Software Synthesis for Trade-off Design

Akiyoshi Sato,† Masato Miki,‡ Toru Yamanouchi,† and Masanobu Watanabe†
†C&C Research Laboratories
‡Transmission Engineering Division
NEC Corporation
4-1-1 Miyazaki, Miyamae-ku, Kawasaki, 216 JAPAN
a-sato@swl.cl.nec.co.jp

Abstract

This paper details a case study of trade-off design in software synthesis. Domain-oriented software synthesis technology enables software designers to encode their own specific knowledge of software design into transformation rules. Thus, generated software is optimal and actually usable for the domain. However, optimal implementation cannot be decided in advance if performance specifications, e.g., memory size limits and execution time limits, are unclear. That is, it is difficult to develop transformation rules which generate optimal software for their domain or situation if performance requirements are unclear beforehand.

This paper proposes a method of trade-off design in software synthesis, and applies this method to develop a file access program generator called POT-DB. The proposed method includes procedures for (1) extracting trade-off parameters as input specifications, (2) designing transformation rules for trade-off parameters to generate programs, and (3) designing performance measurement rules to allow designers to notice the effects of trade-off parameters. Based on the results of applying POT-DB in developing a sales analysis and ordering system, all performance requirements have been satisfied, and application productivity has been improved 1.9 times. Moreover, it is shown that total productivity including the development cost for the POT-DB itself can be improved if POT-DB is applied to at least four application systems. The developed sales analysis and ordering system has been in daily operation with over 10,000 portable terminals at more than one hundred branch stores.

1 Introduction

Many domain-oriented software synthesis systems, or generators, can successfully be applied to improve software productivity [1] [2] [3] [4]. The authors have developed domain-oriented software synthesis systems, e.g., a GUI software generator [7], a switching software generator [10], etc., using the software synthesis shell SOFTEXSHELL [9], which have actually improved software productivity. Domain-oriented software synthesis technology enables software designers to encode their own specific knowledge for software design into transformation rules. Thus, the generated software is optimal and actually usable for the domain.

However, optimal implementation cannot be decided in advance, if performance specifications, e.g., memory size limits and execution time limits, are unclear. That is, it is difficult to develop transformation rules which generate optimal software for their domain or situation if performance requirements are unclear beforehand. Consequently, it is important that an implementation method be controllable from the input specifications for a generator. In the case of a trade-off design between memory size and execution time, for example, a size of read buffer and sorting algorithm need to be defined as input specifications for generators. Moreover, to modify the input specifications in satisfying performance requirements, software designers need to know the effects of these performance specifications on memory size and execution time.

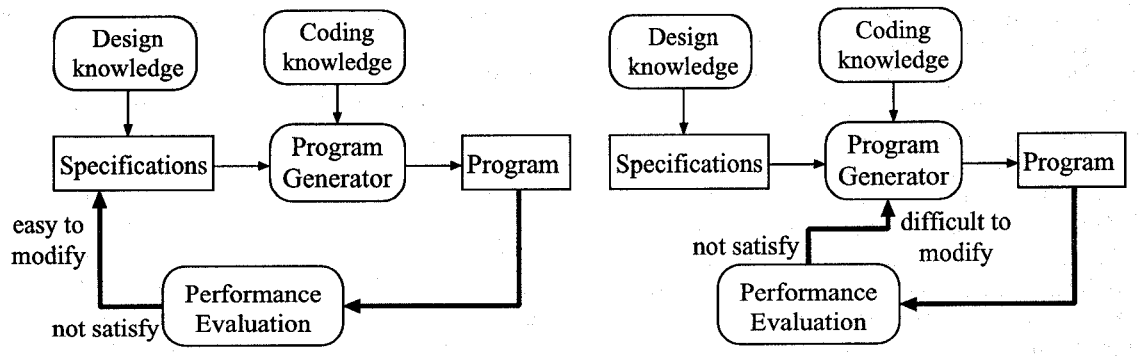
This paper proposes a method of trade-off design in software synthesis. The proposed method includes procedures for (1) extracting trade-off parameters as input specifications, (2) designing transformation rules for trade-off parameters to generate programs, and (3) designing performance measurement rules to allow designers to notice the effects of trade-off parameters. The following sections describe a method using as a case study which describes the trade-off design process with domain-specific generators.

2 Problem and Motivation

2.1 Trade-off Design Problem in Software Synthesis

In the software industry, designers must routinely design software approaching the absolute limits of the hardware resource to reduce cost, that is, for example, equipment with built-in software and real-time software. In these cases, software designers must satisfy not only function requirements but also performance requirements, e.g., memory size limits and execution time limits. In general, memory size and execution time have a *trade-off relationship*, that is, a larger memory size shortens the execution time, and vice versa. Actually, at NEC, software designers must take the trade-off relationship between memory size

1068-3062/96 $5.00 © 1996 IEEE
Proceedings of KBSE '96



(a) Easy to modify design specifications (b) Difficult to modify generator

Figure 1: Trade-off design problem in software synthesis.

and execution time into consideration in terms of exacting performance requirements.

Trade-off design between memory size and execution time needs to be done in both the designing phase and the coding phase. From the viewpoint of automating a coding phase by generators,

- For trade-off in the *designing phase,* generators support trade-off design by generating programs. That is, trade-off designers can immediately know the performance evaluation results by executing a generated program without programming. Trade-off designers can trade-off design by describing the trade-off parameters as design specifications which can be easily modified, as shown in Figure 1 (a).

