TR-687

An Application of Artificial Intelligence to Prototyping Process in Performance Design for Real-Time Systems

by

S. Honiden, N. Uchihira & K. Itoh (Toshiba)

September, 1991

An Application of Artificial Intelligence to Prototyping Process in Performance Design for Real-Time Systems

Shinichi Honiden[1] Naoshi Uchihira[1] Kiyoshi Itoh[2]

1 Systems and Software Engineering Laboratory, Toshiba Corporation
70 Yanagi-cho, Saiwai-ku, Kawasaki 210, Japan
honiden@ssel.toshiba.co.jp

2 Faculty of Science and Technology, Sophia University
7-1 Kioi-cho, Chiyoda-ku, Tokyo 102, Japan
itohkiyo@hoffman.cc.sophia.ac.jp

ABSTRACT

This paper describes an application of artificial intelligence technology to the implementation of a rapid prototyping method in Object-Oriented Performance Design (OOPD) for real-time systems. A prototyping process is composed of three steps: Prototype construction, Prototype execution, and Prototype evaluation. The authors present the following artificial intelligence based methods and tools to be applied to each step. In the prototype construction step, a rapid construction mechanism, using reusable software components, is implemented based on the *planning* method. In the prototype execution step, a hybrid inference mechanism is used to execute the constructed prototype which is described in declarative knowledge representation. In the prototype evaluation step, an expert system, which is based on *qualitative reasoning*, is implemented to detect and diagnose bottlenecks and generate an improvement plan for them.

1. INTRODUCTION

Object-oriented technology can provide designers with practical, productive ways to develop software in most of application areas. As for real-time applications, object-oriented technology has been practically employed in a number of systems. Recently, various multiprocessors have become commercially available and have been used in many applications. Since stringent performance requirements are inevitable for real-time applications, it is very important to predict the target system performance precisely during the design phase, in order to determine the optimal software and hardware configurations which will satisfy users' requirements. However, in some real-time systems on multiprocessors, it is quite difficult to adjust some of the performance factors, such as load-balance on a given multiprocessor architecture, to satisfy the performance requirements. In practice, the performance design activities in the object-oriented design (called Object-Oriented Performance Design: OOPD) process tend to be empirical, because there are few algorithms which can derive the optimal software configurations that satisfy the performance requirements under a given hardware configuration. In other words, OOPD activities are usually carried out on a trial-and-error basis. These activities may compose a prototyping approach to improving ill-defined problems where only few algorithms are available. Artificial intelligence technology has been effectively used to tackle ill-defined problems. Various ill-defined problems are actually seen in the software engineering field including programming tasks. Several attempts have been made in applying the artificial intelligence techniques to these ill-defined problems in the software engineering field. Typical ones are GIST [Coh84, Swa83], Programmer's Apprentice [Ric78], $\phi 0$ [Bar82], Glitter [Fic85], Data-Representation Advisor [Kat81], SC [Dow90], and AFFIRM [Ger80]. These methods seem to be successfully applied to the particular phases or domains. Individual methods can be used to handle several steps in the software development process, but none of them can cover the overall development process nor can be used to fully implement the prototyping process.

A prototyping process is defined to be composed of three steps: prototype construction, prototype execution, and prototype evaluation [Ito89a]. The authors present the following artificial intelligence based methods and tools to be used in these three steps.
1) In the prototype construction step, a rapid construction mechanism, using reusable software components, is implemented, based on the *planning* method.
2) In the prototype execution step, a hybrid inference mechanism is used to execute the prototype which is described in declarative knowledge representation.
3) In the prototype evaluation step, an expert system, which is based on *qualitative reasoning*, is implemented to detect and diagnose bottlenecks and generate appropriate improvement plans.

In this paper, Section 2 describes the requirements of the prototyping process for real-time systems. Section 3 describes the presented method which satisfies the requirements. Section 4 describes OOPD which consists of three prototyping phases with application examples. Section 5 compares the presented method with related work and Section 6 evaluates it.

2. REQUIREMENTS OF PROTOTYPING PROCESS FOR REAL-TIME SYSTEMS

This section describes the requirements that should be satisfied for artificial intelligence to be applied or used to implement a prototyping process for real-time systems. In the previous section, the authors have mentioned that a prototyping process consists of the prototype construction, execution, and evaluation steps, as shown in Fig.1. This process may be repeated until the constructed prototype satisfies all the users' requirements. Note that, since rapidness is essential to prototyping, all the three steps need to be accomplished quickly and the number of iterations should be minimized. The characteristics and requirements of the three steps vary depending on the target applications. This section describes the fundamental requirements for implementing the three steps in the performance design for real-time systems.

Generally, a performance design task is composed of the following activities: performance model construction, performance measurement, performance diagnosis and generation of improvement plans. The prototype construction step, prototype execution step, and prototype evaluation step correspond to the performance model construction, the performance measurement, and the performance diagnosis and generation of improvement plans, respectively, as shown in Fig.1.

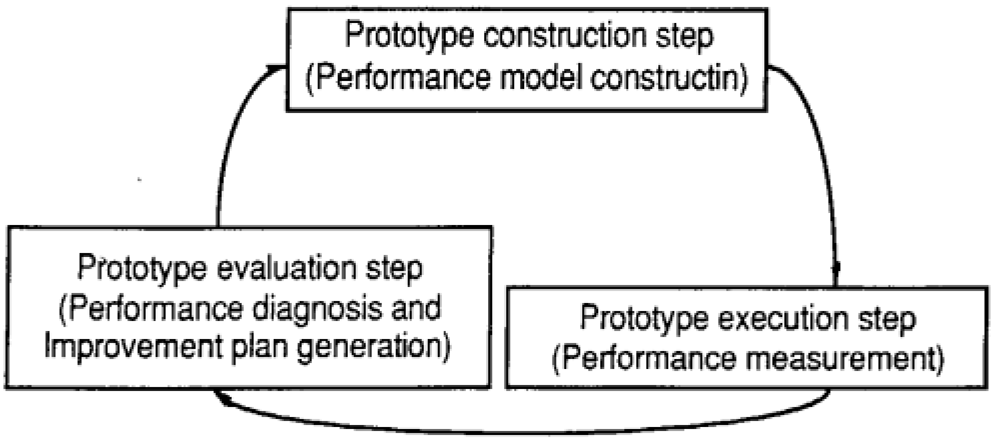

Prototype construction step
(Performance model constructin)
Prototype evaluation step
(Performance diagnosis and
Improvement plan generation)
Prototype execution step
(Performance measurement)

Fig.1 Prototyping process in performance design

2.1 Prototype construction step

Because most of the real-time systems contain many software modules, it is essential to any prototyping method to be able to treat large-scale programs, and increased software productivity during prototyping is required in order to construct prototypes rapidly. Software reuse methods have been considered as one of the most effective methods for increasing software productivity, and are employed in several domains [Hon86b, Jon84]. Various finding mechanisms, such as keyword, case grammar, and formula, have been proposed to retrieve reusable components. However, these methods can retrieve only one reusable component, using one specification statement at a time. The number of the specification statements is then proportional to the number of desired reusable components. To satisfy the rapidness requirements in the prototype construction step, the number of specifications necessary to retrieve the reusable components must be minimized.

