tree T_A. At the *abstraction* of this case, the execution time of function invocation is used as the weight, and the threshold t given as 0.95 (95%). The nodes with the mark "$\star$" have very short execution times, therefore, the nodes are eliminated although they are numerous. And, the intermediate nodes are merged to the ancestor nodes because they are in the same object of their ancestor. The tree T_A created by the abstraction has 155 nodes. The number of nodes will be reduced considerably by the abstraction method. And the behavioral model created with the abstract tree is appropriately exact because the created abstract tree T_A satisfies that the amount of execution times in T_A must be over 95% (the threshold t) of the total amount of the execution traces. The tree T_A is transformed into a sequence diagram shown in Fig. 5.

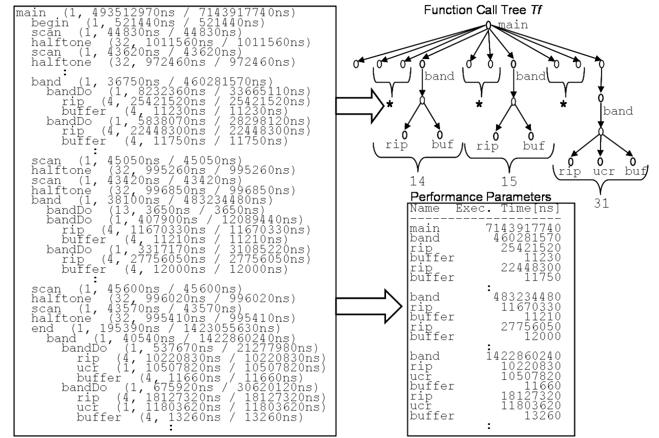

Fig. 3. An Execution Trace of Print Job Processing (as Function Call Tree and Extracted Performance Parameters)

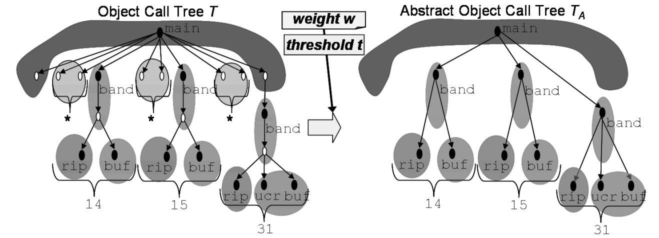

Fig. 4. Abstract Object Call Tree of Print Job Processing

VI. Evaluation

As an evaluation of the reverse modeling method, a comparison of system performance data between model simulation and real system is discussed here. As the target real system,

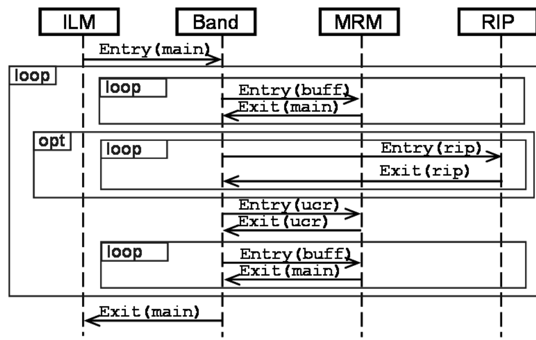

Fig. 5. Abstract Sequence Diagram of Print Job Processing

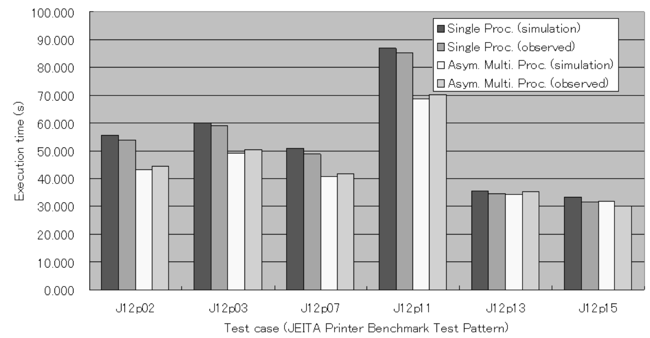

Fig. 6. Evaluation with JEITA Printer Benchmark Test Patterns

we made a MFP product prototype. The MFP prototype was developed on a FPGA board with dual-core PowerPC processors. The source code of existing MFP product was ported to the platform, which was designed for single processor system. After porting, we re-designed the MFP software for AMP architecture. There are two major software components in MFP print job processing. So one software component was allocated to one PowerPC processor core, and another software component was allocated to another PowerPC processor core, and they are working in parallel. As the results of MFP prototype development, we made the prototype for single processing system and the prototype for AMP architecture.

