ソフトウェア工学 70－3
（1989．12．8）

# ＡＩ技術によるリアルタイム・システムのソフトウェア・プロトタイピング

本位田 真一[1] 内平 直志[1] 松本 一教[1] 伊藤 潔[2]

1 （株）東芝 システム・ソフトウェア技術研究所
2 上智大学 理工学部一般科学研究室情報科学部門

リアルタイム・システムを対象としたＡＩ技術を用いたソフトウェア・プロトタイピング手法について述べる。まず、プロトタイプの構築ステップ，実行ステップ，評価ステップから構成されるプロトタイピング・プロセス、およびリアルタイム・システムを対象とした場合の各々のステップにおける要求項目を明らかにし、各々の要求を満たす種々のＡＩ技術について論じる。特に、リアルタイム・システムの場合には、構築したプロトタイプが与えられた性能要求を満たすかどうかの性能評価ステップが必要であり、そのためにプロトタイプを待ち行列ネットワークに帰着させ、定性推論手法によりボトルネック診断を行う手法を提案する。

# An Application of Artificial Intelligence to the Process of Prototyping Real-time Systems

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

1 Systems and Software Engineering Laboratory, Toshiba Corporation
70 Yanagi-cho, Saiwai-ku, Kawasaki 210, Japan

2 Faculty of Science and Technology, Sophia University
7-1 Kioi-cho, Chiyoda-ku, Tokyo 102, Japan

This paper describes the role of artificial intelligence in the prototyping process for real-time systems. A prototyping process consists of three steps: prototype construction, prototype execution, and prototype evaluation. This process may be repeated until the users' requirements are all satisfied. In the prototyping process, all three steps should be accomplished rapidly or the number of iterations of this process should be reduced, so as to satisfy the rapidity property of the prototyping process. The authors present the following artificial intelligence based methods and tools to be applied in 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 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 automate the diagnosis and improvement of performance evaluation.

# 1. INTRODUCTION

Recently, various multiprocessors have become commercially available and have been used in many applications. The objective of multi-processing is to process the application software quickly to satisfy performance requirements. In several applications, severe performance requirements are needed. For example, a real-time process control system requires much more precision timing than a business application system. The response time is defined as the elapse time between receipt of signals from the external environment and generation of required output. If the response time exceeds the allowable limit, several malfunctions will certainly occur. If the real performance is discovered to be out of the allowable range after shipping from the factory, a substantial amount of time and labor will inevitably be spent in maintenance activities, such as reconfigurations and modifications. To avoid such a situation, it is very important to predict the target system performance precisely during the design phase, in order to find the optimal software and hardware configurations which satisfy users' requirements. However, in several real-time systems on the multiprocessor, it is quite difficult to adjust several factors such as load-balance on the given multiprocessor architecture to satisfy the performance requirements. In practice, the performance evaluation tends to be empirical, because there are few algorithms which derive the optimal software configurations to satisfy the performance requirements under a given hardware configuration. The simulation method still plays an important role in detailed performance evaluation. Simulations are usually carried out repeatedly by varying several performance factors, until simulation results satisfy the system requirements. In other words, the performance evaluation activity is carried out on a trial-and-error basis, and this activity may be a prototyping-like approach. However, this method is not equal to the prototyping approach, because it does not satisfy the rapidity which is the major property in the prototyping method. The following problems, existing in the performance evaluation activity, are the reason for this opinion.

(a) The process to construct the performance model for performance evaluation is not often included in a software development main-stream on which the function model of software is mainly constructed. The consistency check is also required between function model and performance model. The cost to construct the performance model may not be low.
(b) As there are few algorithms which can be used to evaluate the simulation results rapidly, analysis of the simulation results is a time-consuming activity. In the performance model, one of the typical analysis activities is the diagnosis for bottlenecks on the multiprocessors.
(c) It is difficult to generate the adequate improvement plans which consist of appropriate performance tuning parameters to eliminate the bottlenecks found in analysis activities. Inadequate plans may force useless simulation again.

This simulation method in performance evaluation is considered to be a prototyping method, on condition that all problems, as mentioned above, must be solved.