2.2 Prototype execution step

One of the major properties essential to prototyping is executability, meaning that the following requirements should be satisfied.

(a) Rapid execution.

(b) Execution without complex preparation.

(c) Visual execution.

(d) Arbitrary interruption and re-starting during execution.

(e) Execution while displaying results which are easy to evaluate.

From the standpoint of performance design, since the execution of simulation usually takes a long time, it is necessary to produce visual performance data during the execution, and to collect and evaluate it effectively. For example, during the execution, it is important to display the queue lengths, as well as dead-lock detection in real-time. Performance statistics factors including the queue length at each server, the utilization rate and the wait time at each server as well as the response time are also required to be collected.

2.3 Prototype evaluation step

A prototyping method that does not support rapid prototype evaluation cannot be regarded as a rapid prototyping method, even if it supports rapid prototype construction and execution. This is because an inadequate evaluation may result in a useless repetition of the prototyping cycle and a time-consuming evaluation may slow down the whole prototyping process. These factors violate one of the main prototyping property of rapidity. To solve this problem in the prototype evaluation step, the following requirements should be satisfied.

Rapid detection of a bug which would produce undesired output and rapid generation of appropriate improvements.

From the viewpoint of performance design, the functions that achieve the above requirements are indispensable for rapid detection of bottlenecks and rapid generation of performance improvement plans which include several appropriate performance parameters. At present, because there are few standard methods which can offer support to satisfy these requirements, it is time-consuming and difficult for a non-expert designer, who has limited experience and is not familiar with performance design, to satisfy these requirements. It is necessary to automate this step, for non-experts to accomplish the design tasks accurately, rapidly, and appropriately. As mentioned previously, using a simulator generally takes a much longer time to accomplish the prototype execution step. Therefore, the prototyping cost can be lowered by reducing the iterations in the prototype execution step. Reducing the prototyping cost also depends on the rapid generation of appropriate improvement plans. In real-time systems on multiprocessors, it is important to validate the performance of the constructed prototype, which is mapped to given multiprocessors. The constructed prototype must be evaluated on the given multiprocessors in the prototype evaluation step.

3. PROTOTYPING PROCESS

This section describes the prototyping process support tools used to satisfy the requirements described in Section 2.

3.1 Prototype construction and execution steps

In real-time systems, the combination of declarative knowledge description and actor-based object modeling is considered to be one of the effective methods to develop a prototype. The inter-relationships among objects are described with actor-based object modeling and the inner behaviors of each object are described using declarative knowledge. The authors adopted MENDEL as the executable specification language which provides the above functions. MENDEL is a Prolog based concurrent object-oriented language [Hon86a, Uch87, Hon89, Hon90], which can be used as a functional and performance prototype construction tool and a prototype execution tool.