However, Figure 1 (b) shows that,

- For trade-off in the *coding phase,* generators decide the fixed implementation using their own decisions. Thus, for trade-off design, designers must modify the design decisions encoded in the transformation rules. However, this modification is difficult for the designers, because this situation is the same as a C-language programmer having to modify a C-language compiler.

In short, generators are very useful for trade-off design without programming by modifying the design specifications. However, generators can be very difficult for trade-off design if the key parameters for trade-off have been encoded into their transformation rules. The explicitness of the trade-off parameters is especially important. This is a crucial problem for trade-off design in software synthesis.

2.2 Example of Trade-off Design

This subsection describes the trade-off design problem through an actual example. First, the file access program generator, called POT-DB (portable terminal database system), is explained as an example. Figure 2 shows the POT-DB system structure. POT-DB is a software synthesis system which generates file access programs (*API:* application interface program and *Loader:* index construction program) and a memory size document from file access specifications (logical data access specifications, index specifications, and file structure specifications).

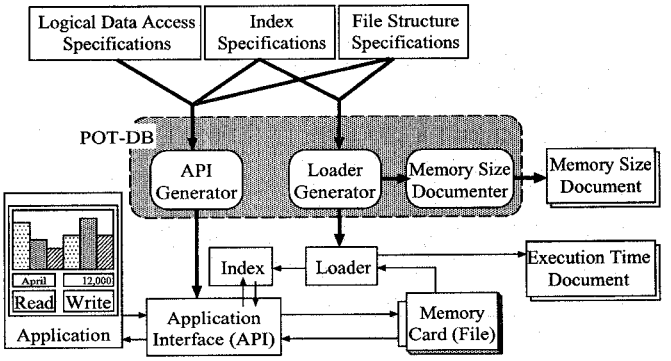


Figure 2: POT-DB system structure.

The file access program is executed with portable terminals, which is running on MS-DOS. The portable terminals have 640 Kbytes of a main memory, 2 Mbytes of an extended memory, a memory card and an Intel 486 processor. In general, a portable terminal needs to have a limited hardware resource, e.g., small size of memory and a low performance processor, for cost reduction. Thus, in the case of this file access program, designers must design and modify software again and again to satisfy the performance requirements. Therefore, POT-DB is useful for trade-off design to improve software productivity.

The loader program constructs many indexes of files on a memory card. The application program can logically read/write the files on a memory card through the application interface (API) without implementation knowledge of the file structure. The application interface program uses the index to read/write the files on a memory card. POT-DB helps programmers and designers through loader program generation and application interface program generation. Moreover,

POT-DB also helps them to design trade-off by generating a memory size document and an execution time document. Designers can easily modify the specifications to satisfy performance requirements by taking these documents into consideration.

Figure 3 shows a logical file access mechanism using a index. The loader program constructs indexes as lists of address pointers which are sorted based on their key items. For example, `sales-index` has a list of address pointers as `<003, 001, 002>`, which is sorted based on its key item as `sales`. Thus, the application program can read the data (`<$5,300, 8895, orange juice>`) in the first place, which is the largest sales based on the index `sales-index`, through an application interface.

By using POT-DB, many application systems can easily be developed. The target application system in this paper is a sales analysis and ordering system, which displays sales in various styles, e.g., daily sales graphs, monthly sales tables, etc., and it is used for placing an order for an article.

Table 1: Performance requirements for the sales analysis and ordering system.

Requirements	Values / Limits
Function	Number of indexes: about 20-30 Item number of each index: max 4000 items
Memory size	Heap memory size: max 121 Kbytes
Execution time	Index construction: max 8.0 seconds for total indexes

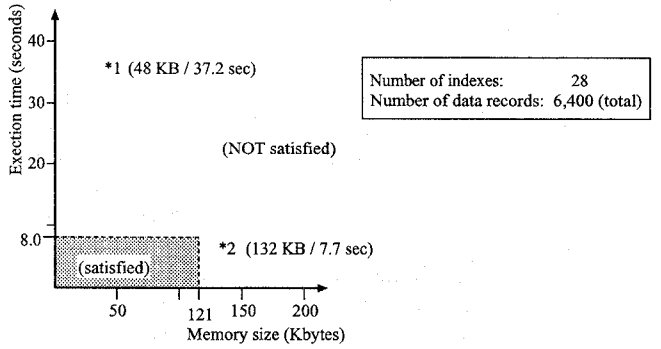


Figure 4: Performance evaluation results for naive implementation.

Performance requirements for the sales analysis and ordering system are listed in Table 1. To evaluate of these performance requirements, a loader program has been arbitrarily developed as an experiment. This arbitrary implementation of a loader program makes 28 indexes for a total of 6,400 data records, and each index has the same implementation coding style. Figure 4 shows the results of evaluation. The results of a memory-priority program which needs 48 Kbytes of memory and 37.2 seconds of execution time are indicated by *1 in Figure 4. This result do not satisfy the execution time requirements. Conversely, the results of a time-priority program are indicated by *2, which needs 132 Kbytes of memory and 7.7 seconds of execution time. These results do not satisfy the performance requirements, because the memory size exceeds the limit of 121 Kbytes.