The authors present an application of artificial intelligence to a rapid prototyping method in the performance design to solve these problems.

Knowledge-based system, as an artificial intelligence method, has been used effectively to tackle the ill-defined problems where few algorithms exist. In the software engineering domain, various ill-defined problems also exist such as programming tasks. Several attempts made in applying the artificial intelligence method to these ill-defined problems in the software engineering domain have been presented. Typical examples are GIST [Coh84], Programmer's Apprentice [Ric78], $\phi$0 [Bar82], Glitter [Fic85], Data-Representation Advisor [Kat81] and AFFIRM [Ger80]. These methods seem successful in the particular phases or domains in which they are applied. Individual method can be applied to some steps in the software process, but does not cover the software process as a whole and does not fully implement the prototyping process.

Hence, artificial intelligence based standard methods, which actually implement the prototyping process, are rare. One of the major reasons for the lack of standard methods is that the following items have not been clearly defined:
(1) The software prototyping process, which consists of several steps.
(2) The requirements pertinent to each step in the prototyping process, in order to apply artificial intelligence.
(3) The implementation method based on artificial intelligence to satisfy the requirements in each step.
(4) The integration method which consists of various methods in order to cover the whole prototyping process.

The objective of this paper is to clarify the above items (1)-(4) in the performance design for real-time systems.

A prototyping process is defined to consist of three steps: prototype construction, prototype execution, and prototype evaluation [Ito89a]. This process may be repeated until the users' requirements are all satisfied. In the prototyping process, all three steps should be accomplished rapidly or the number of iterations of this process should be reduced, so as to satisfy the rapidity property of the prototyping process. Moreover, as the performance

feature characteristic is that a long time execution step exists due to the use of the simulator, the other steps must compensate for the execution step length. That is, the following items are firm requirements for the prototyping process in the performance design.
(i) Performance evaluation activity should be incorporated naturally in the software development process.
(ii) As real-time systems consist of many software modules, when constructing prototypes, increasing software productivity is required.
(iii) The prototype evaluation step, which has not been automated up till the present, should be implemented.
The authors present the following artificial intelligence based methods and tools in 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 automate the diagnosis and improvement of performance evaluation.

In this paper, Section 2 describes the requirements for the prototyping tools for real-time systems. The prototyping process is accomplished in three steps: prototype construction, prototype execution, and prototype evaluation. The requirements for implementing each step are also described. Section 3 describes prototype construction and execution steps, Section 4 describes prototype evaluation step, Section 5 evaluates the presented method and compares related work and Section 6 summarizes the unique features of our presentation.

# 2. PROTOTYPING PROCESS FOR REAL-TIME SYSTEMS

This section describes the requirements such that artificial intelligence can be applied or used to implement the prototyping process for real-time systems.

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

In the previous section, the authors have mentioned that the 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 the rapidity property for the prototyping process has to be satisfied, all three steps in the process need to be accomplished rapidly or the iterations of the process minimized. In practice, it is often impossible to accomplish all the steps rapidly. For example, much effort and time is often taken in a particular step in the prototyping process. In such a case, the other steps must compensate for steps which cannot be accomplished rapidly.

Hence, characteristics and requirements for the three steps depend on the prototyping applications. This section describes the requirements needed to implement the three steps in performance design for real-time systems.

## 2.1 Prototype construction step

Various methods are available to construct a prototype rapidly, such as the use of a executable specification language, a pictorial editor, and a reusable component retrieval tool. The requirements in this step depends on the method adopted. For example, with a specification language, easy modelling ability is included.

Most real-time systems consist of many software modules. As any prototyping method should be able to treat large-scale programs, increasing software productivity is required in order to construct prototypes rapidly. Software reuse methods have been considered among the most effective methods in increased software productivity and are used in several domains [Hon86b, Jon84]. Various specification methods, such as keyword, case grammar, and formula, have been presented to retrieve the reusable component. However, these methods can only retrieve one reusable component using one specification statement at a time. The size of the specification is then proportional to the number of desired reusable components. To achieve the rapidity property in the prototype construction step, the size of the specifications necessary to retrieve the reusable components must be as small as possible.