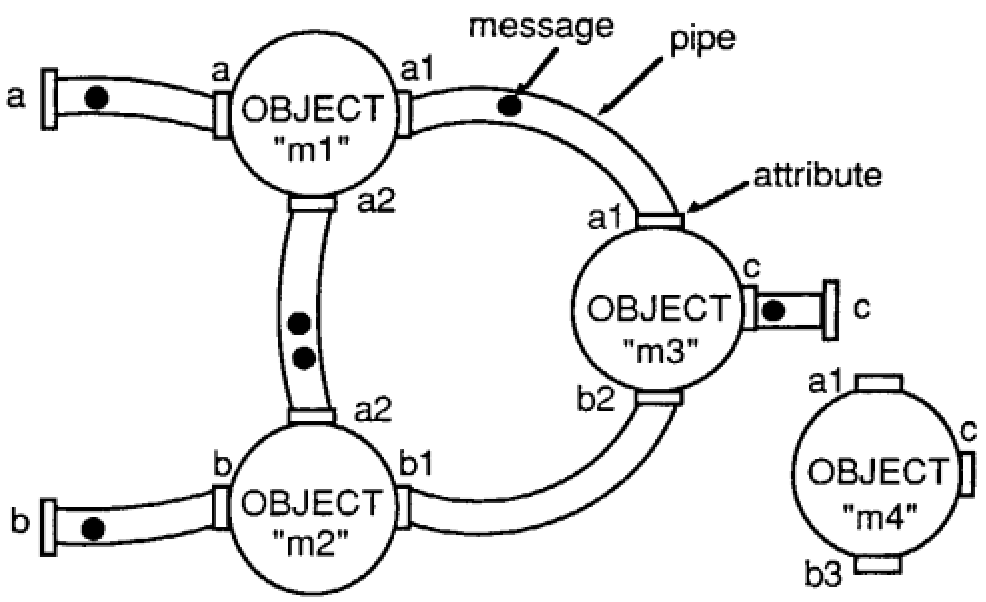


Fig. 2 Interconnection among MENDEL OBJECTs

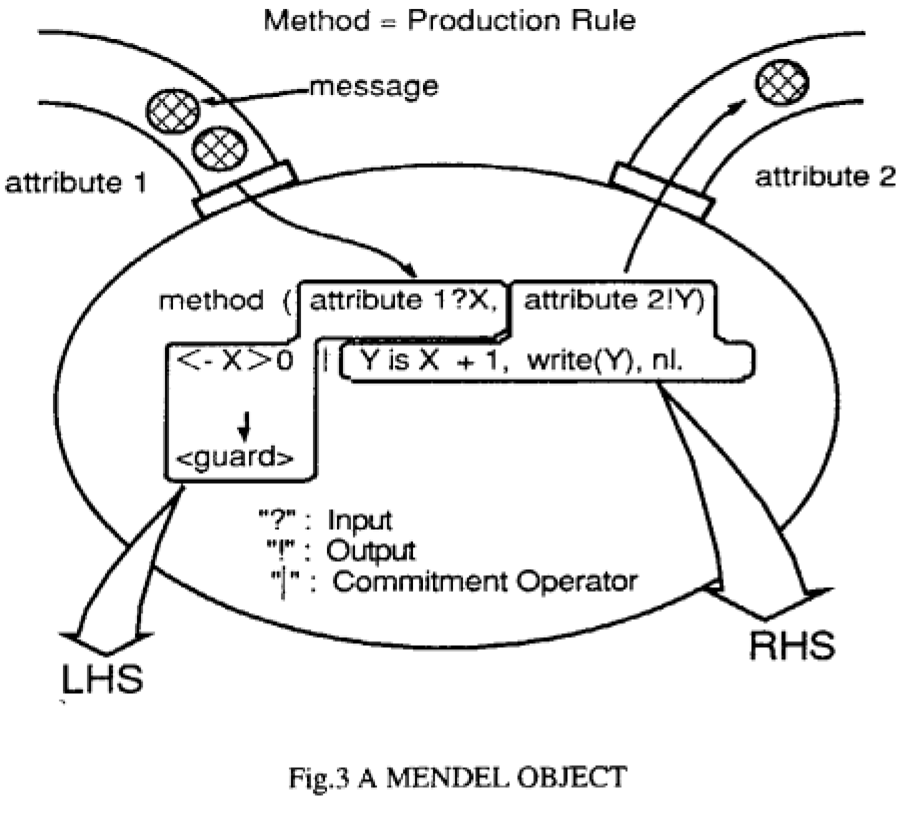

Method = Production Rule
message
attribute 1
attribute 2
method (attribute 1?X, attribute 2!Y)
<- X>0 | Y is X + 1, write(Y), nl.
<guard>
"?" : Input
"!" : Output
"|" : Commitment Operator
LHS
RHS

Fig.3 A MENDEL OBJECT

3.1.1 MENDEL object : A concurrent reusable component

Since an OBJECT in MENDEL is a concurrent processing unit, it can be regarded as a task or a process. Each OBJECT has pipe caps and can transmit messages only through the pipe caps, as shown in Fig.2. An attribute is assigned to each pipe cap and is used to identify input/output messages. Messages are transmitted between OBJECTs through the transmission pipe attached to the pipe caps. Each OBJECT consists of a block of working-memory and several METHODs, as shown in Fig.3. Each message consists of an attribute name, an input/output identifier- "?" or "!", and a variable name. If a message preceded by a "?" is received for a METHOD's variable, the METHOD's Prolog clauses are invoked. When the METHOD is executed, the variable preceded by an "!" will be unified and sent to the other OBJECTs as a message. Each METHOD is regarded as a production-rule and is used in the forward inference mechanism. A METHOD consists of a left-hand side (LHS) and a right-hand side (RHS). An LHS contains input messages and an RHS contains output messages. Both LHS and RHS contain internal state variables, which are stored in the working-memory. METHOD selection in a conflict set is non-deterministic. The body part of a METHOD consists of Prolog clauses. Since the Prolog system can be regarded as a backward inference engine, each METHOD includes a backward inference engine. The overall architecture is a distributed production system, in which each OBJECT has inherent working-memory and both forward and backward inference mechanisms. In MENDEL, a simple synchronization mechanism is achieved by using a METHOD selection mechanism, similar to Dijkstra's guarded command. The OBJECT is suspended, until it receives all required messages.

3.1.2 Planning

The authors extended MENDEL to contribute to rapid prototype construction, using reusable components, by introducing a *planning* method. One method to satisfy the requirements, mentioned in Section 2.1, is *planning* which can generate an action sequence or action program for an agent, such as a robot [Nil82]. Input to *planning* includes the initial world, a set of actions which change the world, and the final world. Output from *planning* is a sequence of actions which is represented by an acyclic-directed graph. As each action can be regarded as a reusable component and the world can be input and output specifications, the sequence of actions is a set of reusable components necessary to satisfy the input and output specifications. Each reusable component contains its own specification, called an *F-rule* [Nil82]. An *F-rule* consists of *preconditions*, *add formulas*, and *delete lists*. A *precondition* is corresponding to an input data item into the component, an *add formula* is corresponding to an output data item from the component, and a *delete list* includes the input data which has not appeared in the *add formula*. In an acyclic-directed graph, each node corresponds to a reusable component and each arc corresponds to the data flow between reusable components. Also, an acyclic-directed graph can be translated into a *task graph*, which has been used for resource allocation in multiprocessors [Gon77]. By using the *planning* method, the designer can retrieve and interconnect several reusable components at one time, only by giving input and output specifications.

MENDEL has employed the software reuse approach to increase software productivity. In MENDEL, the connection between pipe caps is accomplished automatically by the *planning* method. The *planning* method selects the required OBJECTs and connects the transmission pipes to create the message passing route from input specifications to output specifications. It carries out reusable component retrieval and interconnection by determining the reusable components which will satisfy the given input-output specifications. An automatic retrieval and interconnections are accomplished according to the following principles:
(a) A pair of pipe caps, having the same or similar attributes, can be interconnected using a semantic network, which consist of several attributes.
(b) All required output specifications must be reachable from given input specifications through connected objects and pipes.

The authors explain basic mechanism of the retrieval and interconnection using Fig.2. In Fig.2, input specification consists of "a" and "b" as external inputs, and output specification is "c" as an external output. At first, system retrieves OBJECT "m1", because OBJECT "m1" has attribute "a" which corresponds to the external input "a." Next, system retrieves OBJECT "m2", because OBJECT "m2" has attributes "a2" and "b" which correspond to "a2" of OBJECT "m1" and the external input "b." Finally, system retrieves OBJECT "m3," because OBJECT "m3" has attributes "a1", "b2", and "c" which correspond to "a1" of OBJECT "m1", "b1" of OBJECT "m2", and an external output "c", respectively. In this case, OBJECT "m4" can also be retrieved. It is assumed that "b2" of OBJECT "m3" is more similar to "b1" of OBJECT "m2" than "b3" of OBJECT "m4." The similarity is computed by the distance between nodes on the semantic network.

MENDEL has a hierarchical planning mechanism, similar to [And89]. In MENDEL, the strategy for assigning *criticality values* to the literals of an *F-rule*'s *precondition* is based on the information from the reusable component generation process of *Object-Oriented analysis* (OOA) [Coa90]. That is, in OOA, attributes in an object are classified into two groups: attributes which are newly declared in the object and those which are inherited from its upper objects. In this case, MENDEL assigns a higher value to the former attribute and a lower value to the latter attribute.

As a prototype execution tool, MENDEL provides visual execution, where an activated object can be recognized as a blinking one displayed on the screen and message queues and message contents are displayed in real-time. Interconnected OBJECTs in MENDEL form a queueing network, in which each OBJECT in MENDEL represents a server and each message in MENDEL represents a transaction. The statistical data, collected during execution, are passed to the prototype evaluation step to be analyzed.

3.2 Prototype evaluation step

In the prototype evaluation step, any bottlenecks should be detected rapidly and appropriate performance improvement plans for the bottlenecks should be generated. In particular, the reduction of prototyping cost depends on the ability to produce improvement plans. In other words, the prototyping evaluation step should accomplish appropriate parameter tuning to reduce prototyping cost.