According to an evaluation of the above results, the naive implementations cannot satisfy the performance requirements, and a more detailed design needs to be developed to satisfy the requirements. In the above two implementations, each index of the 28 indexes has the same implementation model (or coding style). However, each index should have its own optimal implementation model. Consequently, the designers need a *trade-off design* between memory size and execution time, and need to select an implementation model which is suitable for each index. Trade-off design can be described as an activity moving an evaluation result into a solution space on a problem space by trial and error.

In general, this example indicates that the naive implementation of a software synthesis system may satisfy function requirements, however, it is difficult to satisfy exacting performance requirements. Therefore, a technique for trade-off design in software synthesis is very important in satisfying performance requirements. This is the motivation for this paper.

3 Software Synthesis Mechanism for Trade-off Design

As stated above, there is a great need to satisfy the following conditions for trade-off design in software synthesis.

- Specifications modification must be modification of input specifications for a generator with trade-off parameters rather than modification of the generator itself.
- Performance requirements must be satisfied with modification of trade-off parameters, that is, a solution exists.

This section proposes a total development process for trade-off design with a software synthesis system so that the process satisfies the above two conditions, and it describes each substep of the process with the POT-DB example.

3.1 Total Process of Trade-off Design

3.1.1 Process Policy

The proposed process allows human designers to control trade-off parameters rather than designing them automatically without human designers. This approach indicates that trade-off design should be done by human designers. It is supported that (1) the

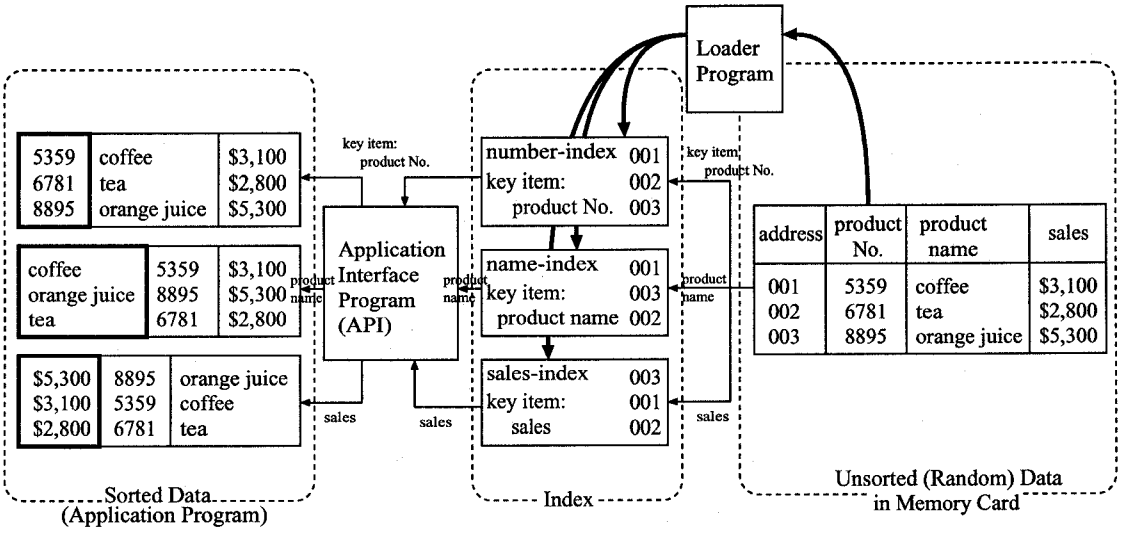


Figure 3: Logical file access mechanism using indexes.

threshold for performance should be decided by humans, and (2) evaluation values depend on individual execution situations. Based on these points, the support of human designers in trade-off design must be more effective than a fully-automatic approaches. The human-centered approach of the DODEs (domain-oriented design environment approaches) [2] [4] is the same approach as this paper's.

3.1.2 Trade-off Design Process

The following two approaches are presented to solve the problem:

- Transformation rules are designed so that there is no need for modification for trade-offs.
- Transformation rules are designed to be modified easily.

This paper employs the first, and proposes the following total process for trade-off design as also shown in Figure 5. The whole process can be divided into two parts: application engineering and method engineering.

Method engineering [5] [6], in general, is responsible for designing and constructing a development method that provides application engineers with systematic procedures, tools and guidance on how to deploy one or more notations (including trade-off parameters) for describing a problem or solution domain. The following five substeps construct a process in method engineering, which take performance requirements and provide the application engineers with notation for performance specifications, program generators and performance measurement tools.

Another part of the whole process in Figure 5 is *application engineering,* which is responsible for designing application systems using the assets provided by method engineering, i.e. the notation of performance specifications, the program generators and the performance measurement tools in the case of trade-off design. Application system design can be efficiently done by using these assets from method engineering.

The proposed five substeps of trade-off design in method engineering are as follows:

(1) Software model design — to determine an input specification model and an output software model for a program generator.

(2) Trade-off parameter design (Design decision analysis) — to determine key points as parameters for performance trade-offs using prototyping and evaluation program.

(3) Solution existence check — to evaluate performance of a trial program to satisfy performance requirements.

(4) Software synthesis system design — to design transformation rules which generate programs from specifications, so that implementation can be controlled by trade-off parameters.

(5) Performance measure design — to design performance (memory size and execution time) evaluation rules, which are added to transformation rules.