An easy modelling method is also necessary to describe each reusable component for the prototype construction step. In cases where declarative knowledge is used for modelling, it is more advantageous than non-declarative (procedural) knowledge, because the prototype is easily refined or modified.

The process to construct the performance model for performance evaluation is not often included in a software development main-stream, on which the function model is constructed. Naturally, the cost involved in constructing the performance model may not be low. Also, the consistency check is required between function model and performance model. Therefore, performance evaluation activity should be incorporated naturally in the software development process.

### 2.2 Prototype execution step

One of the major properties of prototyping is executability. This property means meeting the following requirements.
(a) To execute rapidly.
(b) To execute without complex preparation.
(c) To execute visually.
(d) Arbitrary interruption and re-starting during execution.
(e) To execute while displaying results which are easy to evaluate.

From the aspect of performance design, since simulation execution time is long, there is a need to display visual performance data during execution, and to collect the performance data to be evaluated effectively. For example, during execution, it is important to display the queue length dynamically as well as dead-lock detection. Performance statistics factors such as queue length at each server, utilization rate and wait time at each server, and response time are also required. Also, an on-line simulator should be implemented, in order to interrupt and re-start prototype execution arbitrarily because the simulator execution time is long.

### 2.3 Prototype evaluation step

In some prototyping methods, even if the prototype construction and execution steps are accomplished rapidly, any method which requires much time and effort in the evaluation step, will not be regarded as a rapid prototyping method. This is because inadequate evaluation may force useless repetition of the prototyping process and time-consuming evaluation is merely slow. These factors violate the major prototyping property of rapidity. To solve this problem in the prototype evaluation step, the following function is required.

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

This is particularly true from the viewpoint of performance design, because this requirement means rapid detection of bottlenecks and rapid generation of performance improvement plans, which consist of several appropriate performance factors. However, there are few standard methods which can satisfy this requirement and the requirement becomes time consuming and difficult for a non-expert who has no experience and is not familiar with performance evaluation. It is necessary to automate this step, in order to accomplish it accurately, rapidly, and appropriately for non-experts. As mentioned previously, much time is generally taken to accomplish the prototype execution step due to the use of a simulator. Therefore, the prototyping cost can be lowered by reducing the iterations for the prototype execution step. Reducing prototyping cost also depends on the rapid generation of appropriate improvements.

In real-time systems on a multiprocessor, it is important to validate the performance in the constructed prototype, which is mapped to the given multiprocessor. Therefore, the constructed prototype must be evaluated on the given multiprocessor in the prototype evaluation step.

## 3. PROTOTYPE CONSTRUCTION and EXECUTION STEPS

In real-time systems, the combination of declarative knowledge description and actor-based object modelling is considered to be one of the effective methods to describe the prototype. The inter-relationships among objects are described by actor-based object modelling and the inner behavior of each object is described by declarative knowledge. The authors adopt MENDEL as the executable specification language to satisfy this requirement. MENDEL is a Prolog based concurrent object-oriented. language [Hon86a,Uch87,Hon89]. MENDEL is used as a functional and performance prototype construction tool and a prototype execution tool.

### 3.1 MENDEL object = A concurrent reusable component

Since the OBJECT in MENDEL is a concurrent processing unit, the OBJECT can be regarded as a task or a process. Each OBJECT has finite pipe caps and can transmit messages only through the pipe caps. 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 connected to the pipe caps. Each OBJECT consists of one working-memory and several METHODs, which are declared as follows.

METHOD(attribute?variable...attribute!variable) <- <guard> | Prolog clauses.

Each message consists of an attribute name, an input/output identifier- "?" or "!", and a variable name. If a METHOD's variable after an attribute"?" has been received, that METHOD's Prolog clauses are executed. When the METHOD is executed, the variable after attribute"!" will be unified and sent to the other OBJECTs. Each METHOD is regarded as a production-rule and is used by the forward inference

mechanism. A METHOD consists of a left-hand side (LHS) and a right-hand side (RHS). LHS contains the input messages and RHS contains the 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 in a METHOD consists of Prolog clauses. As the Prolog system can be regarded as backward inference engine, each METHOD includes the backward inference engine. The overall architecture is a distributed production system in which each OBJECT has inherent working-memory and both a forward and a backward inference mechanism.