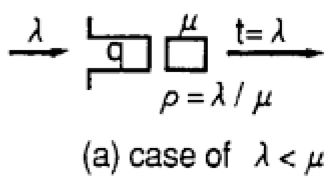
In MENDEL, the parameters to be tuned are as follows:
(1) The distribution of messages among OBJECTs.
Assume that identical OBJECTs are distributed among several processors for load-balancing, and that a particular OBJECT is busy and the others are not so busy. In this case, a message sent from an OBJECT to the busy OBJECT can be sent to an alternative OBJECT which is not so busy.
(2) A reusable component itself, which corresponds to an OBJECT on a particular processor.

Generally, there exist several reusable components in the library to satisfy the functional requirements. In this case, an alternative reusable component can be selected. Note that the reusable component having the shortest execution time, does not always satisfy the performance requirements of the given hardware configuration, and an important factor for performance is load-balancing. Viewpoint from the queueing network, as shown in Fig.4, the parameters to be tuned are, "μ"s for servers which indicate the OBJECT execution time and "r"s for entities on branching points which indicate the number of messages among OBJECTs. Note that "r"s having a functional meaning, such as the message attribute, should not be changed and that "r"s indicating a load-distributed factor can be changed.

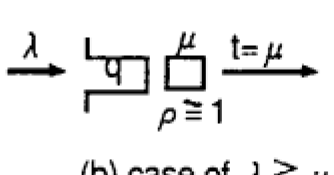
In performance design, there are a large number of parameter candidates to be tuned. For non-experts, in order to select appropriate parameters, the authors adopt a knowledge engineering technique based on *qualitative* and qualitative reasonings. These reasoning methods can be designed by modeling the performance design experts' reasoning process. On the basis of heuristics and knowledge obtained from evaluation experts, the authors have developed two knowledge-based expert systems, BDES (Bottleneck Diagnosis Expert System) and BIES (Bottleneck Improvement Expert System). BDES qualitatively diagnoses or identifies bottlenecks and their sources, and generates qualitative improvement plan. BIES quantitatively estimates the effects of the improvement for bottleneck and their sources on the whole queueing network [Ito89b, Ito90]. BDES and BIES are based on "*qualitative reasoning*" and "quantitative reasoning," respectively.

3.2.1 Bottleneck diagnosis expert system (BDES)

BDES diagnoses bottlenecks and their sources by reviewing for the queueing network and all its parameters. The bottleneck sources are the factors which govern bottlenecks, for example, low "μ"s or high "λ"s for servers, high "r"s on branching points, and their inter-relationships in the whole queueing network. The servers with the highest "ρ"s are bottlenecks, i.e., they are very busy. BDES judges that the servers whose "ρ"s >= 0.7 are or may be bottlenecks. 0.7 is called a bottleneck landmark (BL) in a *qualitative reasoning*. Moreover, BDES detects one or more alternative improvement plans for one bottleneck. On the basis of *qualitative reasoning*, the authors have designed qualitative behavior expressions for a single server, as shown in Fig.4.



(a) in the case of ρ < BL,i.e., ρ = -
(a1) for λ
d ρ =±<-- d λ = ±(λ and ρ change in the same direction.)
dt =±<-- d λ = ±(λ and t change in the same direction.)
(a2) for μ
d ρ =∓<-- d μ = ±(μ and ρ change in the reverse direction.)
dt =0 <-- d μ = ±(although μ changes, t does not change.)



(b) in the case of ρ > BL,i.e., ρ = + or ρ = 0
(b1) for λ
d ρ =−<-- d λ = −(as λ decreases, ρ decreases.)
dt =0 <-- d λ = −(although λ decreases, t does not change.)
(b2) for μ
d ρ =−<-- d μ = +(as μ increases, ρ decreases.)
dt =+<-- d μ = +(as μ increases, t increases.)
(b3) for q
dq =−<-- d ρ = −(as ρ decreases,q decreases.)
d ρ =−<-- dq =−(as q decreases, ρ decreases.)

λ : average arrival rate of entities for a server
μ : average servicing rate of a server for entities
t : average throughput of entities by a server
ρ : average utilization rate for a server
q : average queue length of entities in front of a server

Fig. 4 Unit server model

In order to increase or decrease some parameters of a particular server, all the equations for servers included in the queueing network must be considered. For example, it is assumed that the bottleneck server "s4" in Fig.7 is to be improved. In order to decrease "λ" of server "s4," the designer may decrease "r" from "s2" to "s4" and increase "r" from "s2" to "s5." Increasing "r" from "s2" to "s5" may cause "λ"s of the servers such as "s5", "s9" and "s10" to be increased. That is, new other bottlenecks may occur downstream from the change point. All equations for the servers included in the queueing network are necessary to improve a particular bottleneck. Since the number of states and state transitions is big, qualitative simulation takes much time. Then, BDES introduces the heuristics on queueing network substructures, such as loop, joint, branch, and tandem, into *qualitative reasoning* for effective qualitative simulation.

Several kinds of the queueing network have been utilized in various applications. Two major categories are open and closed types. In each category, typical examples are the synchronized type of queueing network and the multi-entity type of queueing network. Then, the authors have also developed BDES-S (BDES for open Synchronized queueing network) [Ito91ab] and BDES-MF (BDES-S for Multi-Flow network).

3.2.2 Bottleneck improvement expert system (BIES)

Based on the qualitative improvement plan produced by BDES, BIES quantitatively improves the bottleneck and bottleneck sources, i.e., it increases low "μ"values, decreases high "λ"values and decreases high "ρ"values. Moreover, BIES estimates the effects of its operations on the whole queueing network. The effects are estimated by computing new "λ" and "t" values for servers in the whole queueing network to be affected by the improvement. Based on the flow balancing, BIES forces "ρ" to decrease to a constant value, called BIF (Bottleneck Improvement Factor):

Only if a bottleneck server can be improved,

new "μ" = original "λ" / BIF.

Otherwise,

new "λ" = original "μ" / BIF.

The BIF value is varied from 0.7 to 0.6, according to the "q" of the server. When "q" is pretty high, its BIF is automatically set to 0.6, on the basis of the experts' heuristics. After applying this equation, the "ρ" and "t" values of the server can be improved so that "ρ" = BIF and "t" = "λ." The new "t" can be transmitted as the "λ" of the just downstream servers.

The authors have also developed BIES-S (BIES for open Synchronized queueing network) and BIES-MF (BIES-S for Multi-Flow network).