A closely related body of work has been done by Smith end Setliff [8]. They put more emphasis on automating the exploration of the design space, rather than lifting decisions to the level of the specifications, however. The following subsections explain the above-

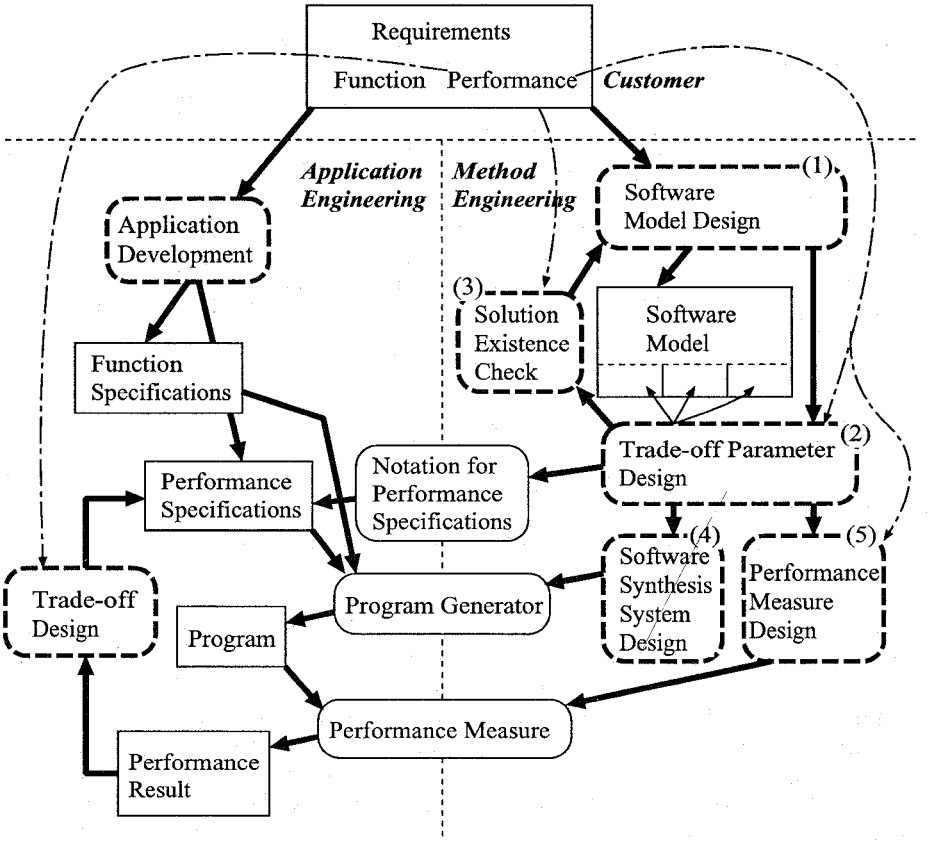


Figure 5: Total process for trade-off design.

mentioned substeps with the POT-DB example.

3.2 Software Model Design

The first substep is designing a software model which is an input specification model and an output software model for a program generator. This software model should be designed individually based on a specific domain or system. Consequently, a general method for designing a software model is difficult to define in advance. This subsection describes a software model employed in the POT-DB example. A main part of the trade-off design is the *Loader* subsystem of the POT-DB system. A software model of this subsystem is described in Figure 6.

The loader generator generates a loader program from two types of specifications, index specifications and file structure specifications. Index specifications include two items for each index, which are `index name`, and `key items`. Key items mean the items which define the order of file records for sorting. File structure specifications include two items which are `file name` and `record layout`.

The loader program consists of four parts, that is, the main part, the sorting algorithm, the reading part and the storage part. All of the four parts are in the main memory. The *Index* is in an EMS (extended memory system) for MS-DOS. The total memory size of indexes is estimated to be at least 100 to 200 Kbytes. This size is quite large comparing with 640 Kbytes of a main memory, which is the limits for MS-DOS. This is the reason why indexes use the EMS. The memory card has unsorted (random) data. The loader program reads the data from the memory card,

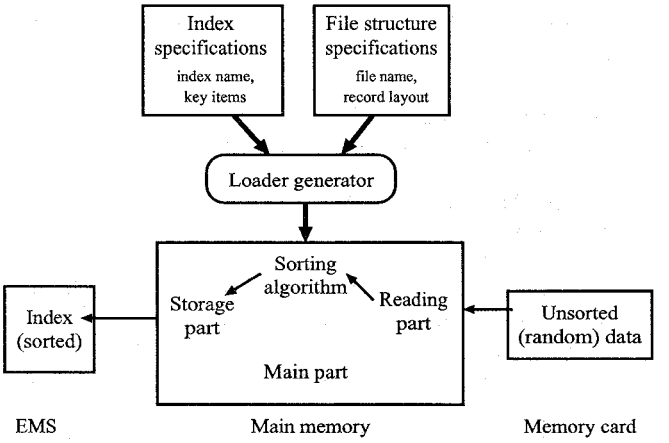


Figure 6: Software model for POT-DB loader subsystem.

sorts the data according to the key items, and writes the sorted index to the EMS.