### 3.2 Planning

The authors extend MENDEL to contribute to the rapid prototype construction, using reusable components, by introducing a planning method. One method to satisfy the requirements mentioned in 2.2 is "planning" which achieves the generation of an action sequence or action program for an agent such as a robot [Nil82]. Input for planning includes the initial world, a set of actions which change the world, and the final world. Output from planning forms the sequence of actions which is 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 the set of reusable components necessary to satisfy the input and output specifications. Each reusable component has the specification itself, called the *F-rule* [Nil82]. *F-rule* consists of *precondition*, *add formula*, and *delete list*. *Precondition* corresponds input data into the component, *add formula* corresponds output data from the component, and *delete list* includes the input data not appeared in add formula. For the acyclic-directed graph, each node corresponds to a reusable component and each arc corresponds to the data flow between reusable components in the acyclic-directed graph. Also, the acyclic-directed graph can be translated into the *task graph* which has been used for resource allocation in multiprocessor [Gon77]. By using the "planning" method, the user can retrieve and interconnect several reusable components at one time by giving input and output specifications only.

MENDEL applies software reusability to increase software productivity. In MENDEL, the binding between pipe caps is accomplished automatically by "planning" method. "planning" method selects the necessary OBJECTs and binds the transmission pipes to create the message passing route from input specifications to output specifications. It performs reusable component retrieval and interconnection by determining the reusable components which will satisfy the given input-output specifications. The automatic retrieval and interconnection are performed according to the following principles:

(a) A pair of pipe caps having the same or similar attribute meaning can be interconnected using semantic networks which consists of several attributes.
(b) All required output specifications must be reachable from given input specifications through connected objects and pipes.

MENDEL has a hierarchical planning mechanism similar to ABSTRIPS [Sac74]. In MENDEL, the assignment strategy for *criticality values* to the literals of a *F-rule*'s *precondition* is based on the design information from the reusable component generation process by *structured analysis* (SA) [DeM84]. That is, in SA, each bubble corresponds to an individual reusable component. Input to the bubble is classified into two groups: input which had appeared in an already upper structure and input which has newly appeared in a current structure. In this case, MENDEL assigns a higher value to the former input and a lower value to the latter input.

As a prototype execution tool, MENDEL provides visual execution, where an activated object can be recognized as a blinking object displayed on the screen and message queue and message contents are displayed dynamically. In MENDEL execution, it is assumed that each object is assigned to each processor in a multiprocessor system. The statistical data, collected during execution, are passed to BDES&BIES to be analyzed.

### 3.3 Example

The authors adopt well-known the "LIFT Problem" in [Iws87] as one of the typical real-time system examples. In MENDEL, a concurrent reusable component corresponds to an OBJECT, and the planning method carries out the automatic retrieval and interconnection among reusable components. For this example, by giving the input specification (hall-up-call, hall-down-call, emergency-call, lift-call) and output specification (lift-lamp-control, up-lamp-control, down-lamp-control, motor-control), several OBJECTs are retrieved and interconnected, as shown in Fig.2. In Fig.2, 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 networks, 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. This LIFT problem runs on the multiprocessor system, where OBJECT "o4" including main lift control is distributed to three processors "s4", "s5", and "s6", because of load-balance, as shown in Fig.3. Emergency-call and lift-lamp-control are omitted in 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 factor. In a queueing network, called "NQ10" in Fig.3., "μ"s for all servers, "λ"s for the entries into NQ10, "r"s, and the NQ10 structure are given from 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.