4. OBJECT-ORIENTED PERFORMANCE DESIGN

4.1 Overview

This section describes OOPD which consists of three prototyping phases for real-time systems. In designing real-time systems, it is important to consider both functional and performance aspects. This implies that two prototypes exist in designing real-time systems: functional and performance prototypes. The functional design should be accomplished while satisfying performance requirements. That is, during the function implementation, the performance requirements must be satisfied under several constraints. Examples of constraints are the number of processors and a task configuration. These constraints may affect the real performance and are determined in accordance with the functional design actually developed. Therefore, prototyping should be accomplished under the defined constraints at various design phases.

The prototyping process of OOPD is shown in Fig.5. Each of the three phases has prototyping processes as follows;

<Phase1>

Prototype construction and execution steps

OOPD starts by logical architecture construction. OOPD assumes that all necessary objects for object design have already been stored in the object library at the object-oriented analysis stage. In the prototype construction step, the logical architecture, which consists of several objects that satisfy the functional requirement, is designed and mapped into the physical architecture of the given multiprocessors. The logical architecture is constructed rapidly using the planning method. Each object in the logical architecture is specified by MENDEL. In <phase1>, since object design is not yet done, only the functional outlines of objects are clarified and the contents of objects are not refined. Since objects are black boxes here, the processing time for each object is estimated by its functional outline taken into account. At the same time, the number of messages arriving at each object is estimated. This estimation data is used to perform simulation. The simulation result is used as the input data in the prototype evaluation step.

Prototype evaluation step

The prototype evaluation step enables the re-estimation of the logical architecture from the viewpoint of message quantity and processing time for objects. In OOPD, the initial logical architecture which consists of appropriate objects is very important to minimize corrections in the subsequent design steps. The estimation of the objects' structure in terms of their performance here has much significance.

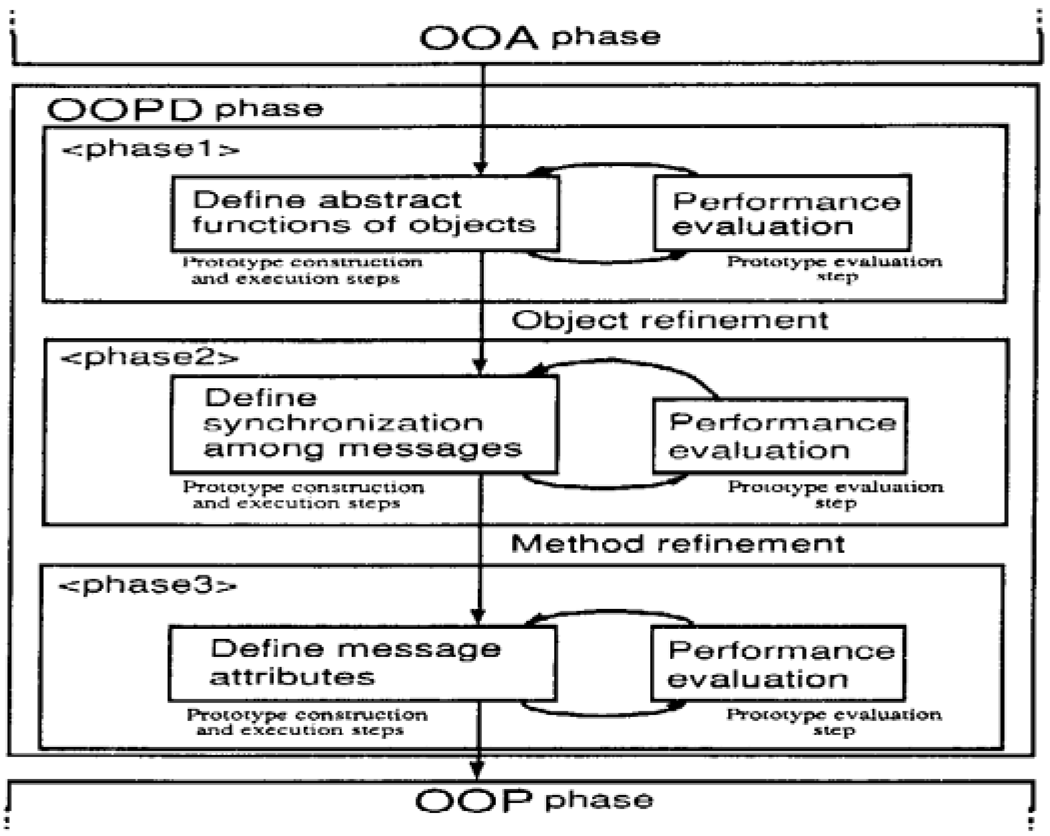

OOA phase
OOPD phase
<phase1>
Define abstract functions of objects
Performance evaluation
Prototype construction and execution steps
Prototype evaluation step
Object refinement
<phase2>
Define synchronization among messages
Performance evaluation
Prototype construction and execution steps
Prototype evaluation step
Method refinement
<phase3>
Define message attributes
Performance evaluation
Prototype construction and execution steps
Prototype evaluation step
OOP phase

Fig.5 OOPD overview

<Phase2>

Prototype construction and execution steps

In <phase2>, an object obtained in <phase1> is refined by defining input and output of the methods that compose the object. With regard to the processing of the methods, the problem of synchronization of several messages arriving at the method must be managed. If a method needs several messages, it must wait until all messages arrive. Generally, the synchronization that affects the performance of concurrent programs must be examined.

Prototype evaluation step

BDES-S and BIES-S can be used to examine the performance degradation caused by synchronization. Also, message distribution among objects and method assignment in objects are checked.

<Phase3>

Prototype construction and execution steps

Each method specified up to <phase2> must be formulated as an algorithm. The algorithm shows how the methods are executed. Up to <phase 2>, it is assumed that there is only one type of attribute for messages during the design process. Here in this phase, however, as the processing of each method is refined, the specific attributes of the messages required for a method are clarified. Depending on the attributes of the messages, the method execution process varies and, as a result, the processing time of the object varies. Therefore, message types must be taken into account for the performance evaluation in this phase.

Prototype evaluation step

BDES-MF and BIES-MF can be used to estimate a method implementation. The performance of the algorithm of each method is checked. If a bottleneck occurs in a certain object, the algorithm of the method is should be re-designed.

4.2 Example

The authors adopted the well-known "LIFT Problem" in [Iws87] as a typical real-time system example. In <phase1>, each of concurrent reusable components is implemented as an OBJECT in MENDEL, and the *planning* method carries out the automatic retrieval and establishes interconnections among reusable components.