3.3 Design Decision Analysis

The second substep of the trade-off design process is designing trade-off parameters. In a conventional transformation process, trade-off design is usually done at the transformation rule in designing phase, and the fixed decision is usually implemented as transformation rules. However, the proposed method extracts trade-off parameters from the software implementation model, and the values of these parameters can be defined as input specifications for the generators. Trade-off parameter candidates are evaluated by the following criteria:

- which are related to trade-offs — to be embedded in an input specification model,
- which are independent of trade-offs — to be embedded in transformation rules.

However, in general, there are no optimal concrete criteria. That is, trade-off parameters should be defined with their domain knowledge. In this subsection, for the POT-DB example, design decision analysis was conducted as follows.

Memory card read part — from the viewpoint of execution time, a memory card is quite slow in reading data. The access time for a memory card actually takes 50% of the whole execution time. Moreover, the number of accesses is very large. A key point is to reduce the number of memory card accesses. To achieve this, a read buffer has been set up in the main memory. As a result, **read buffer size is a trade-off parameter.**

Index storage part — Indexes are stored in an EMS rather than in the main memory. There is a buffering problem between the EMS and the main memory, because EMS access is slower than main memory access. Thus **EMS buffer size is a trade-off parameter. Buffer allocation timing is also a trade-off parameter** because each index needs its own buffer.

Sorting algorithm part — Each index record needs to be sorted based on its key items. Thus, the sorting algorithm is a candidate for a trade-off parameter. However, the insertion sorting algorithm has been chosen as a fixed sorting algorithm in the transformation rules, because a file record may be added sporadically. According to application data characteristics, there are some cases where file records in a memory card have already been sorted. Thus **a sorting on/off switch is a trade-off parameter.**

From the above analysis, the following four parameters are trade-off parameters in the POT-DB system.

1. memory card read buffer size,
2. EMS buffer size,
3. buffer allocation timing, and
4. sorting on/off switch.

The above trade-off relationships may seem to be quite simple. However, memory size limits and execution time limits are unclear beforehand. Consequently, the problem is not so simple even in this case. In designing of a software synthesis system, the nondeterminism of the evaluation function for trade-off relationships is a crucial problem of this paper. That is, *this paper gives a method to delay the design decision (or the determination of the evaluation function) for trade-off relationships using the method engineering approach.*

3.4 Solution Existence Problem

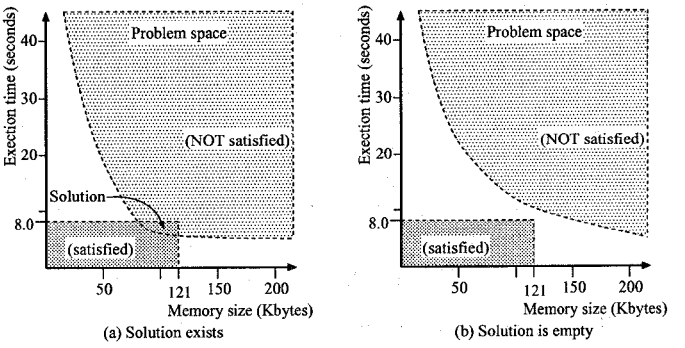
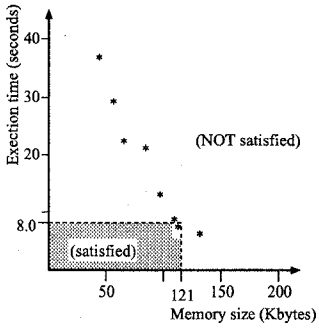


Figure 7: Solution existence problem in trade-off design.

The third substep of the trade-off design process is confirming the existence of a solution. Solution here means satisfaction of performance requirements. Extracting trade-off parameters from the software model corresponds to defining the performance range (i.e. the problem space) with the trade-off parameters. Therefore, it is necessary that the intersection between the problem space and solution space is nonempty (shown in Figure 7 (a)), because performance requirements cannot be satisfied if the intersection is an empty space (shown in Figure 7 (b)).

Although the solution existence problem is essential, solution existence confirmation is very difficult to obtain in advance, because it is a solution itself. That is, there is no need to confirm solution existence if the solution is already known. Consequently, this paper has approached this problem using trial program development and evaluation. First, a sample program was developed, which was comparable to the target program (sales analysis and ordering system). Next, the program was evaluated with various values for each trade-off parameter (the above-mentioned four parameters). The results of performance evaluation for the trial program are shown in Figure 8. The solution space is quite small, but it really exists. Consequently,



buffer size	memory allocation mode individual	lump
1	<48.0, 37.2>	< 96.0, 13.5>
10	<56.0, 29.4>	<104.0, 8.5>
20	<64.0, 22.5>	<120.0, 7.9>
30	<83.6, 21.9>	<131.6, 7.7>

<heap memory size (Kbytes), index creation time (seconds)>

Number of indexes: 28
Number of data records: 6,400 (total)

Figure 8: The results of performance evaluation for the trial program.

the sales analysis and ordering system which is to be developed using POT-DB is expected to satisfy the performance requirements[1].

3.5 Designing Transformation Rules for Trade-off Parameters

The fourth substep of the trade-off design process is designing generators. Generators consist of sets of transformation rules. Transformation rules process trade-off parameters so that a generated program constructs the problem space. This subsection proposes three types of rule description methods, shown in Figure 9, as follows:

3.5.1 Full Generation in Rules (Type 1)

This type of transformation rule makes design decisions with trade-off parameters, and generate the implementation algorithm. The rule implementation cost tends to be higher than for other types, because both design decision and algorithm generation are done in the rules. However, from these full generation characteristics, the design decision and implementation algorithm can be optimized for each trade-off parameter in the rules, even if a library or a platform are difficult to modify. Thus, this type is easy to use to embed performance (e.g., memory size and execution time) evaluation routines in the implementation algorithm.