# 4. PROTOTYPE EVALUATION STEP

In the prototype evaluation step, any bottlenecks should be detected rapidly and pertinent performance improvement plans to eliminate the bottlenecks should be generated appropriately. In particular, reducing prototyping cost depends on the appropriate improvement plans. In other words, the prototyping evaluation step accomplishes appropriate parameter tuning to reduce prototyping cost. In MENDEL, the parameters to be tuned are as follows:
(1) The amount of messages among OBJECTs.
Assume that same OBJECTs are distributed on several processors, because of load-reducing, and that the particular OBJECT is busy and others are not so busy. In this case, the message from the OBJECT, which was sent to the busy OBJECT, can be sent to an alternative OBJECT, which is not so busy.

(2) The reusable component itself which corresponds OBJECT on the particular processor.
Generally, there exist several reusable components in the library to satisfy the functional requirements. In this case, alternative reusable component can be selected. Note that the reusable component having fastest execution time does not always satisfy the performance requirements on the given hardware configuration, and important factor for performance should be load-balance.

As the authors have mentioned in 3.3, interconnected OBJECTs in MENDEL form the queueing networks, in which each OBJECT in MENDEL corresponds to a server and each message in MENDEL corresponds to a transaction.

The process for improving bottlenecks elimination consists of 2 phases with measurement quantity for the queueing network. The 1st phase can qualitatively diagnose or identify bottlenecks and their sources. The 2nd phase can quantitatively estimate the effects of the improvement in bottleneck elimination and their sources on the whole queueing network. Personnel can augment the parameter tuning process ability by applying knowledge engineering. It can reduce the number of total repetition in the prototyping 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), which can augment the ability of the 1st and 2nd phases, respectively [Ito89b, Ito90]. BDES and BIES are based on "*qualitative reasoning*" and "quantitative reasoning", respectively.

## 4.1 Queueing network

A queueing network is often adopted in order to evaluate the performance for objective systems. In a queueing network, "entity", e.g., transactions, messages or materials, are handled by a lot of "servers", e.g., processors, channels or production machines, which are interconnected in the objective systems. The queueing network parameters are as follow:
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 of a server,
q: average queue length of entities in front of a server,
r: branching probability of entities at a branching point.

The system performance, e.g., "ρ"s and "t"s for the whole system and individual servers, can be evaluated by queueing network-based methods. Bottlenecks, which involve a risk in regard to reducing the system performance, occur at one or more servers with higher "ρ"s in queueing network. The bottlenecks may give rise to long queues of many entities waiting for service by the bottleneck servers. Bottlenecks occur at one or more servers with high utilization rates and long demand queues in the queueing network. The existence of bottlenecks reduces the target system performance. The diagnosis of bottlenecks and their sources becomes more difficult in proportion to the magnitude of the queueing network.

The authors' goal is to apply qualitative and quantitative reasoning to the queueing network, in order to improve its bottlenecks elimination. This process is accomplished for one or more parameters, which are bottleneck sources. 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 amount of messages among OBJECTs. Note that "r"s having a functional meaning, such as a message attribute, should not be changed and that "r"s indicating a load-distributed factor can be changed.

## 4.2 Bottleneck diagnosis expert system (BDES)

BDES analyses the message conflicts for the specific object, detects the bottlenecks and generates the improvement plans. BDES can diagnose the bottlenecks and their sources by automatic review for the queueing network and all its parameters. The bottleneck casues are not only local to bottleneck servers but are global with respect to the whole queueing network. The bottleneck sources are factors which govern bottleneck occurring, e.g., low "μ"s for servers, 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., the servers are very busy. BDES can judge that the servers whose "ρ"s >= 0.7 are or may be bottlenecks. 0.7 is called a bottleneck landmark (BL). Moreover, BDES can detect one or more alternative improvement plan for one bottleneck elimination.