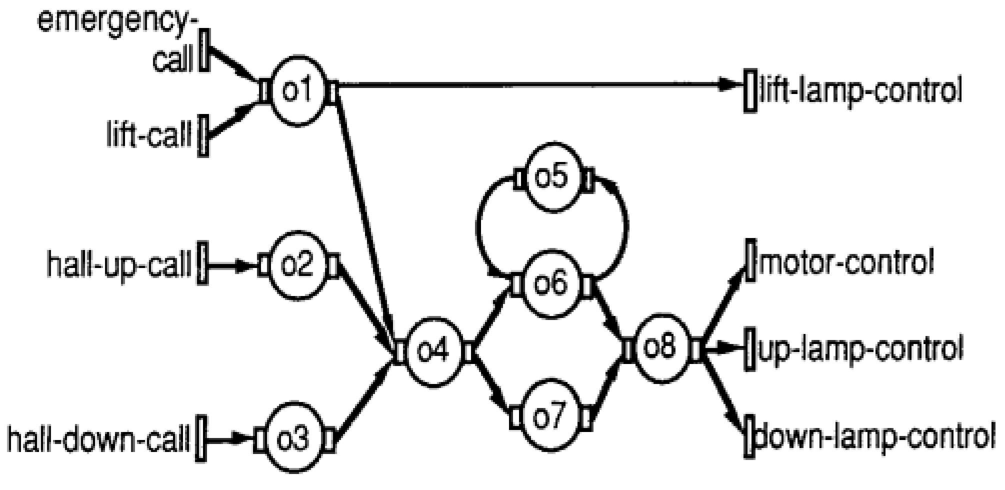


Fig. 6 Interconnected MENDEL OBJECTs which correspond to the functional model

For this example, by giving the input specification (hall-up-call, hall-down-call, emergency-call, lift-call) and the output specification (lift-lamp-control, up-lamp-control, down-lamp-control, motor-control), several OBJECTs are retrieved and interconnected, as shown in Fig.6.

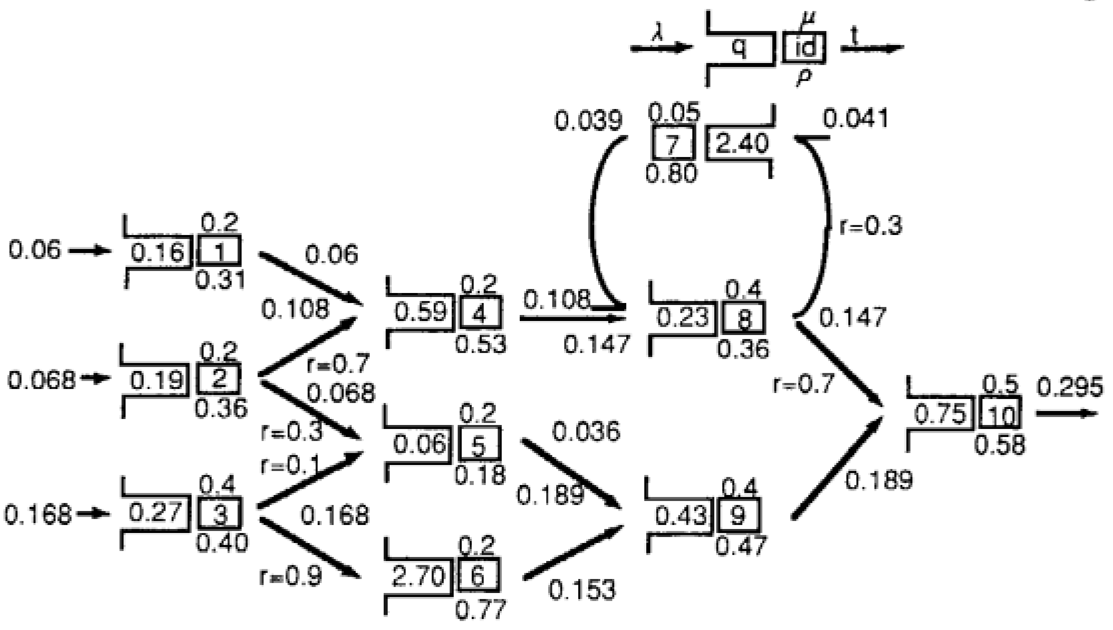


Fig. 7 Queueing network mapped from MENDEL OBJECTs in Fig.6

In Fig.6, OBJECTs "o1", "o2", "o3", "o4", "o5", "o6", "o7" and "o8" are retrieved and interconnected by the *planning* method. Interconnected objects in MENDEL form the queueing network, in which each object in MENDEL corresponds to a server and each message in MENDEL corresponds to a transaction. It is assumed that each OBJECT is assigned to each processor in a multiprocessor system which consists of 10 processors, as shown in Fig.7. OBJECTs "o1", "o2", and "o3" are assigned to "s1", "s2", and "s3", respectively. OBJECT "o4", including the main lift control, is distributed to three processors "s4", "s5", and "s6", because of load-balancing. And, OBJECTs "o5", "o6", "o7", and "o8" are assigned to "s7", "s8", "s9", and "s10", respectively. Lift-call, hall-up-call, and hall-down-call correspond to external inputs to "s1", "s2", and "s3" (called ga, gb, and gc), respectively. Figure 7 also shows the performance data from the simulation. For example, lift-call's arrival rate is 0.06 sec and "s7"'s utilization rate is 0.8 which shows a bottleneck. Emergency-call and lift-lamp-control are omitted from this queueing network, because they are out of statistical measurement. "r" from "s2" to "s4" and "r" from "s2" to "s5" indicate the load-distributed factor, because the message from "s2" may be sent to either "s4" or "s5." "r" from "s3" to "s5" and "r" from "s3" to "s6" indicate the load-distributed factors. In a queueing network in Fig.7, "μ"s for all servers, "λ"s for the entries into the network, "r"s, and the network structure are given, from the prototype construction step before the prototype execution. "ρ"s and "t"s for all servers and "λ"s for all servers can be obtained by the prototype execution.

```
server with maximum ρ: (s7 .800)

server whose ρ ≧ 0.9 none.

server whose ρ ≧ 0.7

   (s6 .770) (s7 .800)

Please input the name of server for diagnosis.
: s7

BDES shows the results of bottleneck
diagnosis.

Parameter to be improved for decreasing ρ of
s7.

*plan-1 increase μ (s7)
*plan-2 decrease  r (s2,s4)
*plan-3 decrease external input, gb
*plan-4 decrease external input, ga
```

Fig. 8 Improvement plans generated by BDES diagnosis

The designer can determine and locate bottlenecks by BDES. Figure 8 shows a list of servers and their "ρ"values by BDES. In Fig.8, for example, the designer can select bottleneck "s7." For improving the bottleneck on "s7", qualitative simulation can be applied. Block 2 in Fig.9 shows that only ρ7 and t4 are used in the loop consisting of "s7" and "s8." Figure 9 shows the bottleneck improvement process for "s7" with these heuristics, in which a dotted line box at the top represents the goal of *qualitative reasoning*, i.e., "decrease ρ7." The other 4 dotted line boxes represent the results from *qualitative reasoning*, i.e., the improvement plans for decreasing ρ7. BDES diagnoses the sources of the bottleneck "s7" and produces 4 improvement plans for bottleneck improvement, which are alternatives for the bottleneck "s7." For example, the designer can select Plan 2. BIES quantitatively improves the parameters, as shown bellow.