The reverse modeling and simulation for performance evaluation were done according to the following procedure. At first, execution traces were captured by observation of the prototype for single processing system. JEITA Printer Benchmark Test Patterns were used for the system observation. And, the model of single processing system was created by the reverse modeling method from these execution traces. At the same time, the performance parameters were extracted from the execution traces. After that, the model of AMP architecture was created by parallelizing the model of single processing system. And then, performance results were evaluated by simulating the model of AMP architecture with the performance parameters. Finally, the performance evaluation results were compared with the performance data by the system observation of the MFP prototype of AMP architecture.

Fig. 6 shows the comparison of the model simulation results and the MFP prototypes. There are the prototype for single processing system ("Single Proc. (simulation)"), the model of single processing system ("Single Proc. (observed)"), the prototype for dual-core AMP architecture ("Asym. Multi. Proc. (simulation)"), and the model of dual-core AMP architecture ("Asym. Multi. Proc. (observed)"). Six test patterns in J12 set of JEITA Printer Benchmark Test Patterns were used for the comparison. For example, "J12p07" means that the test case uses the page 07 test pattern in J12 set. Each test case consists of the four same pages, and the test case "J12p07" is composed of the four same pages as the page 07 in J12 set. For most print test patterns, the performance evaluation results of the model of dual-core AMP architecture are nearly equal to the results of MFP prototype for dual-core AMP architecture. Note that, for the AMP architecture cases, there are performance overhead about data transfer between processors via the internal bus and memory access. The performance overhead is relatively bigger at the cases of small print test patterns (J12p13 and J12p15) than the cases of large print test patterns (J12p02, J12p03, J12p07 and J12p11).

VII. Conclusions

This paper presents a reverse modeling method for performance evaluation using dynamic analysis of legacy embedded systems. The method creates abstract behavioral feature of models by eliminating execution traces of existing products. Abstraction of behavioral feature by eliminating execution traces is the key technology, which is done by considering every function invocation's weight dominant to the total amount how much the function will effect the entire system performance. A case study about applying this method to real MFP is presented in this paper as an evaluation of the reverse modeling method.

References

[1] L. C. Briand, Y. Labiche, and J. Leduc. Toward the reverse engineering of UML sequence diagrams for distributed Java software. *IEEE Transactions on Software Engineering*, 32(9):642–663, September 2006.

[2] S.-U. Jeon, J.-E. Hong, and D.-H. Bae. Interaction-based behavior modeling of embedded software using UML 2.0. In *Proceedings of Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC 2006)*, pages 351–355, Gyeongju, Korea, April 2006. IEEE Computer Society.

[3] H. Liang, J. Dingel, and Z. Diskin. A comparative survey of scenario-based to state-based model synthesis approaches. In *Proceedings of the 2006 International Workshop on Scenarios and State Machines (SCESM 2006, co-located to ICSE 2006)*, pages 5–12, Shanghai, China, May 2006. IEEE Computer Society / ACM.

[4] N. Ohba and K. Takano. Hardware debugging method based on signal transitions and transactions. In *Proceedings of the 11th Asia South Pacific Design Automation Conference (ASP-DAC 2006)*, pages 454–459, Yokohama, Japan, January 2006.

[5] C. A. Prete, G. Prina, and L. Ricciardi. A trace-driven simulator for performance evaluation of cache-based multiprocessor systems. *IEEE Transactions on Parallel and Distributed Systems*, 6(9):915–929, September 1995.

[6] P. Tonella and A. Potrich. Reverse engineering of the interaction diagrams from C++ code. In *Proceedings of the 19th International Conference on Software Maintenance (ICSM 2003)*, pages 159–168, Kloveniersburgwal 29, Amsterdam, The Netherlands, September 2003. IEEE Computer Society.