On the basis of *qualitative reasoning*, the authors have designed the qualitative behavior expressions (QLBE) for a single server in the following:

(a) in the case of ρ < BL, i.e., ρ = -,
(a1) for λ,
dρ = ± <-- dλ = ± ( λ and ρ change in the same direction.)
dt = ± <-- dλ = ± ( λ and t change in the same direction.)
(a2) for μ,
dρ = ∓ <-- dμ = ± ( μ and ρ change in the reverse direction.)
dt = 0 <-- dμ = ± ( although μ changes, ρ does not change.)
(b) in the case of ρ>=BL, i.e., ρ = + or ρ = 0,
(b1) for λ,
dρ = - <-- dλ = - ( as λ decreases, ρ decreases.)
dt = 0 <-- dλ = - ( although λ decreases, t does not change.)
(b2) for μ,
dρ = - <-- dμ = + ( as μ increases, ρ decreases.)
dt = + <-- dμ = + ( as μ increases, t increases.)
(b3) for q,
dq = - <-- dρ = - ( as ρ decreases, q decreases.)
dρ = - <-- dq = - ( as q decreases, ρ decreases.)
QLBEs in (b) are in particular called qualitative bottleneck elimination improvement expression (QL-BIE).

For improving bottleneck elimination, s6 and s7, qualitative simulation can be applied. In the qualitative simulation, possible states and state transitions are exhaustively enumerated. As the number of the states and state transitions are high, it takes much time in qualitative simulation. BDES introduces the heuristics on queueing network substructures, such as loop, joint, branch, and tandem, into *qualitative reasoning* for effective qualitative simulation. Block 2 of Fig.4 shows that only ρ7 and t4 are used in the loop consisting of s7 and s8. Figure 4 shows the bottleneck elimination improvement process for s7 with these heuristics, in which a dashed line box at the top represents the goal of qualitative reasoning, i.e., "decrease ρ7". The other 4 dashed line boxes represent the results by *qualitative reasoning*, i.e., the improvement plans for decreasing ρ7.

### 4.3 Bottleneck improvement expert system (BIES)

According to one qualitative improvement plan by BDES, BIES can quantitatively improve the bottleneck elimination and bottleneck sources, i.e., it can increase low "μ"s, decrease high "λ"s and decrease high "r"s. Moreover, BIES can estimate the effects on the whole queueing network. Estimating the effects is carried out by the automatic computation of new "λ"s for servers and new "t"s for servers in the whole queueing network to be effected by the improvement.

This automatic computation is based on heuristics about so-called "flow balance". Flow balance means that, if the bottleneck elimination can be improved, the input quantity is equal to the output quantity at any part of the queueing network. For example, "λ" for the server is equal to its "t", and "λ" for the loop is equal to its "t" if there is no bottleneck server. On the basis of the flow balance, BIES forces "ρ" to decrease to a constant value, called BIF ( Bottleneck Improvement Factor), with the use of the following QT-BIEs (QualiTative Bottleneck Improvement Equation ) for the improvement:
Only if the 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" for 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 for the server can be improved so that "ρ" = BIF and "t" = "λ". The new "t" can be transmitted as the "λ" for the just downstream servers. The improvement process with QT-BIE can be repeated in the more downstream direction.

### 4.4 Example of diagnosis and improvement

Personnel can determine and locate bottlenecks by BDES. Figure 5 shows the list of servers and their "ρ"s by BDES. In Fig.5, for example, personnel can select bottleneck s7. BDES can diagnose the sources for bottleneck s7 and produce 4 improvement plans for bottleneck elimination improvement, which are alternatives for bottleneck s7. For example, personnel can select Plan 2. BIES can quantitatively improve the parameters in Fig.6. In order to improve bottleneck elimination 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.

Personnel can accomplish measurements for NQ10 using new parameters and obtain the new measure quantity. They can compare the second measure quantity with the first. Table 1 shows a comparison between 2 kinds of "ρ" values obtained by the first measurement, by the tuning process by BDES and BIES, and by the second measurement. Table 1 shows appropriate improvement for bottleneck elimination.

## 5. DISCUSSION

This section evaluates the presented method and compares the related work. The principal characteristics of the presented method consist of the methodology employed in the prototyping process, MENDEL as an executable specification language and rapid prototype construction tool, and qualitative reasoning used for prototype evaluation method.