r(s2,s5)	0.300	---> 0.780
r(s2,s4)	0.700	---> 0.220

In order to improve the bottleneck of "s7", BIES can quantitatively modify "r" from "s2" to "s4." In this case, "r" from "s2" to "s4" can be modified, because this "r" indicates the load-distributed factor and the message from "s2" can be sent to either "s4" or "s5." In MENDEL, the message from OBJECT can be sent to the same OBJECTs on individual processors. The designer can accomplish the measurement using new parameters and obtain a new measured quantity. They can compare the second quantity with the first. Table 1 shows a comparison between two kinds of "ρ" values obtained by the first and second measurements, of the tuning process performed by BDES and BIES. Table 1 shows an appropriate improvement for bottleneck.

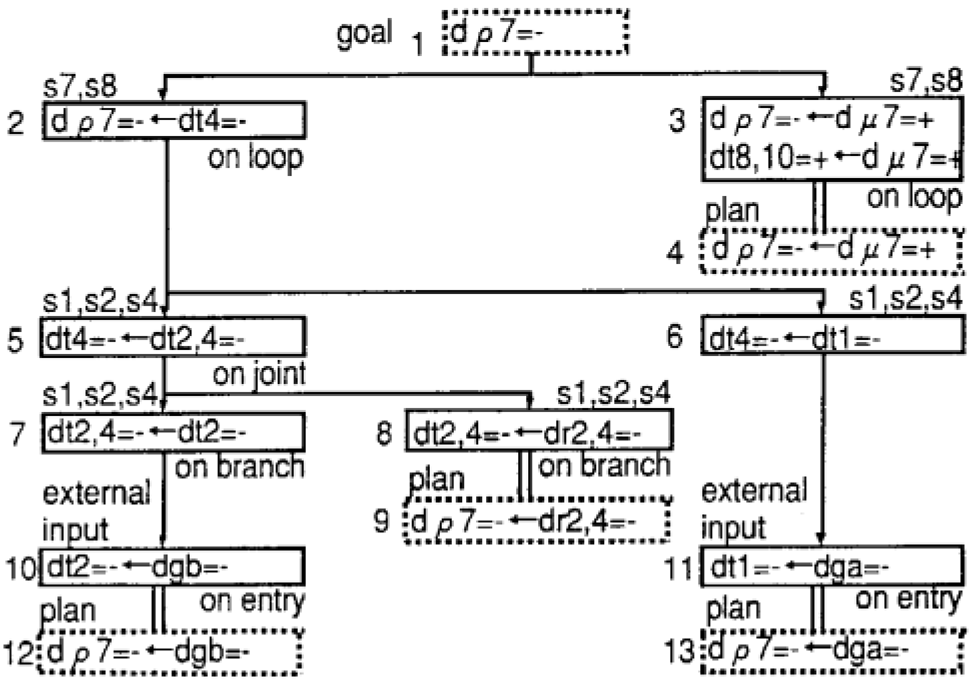


Fig. 9 Qualitative reasoning process for improving ρ's value on server "s7"

Table 1. Comparison of ρ's and queue lengt
between two measurements

	measurement1		improved by BIES	measurement2	
	ρ	q		ρ	q
s7	0.80	2.40	0.65	0.58	0.52

In <phase 2>, how the synchronization affects the performance of concurrent programs must be examined. By refining each object, the designer can find that OBJECT "o8" starts only when it receives messages from both OBJECTs "o6" and "o7." OBJECT "o8" is assigned to "s10" in Fig.7. Then, "s10" can start only when "s10" receives both messages from "s8" and "s9", and it may be bottleneck server. BDES-S and BIES-S are able to model such "s10" as a synchronized server and can generate improvement plan for "s10." In this case, "r" from "s2" to "s4" should be modified.

In <phase3>, by refining each method in each object, the designer can find that the types of messages arriving at OBJECT "o4" from OBJECT "o1" are different from those from OBJECT "o2", and the message processing time is different between them. BDES-MF and BIES-MF are designed to apply to the queuing network which processes multi-entities. These expert systems assist the designer to improve the performance of the prototype with the messages' types taken into account.

5. RELATED WORK

This section compares the method presented in this paper with the related work. The principal characteristics of the presented method include those of OOPD, the methodology employed in the prototyping process, MENDEL as an executable specification language and rapid prototype construction tool, and *qualitative reasoning* used for the prototype evaluation method.

First, while several tools based on object-oriented design have been presented, few of them support performance design as does OOPD.

Second, the presented prototyping process has two characteristics; the application domain is limited to performance design for real-time system on multiprocessor systems, and a special emphasis is put on the prototype evaluation step. Various prototyping methods for real-time systems have been proposed [e.g. Luq88]. However, these methods do not support the overall performance design for statistical features. No prototyping method, which emphasizes the prototype evaluation step, has been presented. The authors' method is considered to be general-purpose and applicable to several other domains, in which the prototype evaluation step is needed for a more complex system.

Third, as for MENDEL, two points, one of which is for executable specification language and the other is for *planning* methods, should be discussed. Various executable specification languages have been presented and experimentally used. They are classified into two groups: the Operational approach, such as GIST [Coh84] and PAISLey [Zav84], and the Functional approach, such as MODEL [Pry84] and RPS [Dav82]. MENDEL employs the operational approach. Concerning the combination of actor-model and declarative knowledge representation, one of the languages most similar to MENDEL is Orient 84/K [Ish87], which is an object-oriented concurrent programming language. The main difference between MENDEL and Orient 84/K is that Orient 84/K has several message-scheduling mechanisms and a parallel control mechanism as a programming language and does not support a hybrid inference engine. As for planning, MENDEL's planning ability is very simple compared with [And89].

Fourth, as for *qualitative reasoning*, the relation to queueing theory and the main difference from other work should be discussed. General analytical method based on queueing theory generates "ρ"s , "t"s and "λ"s for all servers, using given "λ"s from external, all "μ"s and "r"s. On the other hand, BDES&BIES generates some "μ"s and some "r"s in order to improve the bottleneck, using given "ρ"s , "t"s and "λ"s for all servers. Several applications of *qualitative reasoning* to a design process have been proposed [e.g. Wil90]. The difference from other work in applying qualitative reasoning to software engineering [e.g. Dow90] resides in the prototype evaluation step, where *qualitative reasoning* is indispensable to effectively implement a human's heuristics in performance design. Furthermore, no related work on the queueing network model has been presented in the *qualitative reasoning* domain. BDES&BIES present a combination method for *qualitative reasoning* and quantitative reasoning on a queueing network model. BDES selects several parameters to be tuned using *qualitative reasoning*, and BIES determines the parameter value using quantitative reasoning.