In the POT-DB example, transformation rules for the trade-off parameter related to EMS buffer allocation timing were developed with `Type 1`. The range of this parameter is (1) lump allocation and (2) individual allocation. A rule for the lump allocation is shown as `Rule 1`, and a rule for the individual allocation is shown as `Rule 2`. In these rules, `f(lump)` and `f(individual)` correspond to design decision, and the left-hand-side of the two rules corresponds to two types of algorithms.

The lump allocation algorithm allocates a heap memory in the lump for all indexes. Therefore, although there is a need for large scale memory at the same time, the number of allocation times is one for each index. Thus execution time can be reduced with the lump allocation method. On the other hand, the individual allocation algorithm allocates at most one buffer at the same time. Consequently, maximum memory size is limited to a maximum buffer size among indexes. However, there is more execution time than with the lump allocation algorithm, because of many allocations.

```
Rule 1:
   f(lump) ->
         malloc(1), malloc(2),
         read(1.1), read(2.1), read(1.2), read(2.2), ...
         free(1), free(2)
Rule 2:
   f(individual) ->
         malloc(1), read(1.1), free(1),
         malloc(2), read(2.1), free(2),
         malloc(1), read(1.2), free(1),
         malloc(2), read(2.2), free(2), ...
```

3.5.2 Design Decision Generation in Rules (Type 2)

This type of transformation rule only makes design decisions. A library and a platform have some types of implementation algorithms. Consequently, the rule implementation cost tends to be inexpensive. Rule modification cost depends on variations in the algorithm. There are some interfaces between the design decision in the rules and a library or a platform. This type is useful in using the algorithms of a library and a platform from a generator. These types of rules are shown as `Rule 3` and `Rule 4`. In the POT-DB example, the sorting algorithm is implemented as a subroutine in a library. Thus, the sorting on/off swith trade-off parameter was implemented with `Type 2` as `Rule 3` and `Rule 4`.

```
Rule 3:
   f(sorting_switch_on) ->
         call_library(insertion_sorting_subroutine)
Rule 4:
   f(sorting_swith_off) ->
         call_library(empty)
```

3.5.3 Macro Data Generation in Rules (Type 3)

This type of transformation rule generates macro data rather than program procedures. An example of this type of rule is shown as `Rule 5` which processes the EMS buffer size trade-off parameter in the POT-DB example. This type tends to be very inexpensive for developing transformation rules, because almost all rules only exchange the syntax of trade-off parameters. This is useful in encoding fixed coding knowledge into a library or a platform rather than into a transformation rule.

```
Rule 5:
    f(bufsize) ->
          make_macro("#define EMS_BUFFERSIZE bufsize")
```

[1] However, the solution space nearly equals zero in this case. That is, there is no margin. In this situation, it is quite a gamble to start designing transformation rules. From the trade-off design view point, it is a very important large solution space.

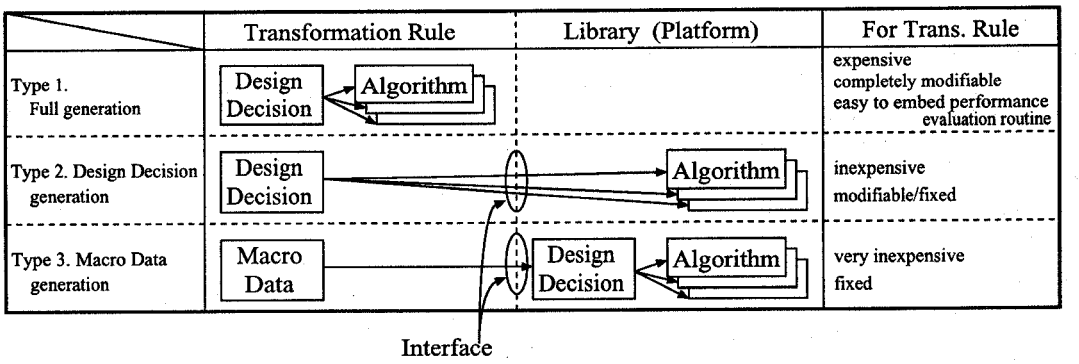


Figure 9: Types of designing transformation rules for trade-off parameters.

3.6 Performance Measurement Rule

Performance measurement and identifying effects of trade-off parameters on performance are very important for trade-off design, because the trade-off relationship must be controllable. That is, even if enough trade-off parameters can be explicitly defined in input specifications for a generator, when the effects of the trade-off parameters on performance are unclear, designers cannot specify the optimal values for the trade-off parameters to satisfy performance requirements.

The last substep of the trade-off design process is designing performance (memory size and execution time) evaluation rules, which are added into transformation rules. This subsection proposes the following two methods: the dynamic performance measurement method and the static performance measurement method.