First of all, the presented prototyping process includes two characteristics: application domain such as performance design for real-time system on multi-processor systems, and emphasis on prototype evaluation step. Various prototyping methods for real-time systems have been presented [Luq88]. However, these methods do not include the overall performance design from statistical features, and treats only a part of it. No prototyping method, which emphasizes the prototype evaluation step, has been presented. This methodology is considered to be general-purpose and can be applicable to several other domains. In particular, the prototype evaluation step is needed for a more complex system.

Second, for MENDEL, following two discussions are needed, an executable specification language, and a planning method. Various executable specification languages have been presented and used. They are classified into two groups: Operational approach such as GIST [Coh84] and PAISLey [Zav84] and Functional approach such as MODEL [Pry84] and RPS [Dav82]. MENDEL belongs to operational approach. The disadvantage in MENDEL is the weakness of verification because only synchronization part is verified using temporal logic specification [Hon89]. As for the combination of actor-model and declarative knowledge representation, one of the most similar language 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 parallel control mechanism as a programming language and does not support hybrid inference engine. For planning, MENDEL's planning ability is the same as ABSTRIPS's [Sac74], and the limitation for MENDEL includes that for ABSTRIPS.

Third, for qualitative reasoning, the authors must mention the following items: the relation to queueing theory and the main difference from the other works. The BDES&BIES application range is beyond the range of analytical methods, which are based on "queueing theory", because it is not easy for analytical methods to produce distinct improvement plans for bottleneck elimination in particular, in a transient state. The difference from other works is the application to the prototype evaluation step where qualitative reasoning is indispensable to effectively implement human's heuristics in performance design. Also, no related work on the queueing network model exists in the qualitative resoning domain. BDES&BIES present a combination method of qualitative resoning and quantitative reasoning on queueing network model. That is, in performance tuning, there are so many parameters to be tuned. BDES selects several parameters to be tuned using qualitative reasoning and BIES determines the parameter value using quantitative reaosning.

The current combination of MENDEL and BDES&BIES is considered to be an appropriate prototyping method, on condition that the number of OBJECTS is nearly equal to the number of processors in multiprocessor or that non-conflict for execution on the same processor is assured, and that no-synchronization among messages occurs. The authors plan to overcome this limitation by extending BDES&BIES to introduce a hybrid queueing network, in which the conflict on the same processor is treated and synchronized-queueing-network similar to TPQN [Cha89] in which synchronization among messages is treated.

## 6. SUMMARY

This paper describes the role of artificial intelligence in implementing a prototyping process for real-time systems. The authors present the methods for each of the following three steps in the prototyping process.

(i) In the prototype construction step, rapid construction with reusable software

components is implemented based on a "planning" method. Each reusable component is described in a declarative knowledge representation.
(ii) In the prototype execution step, a visual execution is used to support the display of the prototype behavior and shows the performance data during the execution. This execution is implemented by the hybrid inference mechanism.
(iii) In the prototype evaluation step, an expert system based on qualitative reasoning is implemented in order to automate diagnosis and improvement.

## Acknowledgment

Parts of this work have been supported by the Japanese Fifth Generation Computer Project and its organizing institute ICOT, as a subproject in the Intelligent Programming System. The authors would like to thank Ryuuzou Hasegawa of ICOT for his encouragement and support. The authors are grateful to Seiichi Nishijima and Takeshi Kohno of Systems & Software Engineering Laboratory, Toshiba Corporation, for their providing the essential support. The authors also wish to thank Jun Sawamura and Keisuke Shida for their helpful cooperation in developing BDES&BIES.

## References

[Bar82] D.Barstow et al. : An Automatic Programming System to Support an Experimental Science, *Proc. of 6th ICSE*, pp.360-366, 1982.
[Cha89] C.K.Chang et al. : Modeling a Real-Time Multitasking System in a Timed PQ Net, *IEEE Software*, pp.46-51, March 1989
[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
[DeM80] T.DeMarco : *Structured Analysis and System Specification*, Yourdon, New York, 1980
[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.
[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 Development Software, *JOHO SHORI*, Vol.30, No.4, pp.387-395, 1989
[Ito89b] K.Itoh etal. : 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, to appear in *Trans. on JSAI*.
[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.
[Sac74] E.D.Sacerdoti : Planning in a Hierarchy of Abstraction Spaces, *Artificial Intelligence* 5, pp.115-135, 1974
[Uch87] N.Uchihira et al. : Concurrent Program Synthesis with Reusable Component using Temporal Logic, *Proc. of Compsac '87*, pp.455-464, 1987.
[Zav84] P.Zave : The Operational versus the Conventional Approach to Software Development, *Comm. ACM*, Vol.27, No.2, pp.104-118, 1984