6. CONCLUSION

This paper describes the role of artificial intelligence in implementing a prototyping process in performance design for real-time systems. The presented method has several limitations and assumptions as follows;
(1) Dynamic object creation is not permitted.
(2) Communication cost is not considered.
(3) Closed type queueing network is not supported.
(4) When required messages are received, the object execution is started without waiting for completion of another object execution.
Among above points, (4) may be the most critical limitation from the viewpoint of applicability. In MENDEL execution, it is assumed that individual objects are assigned to individual processors in a multiprocessor system. In practice, several objects may be mapped to the same processor. Then, the authors are now implementing a hierarchical queueing network version in order to solve this problem.

Acknowledgments

The research in Section 3.1 has been partially supported by the Japanese Fifth Generation Computer Project and its organizing institute ICOT. The authors would like to thank Ryuzou Hasegawa of ICOT for his encouragement and support. The research in Section 3.2 has been partially supported by the Japanese Ministry of Education, Science and Culture. The authors are grateful to Seiichi Nishijima and Yutaka Ofude of Systems & Software Engineering Laboratory, Toshiba Corporation, for providing essential support. The authors also wish to thank Jun Sawamura and Keisuke Shida, Sophia University, for their helpful cooperation in developing BDES&BIES.

References

[And89] J.S.Anderson and S.Fickas : A Proposed Perspective Shift : Viewing Specification Design as a Planning Problem, *Proc. of 5th International Workshop on Software Specification and Design*, pp.177-184, 1989

[Bar82] D.Barstow et al. : An Automatic Programming System to Support an Experimental Science, *Proc. of 6th ICSE*, pp.360-366, 1982.

[Coa90] P.Coad and E.Yourdon : *Object-Oriented Analysis*, Prentice-Hall, 1990

[Coh84] D.Cohen : A Forward Inference Engine to Aid in Understanding Specifications, *Proc. of AAAI-84*, pp.56-60,1984.

[Dav82] A.M.Davis : Rapid Prototyping using Executable Requirements Specifications, *ACM SIGSOFT*, Vol.7,No.5, pp.39-44, 1982

[Dow90] K.Downing and S.Fickas : Specification Criticism via Policy-Directed Envisionment, CIS-TR-90-05, University of Oregon, 1990

[Fic85] S.Fickas : Automating the Transformational Development of Software, *IEEE Trans. Software Eng.*, Vol.11, No.11, pp.1268-1277, 1985

[Ger80] S.Gerhart et al. : An overview of AFFIRM: A Specification and Verification System, *Inform. Proc.*, Vol.80, pp-343-347, 1980.

[Gon77] M.J.Gonzalez : Deterministic Processor Scheduling, *Computing Surveys*, Vol.9, No.3, pp.173-204, 1977

[Hon86a] S.Honiden et al. : MENDEL: Prolog based Concurrent Object Oriented Language, *Proc. of Compcon '86*, pp.230-234, 1986.

[Hon86b] S.Honiden et al. : Software Prototyping with Reusable Components, *Journal of Information Processing*, Vol.9, No.3, pp.123-129, 1986, also in IEEE tutorial 'Software Reuse: The State of the Practice', 1988.

[Hon89] S.Honiden et al. : An Application of Structural Modeling and Automated Reasoning to Concurrent Program Design, *Proc. of HICSS-22*, 1989.

[Hon90] S.Honiden et al. : An application of Structural Modeling and Automated Reasoning to Real-Time Systems Design, *The Journal of Real-Time Systems*, Vol.1, No.3, Kluwer Academic Publishers,1990

[Ish87] Y.Ishikawa and M.Tokoro : Orient 84/K : An Object-Oriented Concurrent Programming Language for Knowledge System, *Object Oriented Concurrent Programming* (ed. by Yonezawa and Tokoro), MIT Press, 1987

[Ito89a] K.Itoh et al. : Tools for Prototyping for Developing Software, *JOHO SHORI*, Vol.30, No.4, pp.387-395, 1989

[Ito89b] K.Itoh et al. : Knowledge-based Parameter Tuning for Queueing Network Type System -A New Application of Qualitative Reasoning, *Proc. of IFIP CAPE'89.*

[Ito90] K.Itoh et al. : A Method for Diagnosis and Improvement on Bottleneck of Queueing Network by Qualitative and Quantitative Reasoning, *Trans. on JSAI*, Vol.5, No.1, 1990.

[Ito91a] K.Itoh et al. : Qualitative Reasoning Based Parameter Tuning on Bottleneck of Synchronized Queueing Network, *Proc. of Compsac '91*, 1991

[Ito91b] K.Itoh et al. : Parameter Tuning on Bottleneck of Synchronized Queueing Network by Qualitative Reasoning, *Trans. on JSAI*, Vol.6, No.6, 1991

[Iws87] *Proc. of 4th International Workshop on Software Specification and Design*, CS Press, Los Alamitos, Calif, 1987.

[Jon84] T.C.Jones : Reusability in Programming: A survey of the State of the Art, *IEEE Trans. Software Eng.*, Vol.SE-9, pp.488-494, 1984.

[Kat81] S.Katz et al. : An Advisory System for Developing Data Representations, *Proc. of 7th. IJCAI*, pp.1030-1036, 1981.

[Luq88] Luqi et al. : Rapidly Prototyping Real-Time Systems, *IEEE Software* September, pp.25-36, 1988.

[Nil82] N.J.Nilson, *Principles of Artificial Intelligence*, Springer-Verlag, 1982

[Pry84] N.S.Prywers : Automatic Program Generation in Distributed Cooperative Computation, *IEEE Trans. Syst. Man. Cyber.*, Vol.14, No.2, pp.275-286, 1984

[Ric78] C.Rich et al. : Initial Report on a LISP Programmer's Apprentice, *IEEE Trans. Software Eng.* ,Vol.4, No.6, 456-467, 1978.

[Swa83] W.Swartout : The Gist behavior explainer, *Proc. of AAAI-83*, 1983

[Uch87] N.Uchihira et al. : Concurrent Program Synthesis with Reusable Component using Temporal Logic, *Proc. of Compsac '87*, pp.455-464, 1987.

[Wil90] B.Williams : Interaction-based Invention : Designing Novel Devices from First Principles, *Proc. of AAAI-90*, 1990

[Zav84] P.Zave : The Operational versus the Conventional Approach to Software Development, *Comm. ACM*, Vol.27, No.2, pp.104-118, 1984