3.6.1 Dynamic Performance Measurement

Dynamic performance here means performance with the execution of a program, for example execution time and the size of a dynamic allocation memory. The above-mentioned Type 1 is suitable for dynamic performance measurement, because the implementation algorithm is designed in the transformation rules. Detailed performance data can be measured with Type 1 rules. In the POT-DB example, the following Rule 6 is an extension of the above-mentioned Rule 2 to allow execution time for each cycle of *memory allocation – buffer read – memory free* to be measured.

```
Rule 6:
  f(individual) ->
        insert("#ifdef DEBUG"),
        insert("   time_measure_point(1);"),
        insert("#endif"),
              malloc(1), read(1.1), free(1),
        insert("#ifdef DEBUG"),
        insert("   time_measure_point(2);"),
        insert("#endif"),
              malloc(2), read(2.1), free(2)
        insert("#ifdef DEBUG"),
        insert("   time_measure_point(3);"),
        insert("#endif"),
```

...

3.6.2 Static Performance Measurement

Static performance here means performance without the execution of a program, for example, the static allocated memory size specified in a program and the name of the sorting algorithm. In the POT-DB example, the following Rule 7 is an extension of the above-mentioned Rule 5 to notify trade-off designers of buffer size.

```
Rule 7:
  f(bufsize) ->
        macro_with_document(
            make_macro("#define EMS_BUFFERSIZE bufsize"),
            document(">>>EMS buffer size is ",
                    multiply(bufsize, 512),
                    " bytes."))
```

4 Evaluations

4.1 Performance Requirements

This section evaluates the results of applying the proposed trade-off design method to the development of the sales analysis and ordering system with POT-DB. Table 2 lists an abstract process for the trade-off design of the sales analysis and ordering system with POT-DB. First, each index of the total 28 indexes was implemented for optimizing memory size. Then, by controlling the trade-off parameters, the memory size restriction is gradually relaxed so that the execution time requirements are to be satisfied.

At the beginning, the trade-off parameter of each index for *sorting on/off switch* is settled *off* if the index has already been sorted. The performance results in this case are indicated by *1 in Table 2 and also in Figure 10, which are not satisfy the execution time requirements. Next, the trade-off parameter for *timing to EMS buffer allocation timing* is optimized. The performance results in this case are indicated by *2, which are better but not satisfy the execution time. Then, the performance results by optimizing the *EMS buffer size* trade-off parameter are indicated by *3. It takes 10.2 seconds of execution time and 73 Kbytes of memory, however, it is not satisfied the time requirements. Finally, the *memory card read buffer size*

Table 2: Performance requirement evaluation results.

	α	β	γ	Time (sec)	Memory (Kbytes)
*1	each time	normal	1	50.0	34
*2	delayed	normal	1	20.5	50
*3	delayed	fast	1	10.2	73
*4	delayed	fast	20	7.3	80

α: the timing to release EMS buffer memory for index registration

β: buffer size for index registration(between 16 Kbytes and the size of the entire buffer)

γ: buffer size for memory card read(between 1 to 20 records)

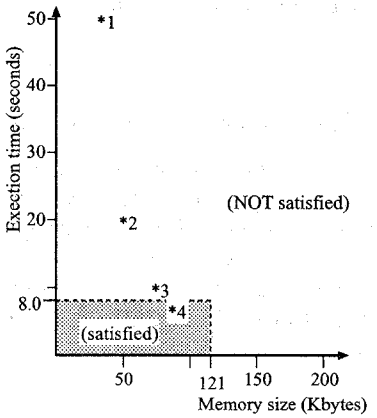


Figure 10: Performance evaluation results for the sales analysis and ordering system.

trade-off parameter is optimized. Then, the performance results indicated by *4 satisfy the performance requirements, which take 7.3 seconds and 80 Kbytes.

Table 3: An example of the memory size and execution time documents.

Index Name	Memory Size (KByte)	Execution Time (second)
number-index	31.2	1.4
name-index	87.4	2.8
sales-index	26.1	1.9
Total	144.7	6.1

During the above trade-off design, the designers took the memory size documents and the execution time documents (see Table 3) generated by POT-DB into consideration to choose the optimal values of the trade-off parameters. These documents greatly contributed to the quick convergence of the performance results, that is, the satisfaction of performance requirements. For example, in the case of Table 3, the *name-index* takes 87.4 KBytes of memory. From this table, the designers can know the key index of the memory size is the *name-index*, and the trade-off parameters of the *name-index* should be tuned.

As a result, all performance requirements have been satisfied with trade-off design, the developed sales analysis and ordering system has been in daily operation with over 10,000 portable terminals at more than one hundred branch stores.

4.2 Productivity

This section evaluates the productivity of developing a sales analysis and ordering system by using the proposed trade-off design method. It also evaluates overall productivity including the development cost for software synthesis systems (or POT-DB).

4.2.1 Productivity of Application Development

Table 4 compares the development costs for the data management sub-system of the sales analysis and ordering system using the proposed method with POT-DB and a conventional method.

Table 4: Development cost results with POT-DB

	Proposed method with POT-DB	Conventional method
Description size	25 logical data access spec. 28 index spec. 19 file structure spec. 1 Kline of C codes	9 Klines of C codes
Cost	4.9 man-months (measured)	9.4 man-months (estimated by 1 Kline = 160 man-hour)

Using POT-DB, it took 4.9 man-months to define 25 logical data access specifications, 28 index specifications, 19 file structure specifications and 1 Kline of C program codes. It is estimated that a program with the same function can be developed by hand-coding 9 Klines of C program codes, which is estimated to take 9.4 man-months based on the standard productivity of C programs, which is 1 Kline = 160 man-hours. Therefore, the productivity of application development with the proposed method is 1.9 times higher than that with the conventional method.