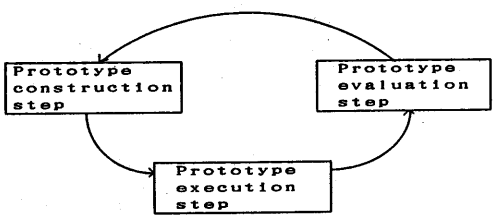


Fig.1 Prototyping process

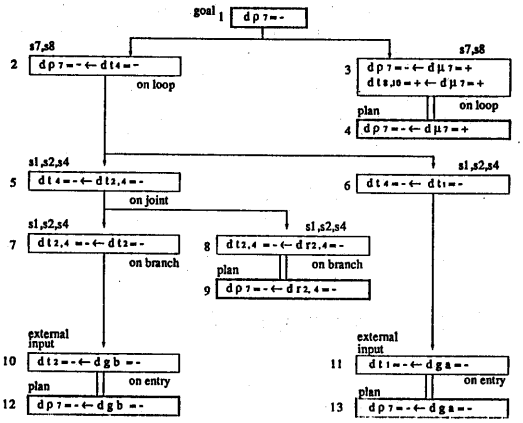


Fig.4 BDES qualitative reasoning process

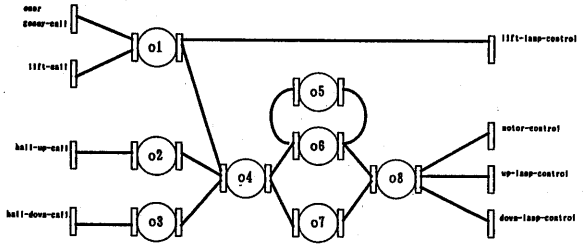


Fig.2 MENDEL OBJECTs

```
server with maximum ρ : (s7 .800)
server whose ρ ≧0.9 none.
server whose ρ ≧0.7
(s6 .770)(s7 .800)
Please imput 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),
         increase r (s2.s5)
*plan-3  decrease external input, gb
*plan-4 decrease external input, ga
```

Fig.5 BDES diagnosis process

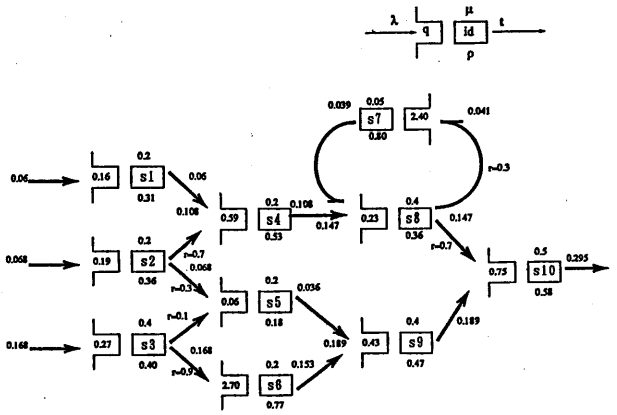


Fig.3 Queueing network

```
[network : nq10,
 bottleneck server to be improved   : s7,
 1st improvement by plan-2]

parameter to be improved
    r (s2,s5)   0.300 →0.780
    r (s2,s4)   0.700 →0.220
```

Fig.6 Improved parameters by BIES

Table.1 Comparison of $\rho$s and queue length

| | measurement1 | | $\rho$ improved | measurement2 | |
|---|---|---|---|---|---|
| | $\rho$ | q | by BIES | $\rho$ | q |
| s7 | 0.80 | 2.40 | 0.65 | 0.58 | 0.52 |