4.2.2 Productivity including the Generator Development Cost

When evaluating the productivity of application systems with special purpose program generators, it is important to consider the development cost of the program generators. If the productivity improvement is not substantially greater than the development cost for the program generator, it is merely shifting the application development cost to the generator development cost, and not improving total productivity. In other words, only when the following formula holds, can it be said that the total productivity has been improved [7].

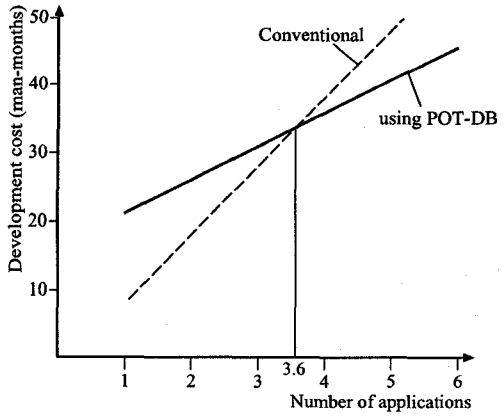


Figure 11: Productivity including the generator development costs.

Table 5: Generator development cost.

Sub-generator	man-month
Loader generator	7
API generator	7
Specification checker	1
Memory usage documenter	1
Total	16

(Application development cost reduction) − (Generator development cost) > 0

Table 5 shows the development cost for POT-DB.

As shown in the previous subsection, the application development cost reduction is (9.4 − 4.9) = 4.5 man-months. Therefore, the total cost reduction by applying POT-DB to a single sales analysis and ordering system is:

(Application development cost reduction) − (Generator development cost)
= 4.5 man-months − 16 man-months
= − 11.5 man-months

resulting not in total cost reduction, but in a 11.5 man-month cost increase.

However, the above evaluation assumes that only a single application system is developed with POT-DB. The reality is that POT-DB can be used to develop more than one application system, making it possible to reduce total costs. Figure 11 shows the total cost reduction/increase when numbers of application systems of equivalent size are developed with POT-DB. It shows that when at least four application systems are developed with POT-DB, total development cost including the POT-DB development cost can be reduced.

5 Conclusion

This paper has proposed a method of trade-off design in software synthesis, and this method has been applied to develop a file access program generator called POT-DB. Proposed method emphasizes lifting decisions to the level of the specifications, and delays the design decision (or the determination of the evaluation function) for trade-off relationships using the method engineering approach.

Based on the results of applying POT-DB in developing a sales analysis and ordering system, all performance requirements were satisfied, and application productivity was improved 1.9 times. Moreover, it was shown that total productivity including the development cost for POT-DB itself could be improved if POT-DB was applied to at least four application systems. The developed sales analysis and ordering system has been in daily operation with over 10,000 portable terminals at more than one hundred branch stores.

Acknowledgements

We wish to thank Toshihiko Takahashi, Chikara Isobe, Yukihiro Yasunobu, Yasushi Arima, Michiyuki Ushinmei, Mitsuo Seino, and Masahiro Yamamoto for their support during this research.

References

[1] Barstow, D.: Domain-Specific Automatic Programming, *IEEE Trans. Software Engineering,* Vol. SE-11, No. 11, 1985.

[2] Fischer, G.: Domain-Oriented Design Environments, *Automated Software Engineering,* Vol. 1, pp. 177-203, 1994.

[3] Keller, R., and Rimon, M.: A Knowledge-based Software Development Environment for Scientific Model-building, *The 7th Knowledge-Based Software Engineering Conference,* pp. 192-201, 1992.

[4] Lowry, M., Philpot, A., Pressburger, T., and Underwood, I.: A Formal Approach to Domain-Oriented Software Design Environments, *The 9th Knowledge-Based Software Engineering Conference,* pp. 48-57, 1994.

[5] Nuseibeh, B.: Meta-CASE Support for Method-Based Software Development, *Proc. of 1st International Congress on Meta-CASE,* UK, 1995.

[6] Nuseibeh, B., Finkelstein, A., and Kramer, J.: Method Engineering for Multi-Perspective Software Development, *Information and Software Technology Journal,* Butterworth-Heinemann, February, 1996.

[7] Sato, A., Tomobe, M., Yamanouchi, T., Watanabe, M., and Hijikata, M.: Domain-Oriented Software Process Re-engineering with Software Synthesis Shell SOFTEX/S, *The 10th Knowledge-Based Software Engineering Conference,* pp. 97-104, 1995.

[8] Smith, T., and Setliff, D.: Towards Supporting Design Phase Synthesis, *The 8th Knowledge-Based Software Engineering Conference,* pp. 20-27, 1993.

[9] Yamanouchi, T., Sato, A., Tomobe, M., Takeuchi, H., Takamura, J., and Watanabe, M. : Software Synthesis Shell SOFTEX/S, *The 7th Knowledge-Based Software Engineering Conference,* pp. 28-37, 1992.

[10] Yamanouchi, T., Sato, A., Watanabe, H., Yamanaka, H., and Watanabe, M. : Software Synthesis Shell SOFTEX/S and Its Application to Switching Software, *ATR International Workshop on Communications Software Engineering,* pp. 155-166, 1994.